NVISO Incident Response

nviso
WWW.NVISO.EU

BRICKSTORM Backdoor Analysis

A Persistent Espionage Threat to European Industries

April 2025

This report delivers a comprehensive technical analysis of BRICKSTORM, an espionage backdoor linked to the China-nexus cluster UNC5221.

NVISO investigates newly identified Windows-based BRICKSTORM variants, expanding on its previous presence in Linux environments. These backdoor samples, linked to long-term espionage campaigns, target European industries in key sectors of Chinese interest.

The analysis details adversary techniques, infrastructure, and defensive challenges, concluding with actionable mitigation and threat-hunting recommendations.

ATHENS BRUSSELS FRANKFURT MUNICH VIENNA

Contents

1. Introduction 3

2. Detailed Analysis

2.1. Capabilities 4

2.1.1. File Manager 5

2.1.2. Network Tunneling 7

2.1.3. Configuration 7

2.2. Protocol 8

2.2.1. Address Resolution 8

2.2.2. Command & Control 10

2.2.3. File Manager API 13

2.2.4. Network Tunneling 16

2.3 Infrastructure 17

3. Conclusions & Recommendations 20

Appendix

4. Indicators of Compromise 21

5. Detection & Hunting Rules

5.1. YARA 22

5.2. Suricata (Network) 23

5.3. Kusto (KQL) Hunting 24

6. 2nd Layer TLS Certificate (2023-2024) 26

7. 2nd Layer TLS Certificate (2024-2025) 28

About NVISO 30

Introduction

NVISO has recently identified new information related to BRICKSTORM, a previously identified[1] backdoor linked to the China-nexus cluster UNC5221. This report provides a technical analysis of two newly identified BRICKSTORM samples which were affecting Windows environments. These samples are part of the BRICKSTORM backdoor family, previously only sighted on a Linux vCenter server. The newly identified backdoor variants are believed to have been employed in long running cyber espionage campaigns since at least 2022, targeting European industries of strategic interest to the People's Republic of China (PRC).

Over the past years, the number of persistent intrusions attributed to China-nexus actors has drastically increased. Numerous reports of compromises affecting both critical infrastructure as well as industries of strategic interest outline the continuous efforts deployed by the PRC. As opposed to the more common extortion-driven intrusions, PRC-associated intrusions have been observed to employ a high degree of discretion, enabling the actors to remain undetected for extended duration. To achieve this strategic long-term placement, PRC-attributed intrusions increasingly involve the usage of previously unknown vulnerabilities (a.k.a., zero-days) alongside low-noise backdoors such as the hereafter documented BRICKSTORM family. PRC-aligned actors furthermore distinguish themselves through the effective targeting of network appliances and subsequently achieved persistent access.

The PRC's cyber operations are amongst the most active offensive programs, backed by a diverse network of military, state and state-aligned operators. The PRC's focus on espionage operations has long been linked to China's political strategy which considers the strengthening of their economy as a matter of national security. To this end, PRC operations are known to heavily involve the theft of intellectual property (IP) and trade secrets pertaining to China's long-term strategic goals to further develop the manufacturing sector and establish itself as a technology-intensive leader.

NVISO and its partners provide this detailed BRICKSTORM analysis to highlight employed adversary techniques and infrastructure. This report outlines BRICKSTORM's inner workings and subsequent defensive challenges before concluding with mitigation and threat hunting recommendations.

[1] https://cloud.google.com/blog/topics/threat-intelligence/ivanti-post-exploitation-lateral-movement

Detailed Analysis

The two newly identified BRICKSTORM executables provide attackers with file manager and network tunneling capabilities. Through these backdoors, adversaries can browse the file system, create/delete arbitrary files and folders as well as tunnel network connections for lateral movement. The BRICKSTORM family resolves its Command & Control servers through DoH (DNS over HTTPS), hindering most network monitoring solutions.

The two Windows samples were written in Go 1.13.5 (released in 2019) and did not export any functions. The adversaries relied on persistence mechanisms such as scheduled tasks for execution. As noted by Mandiant, the recent sample's Go package is named `wssoft`. The second older BRICKSTORM sample however had the Go package name `wsshell`, indicating that the backdoor has historically undergone renaming.

2.1. Capabilities

BRICKSTORM provides attackers with file manager and network tunneling capabilities. As a notable difference to Mandiant's BRICKSTORM report, the Windows samples discussed here are not equipped with command execution capabilities. Instead, adversaries have been observed using network tunneling capabilities in combination with valid credentials to abuse well-known protocols such as RDP or SMB, thus achieving similar command execution.

The avoidance of command execution directly from the backdoor is suspected to be a deliberate choice in order to evade detection by modern security solutions that analyze parent-child process relationships.

Detailed Analysis

2.1.1. File Manager

The BRICKSTORM file manager exposes an HTTP API and rudimentary UI (User Interface) encapsulated within the protocol (see "2.2.3. File Manager API"). The backdoor's JSON-based API provides a wide range of file-related actions such as uploading, downloading, renaming, and deleting files. Adversaries can furthermore create or delete directories as well as list their contents.

The BRICKSTORM panel is served by the malware itself and proxied through its protocol towards the Command & Control server. Its main component, as displayed in *Figure 1*, allows the adversary to select a drive they wish to browse.

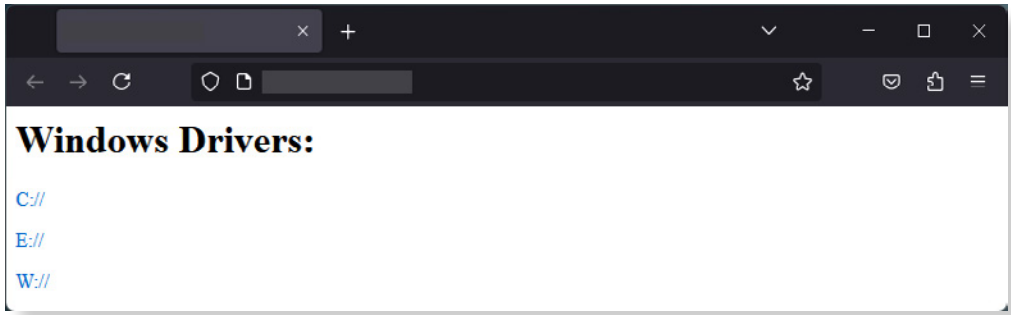

Figure 1: The BRICKSTORM user interface to select a drive, seen from the Command & Control perspective.

Once the drive is selected, BRICKSTORM allows the adversary to browse through the file system and download desired files (see *Figure 2*).

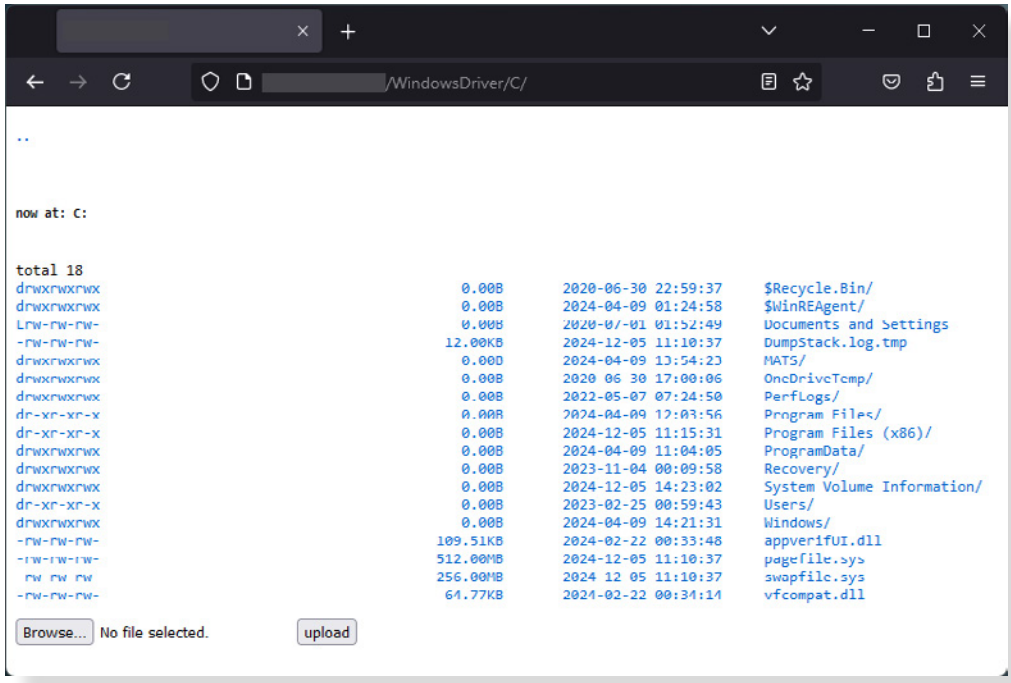

Figure 2: The BRICKSTORM user interface to browse the file system, seen from the Command & Control perspective.

Detailed Analysis

Utilizing the above UI, the adversary can furthermore upload arbitrary files. Once an upload is performed, the UI confirms both file path and computed SHA256 hash (outlined in *Figure 3*).

