BladedFeline: Whispering in the dark

By ESET Research

Archived: 2026-04-05 15:09:19 UTC

In 2024, ESET researchers discovered several malicious tools in the systems used by Kurdish and Iraqi government officials. The APT group behind the attacks is BladedFeline, an Iranian threat actor that has been active since at least 2017, when it compromised officials within the Kurdistan Regional Government (KRG). This group develops malware for maintaining and expanding access within organizations in Iraq and the KRG. While this is our first blogpost covering BladedFeline, we discovered the group in 2023, after it targeted Kurdish diplomatic officials with the Shahmaran backdoor, and previously reported on its activities in ESET APT Activity reports Q4 2023-Q1 2024 and Q2 2024-Q3 2024.

The array of tools utilized in the recent campaign shows that since deploying Shahmaran, BladedFeline has continued to develop its arsenal. We found two reverse tunnels, a variety of supplementary tools, and most notably, a backdoor that we named Whisper and a malicious IIS module we dubbed PrimeCache. Whisper is a backdoor that logs into a compromised webmail account on a Microsoft Exchange server and uses it to communicate with the attackers via email attachments. PrimeCache also serves as a backdoor: it is a malicious IIS module related to what we referred to as Group 2 in our 2021 paper Anatomy of native IIS malware. Significantly, PrimeCache also bears similarities to the RDAT backdoor used by the Iran-aligned OilRig APT group.

Based on these code similarities, as well as on further evidence presented in this blogpost, we assess with medium confidence that BladedFeline is a subgroup of OilRig, an Iran-aligned APT group going after governments and businesses in the Middle East. We have previously reported on other activity linked to OilRig. To avoid confusion, we have since refined our OilRig tracking, and we now track both of those operations under a separate subgroup – Lyceum – within OilRig.

BladedFeline has worked consistently to maintain illicit access to Kurdish diplomatic officials, while simultaneously exploiting a regional telecommunications provider in Uzbekistan, and developing and maintaining access to officials in the government of Iraq. This blogpost details the technical aspects of the initial implants delivered to BladedFeline's targets, the links between the victims, and lays the groundwork for associating this subgroup with OilRig.

> **Key points of the blogpost:**
>
> - BladedFeline compromised officials within the Kurdistan Regional Government at least as early as 2017.
> - The initial implants used there can be traced back to OilRig.
> - We discovered BladedFeline after its operators compromised Kurdish diplomatic officials with the group's Shahmaran signature backdoor in 2023.
> - This APT group has also infiltrated high-ranking officials within the government of Iraq.
> - We assess with medium confidence that BladedFeline is a subgroup within OilRig.
> - We analyze two reverse tunnels (Laret and Pinar), a backdoor (Whisper), a malicious IIS module (PrimeCache), and various supplementary tools.

BladedFeline overview

BladedFeline is an Iran-aligned cyberespionage group, active since at least 2017 according to ESET telemetry. We discovered the group in 2023 when it deployed its Shahmaran backdoor against Kurdish diplomatic officials. Shahmaran, named after a mythical half-snake, half-woman creature from Iranian folklore, is a 64-bit portable executable that we found in the target's Startup directory. This simple backdoor doesn't use any compression or encryption for network communications. After checking in with the C&C server, the backdoor executes any operator commands provided, which include uploading and downloading additional files, requesting specific file attributes and providing file and directory manipulation API.

As evidenced by the campaign toolset we describe in this blogpost; since deploying Shahmaran, BladedFeline has continued to develop its malware in order to retain and even further extend its access to the KRG and to high levels within the government of Iraq (GOI). We uncovered the campaign in 2024 after finding BladedFeline's Whisper backdoor, PrimeCache IIS backdoor, and a set of post-compromise tools in the networks of Kurdish diplomatic officials, Iraqi government officials, and a regional telecommunications provider in Uzbekistan.

We detected and collected one version of Whisper and found another on VirusTotal, uploaded by a user in Iraq. They are virtually identical, and we were able to determine the likely identity of the VirusTotal uploader, based on data in the Whisper sample and other samples uploaded under the same submitter ID. PrimeCache, Flog (a webshell), and Hawking Listener (an early-stage implant that listens on a specified port) were all uploaded to VirusTotal by the same submitter ID who uploaded the Whisper samples. Based on the Whisper link and the close timeframe (both were uploaded within a matter of minutes) we believe it was deployed by BladedFeline to a victim in Iraq's government. Some of the tools mentioned below in the *Timeline* are discussed later in the report (e.g., Slippery Snakelet).

Timeline

```
2017-09-21 ● VideoSRV reverse shell on KRG system
           |
2018-01-30 ● RDAT backdoor on KRG system
           |
2019-07-09 ● Custom Plink on KRG system
           |
2021-05-01 ● Sheep Tunneler on KRG system
           |
2023-01-23 ● LSASS dumped on KRG system
           |
2023-02-01 ● Shahmaran backdoor on KRG system
           |
2023-03-25 ● First victim targeted at a telecommunications company in Uzbekistan
           |
2023-06-12 ● Shahmaran version 2 on KRG system for access maintenance
           |
2023-12-14 ● BladedFeline operators executing CLI commands on KRG system
           |
2023-12-16 ● Slippery Snakelet backdoor on KRG system
           |
2023-12-20 ● P.S. Olala (a PowerShell executor) on KRG system
           |
2023-12-20 ● PsExec on KRG system
           |
```

```
2024-01-07 ● Whisper backdoor on KRG system
           |
2024-02-01 ● Laret reverse tunnel on KRG system
           |
2024-02-20 ● Pinar reverse tunnel on KRG system
           |
2024-02-29 ● PrimeCache malicious IIS module uploaded to VirusTotal
           |
2024-03-11 ● Whisper version 2, Flog, and Hawking Listener uploaded to VirusTotal
```

Attribution

Our attribution of this campaign to BladedFeline is based on the following:

- The campaign targets members of the KRG, as have previous attacks conducted by BladedFeline.
- The original attack activity targeting the KRG organization allowed us to identify successive malware, as BladedFeline has attempted to maintain and expand access to the organization.
- Further analysis of the attacks led us to identify the telecommunications victim in Uzbekistan.
- At the same time, looking into the Whisper backdoor helped us identify the GOI victim.

