

APT36: A Nightmare of Vibeware

B bitdefender.com/en-us/blog/businessinsights/apt36-nightmare-vibeware

Radu Tudorica



I'd like to thank my coauthors Adrian Schipor, Victor Vrabie, Marius Baci, and Martin Zugec for their invaluable contributions to this research.

TL;DR. Pakistan-based threat actor APT36, also known as Transparent Tribe, has pivoted from off-the-shelf malware to "vibeware", an AI-driven development model that produces a high-volume, mediocre mass of implants. Using niche languages like Nim, Zig, and Crystal, the actor seeks to evade standard detection engines while leveraging trusted cloud services, including Slack, Discord, Supabase, and Google Sheets, for command and control.

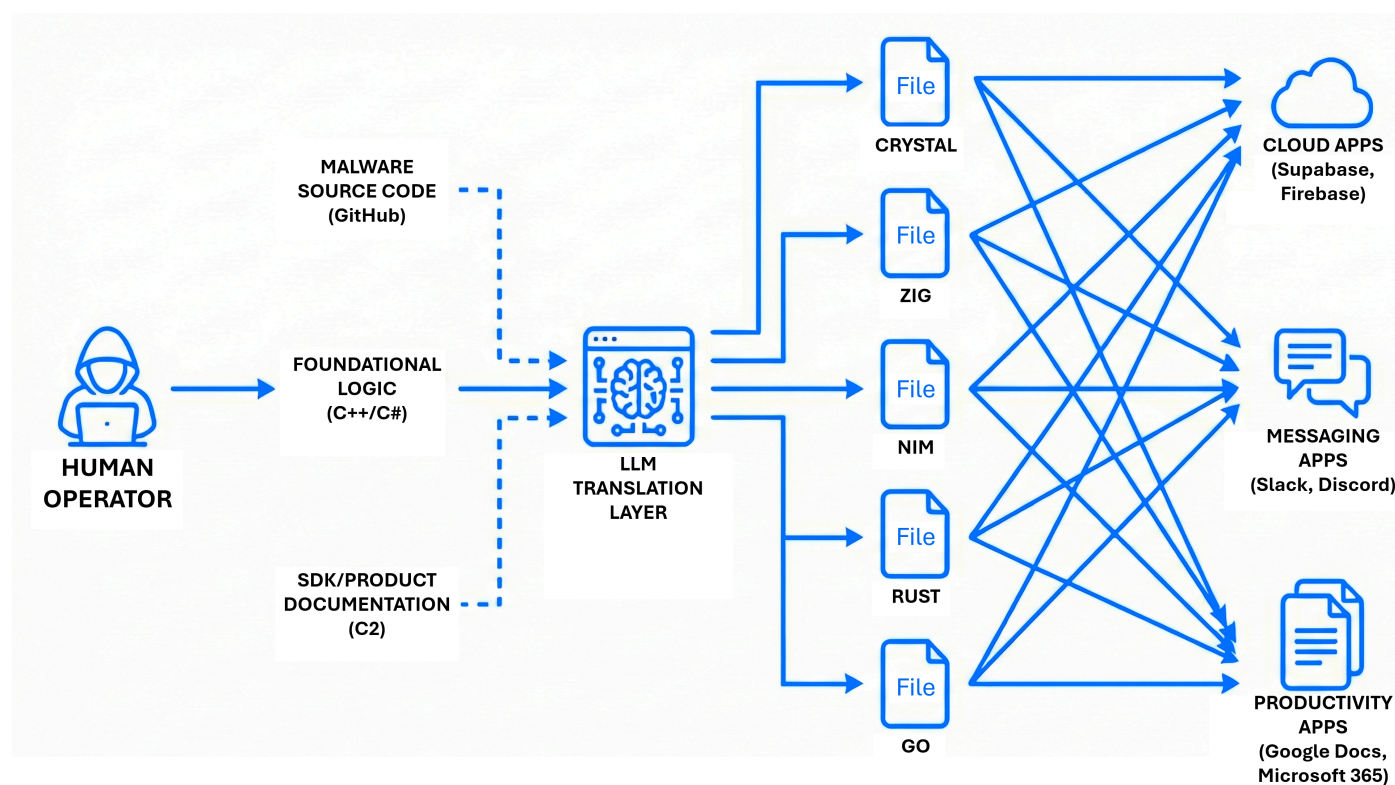
This research provides a direct look at the new reality of vibeware, an emerging malware category deployed by the Pakistani threat group APT36, primarily targeting the Indian government and diplomatic missions. Rather than a breakthrough in technical sophistication, we are seeing a transition toward AI-assisted malware industrialization that allows the actor to flood target environments with disposable, polyglot binaries. The limitations of this approach are obvious when looking at this malware fleet. We observed an instance where a basic Go binary was deployed to steal browser credentials, but developers left a template placeholder where the command and control URL should have been, meaning the tool could never actually exfiltrate data.