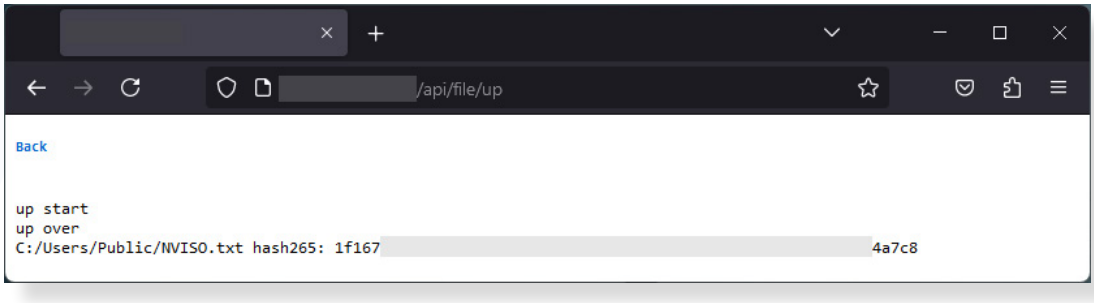

Figure 3: The BRICKSTORM user interface's post-upload confirmation, seen from the Command & Control perspective.

While the above UI does not provide a mechanism to create directories and delete/rename files. The JSON-based API does provide the necessary alternatives. The following snippet provides an example of such an API usage.

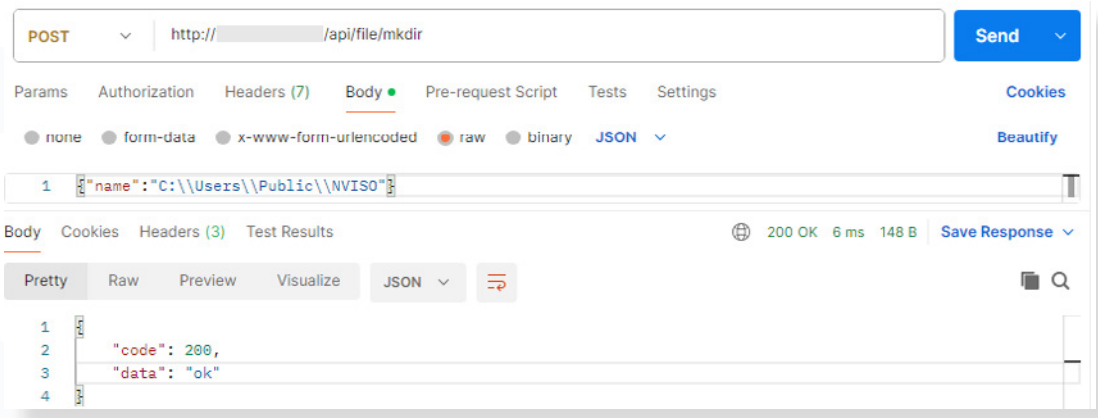

Figure 4: BRICKSTORM's API, seen through Postman from the Command & Control perspective.

Detailed Analysis

2.1.2. Network Tunneling

In addition to its file manager capabilities, BRICKSTORM provides network tunneling capabilities. Specifically, the observed variants support TCP, UDP and ICMP relaying. Attackers may leverage network tunneling to, for example, relay RDP and SMB connections.

Network-tunneling capabilities are especially effective given that attackers have been observed deploying BRICKSTORM on domain-joined devices after having obtained valid privileged credentials. Combined, these techniques provide long-lasting opportunities for network-wide lateral-movements.

2.1.3. Configuration

The BRICKSTORM configuration displays its customization flexibility. The observed configurations defined the `AuthKey` (used as part of the authentication described in section "2.2.2. Command & Control"), the `ServerAddr` and `WsAddr`, used to establish the TLS and WebSocket connection as well as the `DohHost` to resolve the domains (section "2.2.1. Address Resolution").

```
type core.Server struct {
        AuthKey    string
        ServerAddr string
        WsAddr     string
        IPAddrs    []string  // Introduced in later variants
        TmpFile    string
        TlsConfig  tls.Config
        UserAgent  string
        DohHost    []string
        Mu         sync.Mutex                     // Not a configuration setting
        doWork     map[uint8]func(net.Conn) error // Not a configuration setting
        IsLog      bool
}
```

While configurable and used by the application, the `TmpFile`, `UserAgent` & `IsLog` settings were always left empty in observed samples. In contrast, the `TlsConfig.InsecureSkipVerify` setting was always set to `true` in the default configuration.

As an evolution between the two Windows BRICKSTORM variants, the inclusion of the new `IPAddrs` functionality setting was observed. While left empty in observed configurations, the setting offers the ability to embed cloud provider IP addresses directly, rather than requiring the regular DNS resolution documented in section "2.2.1. Address Resolution"). While attacker infrastructure was observed shifting between cloud providers, BRICKSTORM's authentication key remained static throughout the campaign.

Detailed Analysis

2.2. Protocol

The BRICKSTORM protocol is designed to circumvent popular network-level security solutions such as TLS inspection or DNS monitoring. All BRICKSTORM Command & Control activities occur over a single connection which is multiplexed (i.e., shared) on multiple occasions.

Should BRICKSTORM lose connectivity, the communications are reattempted after a 40 minute sleep interval with a jitter up to ± 10 minutes.

2.2.1. Address Resolution

Observed BRICKSTORM Command & Control infrastructure leverages public cloud services as a front; Abused solutions include Cloudflare Workers as well as Heroku applications. Similarly, BRICKSTORM resolves its Command & Control domains through public DoH (DNS over HTTPS) providers which encapsulates plaintext DNS messages within secure HTTPS connections. Within multiple samples BRICKSTORM relied on Quad9, NextDNS, Cloudflare and Google for domain name resolution (see *Figure 5*). Due to DoH's usage of HTTPS, regular network-level DNS monitoring is circumvented.

```
; string DohHost
DohHost         string <offset aQuad9_1, 19h>
                                        ; DATA XREF: main_main+122↑o ; "https://9.9.9.9/dns-query"
                string <offset aNextDNS_1, 1Eh> ; "https://45.90.28.160/dns-query"
                string <offset aNextDNS_2, 1Eh> ; "https://45.90.30.160/dns-query"
                string <offset aQuad9_2, 21h> ; "https://149.112.112.112/dns-query"
                string <offset aQuad9_3, 1Ah> ; "https://9.9.9.11/dns-query"
                string <offset aCloudflare_1, 19h> ; "https://1.1.1.1/dns-query"
                string <offset aCloudflare_2, 19h> ; "https://1.0.0.1/dns-query"
                string <offset aGoogle_1, 19h> ; "https://8.8.8.8/dns-query"
                string <offset aGoogle_2, 19h> ; "https://8.8.4.4/dns-query"
                string <offset aQuad9_4, 20h> ; "https://149.112.112.11/dns-query"
```

Figure 5: BRICKSTORM's DNS Configuration.

The malware does not perform validation of the DoH-providers' TLS certificates; organizations equipped with TLS inspection may identify DNS over HTTPS activity through the `application/dns-message` header, visible in the following decrypted request. While BRICKSTORM has support for custom user-agents, our observed samples did not define any. It is furthermore worth noting that DoH encapsulates the DNS message within the `POST` request body, often not logged by organizations.

```
POST /dns-query HTTP/1.1
Host: REDACTED
Content-Length: 50
Connection: close
Content-Type: application/dns-message
```

Detailed Analysis

As part of its DoH library, BRICKSTORM issues a DNS request containing a single question with a hard-coded type of value 1 (see *Figure 6*) for 32-bit IPv4 addresses. As a consequence, the observed BRICKSTORM samples only operate over IPv4.

```
; dnsmessage_Question default_question
default_question db 0FFh dup(0)          ; Name.Data
                                         ; DATA XREF: wssoft_libs_doh_createDnsMessage+11E↑r
                                         ; wssoft_libs_doh_createDnsMessage+12C↑o
                 db 0                    ; Name.Length
                 dw 1                    ; Type
                 dw 1                    ; Class
```

Figure 6: BRICKSTORM's DNS Configuration.

While the oldest Windows BRICKSTORM variant historically exclusively relied on DoH resolution for its Command & Control domains, the recent sample introduced support for hardcoded IPs (see "2.1.3. Configuration" on page 7).

This latest addition ensures BRICKSTORM can operate in environments blocking DoH resolution as proposed in Conclusions & Recommendations on page 20.

2.2.2. Command & Control

Once BRICKSTORM has the front service IPs (either through optional hardcoded values or DoH resolution), the backdoor connects to the Command & Control domain over HTTPS (see *Figure 7*).

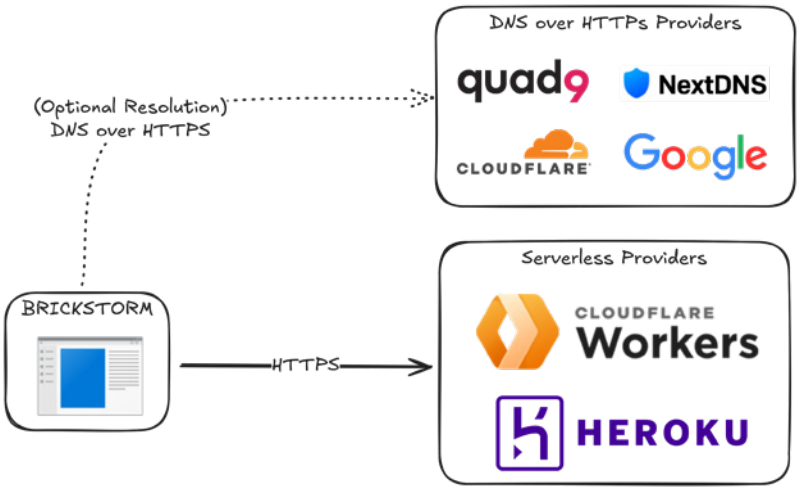

Figure 7: BRICKSTORM's connection to serverless providers over HTTPS.