We assess that BladedFeline is targeting the KRG and the GOI for cyberespionage purposes, with an eye toward maintaining strategic access to high-ranking officials in both governmental entities. The KRG's diplomatic relationship with Western nations, coupled with the oil reserves in the Kurdistan region, makes it an enticing target for Iran-aligned threat actors to spy on and potentially manipulate. In Iraq, these threat actors are most probably trying to counter the influence of Western governments following the US invasion and occupation of the country.

We believe with medium confidence that BladedFeline is a subgroup of OilRig:

- As does OilRig, BladedFeline targets organizations in the Middle East with the purpose of cyberespionage.
- We have found OilRig tools (VideoSRV and RDAT) in a compromised KRG system.
- BladedFeline's malicious IIS module PrimeCache shares code similarities with OilRig's RDAT.

BladedFeline is not the only subgroup of OilRig that we are monitoring: we have already been tracking Lyceum, also known as HEXANE or Storm-0133, as another OilRig subgroup. Lyceum focuses on targeting various Israeli organizations, including governmental and local governmental entities and organizations in healthcare. Major tools we attribute to Lyceum include DanBot, the Shark, Milan, and Marlin backdoors, Solar and Mango, OilForceGTX, and a variety of downloaders using legitimate cloud services for C&C communication.

We will continue to use the name OilRig to refer to the parent group, also known as APT34 or Hazel Sandstorm (formerly EUROPIUM). OilRig is a cyberespionage group that has been active since at least 2014 and is commonly believed to be based in Iran. The group targets Middle Eastern governments and a variety of business verticals, including chemical, energy, finance, and telecommunications. Notable OilRig campaigns include the 2018 and 2019 DNSpionage campaign, targeting victims in Lebanon and the United Arab Emirates; the 2019–2020 HardPass campaign, using LinkedIn to target Middle Eastern victims in the energy and government sectors; the 2020 attack against a telecommunications organization in the Middle East using the RDAT backdoor; and the 2023 attacks targeting organizations in the Middle East with the PowerExchange and MrPerfectionManager backdoors.

OilRig tools used by BladedFeline

We have found two OilRig tools on the KRG machines compromised by BladedFeline.

RDAT

We discovered a previously unreported version of the OilRig backdoor RDAT on two KRG victim systems. Analyzing RDAT, we found that the operational flow (see Unit 42's report for specifics), compilation timestamp (2017-12-26 10:49:35), and file write time (2018-01-30) align with OilRig activity and targeting, particularly with regard to the group's 2017 activity. We observed a file with an SHA-1 of 562E1678EC8FDC1D83A3F73EB511A6DDA08F3B3D and a path of C:\Windows\System32\LogonUl.exe on both systems. The PDB path also corroborates that this binary is RDAT: C:\Users\Void\Desktop\RDAT\client\x64\Release\client.pdb. To date, we have only ever observed RDAT in use by OilRig. Moreover, we have not seen any custom implant sharing between OilRig and other Middle Eastern groups, and it seldom occurs between Iran-aligned threat actors.

Further bolstering the case that BladedFeline is an OilRig subgroup, as with Lyceum, is the analysis linking RDAT with PrimeCache, a malicious IIS module that was uploaded to VirusTotal presumably by the GOI victim. This link is explored in more depth in the *Links with OilRig* section of the blogpost.

VideoSRV

One additional data point on the OilRig and BladedFeline connection is a reverse shell deployed to one of the KRG victims (September 21st, 2017) prior to RDAT getting dropped on the same system (January 30th, 2018). VideoSRV (SHA-1: BE0AD25B7B4834798490817540499653 1CFD74B7), so named for its filename videosrv.exe, has the PDB string C:\Users\v0id\Desktop\reverseShell\clientProxy\x64\Release\ConsoleApplication1.pdb, which bears some similarities to the RDAT PDB string C:\Users\Void\Desktop\RDAT\client\x64\Release\client.pdb.

Technical analysis

Initial access

It is still unclear how BladedFeline is developing access to its victims. What we know is that in the case of the KRG victims, the threat actors obtained access at least as far back as 2017 and have maintained it ever since. As for the GOI victims, we suspect that the group exploited a vulnerability in an application on an internet-facing web server, which allowed them to deploy the Flog webshell.

Toolset

PrimeCache – malicious IIS module

PrimeCache, whose name we derived from the RTTI AVRSAPrimeSelector and its filename (cachehttp.dll), is a passive backdoor implemented as a native IIS module with an internal name of HttpModule.dll. It was uploaded to VirusTotal by the same user who uploaded one of the Whisper backdoor samples. It is a 64-bit C++ DLL with a compilation timestamp of 2023-05-14 06:55:52 and has a minimized PDB string of just HttpModule.pdb. It has a single export: RegisterModule.

PrimeCache is a successor to a collection of unattributed IIS backdoors that we have previously reported as Group 2 (simple IIS backdoors) in our 2021 blogpost, Anatomy of native IIS malware. We obtained those original samples from VirusTotal where they were uploaded by users from Bahrain, Israel, and Pakistan, between 2018 and 2020. Based solely

on the location of the presumed victims, it is possible that those cases were also related to BladedFeline – or, more broadly, OilRig – activities.

Main functionality

PrimeCache's main functionality is implemented in the CGlobalModule::OnGlobalPreBeginRequest handler. This is a unique implementation, differing from its predecessors, which used the CHttpModule::OnBeginRequest handler. PrimeCache filters incoming HTTP requests, only processing those from the BladedFeline operators, which are recognized by having a cookie header with the structure:

F=<command_ID>,<param>;

Note that this value can be standalone or embedded into a longer cookie, surrounded by semicolon (;) characters.

The backdoor works in an unusual way (new with this version as compared with our 2021 analysis). Rather than accepting a backdoor command and all its parameters within a single HTTP request, each action is split into multiple requests. First, the BladedFeline operator sends an individual request for each single parameter; these parameters are stored in a global structure. Then the operator sends another request to trigger the backdoor command. Finally, PrimeCache uses the previously received parameters to execute the specified action, and then clears the cached parameters.

Operator commands

There are three types of requests that can be received by the backdoor, as shown in Table 1.

Table 1. PrimeCache operator commands

<command_ID>	Parameter	Description
1	Format: <key>=<value>	Clears the list of previously stored parameters and adds the new value. Most parameters are encrypted; see *Encryption* below.
0	Not used.	Triggers the backdoor action, using previously transmitted backdoor parameters.
Other	Format: <key>=<value>	Adds the specified value to the list of stored parameters (doesn't clear the list). Most parameters are encrypted; see *Encryption* below.

