Stealth Falcon preying over Middle Eastern skies with Deadglyph

By ESET Research

Archived: 2026-04-06 03:26:56 UTC

For years, the Middle East has maintained its reputation as a fertile ground for advanced persistent threats (APTs). In the midst of routine monitoring of suspicious activities on the systems of high-profile customers, some based in this region, ESET Research stumbled upon a very sophisticated and unknown backdoor that we have named Deadglyph. We derived the name from artifacts found in the backdoor (such as 0x**DEAD**B001, shown also in Table 1), coupled with the presence of a homo**glyph** attack. To the best of our knowledge, this is the first public analysis of this previously undocumented backdoor, used by a group that exhibits a notable degree of sophistication and expertise. Based on the targeting and additional evidence, we attribute Deadglyph with high confidence to the Stealth Falcon APT group.

Deadglyph's architecture is unusual as it consists of cooperating components – one a native x64 binary, the other a .NET assembly. This combination is unusual because malware typically uses only one programming language for its components. This difference might indicate separate development of those two components while also taking advantage of unique features of the distinct programming languages they utilize. Different language can also be harnessed to hinder analysis, because mixed code is more difficult to navigate and debug.

The traditional backdoor commands are not implemented in the backdoor binary; instead, they are dynamically received by it from the command and control (C&C) server in the form of additional modules. This backdoor also features a number of capabilities to avoid being detected.

In this blogpost, we take a closer look at Deadglyph and provide a technical analysis of this backdoor, its purpose, and some of the additional components we obtained. We are also presenting our findings about Deadglyph at the LABScon 2023 conference.

> **Key points of the blogpost:**
>
> - *ESET Research discovered a sophisticated backdoor with unusual architecture that we have named Deadglyph.*
> - *The main components are encrypted using a machine-specific key.*
> - *Traditional backdoor commands are implemented via additional modules received from its C&C server.*
> - *We obtained three out of many modules – process creator, file reader, and info collector.*
> - *We attribute Deadglyph to the Stealth Falcon group.*
> - *Additionally, we found a related shellcode downloader; we postulate it could potentially be used for installation of Deadglyph.*

The victim of the analyzed infiltration is a governmental entity in the Middle East that was compromised for espionage purposes. A related sample found on VirusTotal was also uploaded to the file-scanning platform from this region, specifically from Qatar. The targeted region is depicted on the map in Figure 1.

Figure 1. Victimology of Deadglyph; the related sample was uploaded to VirusTotal from Qatar (in darker color)

Stealth Falcon (also known as Project Raven or FruityArmor) is a threat group linked to the United Arab Emirates according to MITRE. Active since 2012, Stealth Falcon is known to target political activists, journalists, and dissidents in the Middle East. It was first discovered and described by Citizen Lab, which published an analysis of a campaign of spyware attacks in 2016.

In January 2019, Reuters published an investigative report on Project Raven, an initiative allegedly employing former NSA operatives and aiming at the same types of targets as Stealth Falcon. Based on these two reports referring to the same targets and attacks, Amnesty International has concluded (shown in Figure 2) that Stealth Falcon and Project Raven actually are the same group.

Figure 2. Claudio Guarnieri has connected Stealth Falcon with Project Raven

In September 2019, we published research on a backdoor, attributed to Stealth Falcon, that used an unusual technique, Background Intelligent Transfer Service, for C&C communication. We now reveal the result of our in-depth analysis of what presumably is the newest addition to Stealth Falcon's espionage toolset.

Deadglyph backdoor

Deadglyph's loading chain consists of multiple components, as illustrated in Figure 3. The initial component is a registry shellcode loader, which loads shellcode from the registry. This extracted shellcode, in turn, loads the native x64 part of the backdoor – the Executor. The Executor subsequently loads the .NET part of the backdoor – the Orchestrator. Notably, the only component on system's disk as a file is the initial component, which is in the form of a Dynamic Link Library (DLL). The remaining components are encrypted and stored within a binary registry value.