The serverless providers act as reverse-proxies, providing valid SSL certificates while proxying traffic to 2nd-tier Command & Control infrastructure, later outlined in "2.3 Infrastructure" on page 17). The backdoor then upgrades the HTTPS connection to a WebSocket (*Figure 8*) as can be seen in the following decrypted request. As discussed in section "2.2.1. Address Resolution" on page 8, observed BRICKSTORM samples were configured to not define any user-agent.

```
GET /state HTTP/1.1
Host: REDACTED
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Key: REDACTED
Origin: https://REDACTED
Sec-WebSocket-Version: 13
```

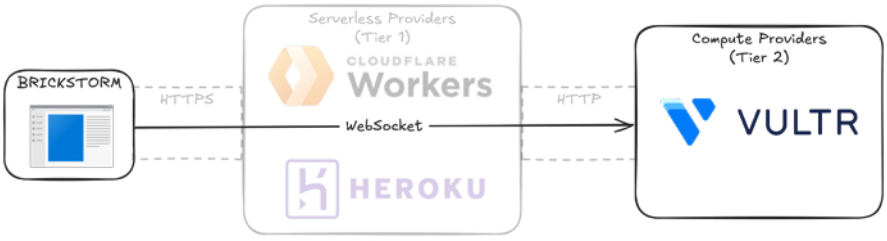

Figure 8: BRICKSTORM's usage of WebSockets through reverse HTTPS proxies.

Detailed Analysis

Once the BRICKSTORM WebSocket is established, the malware creates a nested TLS connection (*Figure 9*). This TLS-in-TLS approach defeats monitoring opportunities at both victim organizations (e.g., outbound HTTPS proxies) and 1st-tier cloud providers which serve the outer HTTPS connection. As for all BRICKSTORM connections, this nested TLS connection does not perform certificate validation. An attacker-issued certificate is provided as "2nd Layer TLS Certificate (2024-2025)" on page 29.

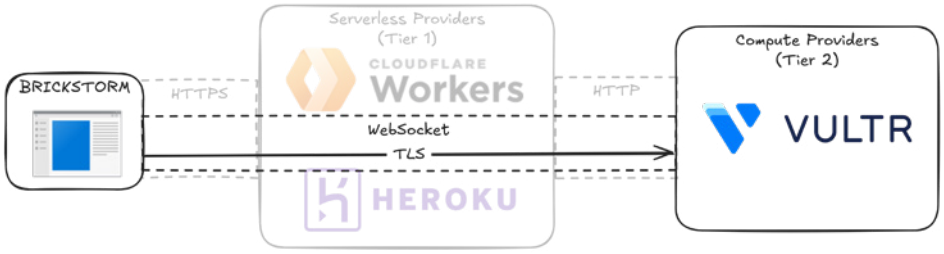

Figure 9: BRICKSTORM's nested TLS usage.

Within the nested TLS connection, the malware performs its own handshake. BRICKSTORM starts by sending over its configured authentication key (see section "2.1.3. Configuration"). If valid, the server acknowledges the authentication (message `0xDD, 0xCF, 0xFC`) which in turn the client acknowledges (message `0xFF, 0xCF, 0xCF`).

Figure 10: BRICKSTORM's handshake over the nested TLS connection.

As soon as the nested TLS connection is authenticated, BRICKSTORM multiplexes the connection to abstract new concurrent Command & Control sessions. This makes it possible to perform concurrent activities such as establishing multiple network tunnels and have concurrent file manager users. To achieve this multiplexing, BRICKSTORM leverages HashiCorp's Yamux[2] library, where the backdoor acts as listening server and the Command & Control infrastructure as the client. The oldest BRICKSTORM sample relied on the similar smux[3] library.

2 https://github.com/hashicorp/yamux
3 https://github.com/xtaci/smux

Detailed Analysis

Once the malware operator wishes to execute a new command (file manager or network tunneling), the attacker opens a new multiplexed session. This multiplexed session is immediately encrypted through a third layer of TLS (*Figure 11*). To date, it is unclear whether this 3-layered TLS connection is terminated within the 2nd-tier infrastructure or proxied through to 3rd-tier infrastructure.

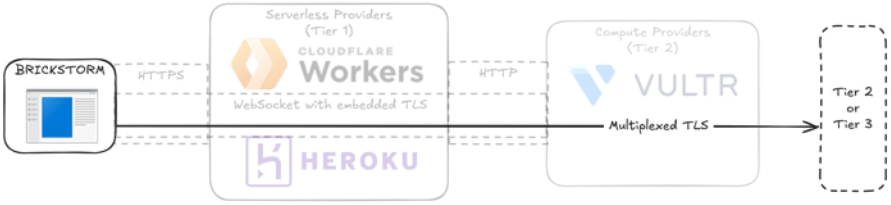

Figure 11: BRICKSTORM's 3-layered TLS connection.

As part of the TLS handshake, BRICKSTORM does not verify the server's certificates. However, as soon as the 3rd-layer TLS connection is established, the malware informs the Command & Control server of the perceived TLS public key (*Figure 12*). This surprising verification tend to suggest that BRICKSTORM assumes the 3rd layer TLS connection might be subject to interception.

Once its own public key is received, the BRICKSTORM server issues the desired command (`0x92` for the web service file manager or `0x12` for the network tunneling), which the malware subsequently acknowledges (code `0x93`).

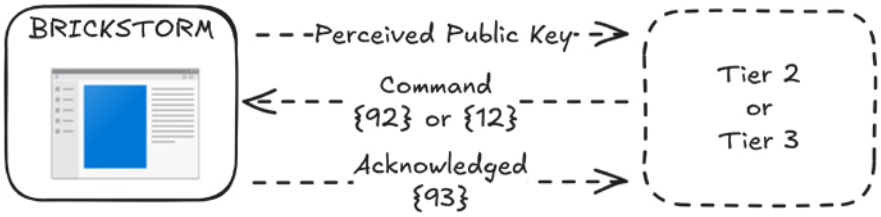

Figure 12: BRICKSTORM's 3rd layer Command & Control sequence.

Once the command acknowledged as valid, the multiplexed layer 3 TLS connection is multiplexed once more before being delegated to the web service (i.e., File Manager API) or network tunneling. This final multiplexing ensures that through a single command, multiple additional sessions can be abstracted. As an example, after having issued the web service command, multiple file manager API calls can simultaneously be issued by the attacker through the established connection.

Detailed Analysis

2.2.3. File Manager API

The identified BRICKSTORM samples shared a similar HTTP API as documented by Mandiant. In addition to the UI available at `/` (and subsequently browsed at `/WindowsDriver`, see section "2.1.1. File Manager" on page 5), the web service provides additional endpoints under the `/api/file` route, highlighted in the table below.

Endpoint	Description
`/up`	Receives the `multipart/form-data` posted from the UI. The file name and content are retrieved from the `uploadFile` file part while destination-directory is part of the `dir` variable. This endpoint returns HTML content as seen in *Figure 3*, including the new file path and its SHA265 hash.
`/delete-dir`	Removes the named empty directory defined in the posted `{"name":"…"}` request. This endpoint returns the `{"code":200,"data":"ok"}` JSON.
`/delete-file`	Removes the named file defined in the posted `{"name":"…"}` request. This endpoint returns the `{"code":200,"data":"ok"}` JSON.

`/stat`	Provides statistics on the file or directory defined in the posted `{"name":"…"}` request. This endpoint returns the beneath JSON, where owner and group are not implemented as shown in *Figure 13*.
	
	`{`
	`  "code": 200,`
	`  "fileinfo": {`
	`  "Name":"…",`
	`  "Size": 0,`
	`  "Mode": 123,`
	`  "ModTime": "…",`
	`  "IsDir": true,`
	`  "Owner": "",`
	`  "Group": "",`
	`  }`
	`}`

`/change-dir`	While this endpoint accepts a directory path through the posted `{"name":"…"}` request, it does not perform any actions (i.e., the working directory remains the same). This no-op endpoint returns the `{"code":200,"data":"ok"}` JSON.
`/list-dir`	Walks the directory defined in the posted `{"name":"…"}` request. This endpoint returns a JSON object similar to the `/stat` endpoint, except for the `$.fileinfo` property being an array.
`/mkdir`	Creates the directory defined in the posted `{"name":"…"}` request. This endpoint returns the `{"code":200,"data":"ok"}` JSON.
`/rename`	Renames the file or directory according to the posted `{"from-name":"…","to-name":"…"}` request. This endpoint returns the `{"code":200,"data":"ok"}` JSON.
`/get-file`	Downloads (and optionally resumes) the contents of the file according to the `{"path":"…","offset":123}` JSON request body. The observed samples did not require this endpoint to leverage the `POST` method.

Detailed Analysis

Endpoint	Description
`/put-file`	Creates or overwrites the file named by the `DestPath` MIME header with the posted data, optionally appending the file if the `Append` header is set to `yes`. This endpoint returns the `{"code":200,"data":123}` JSON, indicating the number of written bytes.

`/slice-up`	Uses the below POST data to create (or append/overwrite if `offset` is set) using the base64-encoded `data`. If the `size` matches the destination file's size, the actual and expected MD5 hashes are compared. This endpoint can be leveraged to upload file chunks or patch existing files.
	
	`{`
	`  "md5": "…",`
	`  "filename": "…",`
	`  "size": 123,`
	`  "offset": 123,`
	`  "data": "…",`
	`}`
	
	This endpoint returns the `{"code":200,"data":"123"}` JSON, indicating the textual number of written bytes.

`/file-md5`	Computes the MD5 hash of the file defined in the posted `{"name":"…"}` request. This endpoint returns the `{"code":200,"data":"…"}` JSON with the hex-encoded MD5.

Table 1: BRICKSTORM's file manager API.