Once the action is triggered (via <command_ID>=0), PrimeCache performs an action, based on the previously obtained parameters, as shown in Table 2. One note on the chart below:

The PrimeCache action is operator command (OpCom) a, the session key is OpCom k, binary data is OpCom b, and the filename is OpCom f.

Table 2. PrimeCache post-operator command actions

PrimeCache action	Session key	Binary data	Filename	Command description	Return value
r	RSA-encrypted session key	AES-encrypted command line	Null	Runs the specified command via popen.	Command output
r2				Runs the specified command via CreateProcessW.	
r3				(Presumably) runs the specified command by sending it to another (unknown) process via the named pipe \\.\pipe\iis, then reads (presumably) the command output from the same pipe.	
u		AES-encrypted file content	Local filename	Creates a local file with the specified name and content.	OK
d		Null		Exfiltrates the given file from the compromised IIS server.	File content

Encryption

Similar to its predecessors, PrimeCache uses both RSA and AES-CBC for its C&C communication. The parameters and the return values are always AES-CBC encrypted using the session key, then base64 encoded. The session key is RSA encrypted; the backdoor has a hardcoded private and public RSA key (not a pair) to handle both directions of the communication.

A statically linked Crypto++ library is used to handle the encryption and decryption operations.

C&C communications

Operator commands are transmitted in the cookie header (another deviation from earlier versions, which used the URL or the HTTP request body). PrimeCache responses are added to the HTTP response body. If a file is being exfiltrated, the Content-Type header is set to attachment, matching the functionality of the previous versions.

The PrimeCache predecessors also used the same encryption scheme, and similar parameter names (a, c, f, k), but all were sent to the backdoor in a single request. The only supported commands were r, u, and d.

Links with OilRig

When we compare PrimeCache with RDAT, as described in the *RDAT* attribution subsection, we see several similarities that support our supposition that BladedFeline is a subgroup of OilRig.

- Both RDAT and PrimeCache use the Crypto++ library, and both parse the backdoor commands using the regular expression [^,]+.

- The payload attempts to parse the decrypted cleartext using the regular expression [^,]+ to get the command value and the command arguments that are split with a comma.

- Both share a function, shown in Figure 1, that executes a shell command and reads the output, which, across our corpus, is found only in these two pieces of malware.

Figure 1. A unique function to execute a shell command, shared between RDAT (left) and PrimeCache backdoors (right)

Whisper backdoor

Whisper is a 32-bit Windows binary written in C#/.NET, named after its PDB strings G:\csharp\Whisper_Trojan_winform\Whisper_Trojan_winform\Whisper_Trojan_winform\obj\Release\Veaty.pdb and Z:\csharp\Whisper_Trojan_winform_for_release\Whisper_Trojan_winform\Whisper_Trojan_winform\obj\Release\Veaty.pdb. It uses a Microsoft Exchange server to communicate with the attackers by sending email attachments via a compromised webmail account. We have seen two versions of the backdoor: we detected and collected one version, and was uploaded to VirusTotal from Iraq. These samples are virtually identical, but we were able to determine the likely identity of the VirusTotal uploader based on data in the Whisper sample and other samples uploaded by that user.

Both these versions of Whisper have timestomped compilation timestamps (2090-04-11 23:38:14 and 2080-12-11 03:50:47). They are built using Costura, presumably to ensure that the victim's system uses the DLLs packaged with the binary and not DLLs in the Global Assembly Cache.

Whisper's operation is not the first time we have observed an OilRig subgroup using cloud services for its C&C protocol. While, unlike with Whisper, there were no emails actually being sent, Lyceum used email drafts for communication between its malware and operators throughout 2022, as we described in a previous blogpost.

Operational workflow

Whisper does not require or accept any arguments. Instead, its dropper – which we dubbed Whisper Protocol after its filename, Protocol.pdf.exe – writes its configuration file to disk alongside it (see the *Whisper Protocol* section). The config file, shown in Figure 2, is in XML format with its key and value strings base64 encoded. It is called by the Specs class of Whisper, which uses a function – DelockItems – to base64 decode the config variables.

Figure 2. Whisper configuration file with its base64-encoded elements (left) and decoded (right)

Figure 3 shows the operational flow of Whisper, which we detail in the following paragraphs.