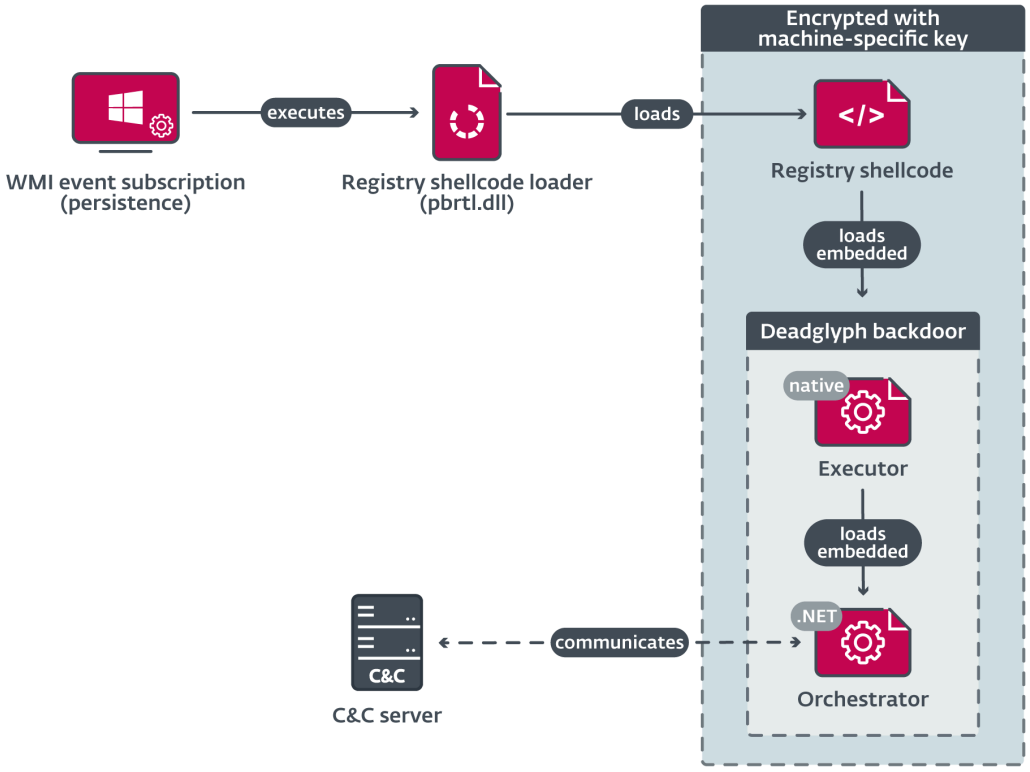

Figure 3. Deadglyph loading chain components

While the precise method of the initial compromise vector is not yet determined, our suspicion is that an installer component is involved in deploying further components and establishing persistence within the system.

In the rest of this section, we analyze each component.

Registry shellcode loader

Deadglyph's initial component is a tiny DLL with a single export, named 1. This component is persisted using Windows Management Instrumentation (WMI) event subscription and serves as a registry shellcode loader. It is executed via the command line rundll32 C:\WINDOWS\System32\\pbrtl.dll,#1.

The registry shellcode loader begins its operation by decrypting the path to the encrypted shellcode stored within the Windows registry, using RC4. We suspect the path is unique for each victim; in the case analyzed here, the registry path was:

Software\Classes\CLSID\{5abc7f42-1112-5099-b082-ce8d65ba0c47}\cAbRGHLg

The root registry key is either HKLM or HKCU, depending on whether the current process is running with elevated privileges or not. The same logic can be found in further components.

Following this, the loader derives a machine-specific RC4 key using the system UUID retrieved from the raw SMBIOS firmware table. Using this key, it loads, decrypts, and then executes the shellcode. It is important to highlight that this key derivation approach ensures that proper decryption won't occur if the loader is executed on a different computer.

Interestingly, the loader can also be configured by a flag in its .data section to use a hardcoded key to decrypt the shellcode, instead of the machine-specific one.