As dependencies, BRICKSTORM leverages the popular Gorilla mux[4] library to serve its HTTP API, with the lesser known nex[5] library being used to perform the encoding and decoding of API requests. Interestingly, the older inactive BRICKSTORM sample furthermore embedded (without using) the goftp[6] library. This unused dependency provides an interesting datapoint given the project was archived from GitHub on July 8th, 2020.

4 https://github.com/gorilla/mux
5 https://github.com/lonng/nex
6 https://github.com/goftp/server

Detailed Analysis

The following capture provides an example of BRICKSTORM's file manager API as could be used from a command & control perspective. As an example, the above-documented `stat` endpoint is being used to obtain file details.

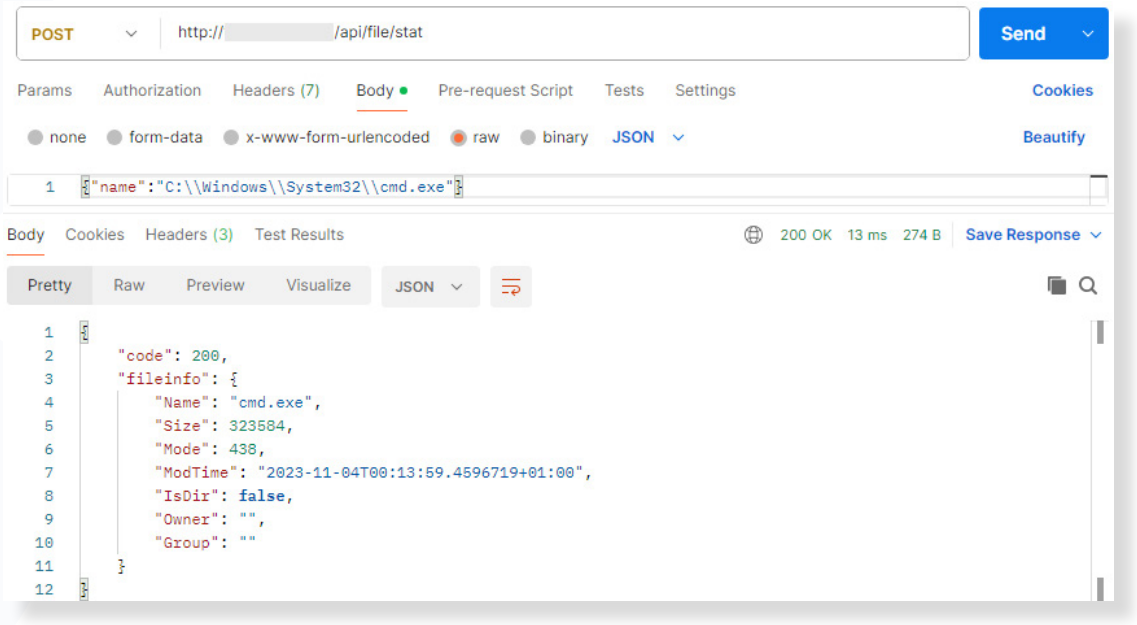

Figure 13: BRICKSTORM's file manager API, seen through Postman from the Command & Control perspective.

Through its extensive file manager, BRICKSTORM allows its operators to freely stage additional data or exfiltrate files of interest. However, given the network tunneling capabilities, adversaries were observed prioritizing interactive file operations through RDP and SMB.

2.2.4. Network Tunneling

For its network tunneling capabilities, BRICKSTORM receives a header configuring the intended tunnel. The first byte defines the targeted network which can be `0x00` for TCP, `0x01` for UDP or `0x02` for ICMP (IPv4). The second byte defines the target's address type. Similar to the SOCKS5 addressing[7], possible values are:

- `0x01` for IPv4, indicating the next 4 bytes compose the IPv4 address.
- `0x03` for a string (e.g., host name), indicating the next byte defines the string length followed by the string's content.
- `0x04` for IPv6, indicating the next 16 bytes compose the IPv6 address.

Following the target address, an additional 2 bytes define the target port in network byte order[8].

Figure 14: BRICKSTORM's pseudo-SOCKS header.

Once the header received, BRICKSTORM establishes the desired connections and tunnels the connection with the established attacker session. For TCP targets, the backdoor has a dial timeout of 60 seconds, ensuring TCP- tunnels which cannot be established are abandoned after a minute.

7 https://www.rfc-editor.org/rfc/rfc1928#section-5
8 https://en.wikipedia.org/wiki/Endianness#Networking

Detailed Analysis

2.3 Infrastructure

For Command & Control, BRICKSTORM's observed configurations exclusively relied on serverless providers such as Cloudflare or Heroku (see *Figure 15* & *Figure 16*). The usage of such providers obfuscates BRICKSTORM's infrastructure due to the shared and distributed nature of the provider's IP addresses. The usage of such serverless providers is common within the threat landscape as distinguishing between legitimate and malicious usage poses a challenge to many organizations.

```
core_Server.ServerAddr.ptr = "https://ms-azure.azdatastore.workers.dev";
core_Server.ServerAddr.len = 40LL;
core_Server.WsAddr.ptr = "wss://ms-azure.azdatastore.workers.dev/state";
core_Server.WsAddr.len = 44LL;
core_Server.AuthKey.ptr = "[redacted]";
core_Server.AuthKey.len = 21LL;

core_Server.ServerAddr.ptr = "https://ms-azure.herokuapp.com";
core_Server.ServerAddr.len = 30LL;
core_Server.WsAddr.ptr = "wss://ms-azure.herokuapp.com/state";
core_Server.WsAddr.len = 34LL;
core_Server.AuthKey.ptr = "[redacted]";
core_Server.AuthKey.len = 21LL;
```

Figure 16: BRICKSTORM's historical Heroku Command & Control configuration.

Detailed Analysis

As Cloudflare workers use a "sub-sub-domain" (i.e., `ms-azure[.]azdatastore[.]workers[.]dev`), and given TLS certificates do not support nested wildcards (i.e., no `*.*.workers.dev`), certificate transparency logs provide valuable insights into when the associated 1st-tier workloads were registered. Specifically, the observed Cloudflare infrastructure was set up as early as November 28th, 2022; with older Heroku infrastructure predating this datapoint. The Mandiant-reported Linux BRICKSTORM sample had its infrastructure set up on January 2nd, 2024.