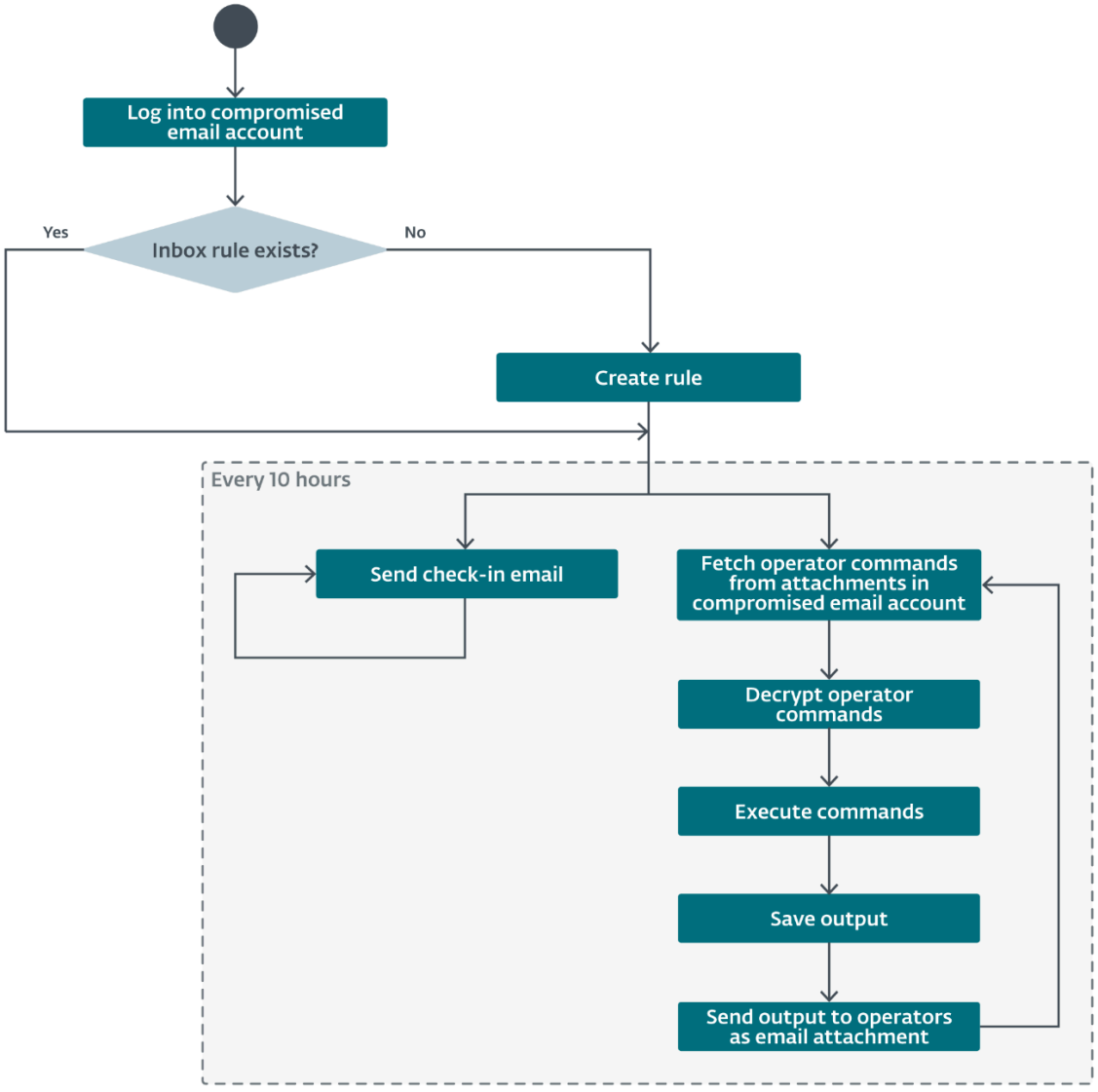

Figure 3. Basic operational flow of Whisper

Whisper's operational flow can be broken down into seven steps:

In Step 1, Whisper uses the credentials from the config file (line 15 in Figure 2) and the Microsoft Exchange Web Services class ExchangeService to attempt to log into compromised webmail accounts. Once Whisper successfully logs into an account, it saves the credentials in memory and writes the following to the log file c:\Windows\Temp\WindowsEventLogs.txt:

---------- ItemContext is set: username [<username>] , use_defaultCred: [credentials>]

If no credentials in the config file are valid, Whisper logs the following error messages to the log file:

---------- there was No Way to access any MailBox.

_____ Extraction function is called.

If an unexpected error is caught, Whisper writes the following to the log file (note the misspelling of the word happened, indicative of a non-native English speaker) and exits using the Environment.Exit(Int32) method. Strangely, the exitCode used, 0, indicates that the process completed successfully.

----------__ an unknown Exception happend. program turned off

Next, in Step 2, Whisper uses the credentials from the previous step to check for inbox rules using the ExchangeService.GetInboxRules method (which *[r]etrieves a collection of Inbox rules that are associated with the specified user*). Using the value in line 13 of the configuration file (key="receive_sign", value="PMO"), Whisper iterates over the inbox rules looking for that value to be specified in one of three places: subject, body, or subjectorbody and for emails matching that value to be sent to a specified location (deleteditems or inbox, depending on the version of Whisper). If the inbox has such a rule, Whisper goes to the next step; otherwise, Whisper creates a rule with the given parameters:

- Rule name: MicosoftDefaultRules.
- Move to folder: deleteditems or inbox.
    - One version of Whisper specifies the deleteditems folder; the other points to the inbox. Both are hardcoded in the separate binaries.
- Mark as read: true.
- Condition: subject contains PMO.
    - The location to look for the string, subject, is hardcoded in both versions of Whisper. The string to look for, PMO, is in the configuration file used by Whisper; we were unable to collect the other configuration file.

In Step 3, Whisper initiates a never-ending do loop that sends a check-in email message from the compromised email account in Step 1 to an email address specified in the configuration file (line 16, key="alive_mail"). The check-in message is sent every 10 hours (line 10 in the configuration file, key="al_time"; in minutes), the subject (line 17, key="alive_msg_subj") is Content, and the message body contains the string defined below:

"Content ID: " + base64_encode("COMPUTERNAME:USERDNSDOMAIN:USERNAME")

Next, in Step 4, Whisper fetches operator commands. It does so by searching the inbox identified in Step 1 for files in a given folder (deleteditems or inbox, depending on the version of Whisper) with attachments where the subject matches a string (supplied in the configuration file; PMO in the only configuration file we collected). For matching emails with attachments, Whisper scrapes the attachment body (which should contain encrypted commands) and stores the sender's email address for use later as the C&C server to which operator command results are uploaded.

In Step 5, Whisper decrypts the operator commands. It does so by first base64 decoding the string containing the command and then decrypting the result using the .NET AES class with a 16-byte initialization vector and the encryption key found in the configuration file (line 18, key="enc_key" value="cXdlcmFzZHp4Y3ZmZ2d0aGhsZGZvZ2g/bHZtZ2xrZyE="). Decrypted commands are in the form of

<cmd_id>;<command_to_execute>. The command ID, commands, and command output are saved in the following format:

base64-encoded(<command_id>: <cmd_id>\n<cmd_output>\n)

Then, in Step 6, Whisper executes the backdoor commands and records the results. Possible commands include:

- Write a file to disk

The data written to disk is:

this is my file content

<filepath>

<filename>

<nbytes-to-write>

The bytes to write are base64 encoded (and decoded before writing to disk). Successful execution returns:

file received properly. wrote to: <filepath>\<filename>

- Send a file to the C&C server

This command is prefixed with this is my required file path followed by \n<unknown_variable>\n<filepath>\<filename>. Whisper reads the contents of the file into memory, base64 encodes them, and returns:

this is my required file <path>\n<unknown_variable>\n<filename>\n<base64_encoded_file_contents>

- Execute a PowerShell script

This command does not have a prefix and instead only contains a plaintext command that PowerShell is capable of executing, postfixed with a pipe after which Whisper appends Out-String. Output is saved in this form:

base64-encoded(<command_id>: <cmd_id>\n<cmd_output>\n)

Finally, in Step 7, Whisper sends the command output in an email message to the C&C inbox found in Step 4. The email is formatted with these particulars:

- sending email address: inbox from Step 1,
- recipient: email address from Step 4,
- subject: Email (from the configuration file, line 14, key="send_sign"),
- message body: Hey There! find your results in the attachment (hardcoded in the binary), and
- attachment: output from the commands in Step 6, encrypted with the same encryption key in Step 5 (configuration file line 18, key="enc_key" value="cXdlcmFzZHp4Y3ZmZ2d0aGhsZGZvZ2g/bHZtZ2xrZyE=").