We spotted a homoglyph attack mimicking Microsoft Corporation in the VERSIONINFO resource of this and other PE components. This method employs distinct Unicode characters that appear visually similar, but in this case not identical, to the original characters, specifically Greek Capital Letter San (U+03FA, Ϻ) and Cyrillic Small Letter O (U+043E, о) in Microsoft Corporation.

Registry shellcode

Comprised of two parts, the registry shellcode consists of a decryption routine and an encrypted body. First, the decryption routine rotates each byte of the encrypted body to the left by one (ROL 0x01). Subsequently, control is transferred to this decrypted body. The decrypted body consists of a PE loader and a PE file, the latter being the Executor, which represents the native part of the backdoor. This loader is responsible for parsing and loading the associated PE file.

Executor

The Executor is the native x64 part of the Deadglyph backdoor, which does the following:

- loads its configuration,
- initializes the .NET runtime,
- loads the embedded .NET part of the backdoor (the Orchestrator), and
- acts as a library for the Orchestrator.

First, two default configurations embedded in the .data section are AES-decrypted. The configurations encompass various parameters, including encryption keys, safety and evasion settings, and the entry point of the subsequent component.

During the initial execution, those two default configurations are stored within the Windows registry, from where they are loaded on subsequent runs, enabling the implementation of updates. The registry path for each configuration is generated with the following format:

{HKCU|HKLM}\Software\Classes\CLSID\{<variable_GUID>}\(Default)

<variable_GUID> is a generated GUID, which is unique to each victim.

Following this, the .NET runtime is initialized, then the Executor RC4-decrypts the .NET part of the backdoor known as the Orchestrator. The Orchestrator is located within the .rsrc section of the Executor. The configuration specifies the Orchestrator's execution method as an entry point. Moreover, a distinct structure is provided to facilitate accessibility of the Executor's functions by the Orchestrator.

After launching the Orchestrator, the Executor acts as a support library for the Orchestrator. The Executor contains many interesting functions; we describe some of them in the following section, in context of their utilization by the Orchestrator and further loaded modules.

Orchestrator

Written in .NET, the Orchestrator is the main component of the Deadglyph backdoor. This component's primary role involves establishing communication with the C&C server and executing commands, often facilitated through the intermediary role of the Executor. In contrast to the preceding components, the Orchestrator is obfuscated, employing .NET Reactor. Internally, the backdoor is referred to as agent, which is a common name for the client part in various post-exploitation frameworks.

Initialization

The Orchestrator first loads its configuration and two embedded modules, each accompanied by its own set of configurations, from resources. These resources are Deflate compressed and AES encrypted. They are referenced by an ID that is SHA-1 hashed into a resource name. An overview of these resources is provided in Table 1.

Table 1. Orchestrator resources

Resource name	ID (decimal)	ID (hex)	Description
43ed9a3ad74ed7ab74c345a876b6be19039d4c8c	2570286865	0x99337711	Orchestrator configuration.
3a215912708eab6f56af953d748fbfc38e3bb468	3740250113	0xDEEFB001	Network module.
42fb165bc9cf614996027a9fcb261d65fd513527	3740250369	0xDEEFB101	Network module configuration.
e204cdcf96d9f94f9c19dbe385e635d00caaf49d	3735924737	0xDEADB001	Timer module.
abd2db754795272c21407efd5080c8a705a7d151	3735924993	0xDEADB101	Timer module configuration.

The configuration of the Orchestrator and embedded modules is stored in XML format. An example of an Orchestrator configuration is shown in Figure 4.

```
<c>
    <k>a7e75147c6fcdc19e9a2d602a7307573</k>
    <a>QQ.jX.Rx</a>
    <b>true</b>
    <c>yr.TV.J2</c>
    <d>true</d>
    <p>3740250113</p>
    <t>3735924737</t>
</c>
```

Figure 4. Orchestrator configuration

The description of Orchestrator configuration entries is shown in Table 2.