We saw similar patterns across the rest of the fleet, where other malware components began to collapse under their own weight as soon as the logic reached a moderate level of complexity. These kinds of mistakes are typical of code that is syntactically correct but logically unfinished. As we stated in our [2026 Predictions](#), “...LLMs are excellent at repackaging existing code in the language of your choice, but the generated code is derivative by its very nature.”

Despite these flaws, the vibeware model offers some strategic advantages.

By adopting niche languages like Nim, Zig, or Crystal, the actor resets the detection baseline for security engines. LLMs make these languages accessible by collapsing the expertise gap, allowing developers to generate functional code in unfamiliar languages by simply porting logic from more common ones.

The vibeware model also naturally facilitates the adoption of Living Off Trusted Services (LOTS) for both command and control and data exfiltration. LLMs are highly effective at generating stable code for platforms like Discord, Slack, and Google Sheets because of the vast number of public SDKs and documentation in their training data, allowing attackers to weaponize these trusted channels for bi-directional communication and silent file theft.

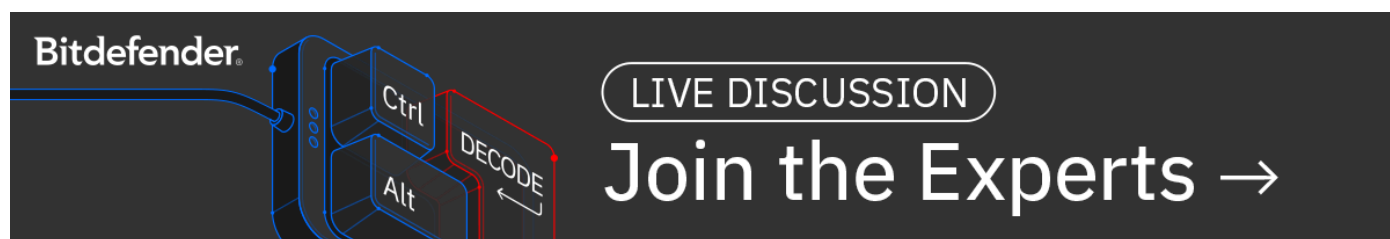


*In the image above, you can understand why the transition toward vibeware can be viewed as a **Distributed Denial of Detection (DDoD)**. In this framework, the objective is not to bypass security through technical brilliance, but to exhaust the defenders through automated volume.*

Identified victims were infected with multiple, parallel implants, each using a different language and communication protocol. If one experimental channel is neutralized, the actor retains access through a separate path. It is important to remember that the industrialization of the malware

production pipeline does not mean the attack itself is automated. While the tooling is vibe-coded and rapidly produced, the actual hacking operations remain a manual effort.

The evidence of AI assistance in this fleet is conclusive. We identified metadata within their projects that points directly to the use of AI-integrated code editors. The binaries themselves frequently contain Unicode emojis in strings. The actor maintains a malware-a-day cadence, producing new variants daily. The reality is not a breakthrough in malware sophistication, but an optimization of the mediocre.