Steps 4–7 continue in a loop using the same check-in schedule from Step 3 until the credentials hardcoded in the configuration file are changed.

Shahmaran backdoor

The Shahmaran backdoor, named after a mythical half-snake, half-woman creature from Iranian folklore, is a 64-bit PE that was found in the startup folder as:

%ROAMINGAPPDATA%\Microsoft\Windows\Start Menu\Programs\Startup\adobeupdater.exe

At system startup, Shahmaran creates a Windows event object, SysPrep. It is possible that the Shahmaran developers chose SysPrep as the event name to blend into the background noise, as SysPrep is part of the Windows imaging process. Windows admins use it to create a standard Windows image (often referred to as a Gold or Golden image) before deployment to enterprise systems. Figure 4 shows the SysPrep event object on a compromised system as seen by Sysinternals' WinObj.

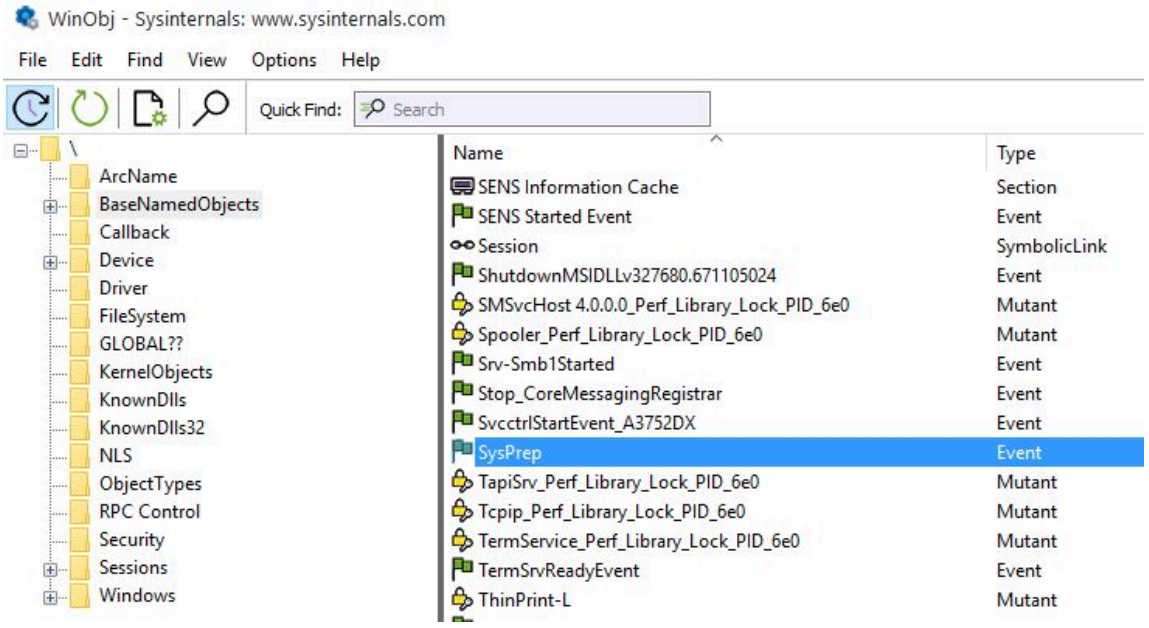

Figure 4. Sysinternals' WinObj showing the SysPrep *event object on a compromised system*

The C&C domain is hardcoded, olinpa[.]com, as is the port, 80, and the User-Agent string, of which there are two. The initial connection to the C&C uses an incomplete User-Agent string (it is missing the closing parenthesis):

Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.0

Subsequent communication with the C&C uses the corrected User-Agent string:

Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.0)

Shahmaran does not use any compression or encryption for network communications. And while the port is hardcoded (80), there are code fragments that check for the port in use and update communication variables if port 443 is used.

After checking in with the C&C server, Shahmaran executes any operator commands provided, returns any output from those commands, then sleeps for 30 seconds before checking in with the C&C server again, ad infinitum. Table 3 shows the available operator commands and their functions.

Table 3. Operator commands and their descriptions

Operator command	Description
1 <path/filename>	Returns the datetime that the specified file was written to disk in UTC, prepended with id= and in the format YYYY/MM/DD HH:MM:SS.
2 <filename> <source> <destination>	Moves the specified file to the specified location. Returns the output of the file move operation prepended with id=.
3 <path/filename>	Deletes the specified file. Returns the output of the file delete operation prepended with id=.
4 <path/directory>	Creates the specified directory. Returns the output of the directory creation operation prepended with id=.
5	Creates a log file in the hardcoded location c:\programdata\~tmp.log, if it does not already exist. If the file already exists, reads the contents and returns them to the C&C server with the file's timestamp in UTC and in the format YYYY/MM/DD HH:MM:SS, then deletes the file. If the file does not exist, returns the filename and path. If an error occurs, returns the error. All returned data is prepended with s=.
6 <path/filename> <data>	Checks for the specified file. If found, writes the provided data to the file and returns s= <provided_filename>. If not found, returns u=<error_code>.
7 <path/filename>	Creates the specified file. Returns s= appended with either the filename (success) or an error code.
8 <path/filename>	Checks for the presence of the specified filename in a compressed folder in the specified location on disk and creates it if it does not exist. Returns s= appended with the filename and the timestamp in UTC in the format YYYY/MM/DD HH:MM:SS. The timestamp is used to determine whether the file was already present or was just created.

After executing an operator command, Shahmaran sends the output to the C&C server using the format t= <operator_command>&<command_output>, such as t=1&s=<file_timestamp>.

Slippery Snakelet backdoor

Slippery Snakelet is a small Python-based backdoor with limited capabilities:

1. executes a command via cmd.exe,

2. downloads a file from a URL, and

3. upload a file to the /newfile/ URI path.