Table 2. Orchestrator configuration entries

Key	Description
k	AES key used for persisting module configurations.
a	Network module initialization method name.
b	Unknown network module-related flag.
c	Timer module initialization method name.
d	Flag enabling usage of machine-specific AES key (system UUID) for resources.

p	Network module resource ID.
t	Timer module resource ID.

After the resource components are loaded, multiple threads are created to carry out distinct tasks. One of these threads is responsible for conducting environment checks, a function implemented within the Executor. Another thread is devoted to establishing periodic communication with the C&C server, enabling the retrieval of commands. Lastly, a set of three threads is employed for the purpose of executing received commands and subsequently transmitting any generated output back to the C&C server.

The environment-checking thread monitors running processes to identify unwanted ones. This thread operates with two distinct lists of process names. If a process on the first list is detected, C&C communication and command execution is paused until the unwanted process no longer exists. If there is a match for any process on the second list, the backdoor immediately quits and uninstalls itself.

Neither list was configured in the analyzed instance, so we don't know what processes might typically be checked for; we believe it is probably intended to evade analysis tools that could detect suspicious activity and lead to discovery of the backdoor.

Communication

The Orchestrator utilizes two embedded modules for C&C communication – Timer and Network. Like the Orchestrator, these modules are obfuscated with .NET Reactor. The configuration for both modules is supplied by the Orchestrator. Within the Orchestrator, a preset configuration for the modules is included; optionally, the Orchestrator can also load an updated configuration version from the registry:

{HKCU|HKLM}\Software\Classes\CLSID\{<variable_GUID>}\<mod_cfg_res_ID>

The backdoor contains an interesting safety measure related to communication. If the backdoor is unable to establish communication with the C&C server for a duration surpassing a predefined threshold, configured within the Executor, a self-uninstallation mechanism is triggered. This time threshold is specified in hours and was set at one hour in the examined case.

This approach serves a twofold purpose. On one hand, it prevents the generation of redundant network requests towards an inaccessible server. On the other hand, it reduces the chances of subsequent detection if the operators lose control over the backdoor.

Timer module

This small module executes the specified callback at a configurable interval. It is used by the Orchestrator in combination with the Network module to communicate with the C&C server periodically. To prevent the creation of detectable patterns in network logs, the execution interval is subject to randomization, based on a percentage specified in the configuration. In the analyzed instance, the interval was set to five minutes, with a ±20% variation introduced for randomness.

Another method to avoid detectable network patterns in periodic communication can be found in generation of requests sent to the C&C server. This mechanism, implemented in the Executor, involves the inclusion of padding of varying length, comprised of random bytes, within the requests, resulting in requests of diverse sizes.

Network module

The Network module implements communication with the C&C servers specified in its configuration. It can send data to a C&C server using HTTP(S) POST requests. Notably, it offers several mechanisms to acquire proxy configuration details. This feature suggests a potential focus on environments where direct internet access is not available.

An example of a decrypted (and beautified) configuration is shown in Figure 5.

```
<ht>
    <ppa/>
    <pdo/>
    <urs>https://chessandlinkss.com/f5684883-ef55-483f-9e83-ac769c4c22fa;https://easymathpath.com/923144a0-a269-4471-9160-e7ad5e1512e3</urs>
    <ua>Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/76.0.3809.132 Safari/537.36</ua>
    <pur/>
    <pus/>
</ht>
```

Figure 5. Network module configuration

Configuration entries contain details related to network communications – C&C URLs, HTTP User-Agent, and optionally a proxy configuration.

When communicating with the C&C server, a custom binary protocol with encrypted content is used underneath HTTPS.

Commands

The Orchestrator receives commands from the C&C server in the form of tasks, which are queued for execution. There are three kinds of tasks processed:

- Orchestrator tasks,
- Executor tasks, and
- Upload tasks.

The first two kinds are received from the C&C server and the third is created internally to upload the output of commands and errors.