Watch our live [Ctrl-Alt-DECODE episode](#) where we explore, *Vibeware Analysis: Is AI-Generated Malware by APT36 a Real Threat or Just Noise?* We'll answer your questions in real-time.

Attribution and Victimology

We assess with medium confidence that the observed malicious activity is linked to **APT36**, also known as Transparent Tribe. This assessment is primarily based on the campaign's reliance on a familiar mix of open-source and commercial tooling, including Havoc, Cobalt Strike, and Gate Sentinel, which is consistent with historically observed TTPs for this actor.

A critical technical artifact is the tool warcode.exe. This specific file was previously identified in older campaigns as a loader for the Havoc framework. Its reappearance alongside vibeware provides a strong indicator of continuity between the established APT36 toolkit and these newer vibe-coded iterations.

The targeting remains highly focused on South Asian regional politics and national security. The primary victims are associated with the Indian government and its embassies in multiple foreign countries. A screenshot of a LinkedIn page was also recovered, showing a list of Indian employees from a government agency that is related to the military domain, indicating the actor's use of professional social networks to identify and profile high-value targets. Secondary targets include the Afghanistan government and several private businesses.

Based on recovered artifacts, the attackers are specifically interested in the following data categories:

- Army personnel documentation
- Foreign affairs and diplomacy
- Strategic and policy documents
- Defense and national security

Investigation into the group's internal infrastructure revealed a recurring username, "Nightmare", across multiple systems. This persona appears to be central to the development or operation of the fleet.



An AI-generated image by this actor was found within their infrastructure.

A notable characteristic of this campaign is the emphasis on resilience. Most identified victims were simultaneously infected with multiple implants.

AI-Generated Malware in 2026

The current threat landscape validates [some of our AI predictions](#) from two years ago. We noted that people should not imagine a complex binary skillfully maneuvering through a network to pinpoint vulnerabilities, but rather "a code with minor customizations, crafted in a language of your preference."

Contrary to popular belief, gaining access to functional malware source code is trivial. GitHub Repositories such as [theZoo](#) or [vx-underground](#) collections provide access to thousands of live samples, proofs-of-concept, and historical source files for researchers and adversaries alike. Because LLMs are trained on these vast, public datasets, they have essentially "read" and internalized thousands of distinct implementations of malicious logic, enabling them to reconstruct and vary these routines with minimal prompting.

But a critical assessment of this capability reveals a nuanced reality. While LLMs are highly proficient at pattern recognition and logic mapping, they do not possess a true understanding of security context. They function as engines that retrieve and reorganize existing patterns rather than inventing new ones. Consequently, the "reconstruction" of malware is often a process of assembling known fragments. This leads to what we identify as the "Mediocre Mass".

AI-Generated Malware Trends: 6-Month Growth

INTERNAL RESEARCH

SOURCE: BITDEFENDER TELEMETRY RESEARCH

Bitdefender's internal telemetry confirms a consistent expansion in automated sample generation starting from the August 2025 low point. The volume has scaled by over 10x in half a year, reflecting an industrialization of the production pipeline.



This chart shows the total number of unique AI-generated malware samples found in our telemetry over the last six months (not limited to APT36). We're planning to publish a more detailed analysis as we collect more data.

The vibe-coded samples in this investigation are not technical masterpieces. They are often generic, inconsistent, and error-prone. While we assigned names to the malware to help track individual samples, the overall relatively low development quality makes them arguably not worth naming. Instead, it would be more accurate to label this kind of malware as "**vibeware**". We have reached a similar conclusion as the [recent report from Google](#)'s GTIG team, which noted that they have "*not yet observed APT or information operations (IO) actors achieving breakthrough capabilities that fundamentally alter the threat landscape.*"

While this malware lacks true technical innovation, it would be a mistake to underestimate the risk it poses. The threat lies in the industrialization of these attacks. We are seeing a convergence of two trends that have been developing for some time: the adoption of exotic, niche programming

languages, and the abuse of trusted services to hide in legitimate network traffic. This combination allows even mediocre code to achieve high operational success by simply overwhelming standard defensive telemetry.