crt.sh ID	Logged At	Not Before	Not After	Common Name	Matching Identities	Issuer Name
15141250474	2024-10-29	2024-10-29	2025-01-27	azdatastore.workers.dev	*.ms-azure.azdatastore.workers.dev	C=US, O=Google Trust Services, CN=WE1
15132043603	2024-10-29	2024-10-29	2025-01-27	azdatastore.workers.dev	*.ms-azure.azdatastore.workers.dev	C=US, O=Google Trust Services, CN=WR1
15140014427	2024-10-29	2024-10-29	2025-01-27	azdatastore.workers.dev	*.ms-azure.azdatastore.workers.dev	C=US, O=Google Trust Services, CN=WE1
15139737246	2024-10-29	2024-10-29	2025-01-27	azdatastore.workers.dev	*.ms-azure.azdatastore.workers.dev	C=US, O=Google Trust Services, CN=WR1
14376824719	2024-08-31	2024-08-31	2024-11-29	azdatastore.workers.dev	*.ms-azure.azdatastore.workers.dev	C=US, O=Google Trust Services, CN=WE1
14376759055	2024-08-31	2024-08-31	2024-11-29	azdatastore.workers.dev	*.ms-azure.azdatastore.workers.dev	C=US, O=Google Trust Services, CN=WR1
14029940509	2024-07-30	2024-07-30	2024-10-28	azdatastore.workers.dev	*.ms-azure.azdatastore.workers.dev	C=US, O=Let's Encrypt, CN=E6
13929597667	2024-07-30	2024-07-30	2024-10-28	azdatastore.workers.dev	*.ms-azure.azdatastore.workers.dev	C=US, O=Let's Encrypt, CN=E6
14029936271	2024-07-30	2024-07-30	2024-10-28	azdatastore.workers.dev	*.ms-azure.azdatastore.workers.dev	C=US, O=Let's Encrypt, CN=R11
13972330754	2024-07-30	2024-07-30	2024-10-28	azdatastore.workers.dev	*.ms-azure.azdatastore.workers.dev	C=US, O=Let's Encrypt, CN=R11
13251592855	2024-06-01	2024-06-01	2024-08-30	azdatastore.workers.dev	*.ms-azure.azdatastore.workers.dev	C=US, O=Let's Encrypt, CN=E1
13251586774	2024-06-01	2024-06-01	2024-08-30	azdatastore.workers.dev	*.ms-azure.azdatastore.workers.dev	C=US, O=Let's Encrypt, CN=E1
13251591200	2024-06-01	2024-06-01	2024-08-30	azdatastore.workers.dev	*.ms-azure.azdatastore.workers.dev	C=US, O=Let's Encrypt, CN=R3
13251586339	2024-06-01	2024-06-01	2024-08-30	azdatastore.workers.dev	*.ms-azure.azdatastore.workers.dev	C=US, O=Let's Encrypt, CN=R3
12624534133	2024-04-03	2024-04-03	2024-07-02	azdatastore.workers.dev	*.ms-azure.azdatastore.workers.dev	C=US, O=Let's Encrypt, CN=E1
12597776520	2024-04-03	2024-04-03	2024-07-02	azdatastore.workers.dev	*.ms-azure.azdatastore.workers.dev	C=US, O=Let's Encrypt, CN=E1
12624534115	2024-04-03	2024-04-03	2024-07-02	azdatastore.workers.dev	*.ms-azure.azdatastore.workers.dev	C=US, O=Let's Encrypt, CN=R3
12585760841	2024-04-03	2024-04-03	2024-07-02	azdatastore.workers.dev	*.ms-azure.azdatastore.workers.dev	C=US, O=Let's Encrypt, CN=R3
11972216158	2024-02-04	2024-02-04	2024-05-04	azdatastore.workers.dev	*.ms-azure.azdatastore.workers.dev	C=US, O=Let's Encrypt, CN=E1
11972211370	2024-02-04	2024-02-04	2024-05-04	azdatastore.workers.dev	*.ms-azure.azdatastore.workers.dev	C=US, O=Let's Encrypt, CN=E1
11972218214	2024-02-04	2024-02-04	2024-05-04	azdatastore.workers.dev	*.ms-azure.azdatastore.workers.dev	C=US, O=Let's Encrypt, CN=R3
11972211476	2024-02-04	2024-02-04	2024-05-04	azdatastore.workers.dev	*.ms-azure.azdatastore.workers.dev	C=US, O=Let's Encrypt, CN=R3
11341238561	2023-12-07	2023-12-07	2024-03-06	azdatastore.workers.dev	*.ms-azure.azdatastore.workers.dev	C=US, O=Let's Encrypt, CN=E1
11341233778	2023-12-07	2023-12-07	2024-03-06	azdatastore.workers.dev	*.ms-azure.azdatastore.workers.dev	C=US, O=Let's Encrypt, CN=E1
11341238298	2023-12-07	2023-12-07	2024-03-06	azdatastore.workers.dev	*.ms-azure.azdatastore.workers.dev	C=US, O=Let's Encrypt, CN=R3
11341233510	2023-12-07	2023-12-07	2024-03-06	azdatastore.workers.dev	*.ms-azure.azdatastore.workers.dev	C=US, O=Let's Encrypt, CN=R3
10670492925	2023-10-09	2023-10-09	2024-01-07	azdatastore.workers.dev	*.ms-azure.azdatastore.workers.dev	C=US, O=Let's Encrypt, CN=E1
10681028679	2023-10-09	2023-10-09	2024-01-07	azdatastore.workers.dev	*.ms-azure.azdatastore.workers.dev	C=US, O=Let's Encrypt, CN=E1
10681416718	2023-10-09	2023-10-09	2024-01-07	azdatastore.workers.dev	*.ms-azure.azdatastore.workers.dev	C=US, O=Let's Encrypt, CN=R3
10670260728	2023-10-09	2023-10-09	2024-01-07	azdatastore.workers.dev	*.ms-azure.azdatastore.workers.dev	C=US, O=Let's Encrypt, CN=R3
10253233348	2023-08-11	2023-08-10	2023-11-08	azdatastore.workers.dev	*.ms-azure.azdatastore.workers.dev	C=US, O=Let's Encrypt, CN=E1
10140335128	2023-08-11	2023-08-10	2023-11-08	azdatastore.workers.dev	*.ms-azure.azdatastore.workers.dev	C=US, O=Let's Encrypt, CN=E1
10253231806	2023-08-11	2023-08-10	2023-11-08	azdatastore.workers.dev	*.ms-azure.azdatastore.workers.dev	C=US, O=Let's Encrypt, CN=R3
10140334498	2023-08-11	2023-08-10	2023-11-08	azdatastore.workers.dev	*.ms-azure.azdatastore.workers.dev	C=US, O=Let's Encrypt, CN=R3
9666372272	2023-06-12	2023-06-12	2023-09-10	azdatastore.workers.dev	*.ms-azure.azdatastore.workers.dev	C=US, O=Let's Encrypt, CN=E1
9637422122	2023-06-12	2023-06-12	2023-09-10	azdatastore.workers.dev	*.ms-azure.azdatastore.workers.dev	C=US, O=Let's Encrypt, CN=E1
9666372090	2023-06-12	2023-06-12	2023-09-10	azdatastore.workers.dev	*.ms-azure.azdatastore.workers.dev	C=US, O=Let's Encrypt, CN=R3
9637746862	2023-06-12	2023-06-12	2023-09-10	azdatastore.workers.dev	*.ms-azure.azdatastore.workers.dev	C=US, O=Let's Encrypt, CN=R3
9155793346	2023-04-14	2023-04-14	2023-07-13	azdatastore.workers.dev	*.ms-azure.azdatastore.workers.dev	C=US, O=Let's Encrypt, CN=E1
9149551389	2023-04-14	2023-04-14	2023-07-13	azdatastore.workers.dev	*.ms-azure.azdatastore.workers.dev	C=US, O=Let's Encrypt, CN=E1
9155788332	2023-04-14	2023-04-14	2023-07-13	azdatastore.workers.dev	*.ms-azure.azdatastore.workers.dev	C=US, O=Let's Encrypt, CN=R3
9150134513	2023-04-14	2023-04-14	2023-07-13	azdatastore.workers.dev	*.ms-azure.azdatastore.workers.dev	C=US, O=Let's Encrypt, CN=R3
9093237458	2023-04-06	2023-04-06	2023-07-05	*.ms-azure.azdatastore.workers.dev	*.ms-azure.azdatastore.workers.dev	C=US, O=Let's Encrypt, CN=R3
9073197006	2023-04-06	2023-04-06	2023-07-05	*.ms-azure.azdatastore.workers.dev	*.ms-azure.azdatastore.workers.dev	C=US, O=Let's Encrypt, CN=R3
9047532897	2023-03-31	2023-03-31	2023-06-29	*.ms-azure.azdatastore.workers.dev	*.ms-azure.azdatastore.workers.dev	C=US, O=Let's Encrypt, CN=R3
9016670984	2023-03-31	2023-03-31	2023-06-29	*.ms-azure.azdatastore.workers.dev	*.ms-azure.azdatastore.workers.dev	C=US, O=Let's Encrypt, CN=R3
9047515537	2023-03-31	2023-03-31	2023-06-29	*.ms-azure.azdatastore.workers.dev	*.ms-azure.azdatastore.workers.dev	C=US, O=Let's Encrypt, CN=R3
9016658717	2023-03-31	2023-03-31	2023-06-29	*.ms-azure.azdatastore.workers.dev	*.ms-azure.azdatastore.workers.dev	C=US, O=Let's Encrypt, CN=R3
8622261058	2023-01-31	2023-01-31	2023-05-01	*.ms-azure.azdatastore.workers.dev	*.ms-azure.azdatastore.workers.dev	C=US, O=Let's Encrypt, CN=E1
8539512743	2023-01-31	2023-01-31	2023-05-01	*.ms-azure.azdatastore.workers.dev	*.ms-azure.azdatastore.workers.dev	C=US, O=Let's Encrypt, CN=E1
8622236488	2023-01-31	2023-01-31	2023-05-01	*.ms-azure.azdatastore.workers.dev	*.ms-azure.azdatastore.workers.dev	C=US, O=Let's Encrypt, CN=R3
8539495255	2023-01-31	2023-01-31	2023-05-01	*.ms-azure.azdatastore.workers.dev	*.ms-azure.azdatastore.workers.dev	C=US, O=Let's Encrypt, CN=R3
8083455763	2022-11-28	2022-11-28	2023-02-26	*.ms-azure.azdatastore.workers.dev	*.ms-azure.azdatastore.workers.dev	C=US, O=Let's Encrypt, CN=E1
8078797581	2022-11-28	2022-11-28	2023-02-26	*.ms-azure.azdatastore.workers.dev	*.ms-azure.azdatastore.workers.dev	C=US, O=Let's Encrypt, CN=E1
8083456828	2022-11-28	2022-11-28	2023-02-26	*.ms-azure.azdatastore.workers.dev	*.ms-azure.azdatastore.workers.dev	C=US, O=Let's Encrypt, CN=R3
8079473999	2022-11-28	2022-11-28	2023-02-26	*.ms-azure.azdatastore.workers.dev	*.ms-azure.azdatastore.workers.dev	C=US, O=Let's Encrypt, CN=R3

Figure 17: BRICKSTORM's Cloudflare TLS certificates as seen in crt.sh certificate transparency logs.

Detailed Analysis

During NVISO's monitoring of the campaign, BRICKSTORM operators were found updating back-end infrastructure which inadvertently exposed their usage of 2nd-tier infrastructure. The following image was captured during temporary downtime of the 1st-tier `ms-azure[.]azdatastore[.]workers[.]dev` infrastructure, exposing the `64[.]176[.]166[.]79` Vultr instance as 2nd-tier infrastructure. Following this maintenance window, a new 2nd-layer TLS certificate was served (see "2nd Layer TLS Certificate (2024-2025)" on page 29).

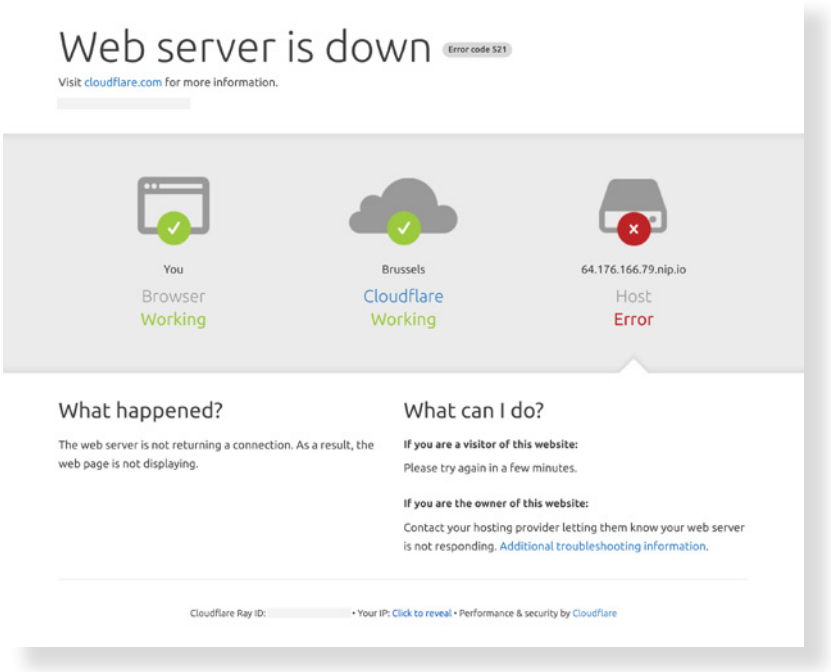

Figure 18: Cloudflare's error message, exposing 2nd-tier Vultr infrastructure.