Orchestrator tasks

Orchestrator tasks offer the ability to manage the configuration of the Network and Timer modules, and also to cancel pending tasks. The overview of Orchestrator tasks is shown in Table 3.

Table 3. Orchestrator tasks

Type	Description
0x80	Set configuration of network and timer modules.
0x81	Get configuration of network and timer modules.
0x82	Cancel task.
0x83	Cancel all tasks.

Executor tasks

Executor tasks offer the ability to manage the backdoor and execute additional modules. It's notable that the traditional backdoor functionality is not inherently present within the binary itself. Instead, these functions are obtained from the C&C server in the form of PE files or shellcode. The full extent of the backdoor's potential remains unknown without these additional modules, which effectively unlock its true capabilities. An overview of module tasks is shown in Table 4, which includes details about the few identified modules. Similarly, Table 5 provides an overview of management tasks associated with the Executor.

Table 4. Executor tasks – modules

Type	Description
0x??–0x63	Unknown

0x64	File reader
0x65	Unknown
0x66	Unknown
0x67	Unknown
0x68	Unknown
0x69	Process creator
0x6A	Unknown
0x6B	Unknown

0x6C	Info collector
0x6D	Unknown
0x6E	Unknown

Table 5. Executor tasks – management

Type	**Description**
0x6F-0x76	Not implemented
0x77	Set Executor configuration
0x78	Get Executor configuration
0x79-0x7C	Not implemented
0x7D	Update
0x7E	Quit
0x7F	Uninstall

The command that sets the Executor configuration can change the:

- unwanted process lists,

- time threshold of C&C communication failure, and
- time limit for execution of additional modules.

Modules

We managed to obtain three unique modules from the C&C server, each corresponding to a different Executor task type, as shown in Table 4. Based on available information, we estimate there are nine to fourteen modules in total. As the modules are in fact backdoor commands, they have one basic operation to execute and then optionally return their output. The modules we obtained are DLLs with one unnamed export (ordinal 1), in which they resolve necessary API functions and call the main function.

When executed, the modules are provided with an API resolution function, which can resolve Windows APIs and custom Executor APIs. The Windows APIs are referenced by a DWORD hash, calculated from the name of the API and its DLL. Small hash values (<41) are treated specially, referencing the Executor API function. The Executor API comprises a total of 39 functions that are accessible to the modules. These functions pertain to a variety of operations, including:

- file operations,
- encryption and hashing,
- compression,
- PE loading,
- access Token Impersonation, and
- utility.

In the rest of this section, we describe the modules that we obtained.

Process creator

Module 0x69 executes the specified command line as a new process and provides the resulting output back to the Orchestrator. The process can be created under a different user, and its execution time can be limited. Notably, an unusual Job API is used in this module's functionality.

This module was served with the command line cmd.exe /c tasklist /v.

We assume it serves as an idle command issued automatically, while the operators wait for something interesting to happen on the compromised computer.

Info collector

Module 0x6C collects extensive information about the computer via WMI queries and passes it back to the Orchestrator. Information about the following is collected:

- operating system,
- network adapters,
- installed software,
- drives,

- services,
- drivers,
- processes,
- users,
- environment variables, and
- security software.

File reader

Module 0x64 reads the specified file and passes the content back to the Orchestrator. Optionally, it can delete the file after reading.

We saw this module used to retrieve the victim's Outlook data file

c:\Users\<redacted>\AppData\Local\Microsoft\Outlook\outlook.ost.

Chain with shellcode downloader

In the process of investigating Deadglyph, we encountered a dubious CPL file signed with an expired certificate and no countersignature with a timestamp, which had been uploaded to VirusTotal from Qatar. Upon closer examination, it became evident that this CPL file functioned as a multistage shellcode downloader, sharing certain code resemblances with Deadglyph. The loading chain is illustrated in Figure 6.