Malware Polyglots

While LLMs are not currently architecting novel attack methodologies, their influence is distinctly visible in enabling the rise of malware polyglots. Historically, pivoting from a well-known framework like .NET or C++ to a less common language required significant time and effort. Today, AI-assisted development has collapsed this barrier, allowing actors with foundational logic skills to cycle through different languages with minimal friction.

Most detection engines rely on signatures and behavioral patterns heavily tuned for common languages like C# or C++ - the way these languages manage memory, handle system calls, or structure their runtime. When a threat actor uses an "exotic" language, they essentially reset the detection baseline. Furthermore, many of these modern languages, such as Rust, Go, and Zig, are inherently OS-agnostic, leading to a growing trend where malware is developed to target multiple operating systems simultaneously with the same code base.

In this research, we identified several different languages:

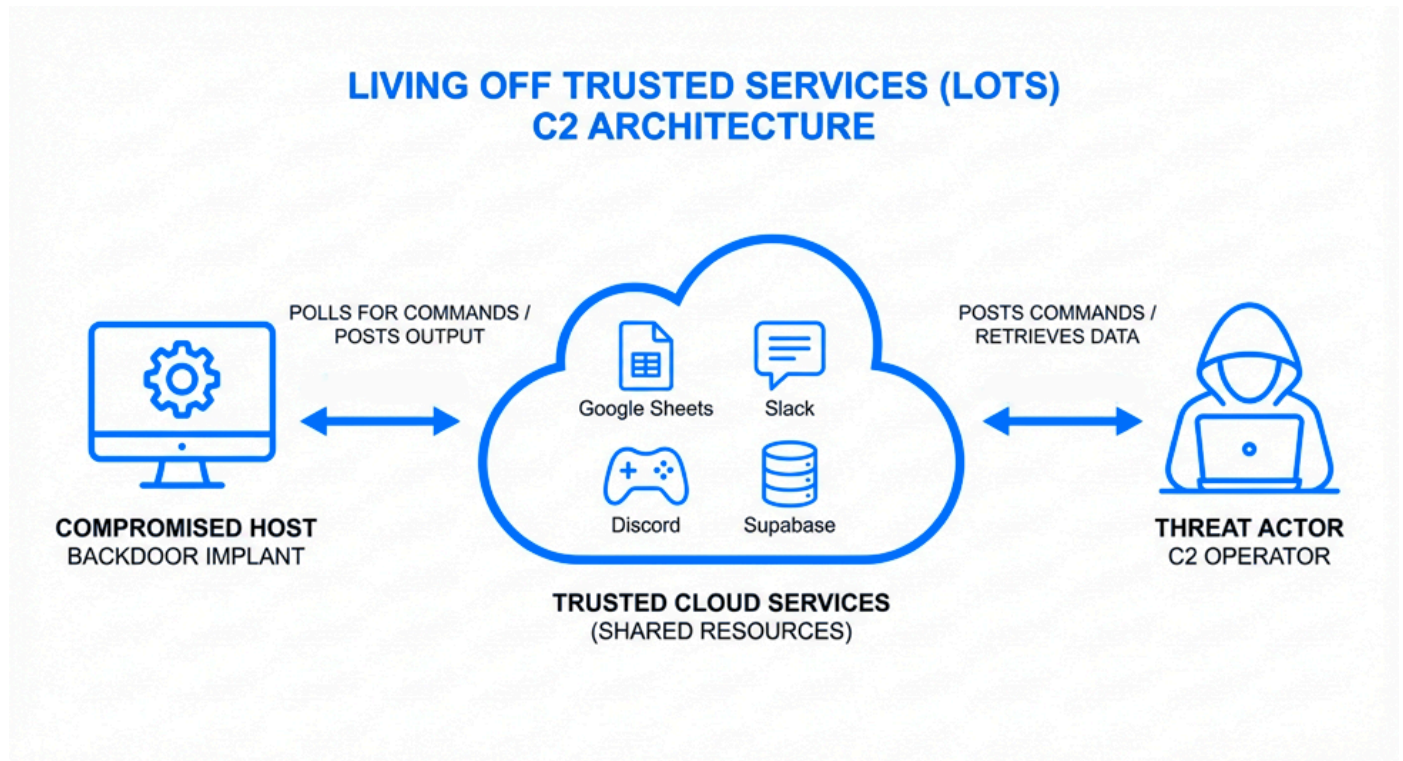
- **Crystal** and **Zig**: These are niche enough to lack established behavioral signatures in many EDR solutions. For example, Warcode and CrystalShell utilize Crystal to provide high performance without the common signatures associated with C++ or C#.
- **Nim**: This serves as a stealthy wrapper for legacy payloads. Because it compiles to C or C++ but uses a unique runtime, it often appears as an "unknown" rather than a "malicious" binary to simple scanners.
- **Rust** and **Go**: These are used for "noisy" operations like file harvesting and exfiltration. Their built-in memory safety and performance ensure that the malware remains stable during intensive tasks, preventing crashes that might otherwise alert a system administrator.