While irrelevant from an architecture perspective, BRICKSTORM operators leveraged `nip.io` to provide their Vultr instance with a DNS record. The 2nd-tier Vultr instance received its proxied Command & Control traffic on the default unencrypted HTTP port, with a default `404 Not Found` message acting as a deception lure.

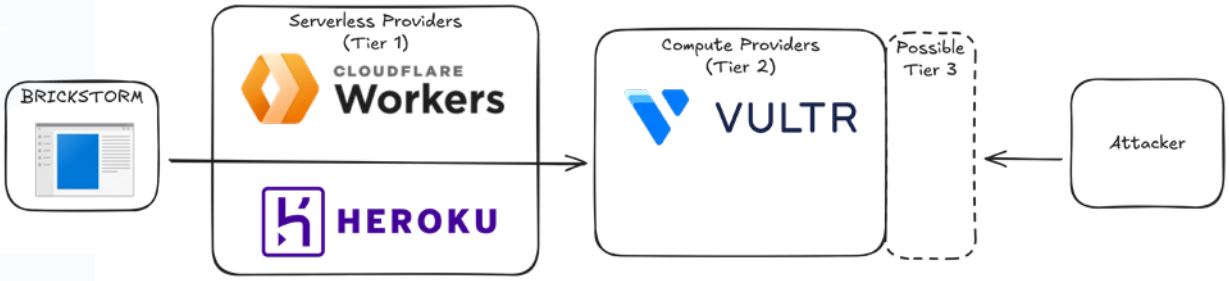

Figure 19: BRICKSTORM's assumed multi-tiered architecture.

As part of its monitoring activities, NVISO identified additional suspected 2nd-tier Vultr instances leveraged for similar campaigns. The 2nd-tier instances could however not be validated as the configuration of the reverse TLS proxies is unknown.

Conclusions & Recommendations

The inner workings of the BRICKSTORM backdoor provide effective means to bypass common security controls such as DNS monitoring and geo-blocking at network level. The usage of legitimate and popular cloud providers blends the Command & Control communications amongst every-day traffic while complicating clustering and attribution.

Although BRICKSTORM's file manager and network tunneling functionality could be considered basic, their effectiveness remains undoubted. These recent discoveries of several year old adversary capabilities, alongside evidence of infrastructure maintenance, highlight the need for at-risk industries to bolster their security posture and continuously audit their environment for rare/uncommon activity. To this end, NVISO provides as appendix detection and hunting rules that highlight similar suspicious activity; Organizations are encouraged to review their environment for similar patterns.

While BRICKSTORM can be configured to operate without DoH (DNS over HTTPS), organizations are nonetheless recommended to block DoH providers on their network; preventing certain BRICKSTORM variations and similar samples from operating. Organizations with strict network controls are encouraged to ensure their TLS inspection either detects or, ideally, blocks nested TLS sessions.

NVISO values the collaboration of affected organizations who, through their willingness to share information, help the community prevent and detect similar intrusions.

NVISO's incident response teams can be reached 24/7 should you be affected by BRICKSTORM or face a similar threat.

Belgium/Global:	+32 2 588 43 80 csirt@nviso.eu
Germany:	+49 69 8088 3829 csirt@nviso.de
Austria:	+43 720 228 337 csirt@nviso.at

Appendix
Indicators of Compromise

The following indicators have been observed in BRICKSTORM intrusions. Organizations encountering the beneath elements in their environment should initiate their incident response procedures.

Organizations requiring additional assistance can reach out to NVISO's 24/7 incident response teams[9] through csirt@nviso.eu.

Indicator	Value
Filename	CreatedUACExplorer.exe
Size	7687168 bytes (7507 KiB)
SHA256	b42159d68ba58d7857c091b5acc59e30e50a854b15f7ce04b61ff6c11cdf0156
SHA1	b4af963d43b6e834a28ad281c2004d348a91b938
MD5	c65d7f8accb57a95e3ea8a07fac9550f
Domain*	ms-azure[.]azdatastore[.]workers[.]dev

**The BRICKSTORM command & control domains are resolved according to section 2.2.1 and can subsequently not be identified through DNS logs.*

Alongside the above sample, the following historical sample was identified.

Indicator	Value
Filename	CreateUACExplorer.exe
Size	7677440 bytes (7497 KiB)
SHA256	42692bd13333623e9085d0c1326574a3391efcbf18158bb04972103c9ee4a3b8
SHA1	e57515297ee77c595eec19c00b2a77bba0171879
MD5	8af1c3f39b60072d4b68c77001d58109
Domain*	ms-azure[.]herokuapp[.]com

**The BRICKSTORM command & control domains are resolved according to section 2.2.1 and can subsequently not be identified through DNS logs.*

[9] https://www.nviso.eu/contact

Appendix
Detection & Hunting Rules

The following detection rules cover Windows samples observed by NVISO; refer to Mandiant's notes[10] for vCenter appliances.

5.1. YARA

```
rule NVISO_BACKDOOR_BRICKSTORM {
  meta:
    description = "Detects the BRICKSTORM backdoor Windows executables"
    author      = "NVISO"
    created     = "2024-11-25"
    md5         = "8af1c3f39b60072d4b68c77001d58109"
    md5         = "c65d7f8accb57a95e3ea8a07fac9550f"
    license     = "Detection Rule License (DRL) 1.1"
    reference   = "https://nviso.eu/blog/nviso-analyzes-brickstorm-espionage-backdoor"

  strings:
    $lib1   = "wsshell/core/task.DoTask" ascii wide
    $lib2   = "wssoft/core/task.DoTask"  ascii wide
    $wss    = "wss://"       ascii wide
    $go     = "/golang.org/" ascii wide
    $doh01  = "https://1.0.0.1/dns-query"  ascii wide
    $doh02  = "https://1.1.1.1/dns-query"  ascii wide
    $doh03  = "https://8.8.4.4/dns-query"  ascii wide
    $doh04  = "https://8.8.8.8/dns-query"  ascii wide
    $doh05  = "https://9.9.9.9/dns-query"  ascii wide
    $doh06  = "https://9.9.9.11/dns-query" ascii wide
    $doh07  = "https://45.90.28.160/dns-query"    ascii wide
    $doh08  = "https://45.90.30.160/dns-query"    ascii wide
    $doh09  = "https://149.112.112.11/dns-query"  ascii wide
    $doh10  = "https://149.112.112.112/dns-query" ascii wide
    $cmd1   = "/get-file" ascii wide
    $cmd2   = "/put-file" ascii wide
    $cmd3   = "/slice-up" ascii wide
    $cmd4   = "/file-md5" ascii wide

  condition:
    uint16be(0) == 0x4D5A
    and any of ($lib*)
    and any of ($doh*)
    and any of ($cmd*)
    and $wss and $go
}
```

[10] https://cloud.google.com/blog/topics/threat-intelligence/ivanti-post-exploitation-lateral-movement

Appendix
Detection & Hunting Rules

5.2. Suricata (Network)

The following IDS (Intrusion Detection System) rule alerts on known BRICKSTORM Command & Control connections.

```
alert tls $HOME_NET any -> $EXTERNAL_NET any (msg:"[NVISO] Observed BRICKSTORM CnC
Domain (ms-azure .azdatastore .workers .dev in TLS SNI)"; flow:established,to_server;
tls.sni; bsize:32; content:"ms-azure.azdatastore.workers.dev"; fast_pattern;
reference:url,nviso.eu/blog/nviso-analyzes-brickstorm-espionage-backdoor;
classtype:domain-c2; sid:1; rev:1; metadata:attack_target Server_Endpoint, created_at
2025_03_25, deployment Perimeter, performance_impact Low, confidence High,
signature_severity Critical, malware_family BRICKSTORM;)
```

Organizations are furthermore encouraged to consider additional public rules such as Proofpoint's `ET INFO Observed Cloudflare workers.dev Domain in TLS SNI`[11], potentially identifying similar infrastructure.

[11] https://rules.emergingthreats.net/open/suricata-7.0.9/rules/emerging-info.rules

5.3. Kusto (KQL) Hunting

The following Kusto queries, intended for threat hunting, are designed for usage within Microsoft Defender's Advanced Hunting due to the dependency on the `FileProfile`[12] function.