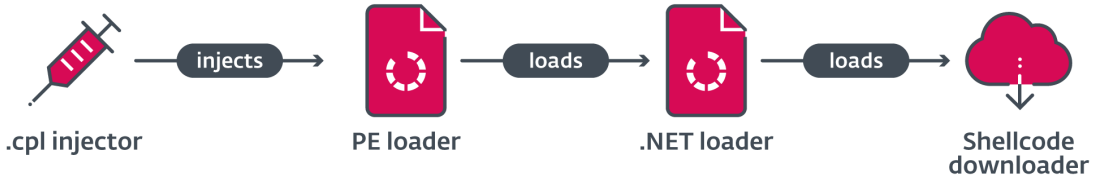

Figure 6. Shellcode downloader loading chain

In its initial form, which serves as the first stage, this file anticipates having a .cpl extension (Control Panel file) and is meant to be executed via a double-click action. Upon execution in this manner, the embedded shellcode undergoes XOR decryption, and the running processes are checked to identify a suitable host process for subsequent injection.

If avp.exe (a Kaspersky endpoint security process) is running, %windir%\system32\UserAccountBroker.exe is used. Otherwise, the default browser is used. Then, it creates the host process in a suspended state, injects the shellcode by hijacking its main thread, and resumes the thread.

The second stage, the shellcode, consists of two parts. The first part of the shellcode resolves API hashes, using the same unique hash calculation technique employed in Deadglyph, and decrypts strings with process names. It starts a self-delete thread tasked with overwriting and subsequently erasing the first-stage file. Following this, the shellcode proceeds to inspect the currently active processes, targeting a security solution.

If any of the specified processes are detected, the shellcode creates a sleeper thread with the lowest priority (THREAD_PRIORITY_IDLE) and allows it to remain active for a duration of 60 seconds before terminating its

operation. This interval is likely implemented as a precautionary measure to evade certain detection mechanisms employed by security solutions. Finally, the shellcode proceeds to invoke the execution of the second part of its code.

The second part of the shellcode loads an embedded PE file with stage three and calls its export with ordinal number 1.

The third stage, a DLL, serves as a .NET loader and contains the payload in its .rsrc section.

To load the payload, the .NET runtime is initialized. During the .NET initialization, two intriguing techniques are performed, seemingly intended to evade Windows Antimalware Scan Interface (AMSI) scanning:

- The .NET loader temporarily hooks the GetModuleHandleW import in the loaded clr.dll, while calling ICorRuntimeHost::Start. The hook tampers with the return value when GetModuleHandleW is called with NULL. It returns a pointer to a dummy PE with no sections.
- It then subtly patches the AmsiInitialize import name string in the .rdata section of the loaded clr.dll to amsiinitialize.

The fourth stage is a .NET assembly, obfuscated with ConfuserEx, that serves as a shellcode downloader. First, it XOR-decrypts its configuration in XML format from its resources. A beautified version of the extracted configuration is presented in Figure 7. The configuration entries contain details related to network communication and blocklisted processes.

```
<c>
    <t>https://joinushealth.com/ksngh63d</t>
    <b>Microsoft-WebDAV-MiniRedir/10.0.22000</b>
    <h>true</h>
    <d>true</d>
</c>
```

Figure 7. Shellcode downloader configuration

Before proceeding, it checks the running processes against a list of blocklisted processes from the configuration. If a match is detected, the execution halts. It is important to note that in the analyzed instance, this blocklist wasn't set up.

Next, it sends an HTTP GET request to the C&C server to retrieve some shellcode, using parameters specified in the configuration (URL, User-Agent, and optionally Proxy). Regrettably, during our investigation we were unable to acquire any shellcode from the C&C server. Nonetheless, we hypothesize that the content being retrieved could potentially serve as the installer for Deadglyph.

Following this, the retrieved shellcode is executed within a newly created thread. After waiting until the shellcode thread finishes execution, the shellcode downloader removes all files located in the directory %WINDIR%\ServiceProfiles\LocalService\AppData\Local\Temp\TfsStore\Tfs_DAV.