C2 via Living Off Trusted Services (C2)

While language diversification complicates binary detections, malware can still be identified by its Tactics, Techniques, and Procedures (TTPs). While the binary might look different, its behavior remains its most durable identifier. To counter this, threat actors are increasingly adopting a "Living Off" philosophy to stay undetected by blending into the target environment's noise. This strategy is standard practice among sophisticated threat actors, and our research indicates that up to 84% of modern cyberattacks now use legitimate, pre-installed tools.

Historically, the "Living Off" concept primarily referred to Living Off the Land Binaries (LOLBins). However, this philosophy is not limited to local binaries. We are observing a significant trend towards **Living Off Trusted Services (LOTS)**. Our research has identified threat actors using

Google Sheets to store dynamic instructions for malware, leveraging cloud databases like Firebase or Supabase for storing metadata and credentials, and utilizing Slack or Discord messages to send real-time instructions or retrieve harvested data.



This image details some of the ways threat actors are leveraging trusted services.

LLMs are exceptionally well-suited for implementing LOTS, as legitimate services are extensively documented with vast quantities of legitimate code examples and SDKs. This abundance of training data allows an LLM to generate robust, functional integration code for complex APIs with high reliability.

For a vibe-coding actor, this is a win-win scenario. They can prompt an AI to produce a stable, C2-ready client using a trusted service API without needing to build their own suspicious infrastructure or understand the technical nuances of the underlying protocol.

Technical Analysis of the Malware Collection

Initial Access

Initial access is believed to occur through malicious emails. We have identified evidence of shortcut files (.LNK) bundled within ZIP archives or ISO images, which are likely delivered as email attachments. These archives act as containers to bypass simple file-extension filters on mail gateways.

One notable technique uses a PDF file as a visual lure. The document contains a high-quality image of a resume with a large, prominent "Download Document" button overlaid. Once a user clicks this button, they are redirected to an attacker-controlled website, which likely triggers an automatic download of an archive containing the previously mentioned shortcut files. This multi-stage redirection adds a layer of social engineering, as the initial PDF appears benign to many scanners while the malicious payload is hosted externally.

ABOUT ME

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.