Given the backdoor's long-running nature, the following query identifies rare processes that have been running for over 10 days while making network connections.

```
let Lookback = 30d;                 //Parameters:
let ProcessAge = 10d;               // The minimal age of the running process
let URLThreshold = 2;               // The limit of contacted URLs
let LocalPrevalenceThreshold = 5;   // The limit of internal sightings
let GlobalPrevalenceThreshold = 20; // The limit of world-wide sightings
// Identify long-running processes performing successful public network connections
DeviceNetworkEvents
| where Timestamp > ago(Lookback)
    and isnotempty(InitiatingProcessSHA256)
    and RemoteIPType == "Public"
    and ActionType == "ConnectionSuccess"
    and InitiatingProcessCreationTime < Timestamp-ProcessAge
| summarize
    DeviceCount=dcount(DeviceId),
    DeviceNames=make_set(DeviceName, LocalPrevalenceThreshold),
    IPCount=dcount(RemoteIP),
    URLCount=dcountif(RemoteUrl, isnotempty(RemoteUrl)),
    arg_max(Timestamp, *)
    by InitiatingProcessSHA256
// Where the executables have rarely been seen publicly
| where URLCount <= URLThreshold and DeviceCount <= LocalPrevalenceThreshold
| as IntermediaryResult
| where assert(toscalar(IntermediaryResult | count) <= 1000, "Too many matches for FileProfile")
| invoke FileProfile("InitiatingProcessSHA256", 1000)
| where GlobalPrevalence <= GlobalPrevalenceThreshold
// Order the results by priority
| project-reorder
    Timestamp,
    DeviceNames,
    GlobalPrevalence,
    InitiatingProcessFolderPath,
    InitiatingProcessCommandLine,
    RemoteIP,
    RemoteUrl,
    SHA256,
    IPCount,
    URLCount
| order by GlobalPrevalence asc, URLCount asc, IPCount desc
```

[12] https://learn.microsoft.com/en-us/defender-xdr/advanced-hunting-fileprofile-function

Appendix
Detection & Hunting Rules

Given the backdoors were not signed, the following query identifies uncommon and unsigned System executables interacting with Cloudflare. Do note it relies on hard-coded ranges due to performance issues combining the `externaldata` and `FileProfile` operators.

```
let Lookback = 30d;
// Define Cloudflare IPs (see https://www.cloudflare.com/ips-v4/#)
let Cloudflare = datatable(Range: string)[
    "173.245.48.0/20" ,   "103.21.244.0/22" ,    "103.22.200.0/22",   "103.31.4.0/22"  ,
    "141.101.64.0/18" ,   "108.162.192.0/18",    "190.93.240.0/20",   "188.114.96.0/20",
    "197.234.240.0/22",   "198.41.128.0/17" ,    "162.158.0.0/15" ,   "104.16.0.0/13"  ,
    "104.24.0.0/14"   ,   "172.64.0.0/13"   ,    "131.0.72.0/22"];
// Identify system processes making Cloudflare connections
DeviceNetworkEvents
| where Timestamp > ago(Lookback) and InitiatingProcessAccountName =~ "System"
| evaluate ipv4_lookup(Cloudflare, RemoteIP, Range)
| summarize Count=count()
    by
    DeviceId,
    DeviceName,
    InitiatingProcessFolderPath,
    InitiatingProcessFileName,
    InitiatingProcessSHA256,
    InitiatingProcessSHA1
// Where the file is not signed (based on Defender telemetry)
| join kind=leftanti (DeviceFileCertificateInfo | where Timestamp > ago(Lookback))
    on $left.InitiatingProcessSHA1 == $right.SHA1
| summarize
    Devices=dcount(DeviceId),
    Count=sum(Count),
    InitiatingProcessFolderPath=make_list(InitiatingProcessFolderPath),
    DeviceName=make_list(DeviceName),
    InitiatingProcessFileName=make_list(InitiatingProcessFileName)
    by InitiatingProcessSHA256, InitiatingProcessSHA1
// Where the file is not signed (based on Microsoft telemetry) and uncommon
| as IntermediaryResult
| where assert(toscalar(IntermediaryResult | count) <= 1000, "Too many matches for FileProfile")
| invoke FileProfile("InitiatingProcessSHA1", 1000)
| where SignatureState == "Unsigned" and GlobalPrevalence < 50000
| mv-expand DeviceName, InitiatingProcessFolderPath, InitiatingProcessFileName
| project-reorder DeviceName, InitiatingProcessFolderPath, SHA256, SHA1, Count
| order by Count desc
```

Appendix
2nd Layer TLS Certificate (2023-2024)

As described in section "2.2.2. Command & Control" on page 10, BRICKSTORM's protocol relies on 3 layers of TLS encryption. While the first layer is provided by legitimate cloud service providers, the second and potentially third layer are attacker-controlled. The following non-redacted x509 certificate has been recovered as part of the 2nd layer TLS handshake and is provided for pivoting and clustering purposes.

```
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number:
            ca:56:1f:f5:5d:fb:f1:03:a9:04:5a:81:c5:d5:5f:02
        Signature Algorithm: sha256WithRSAEncryption
        Issuer: O=
        Validity
            Not Before: Oct 19 13:14:40 2023 GMT
            Not After : Oct 18 13:14:40 2024 GMT
        Subject: O=
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                Public-Key: (2048 bit)
                Modulus:
                    00:b5:37:32:d3:60:59:7f:7e:5e:55:4e:1e:06:7b:
                    97:14:b1:84:11:d4:71:10:9d:8a:b0:93:87:87:f2:
                    77:0b:f5:05:c5:b5:12:79:f6:29:87:eb:43:08:3c:
                    cf:25:88:74:bb:0b:b0:7e:dc:17:5f:a0:dc:15:5c:
                    67:ef:f2:2d:53:39:24:49:e6:cf:25:c0:1c:29:8e:
                    2e:dd:f8:f6:2c:1b:7f:42:49:24:f6:e0:ef:4a:83:
                    5a:9d:84:12:a0:39:4e:1c:1a:9c:88:5b:ac:be:c3:
                    8e:aa:71:0f:c4:85:94:55:b8:d3:9a:57:e8:22:e0:
                    65:a1:0b:af:5f:4e:72:14:d6:33:2e:86:4c:1b:b7:
                    ef:f2:a3:26:28:67:4b:0b:b0:ad:a7:75:79:50:fa:
                    d1:70:b6:f9:ff:7e:d5:a0:7d:4f:e8:0d:7f:ce:a0:
                    35:41:f7:f8:72:ec:c9:11:65:6b:c1:bc:49:be:ae:
                    42:c3:da:23:5d:6c:6d:b0:7a:46:a7:23:fa:7c:69:
                    1a:73:5c:2d:29:0f:9a:03:91:09:fa:a0:a3:18:6a:
                    ca:c5:c0:95:87:38:74:ff:6b:9a:0f:fb:ac:c9:79:
                    1b:01:8d:fe:3a:4e:7e:2b:65:c6:4a:c4:6c:e9:12:
                    e4:3d:aa:71:cb:0e:73:a5:00:2d:0f:67:9d:a8:6a:
                    44:1b
                Exponent: 65537 (0x10001)
        X509v3 extensions:
            X509v3 Key Usage: critical
                Digital Signature, Key Encipherment, Certificate Sign
            X509v3 Extended Key Usage:
                TLS Web Server Authentication
            X509v3 Basic Constraints: critical
                CA:TRUE
            X509v3 Subject Alternative Name:
```

Appendix
2nd Layer TLS Certificate (2023-2024)

```
                DNS:localhost
    Signature Algorithm: sha256WithRSAEncryption
    Signature Value:
        0e:36:bf:ed:4d:33:e1:0f:22:5a:a0:5a:33:8f:1f:ac:a9:c1:
        a8:23:31:7d:56:84:3c:50:d1:eb:10:1f:b6:d8:95:a3:cc:7a:
        03:0d:10:f8:a7:16:98:08:76:cd:df:cf:c7:04:5b:9a:01:f8:
        51:99:8e:d8:32:73:e5:a8:c8:1f:ea:0c:eb:09:64:75:9d:5b:
        b4:d8:aa:73:fa:c7:d9:bc:49:22:94:2d:1f:3d:1d:85:a6:2f:
        9b:e6:15:04:42:53:fb:bf:44:1a:20:01:17:0a:fd:15:aa:15:
        cd:4e:57:cf:3b:5f:7d:55:83:b0:ee:f7:8b:f2:b2:80:32:55:
        14:95:aa:91:8e:43:4b:ad:0f:20:50:f5:ed:db:de:4c:14:61:
        f2:f0:83:cc:f4:79:55:20:82:5d:04:47:1f:f1:50:6c:05:e8:
        bc:9b:6c:7e:97:ec:38:4d:00:fc:dd:b7:f1:fd:62:8c:64:c9:
        88:f2:13:b7:9b:c0:36:0c:b3:0d:d7:fe:2f:5b:b1:cb:32:71:
        36:7c:1a:7b:73:34:b9:07:0c:d7:6f:4a:a7:d7:32:30:82:dc:
        f8:30:30:bf:2c:6a:bf:17:1a:be:1d:c1:ee:c1:c9:9a:ac:85:
        dc:d2:5e:82:51:e0:df:43:5e:52:e2:ec:d2:90:ff:16:3c:85:
        b6:69:59:ac
-----BEGIN CERTIFICATE-----
MIIC7zCCAdegAwIBAgIRAMpWH/Vd+/EDqQRagcXVXwIwDQYJKoZIhvcNAQELBQAw
CzEJMAcGA1UEChMAMB4XDTIzMTAxOTEzMTQ0MFoXDTI0MTAxODEzMTQ0MFowCzEJ
MAcGA1UEChMAMIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAtTcy02BZ
f35eVU4eBnuXFLGEEdRxEJ2KsJOHh/J3C/UFxbUSefYph+tDCDzPJYh0uwuwftwX
X6DcFVxn7/ItUzkkSebPJcAcKY4u3fj2LBt/Qkkk9uDvSoNanYQSoDlOHBqciFus
vsOOqnEPxIWUVbjTmlfoIuBloQuvX05yFNYzLoZMG7fv8qMmKGdLC7Ctp3V5UPrR
cLb5/37VoH1P6A1/zqA1Qff4cuzJEWVrwbxJvq5Cw9ojXWxtsHpGpyP6fGkac1wt
KQ+aA5EJ+qCjGGrKxcCVhzh0/2uaD/usyXkbAY3+Ok5+K2XGSsRs6RLkPapxyw5z
pQAtD2edqGpEGwIDAQABo04wTDAOBgNVHQ8BAf8EBAMCAqQwEwYDVR0lBAwwCgYI
KwYBBQUHAwEwDwYDVR0TAQH/BAUwAwEB/zAUBgNVHREEDTALgglsb2NhbGhvc3Qw
DQYJKoZIhvcNAQELBQADggEBAA42v+1NM+EPIlqgWjOPH6ypwagjMX1WhDxQ0esQ
H7bYlaPMegMNEPinFpgIds3fz8cEW5oB+FGZjtgyc+WoyB/qDOsJZHWdW7TYqnP6
x9m8SSKULR89HYWmL5vmFQRCU/u/RBogARcK/RWqFc1OV887X31Vg7Du94vysoAy
VRSVqpGOQ0utDyBQ9e3b3kwUYfLwg8z0eVUggl0ERx/xUGwF6LybbH6X7DhNAPzd
t/H9YoxkyYjyE7ebwDYMsw3X/i9bscsycTZ8GntzNLkHDNdvSqfXMjCC3PgwML8s
ar8XGr4dwe7ByZqshdzSXoJR4N9DXlLi7NKQ/xY8hbZpWaw=
-----END CERTIFICATE-----
```