Slippery Snakelet has a hardcoded C&C server, zaincell[.]store, and communicates with it via URLs of the form https://zaincell[.]store/request/<UID>, where the <UID> is the victim's login domain and the compromised computer's

name separated by a period then base64 encoded (e.g., victim_domain.computer_name = dmljdGltX2RvbWFpbi5jb21wdXRlcl9uYW1l).

Slippery Snakelet also has this hardcoded User-Agent:

Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/88.0.4324.104 Safari/537.36

The C&C server was disguised as an Arabian Gulf E-Learning site and the default HTML landing page does not contain any commands. When Slippery Snakelet supplies a correctly formatted request (e.g., https://zaincell[.]store/request/<UID>), the C&C server inserts <code> tags such as <code>6wjTyB3Y20KSzU1VUlTagp3aG9hbWkKbnVsbApudWxs</code> into the page, and Slippery Snakelet collects and decodes these.

Slippery Snakelet base64 decodes everything from the eighth character to the end of the string (i.e., Y20KSzU1VUlTagp3aG9hbWkKbnVsbApudWxs in the example above). The decoded output is newline separated and contains the five items described in Table 4

Table 4. Slippery Snakelet arguments and options

Commands	Options	Example
Command Type	cm (execute cmd.exe command) getfl (download a file) sendfl (upload a file)	cm
Command ID	CMID (a random string)	K55UISj
Command | FileUrl | FilePath	Respectively for cm | getfl | sendfl	whoami
Null | SavePath | FilePath	Respectively for cm | getfl | sendfl	null
Null	Unknown	null

Laret and Pinar – reverse tunnels

Laret and Pinar, whose names are derived from the internal names in each respective file, are 32-bit Windows binaries written in C#/.NET. Both have timestomped PE compilation timestamps – a tactic that is common amongst Middle Eastern (and particularly Iran-nexus) threat groups – of 2058-02-07 00:12:48 and 2072-07-10 18:26:15, respectively. Both were found on two systems at the locations in Table 5.

Table 5. Locations of Laret and Pinar on disk, along with filenames

Reverse tunnel	Location
Laret	%APPDATA%\Local\LEAP Desktop\LEAPForm.exe
	<unknown_location>\wincapsrv.exe
Pinar	C:\Program Files\LEAP Office\SystemMain.exe

Reverse tunnel	Location
	C:\Program Files\LEAP Office\winhttpproxy.exe

In the case where we do not have a location on disk for Laret but we do have the filename (wincapsrv.exe), we could see that Laret was downloaded from http://178.209.51[.]61:8000/wincapsrv.exe via PowerShell. Unfortunately, we did not manage to discover where it was written to disk. Attempts to enumerate the IP and download the file were rebuffed by the C&C server, likely indicating that some form of compromised host identification is required in the connection setup (which we do not have).

Regarding writing to disk, BladedFeline operators likely timestomped the file creation date of Pinar to 2017-09-14 14:56:00 on one of the two compromised systems. How the file creation date was timestomped is an open question, but it shows that the attackers have compromised these two systems to such an extent that they probably have administrative rights.

At runtime, both Laret and Pinar rely on a configuration file in the same directory as their binaries for eight required variables, which are listed in Table 6.

Table 6. Laret and Pinar configuration parameters with default hardcoded values

Field	Description	Default value
ssh_host	C&C IP address.	N/A
ssh_port		22
ssh_username	C&C username.	N/A
ssh_pass	C&C password.	N/A
local_port		9666
process_file	File to execute before executing any reverse tunnel actions.	N/A
wait_time_minutes	Time to wait between check-ins with the C&C server.	10f (271)
remote_port	Port number used for port forwarding.	1234

We have thus far not collected the configuration file but have reconstructed its likely content, found in Figure 5, based on code analysis. Reading from the configuration file is done by base64 decoding the encoded string to bytes, which results in strings of space-delimited, hexadecimal-encoded character values, which in turn are decoded into ASCII strings.