EDUCATION

Bachelor of Business Management

Wardens University
2018 - 2020

Bachelor of Business

Wardens University
2018 - 2020

SKILLS

Management Skills

Creativity

Digital Marketing

Negotiation

Critical Thinking

Leadership

LANGUAGE

- English
- Spain

+00-000-1000

hello@wardenswebsite.com

www.wardenswebsite.com

100 Anywhere St., Any City, 00-0000

EXPERIENCE

Product Design Manager

2020 - 2023

Anywhere Industries / 123 Anywhere St., Any City

- Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nunc sit amet sem nec risus agnate arcuamus. In enim turpis, interdum at quam eget, luctus sollicitudin neque. Sed nec nisl, semper at hendrerit a, sollicitudin in eros.

Marketing Manager

2018 - 2020

Anywhere Industries / 123 Anywhere St., Any City

- Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nunc sit amet at quam eget, luctus sollicitudin neque.

Marketing Manager

2017 - 2018

Anywhere Industries / 123 Anywhere St., Any City

- Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nunc sit amet at quam eget, luctus sollicitudin neque.

Marketing Manager

2016 - 2017

Anywhere Industries / 123 Anywhere St., Any City

- Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nunc sit amet sem nec risus agnate arcuamus. In enim turpis, interdum at quam eget, luctus sollicitudin neque.

REFERENCES

Harumi Kobayashi

Wardens Inc. / CEO

Phone: 00-000-1000

Email: hello@wardenswebsite.com

Billy Dupont

Wardens Inc. / CEO

Phone: 00-000-1000

Email: hello@wardenswebsite.com

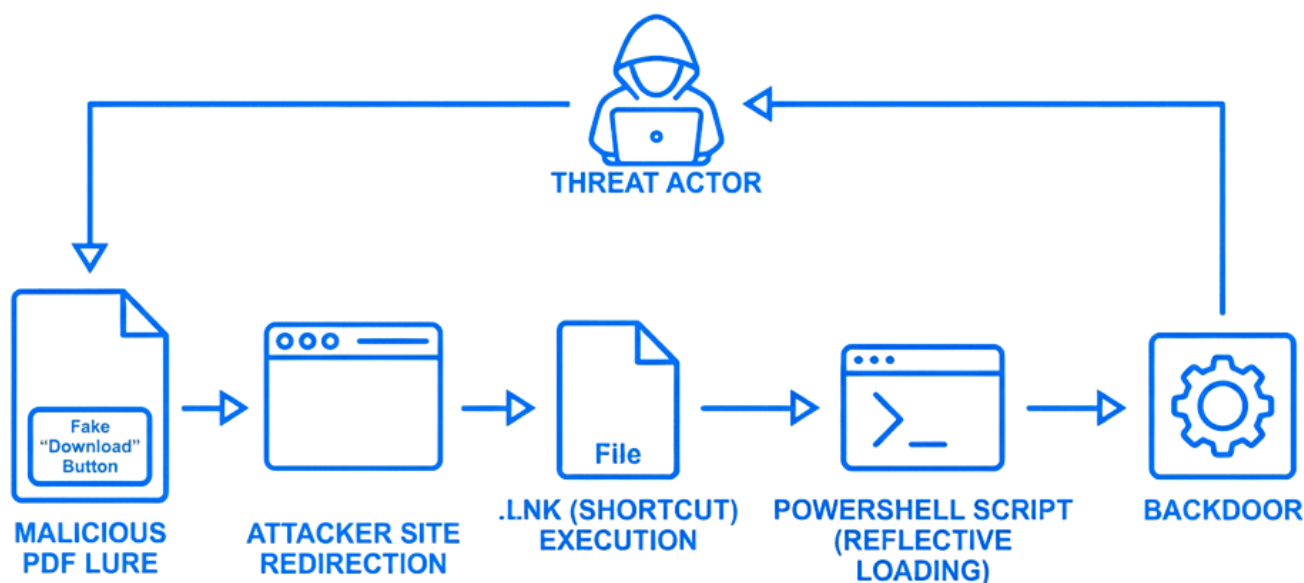
PDF



Download Document

This image shows the threat actor using a PDF file as a visual lure.