Appendix
2nd Layer TLS Certificate (2024-2025)

The beneath x509 certificate has been recovered as part of the 2nd layer TLS handshake following the maintenance window described in section "2.3 Infrastructure" on page 17; It is provided for pivoting and clustering purposes.

```
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number:
            5b:3f:1a:69:74:17:df:2b:0e:a1:52:f0:22:12:f8:d4
        Signature Algorithm: sha256WithRSAEncryption
        Issuer: O=
        Validity
            Not Before: Dec 18 07:49:40 2024 GMT
            Not After : Dec 18 07:49:40 2025 GMT
        Subject: O=
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                Public-Key: (2048 bit)
                Modulus:
                    00:b0:fb:8e:13:72:94:26:b5:01:60:40:dd:41:28:
                    e0:21:37:17:2f:e7:78:af:54:52:c1:a3:1d:de:37:
                    53:0e:f6:4d:b0:a2:f7:9e:2f:d0:6e:c6:6d:62:e4:
                    3d:04:92:d0:6f:9b:0a:19:aa:dc:b0:b3:56:31:d4:
                    1c:fc:52:c6:fe:c6:f8:cf:bb:d7:27:88:ae:14:c3:
                    b7:f4:85:a2:5a:ee:79:2c:f7:32:ce:db:f4:9f:20:
                    22:99:ff:99:64:ac:12:f7:e4:ae:9b:56:68:d2:55:
                    b9:d6:aa:9b:36:84:96:8b:c5:04:cf:50:26:67:ef:
                    75:2f:15:08:79:07:a8:4a:ac:53:15:1a:cd:6b:54:
                    e2:f9:e0:99:f1:34:a4:7e:c4:a9:e5:8e:e5:c1:0a:
                    f0:a6:6c:1d:b2:76:74:8e:52:f8:55:31:80:df:ae:
                    06:84:89:1d:1a:90:d7:32:43:a2:02:56:57:a6:f5:
                    4c:f1:ce:11:de:7d:73:3e:4a:c4:8d:79:39:d9:bd:
                    b6:04:59:d8:5a:83:9e:d5:b5:c2:ef:15:75:c0:07:
                    82:72:e6:06:c8:68:b8:4c:c5:33:43:bf:97:cc:39:
                    a5:28:bb:a3:77:d4:04:3d:ee:54:93:9e:b1:5c:aa:
                    e7:64:77:25:aa:ed:42:1d:4b:06:b0:e8:f1:4a:54:
                    fe:5b
                Exponent: 65537 (0x10001)
        X509v3 extensions:
            X509v3 Key Usage: critical
                Digital Signature, Key Encipherment, Certificate Sign
            X509v3 Extended Key Usage:
                TLS Web Server Authentication
            X509v3 Basic Constraints: critical
                CA:TRUE
            X509v3 Subject Alternative Name:
                DNS:localhost
```

Appendix
2nd Layer TLS Certificate (2024-2025)

```
    Signature Algorithm: sha256WithRSAEncryption
     Signature Value:
         63:5e:92:03:e5:fd:1b:51:6f:78:ed:c3:c5:49:9d:30:f6:06:
         0f:92:75:b2:32:a9:65:e1:a5:76:6b:eb:30:7a:fd:c8:a2:39:
         ca:da:ad:76:11:0e:13:40:81:6c:f9:be:2b:46:d5:58:6d:5b:
         3b:8c:94:7b:d4:95:1e:f2:78:94:fb:99:cc:33:8e:ae:2c:91:
         10:c5:b8:3f:5a:25:2a:b4:6f:c9:8c:32:3b:ea:88:fb:bf:1d:
         e2:34:7e:fb:5d:71:4b:61:8e:1c:28:83:a0:bf:1d:8f:eb:0d:
         ec:2c:20:f4:a0:82:8c:2c:70:e4:60:a5:29:7e:37:86:fe:d7:
         f9:62:7e:b7:d9:3e:f6:ef:59:91:ea:ef:fb:31:93:e3:b9:52:
         5e:ae:6c:4c:94:69:27:36:eb:e6:10:8c:8b:bb:5e:51:0d:1c:
         79:f1:6d:f4:6b:66:51:fa:12:9e:33:62:cc:04:03:6f:86:8d:
         bd:0a:96:db:cc:45:80:1d:1b:a9:cf:9f:22:b0:2b:bf:6c:0c:
         42:61:cb:03:53:71:af:d2:29:d2:e7:c9:b8:61:31:87:11:68:
         04:38:8c:0d:7a:4f:3a:ea:26:fd:8e:2b:d6:cb:86:4b:a8:69:
         76:9c:47:4f:48:4a:23:27:06:99:4c:25:76:92:dd:87:33:e5:
         ee:7f:95:16
-----BEGIN CERTIFICATE-----
MIIC7jCCAdagAwIBAgIQWz8aaXQX3ysOoVLwIhL41DANBgkqhkiG9w0BAQsFADAL
MQkwBwYDVQQKEwAwHhcNMjQxMjE4MDc0OTQwWhcNMjUxMjE4MDc0OTQwWjALMQkw
BwYDVQQKEwAwggEiMA0GCSqGSIb3DQEBAQUAA4IBDwAwggEKAoIBAQCw+44TcpQm
tQFgQN1BKOAhNxcv53ivVFLBox3eN1MO9k2woveeL9Buxm1i5D0EktBvmwoZqtyw
s1Yx1Bz8Usb+xvjPu9cniK4Uw7f0haJa7nks9zLO2/SfICKZ/5lkrBL35K6bVmjS
VbnWqps2hJaLxQTPUCZn73UvFQh5B6hKrFMVGs1rVOL54JnxNKR+xKnljuXBCvCm
bB2ydnSOUvhVMYDfrgaEiR0akNcyQ6ICVlem9UzxzhHefXM+SsSNeTnZvbYEWdha
g57VtcLvFXXAB4Jy5gbIaLhMxTNDv5fMOaUou6N31AQ97lSTnrFcqudkdyWq7UId
Swaw6PFKVP5bAgMBAAGjTjBMMA4GA1UdDwEB/wQEAwICpDATBgNVHSUEDDAKBggr
BgEFBQcDATAPBgNVHRMBAf8EBTADAQH/MBQGA1UdEQQNMAuCCWxvY2FsaG9zdDAN
BgkqhkiG9w0BAQsFAAOCAQEAY16SA+X9G1FveO3DxUmdMPYGD5J1sjKpZeGldmvr
MHr9yKI5ytqtdhEOE0CBbPm+K0bVWG1bO4yUe9SVHvJ4lPuZzDOOriyREMW4P1ol
KrRvyYwyO+qI+78d4jR++11xS2GOHCiDoL8dj+sN7Cwg9KCCjCxw5GClKX43hv7X
+WJ+t9k+9u9Zkerv+zGT47lSXq5sTJRpJzbr5hCMi7teUQ0cefFt9GtmUfoSnjNi
zAQDb4aNvQqW28xFgB0bqc+fIrArv2wMQmHLA1Nxr9Ip0ufJuGExhxFoBDiMDXpP
Ouom/Y4r1suGS6hpdpxHT0hKIycGmUwldpLdhzPl7n+VFg==
-----END CERTIFICATE-----
```

About NVISO

NVISO is a leading European cyber security firm with offices in Brussels, Frankfurt, Munich, Athens, and Vienna. Founded by seasoned experts, we are a pure-play cyber security company and home to world-class professionals who author SANS Institute trainings, speak at major conferences, and lecture at universities across Europe. Knowledge sharing is at the core of our DNA. Our blog posts and publications are widely cited by security professionals globally.

We specialize in preventing, detecting, and responding to cyber security incidents. Our prevention services tackle infrastructure, application, and human challenges, while our detection and response offerings range from on-demand threat hunting to continuous Managed Detection & Response (MDR) services.

Our CSIRT team is recognized as a Trusted Introducer (TI) member, a FIRST member, and a BSI-listed APT Incident Responder. We regularly share our research on the NVISO Labs blog.

→ blog.nviso.eu

→ nviso.eu

EMERGENCY

Belgium/Global:	+32 2 588 43 80 *csirt@nviso.eu*
Germany:	+49 69 8088 3829 *csirt@nviso.de*
Austria:	+43 720 228 337 *csirt@nviso.at*