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <appSettings>
    <!-- ... -->
    <add key="NzMgNzMgNjggNWYgNjggNmYgNzMgNzQ=" value="<base64_encoded_ip_address>"/>
    <add key="NzMgNzMgNjggNWYgNzAgNmYgNzIgNzQ=" value="<base64_encoded_remote_ssh_port_number>"/>
    <add key="NzMgNzMgNjggNWYgNzUgNzMgNjUgNzIgNmUgNjEgNmQgNjU=" value="<base64_encoded_user_name>"/>
    <add key="NzMgNzMgNjggNWYgNzAgNjEgNzMgNzM=" value="<base64_encoded_password>"/>
    <add key="NmMgNmYgNjMgNjEgNmMgNWYgNzAgNmYgNzIgNzQ=" value="<base64_encoded_local_port_number>"/>
    <add key="NzAgNzIgNmYgNjMgNjUgNzMgNzMgNWYgNjYgNjkgNmMgNjU=" value="<base64_encoded_file_name>"/>
    <add key="NzcgNjEgNjkgNzQgNWYgNzQgNjkgNmQgNjUgNWYgNmQgNjkgNmUgNzUgNzQgNjUgNzM=" value="<base64_encoded_wait_time_between_check-ins>"/>
    <add key="NzIgNjUgNmQgNmYgNzQgNjUgNWYgNzAgNmYgNzIgNzQ=" value="<base64_encoded_remote_port_number>"/>
    <!-- ... -->
  </appSettings>
</configuration>
```

Figure 5. Example contents of the configuration file used by Laret and Pinar reverse tunnels

The BladedFeline developers refer to this as Delocking and the opposite (writing to the configuration file) as Enlocking. This probably indicates a passing familiarity with English, but the developers were far from proficient. Other examples of weak translation skills include:

- time Alapsed and client not connected
- aerpoo after
- Waiting connection ...
- error in creaate ssh client

Interestingly, at another point in the reverse tunnels, the developers correctly spelled the word elapsed (time elapsed!), which is indicative of poor coding and lax code review, if any is performed (e.g., there is a lot of command result text output to the command line, as if the reverse tunnels were shipped immediately after successful testing was completed).

The actual function and flow of Laret and Pinar after collecting the parameters from the configuration file is quite banal, but that is probably an intentional effort to blend in. Both look for a filename in the process_file parameter and, if a file matching the supplied name is present, execute it and start two threads:

1. Sets up an SSH connection to the C&C IP in the configuration file using the Core.Renci.SshNet DLL included within the binary. Port 22 is hardcoded as the C&C port and port forwarding is also enabled, using the remote_port variable from the configuration file.
2. Sets up a listener on the port specified in the local_port parameter of the configuration file. Note that any data sent to the listener is done in the clear (i.e., no encryption or obfuscation is used beyond extra \0 characters that are removed at the time of receipt by Laret and Pinar).

If no file is specified in process_file, both Laret and Pinar skip setting up a listener port.

Laret and Pinar only differ significantly in that Pinar sets up a service, called Service1, for persistence prior to executing the two threads. Laret has no means of persistence beyond its process running indefinitely.

Supplementary tools

Flog webshell

Flog is a webshell found uploaded to VirusTotal from Iraq by the same submitter who uploaded one of the versions of Whisper. Based on that and the close timeframe (both were uploaded within a matter of minutes) we believe it was deployed by BladedFeline to the victim in the Iraq government.

Flog, so named for its filename – flogon.aspx – looks for specific input from the BladedFeline operators of the form <password>=<(a|b|c|d)>#<path>

Flog hashes the password, which must match the MD5 checksum 4CC88CE123B0DA8D75C0FE66A39339F6.

Variables (a|b|c|d) are command options:

- a returns, for the path provided, a directory listing and the byte length of each file,
- b creates a file on disk, using the path provided,
- c splits the path variable on a pipe and writes a file to disk where the first part of the path is the filename and the second part is the data to write, and
- d deletes a specified file given in the supplied path.

Hawking Listener

Hawking Listener, so named for its PDB string – C:\Users\g18u04\source\repos\Hawking\Hawking\obj\Release\listner.pdb – is a 32-bit .NET/C# Windows binary with a timestomped compilation time of 2057-11-14 16:59:12. It was also uploaded to VirusTotal by the same user who uploaded Flog and is probably a BladedFeline tool. It implements the .NET HTTPListener class to set up a listener with a hardcoded URL (which we cannot disclose in this case without revealing the identity of the victim). Alternatively, Hawking can be provided at runtime with URLs for the listener socket to monitor.

Hawking listens for a provided QueryString (from a BladedFeline operator) with snmflwkejrhgsey as the key in the key-value pair. Once received, Hawking executes the value in cmd.exe and returns the output. To stop Hawking, operators need only send stop as the key in the QueryString with a non-null variable in the value.

Hawking logs all interactions, runtime arguments, and command output to the file log.txt in its working directory.

P.S. Olala

P.S. Olala is a 32-bit .NET binary named for its intended function (executing PowerShell scripts) and its PDB path G:\csharp\psExecuterService\ewsService\obj\Release\Olala.pdb. It does not accept any runtime arguments. Rather, at runtime, P.S. Olala uses the Run(ServiceBase[]) method of the .NET ServiceBase class to register itself as a service with the Service Control Manager (for persistence).

When the P.S. Olala service is called, it spawns a thread and executes the function mainLoop, shown in Figure 6. Essentially, P.S. Olala is an executor of the PowerShell script stored in %APPDATA%\Local\Microsoft\InputPersonalization\TrainedDataStore.ps1.