Finally, it makes an attempt to delete itself after a 20-second interval, employing the subsequent command, before concluding its operation and exiting:

cmd.exe choice /C Y /N /D Y /T 20 & Del /f /q <current_process_exe_path>

This self-deletion does not make sense in this chain. This is due to the fact that the shellcode downloader is executed within a browser or system process after being injected, rather than operating as an independent executable. Moreover, the initial file was already deleted by the second stage. This observation suggests that the shellcode downloader might not be an exclusive payload of this chain and may also be used separately in other operations.

Conclusion

We have discovered and analyzed a sophisticated backdoor used by the Stealth Falcon group that we have named Deadglyph. It has an unusual architecture, and its backdoor capabilities are provided by its C&C in the form of additional modules. We managed to obtain three of these modules, uncovering a fraction of Deadglyph's full capabilities.

Notably, Deadglyph boasts a range of counter-detection mechanisms, including continuous monitoring of system processes and the implementation of randomized network patterns. Furthermore, the backdoor is capable of uninstalling itself to minimize the likelihood of its detection in certain cases.

Additionally, our investigation led us to the discovery of a compelling multistage shellcode downloader chain on VirusTotal. We suspect this downloader chain is likely leveraged in the installation process of Deadglyph.

> 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
C40F1F46D230A85F702DAA38CFA18D60481EA6C2	pbrtl.dll	Win64/Deadglyph.A	Registry Shellcode Loader.
740D308565E215EB9B235CC5B720142428F540DB	N/A	Win64/Deadglyph.A	Deadglyph Backdoor – Executor.

1805568D8362A379AF09FD70D3406C6B654F189F	N/A	MSIL/Deadglyph.A	Deadglyph Backdoor – Orchestrator.
9CB373B2643C2B7F93862D2682A0D2150C7AEC7E	N/A	MSIL/Deadglyph.A	Orchestrator Network module.
F47CB40F6C2B303308D9D705F8CAD707B9C39FA5	N/A	MSIL/Deadglyph.A	Orchestrator Timer module.
3D4D9C9F2A5ACEFF9E45538F5EBE723ACAF83E32	N/A	Win64/Deadglyph.A.gen	Process creator module.
3D2ACCEA98DBDF95F0543B7C1E8A055020E74960	N/A	Win64/Deadglyph.A	File reader module.
4E3018E4FD27587BD1C566930AE24442769D16F0	N/A	Win64/Deadglyph.A	Info collector module.
7F728D490ED6EA64A7644049914A7F2A0E563969	N/A	Win64/Injector.MD	First stage of shellcode downloader chain.

Certificates

Serial number	00F0FB1390F5340CD2572451D95DB1D92D
Thumbprint	DB3614DAF58D041F96A5B916281EA0DC97AA0C29

Subject CN	RHM LIMITED
Subject O	RHM LIMITED
Subject L	St. Albans
Subject S	Hertfordshire
Subject C	GB
Email	rhm@rhmlimited[.]co.uk
Valid from	2021-03-16 00:00:00
Valid to	2022-03-16 23:59:59

C&C servers

IP	Domain	First seen	Comment
185.25.50[.]60	chessandlinkss[.]com	2021-08-25	Deadglyph C&C server.
135.125.78[.]187	easymathpath[.]com	2021-09-11	Deadglyph C&C server.
45.14.227[.]55	joinushealth[.]com	2022-05-29	Shellcode downloader C&C server.

MITRE ATT&CK techniques

This table was built using version 13 of the MITRE ATT&CK framework.