The shortcut files, once executed by the victim, initiate the download and fileless execution of PowerShell scripts in memory. These scripts then download and execute the primary backdoor. Once the backdoor is active, the attackers transition to an interactive phase, connecting manually to the compromised host to conduct post-exploitation activities.



The image above shows how the threat actor utilizes the malicious PDF to initiate the attack.

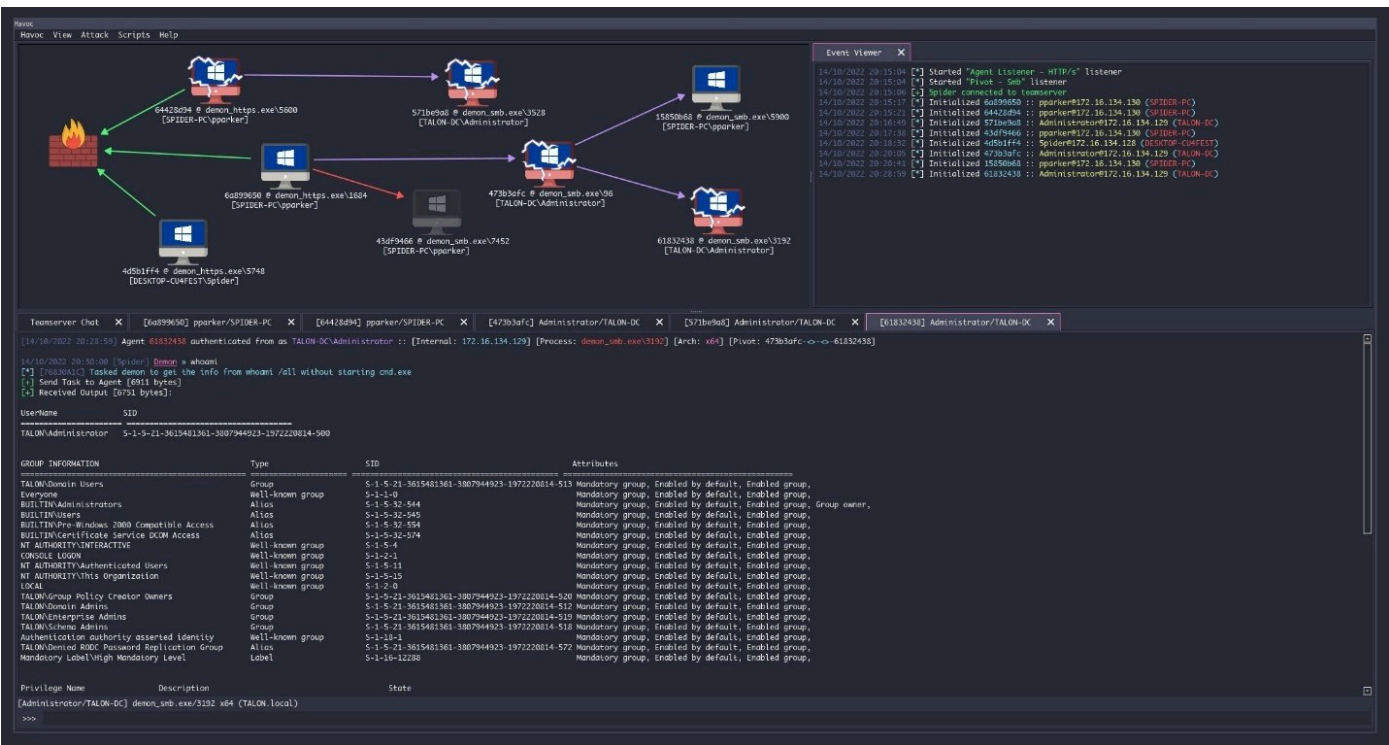
Warcode

APT36 has historically relied on mature, off-the-shelf offensive security frameworks to manage their operations. Tools such as Cobalt Strike and Havoc have been the group's primary choices for maintaining a stable presence within target networks.