```
public static async void mainLoop()
{
    string fPa = "C:\\Users\\[redacted]\\AppData\\Local\\Microsoft\\InputPersonalization\\TrainedDataStore.ps1";
    PowerShell iss = PowerShell.Create();
    string script = File.ReadAllText(fPa);
    iss.AddScript(script).AddParameter("-exec bypass").Invoke();
    m_oTimer1 = new System.Timers.Timer(60000.0);
    m_oTimer1.Elapsed += delegate
    {
        script = File.ReadAllText(fPa);
        iss.AddScript(script).AddParameter("-exec bypass").Invoke();
    };
    m_oTimer1.Enabled = true;
    CancellationTokenSource cancellationTokenSource = new CancellationTokenSource();
    await Task.Delay(-1, cancellationTokenSource.Token).ConfigureAwait(continueOnCapturedContext: false);
}
```

Figure 6. The main function of P.S. Olala

Unfortunately, we were unable to collect any of the TrainedDataStore.ps1 scripts. However, contextual information indicates it is likely an executor of the Whisper backdoor, or one of the reverse tunnels (Laret or Pinar). The entire flow (P.S. Olala → TrainedDataStore → Whisper/Laret/Pinar) is probably an elongated persistence chain aiming to maintain access.

Sheep Tunneler

Sheep Tunneler, a custom tunneling application that we named based on the PDB string C:\Users\sheep\source\repos\MP\MP\obj\Release\MP.pdb), has been observed in the two following locations:

- %APPDATA%\Local\Microsoft\Windows\Ringtones\RingService.exe
- %APPDATA%\Local\Microsoft\Windows\Shell\mspsrv.exe

Sheep Tunneler can be executed in two modes: network tunneling (by using the runtime argument middle) or connect back (by using the arguments cb <ip>:<port>).

Whisper Protocol

Whisper Protocol, so named for its filename (Protocol.pdf.exe) is a 64-bit Python-compiled Windows binary with a compilation timestamp of 2024-03-11 09:01:20. It creates a folder in C:\ProgramData\VeeamUpdate and writes both Whisper and its configuration file to that folder. Whisper Protocol also copies itself to %APPDATA%\Roaming\Microsoft\Windows\Start Menu\Programs\Startup\VeeamUpdate.lnk for persistence. Finally, it executes Whisper and exits gracefully.

Conclusion

BladedFeline is an advanced threat group that specializes in targeting Iraqi and Kurdish victims, specifically governmental officials and organizations. We assess that the group is likely a subgroup of OilRig. We expect to find that BladedFeline will persist with implant development in order to maintain and expand access within its compromised victim set, likely for cyberespionage.

> *For any inquiries about our research published on WeLiveSecurity, please contact us at threatintel@eset.com.*
>
> *ESET Research offers private APT intelligence reports and data feeds. For any inquiries about this service, visit the ESET Threat Intelligence page.*

IoCs

Files

SHA-1	Filename	Detection	Description
01B99FF47EC6394753F9 CCDD2D43B3E804F9EE36	Avamer.pdf.exe	Python/Trojan Dropper.Agent.GI	Python-compiled dropper for Spearal
1C757ACCBC2755E83E53 0DDA11B3F81007325E67	Win_Updates.exe	MSIL/Agent.EUM	Spearal, a BladedFeline backdoor.
272CF34E8DB2078A3170 CF0E54255D89785E3C50	scr8B45.ps1	PowerShell/Trojan Dropper.Agent.AJU	PowerShell script to install Spearal.
37859E94086EC47B3665 328E9C9BAF665CB869F6	ncms_demo.msi	MSIL/Agent.EUM	MSI inside the zip archive that drops and executes a PowerShell script that in turn drops and executes Spearal.
3D21E1C9DFBA38EC6997 AE6E426DF9291F89762A	flogon.aspx	ASP/Agent.BI	Flog webshell.
4954E8ACE23B48EC55F1 FF3A47033351E9FA2D6C	winsmsrv.exe	MSIL/HackTool .Agent.YN	Pinar, a reverse tunnel.

SHA-1	Filename	Detection	Description
562E1678EC8FDC1D83A3F73EB511A6DDA08F3B3D	LogonUl.exe	Win64/OilRig_AGen.A	RDAT backdoor.
66BD8DB40F4169C7F0FCA3D5D15C978EFE143CF8	Protocol.pdf.exe	Python/Trojan Dropper.Agent.FT	Whisper Protocol, the dropper that writes and executes the Whisper backdoor.
6973D3FF8852A3292380B07858D43D0B80C0616E	VeeamUpdate.exe	MSIL/Agent.ERR	Whisper backdoor.
73D0FAA475C6E489B2C5C95BB51DEDE4719D199E	winhttpproxy.exe	MSIL/HackTool.Agent.XY	Pinar, a reverse tunnel.
B8AFC21EF2AA854896B97F1C81B376DCDDE2466D	RunExeActionAllowedList.exe	MSIL/Agent.ERR	Whisper backdoor.
BB4FFCDBFAD40125080C13FA4917A1E836A8D101	MFTD.exe	MSIL/Tiny.GL	Hawking Listener.
BE0AD25B7B48347984908175404996531CFD74B7	videosrv.exe	Generik.BKYYERR	VideoSRV, a reverse shell.
E8E6E6AFEF3F574C1F5228BDB28ABB34F8A0D09A	wincapsrv.exe	MSIL/HackTool.Agent.XY	Laret, a reverse tunnel.
F28D8C5C2283019E6ED788D20240ABC8554CADB5	N/A	MSIL/Agent.EUM	Zip archive that contains an MSI that drops and executes a PowerShell script that in turn drops and executes Spearal.

Network

IP	Domain	Hosting provider	First seen	Details
178.209.51[.]61	N/A	Nine Internet Solutions AG	2023-12-18	Distribution server for BladedFeline's Laret reverse tunnel.
185.76.78[.]177	N/A	EDIS GmbH - Noc Engineer	N/A	C&C used by Spearal.

MITRE ATT&CK techniques

This table was built using version 17 of the MITRE ATT&CK framework.

Tactic	ID	Name	Description
Reconnaissance	T1595.002	Active Scanning: Vulnerability Scanning	BladedFeline probably conducts vulnerability scanning against targets to identify potentially vulnerable, exposed applications.
Resource Development	T1583.001	Acquire Infrastructure: Domains	BladedFeline registers domains to use for C&C servers.
	T1583.003	Acquire Infrastructure: Virtual Private Server	BladedFeline uses VPS services to host C&C servers.
	T1583	Acquire Infrastructure	BladedFeline uses IPs for network infrastructure, including distributing malware and C&C servers.
	T1586.002	Compromise Accounts: Email Accounts	BladedFeline uses compromised email accounts as C&C servers.
Initial Access	T1190	Exploit Public-Facing Application	BladedFeline probably exploits vulnerable public-facing applications for initial access.
Execution	T1059.003	Command and Scripting Interpreter: Windows Command Shell	BladedFeline uses the Windows Command Shell to execute commands on compromised endpoints.
	T1059.007	Command and Scripting Interpreter: JavaScript	BladedFeline uses JavaScript webshells to execute commands on compromised endpoints.
	T1059.001	Command and Scripting Interpreter: PowerShell	BladedFeline uses PowerShell to execute commands on compromised endpoints.
	T1059.006	Command and Scripting Interpreter: Python	BladedFeline uses Python as a dropper for deploying backdoors to compromised endpoints.
	T1559	Inter-Process Communication	BladedFeline uses IPC as a means of local code execution in its malicious IIS module.
	T1569.002	System Services: Service Execution	BladedFeline uses Windows services for malware execution with Whisper and PrimeCache.
Persistence	T1547.001	Boot or Logon Autostart Execution: Registry Run Keys / Startup Folder	The Whisper backdoor creates a LNK file in the startup folder for persistence.

Tactic	ID	Name	Description
	T1546	Event Triggered Execution	PrimeCache is loaded by an IIS Worker Process (w3wp.exe) when the IIS server receives an inbound HTTP request.
Defense Evasion	T1078	Valid Accounts	BladedFeline uses legitimate accounts to exfiltrate data and bypass defenses, and as C&C servers.
	T1140	Deobfuscate/Decode Files or Information	The Whisper backdoor uses base64 encoding to obfuscate data.
	T1070.004	Indicator Removal: File Deletion	The Python dropper for Whisper deletes itself and other install files after a successful installation.
	T1070.006	Indicator Removal: Timestomp	BladedFeline routinely timestomps the compilation timestamps of malware that the group develops.
Credential Access	T1003.001	OS Credential Dumping: LSASS Memory	BladedFeline dumps LSASS from memory to steal credentials.
Command and Control	T1573.001	Encrypted Channel: Symmetric Cryptography	The Whisper backdoor uses AES encryption to send and receive data between the malware and the C&C.
	T1071.001	Application Layer Protocol: Web Protocols	PrimeCache uses standard web protocols for communication with the C&C server.
	T1132.001	Data Encoding: Standard Encoding	PrimeCache uses standard encoding for communication with the C&C server.
	T1573.002	Encrypted Channel: Asymmetric Cryptography	PrimeCache uses RSA and AES-CBC for C&C communication.
	T1105	Ingress Tool Transfer	PrimeCache has the capability to download additional files from the C&C server for local execution.
Exfiltration	T1048.001	Exfiltration Over Alternative Protocol: Exfiltration Over Symmetric Encrypted Non-C2 Protocol	The Whisper backdoor uses AES encryption and email inboxes to send and receive data between the malware and the C&C.
	T1041	Exfiltration Over C2 Channel	PrimeCache exfiltrates data to a C&C server.

Source: https://www.welivesecurity.com/en/eset-research/bladedfeline-whispering-dark/