Tactic	ID	Name	Description
Resource Development	T1583.001	Acquire Infrastructure: Domains	Stealth Falcon has registered domains for C&C servers and to obtain a code-signing certificate.
	T1583.003	Acquire Infrastructure: Virtual Private Server	Stealth Falcon has used VPS hosting providers for C&C servers.
	T1587.001	Develop Capabilities: Malware	Stealth Falcon has developed custom malware, including custom loaders and the Deadglyph backdoor.
	T1588.003	Obtain Capabilities: Code Signing Certificates	Stealth Falcon has obtained a code-signing certificate.
Execution	T1047	Windows Management Instrumentation	Deadglyph uses WMI to execute its loading chain.
	T1059.003	Command and Scripting Interpreter: Windows Command Shell	Shellcode downloader uses cmd.exe to delete itself.
	T1106	Native API	A Deadglyph module uses CreateProcessW and CreateProcessAsUserW API functions for execution.
	T1204.002	User Execution: Malicious File	The shellcode downloader chain requires the user to double-click and execute it.
Persistence	T1546.003	Event Triggered Execution: Windows Management	The initial Deadglyph loader is persisted using WMI event subscription.

		Instrumentation Event Subscription	
Defense Evasion	T1027	Obfuscated Files or Information	Deadglyph components are encrypted. Deadglyph Orchestrator and embedded modules are obfuscated with .NET Reactor. The shellcode downloader is obfuscated with ConfuserEx.
	T1070.004	Indicator Removal: File Deletion	Deadglyph can uninstall itself. The shellcode downloader chain deletes itself and deletes files in the WebDAV cache.
	T1112	Modify Registry	Deadglyph stores its configuration and encrypted payload in the registry.
	T1134	Access Token Manipulation	Deadglyph can impersonate another user.
	T1140	Deobfuscate/Decode Files or Information	Deadglyph decrypts encrypted strings. The shellcode downloader chain decrypts its components and configurations.
	T1218.011	System Binary Proxy Execution: Rundll32	The initial Deadglyph loader is executed using rundll32.exe.
	T1480.001	Execution Guardrails: Environmental Keying	Deadglyph is encrypted using a machine-specific key derived from the system UUID.
	T1562.001	Impair Defenses: Disable or Modify Tools	The shellcode downloader avoids AMSI scanning by patching clr.dll in memory .

	T1620	Reflective Code Loading	Deadglyph reflectively loads its modules using a custom PE loader.
Discovery	T1007	System Service Discovery	A Deadglyph module discovers services using the WMI query SELECT * FROM Win32_Service.
	T1012	Query Registry	The shellcode downloader chain queries the registry for the default browser.
	T1016	System Network Configuration Discovery	A Deadglyph module discovers network adapters using WMI queries SELECT * FROM Win32_NetworkAdapter and SELECT * FROM Win32_NetworkAdapterConfiguration where InterfaceIndex=%d.
	T1033	System Owner/User Discovery	A Deadglyph module discovers users with the WMI query SELECT * FROM Win32_UserAccount.
	T1057	Process Discovery	A Deadglyph module discovers processes using WMI query SELECT * FROM Win32_Process.
	T1082	System Information Discovery	A Deadglyph module discovers system information such as OS version, drives, environment variables, and drivers using WMI queries.
	T1518	Software Discovery	A Deadglyph module discovers installed software using WMI query SELECT * FROM Win32_Product.
	T1518.001	Software Discovery: Security Software Discovery	A Deadglyph module discovers security software using WMI queries SELECT * FROM AntiVirusProduct, SELECT * FROM

			AntiSpywareProduct and SELECT * FROM FirewallProduct. The shellcode downloader chain checks running processes for a security solution.
Collection	T1005	Data from Local System	Deadglyph has a module for reading files.
Command and Control	T1071.001	Application Layer Protocol: Web Protocols	Deadglyph and the shellcode downloader communicate with the C&C server via the HTTP protocol.
	T1090	Proxy	Deadglyph and the shellcode downloader can use HTTP proxy for C&C communication.
	T1573.001	Encrypted Channel: Symmetric Cryptography	Deadglyph uses AES to encrypt C&C communications.
Exfiltration	T1041	Exfiltration Over C2 Channel	Deadglyph uses the C&C channel for exfiltration.

Source: https://www.welivesecurity.com/en/eset-research/stealth-falcon-preying-middle-eastern-skies-deadglyph/