As they pivot towards vibeware, they maintain a hybrid approach, continuing to deploy these established tools as a safety net.

Warcode is a critical part of this fallback strategy. It is a custom shellcode loader written in Crystal language. We have observed the binary, warcode.exe, in common writable locations such as C:\Users\Public\AccountPictures and C:\Users\Public\Downloads. Telemetry shows this specific loader was in use prior to the current campaign, indicating it is a trusted component that the actor considers resilient.

The malware functions by reflectively loading a Havoc C2 agent directly into memory, which then establishes communication with the hardcoded infrastructure at 23[.]152[.]10[.]81.



Havoc is a modern open-source post-exploitation framework designed for stealthy command and control. Source: [Havoc's GitHub Repository](#)

This hybrid strategy ensures that their mission remains resilient. If the experimental tools are blocked or don't function as intended, their reliable, human-audited tools provide a backup channel.

NimShellcodeLoader

The NimShellcodeLoader functions as the experimental counterpart to Warcode. It is written in Nim, a language that remains highly niche in the global development community. It often ranks near the bottom of industry indices like [TIOBE](#) with less than 0.1% market share.

Like its predecessor, it serves as a wrapper for mature, off-the-shelf malware (the internal name of this project is cobaltdropper.nim). This specific loader is designed to deploy a Cobalt Strike beacon, which is embedded directly within the binary. By using an LLM to generate a functional wrapper in a niche language, the attackers can rapidly iterate on their delivery mechanisms.

The technical implementation uses a structured cryptographic flow. The shellcode is encrypted using AES in CBC mode. To generate the decryption key, the loader hashes the hardcoded password Pun7sh3r@123 using SHA256. Once decrypted, the beacon is executed in volatile memory.

The C2 infrastructure utilizes slackin[.]online, which is notably abusing Azure Front Door. By routing traffic through this trusted CDN, the attackers hide their communication within legitimate HTTPS traffic.

CreepDropper

The threat actor utilizes something written in .NET, called CreepDropper. This dropper is a type of malware designed to deliver and install other malicious payloads onto a compromised system. In this campaign, CreepDropper serves as the delivery vehicle for SheetCreep and MailCreep (described in the next section). The executable masquerades as a legitimate browser process chrome.exe, and both its embedded version information and appended (invalid) digital signature are cloned from a legitimate Google Chrome version.

Technically, CreepDropper is a .NET assembly containing a **Task10** class. This class executes a straightforward routine to deploy its payloads to C:\Users\Public\Documents:

- **MailCreep** is dropped as msedge.exe.
- **SheetCreep** is dropped as ds.png.
- **MicrosoftEdgesUpdatesTasksMachineUAs-Task**: This task is configured to execute the MailCreep binary (msedge.exe).
- **MicrosoftEdgesUpdatesTasksMachineUAs-1234-87hy**: This task reads a file into a byte array, reverses the byte order to de-obfuscate the .NET assembly, and calls [System.Reflection.Assembly]::Load to reflectively execute the payload in memory (fileless).

Despite the .png extension, ds.png is a functional malicious library. To execute these payloads, CreepDropper creates two scheduled tasks designed to blend into standard Windows update telemetry.

```
$b=[IO.File]::ReadAllBytes('C:\\Users\\Public\\Documents\\detail.png');  
([System.Reflection.Assembly]::Load([byte[]]($b[($b.Length-  
1)..0])).GetType('Service.Program')::JK())"
```

This specific execution chain allows the malware to remain resident in memory without ever executing a traditional .exe or .dll for the SheetCreep component, raising the bar for detection by standard endpoint security products.

MailCreep and SheetCreep

The final stage of the CreepDropper infection involves executing two functional payloads. MailCreep and SheetCreep. These components provide the threat actor with the actual capabilities required for data theft and command execution. Both families have been previously documented by [Zscaler research](#).

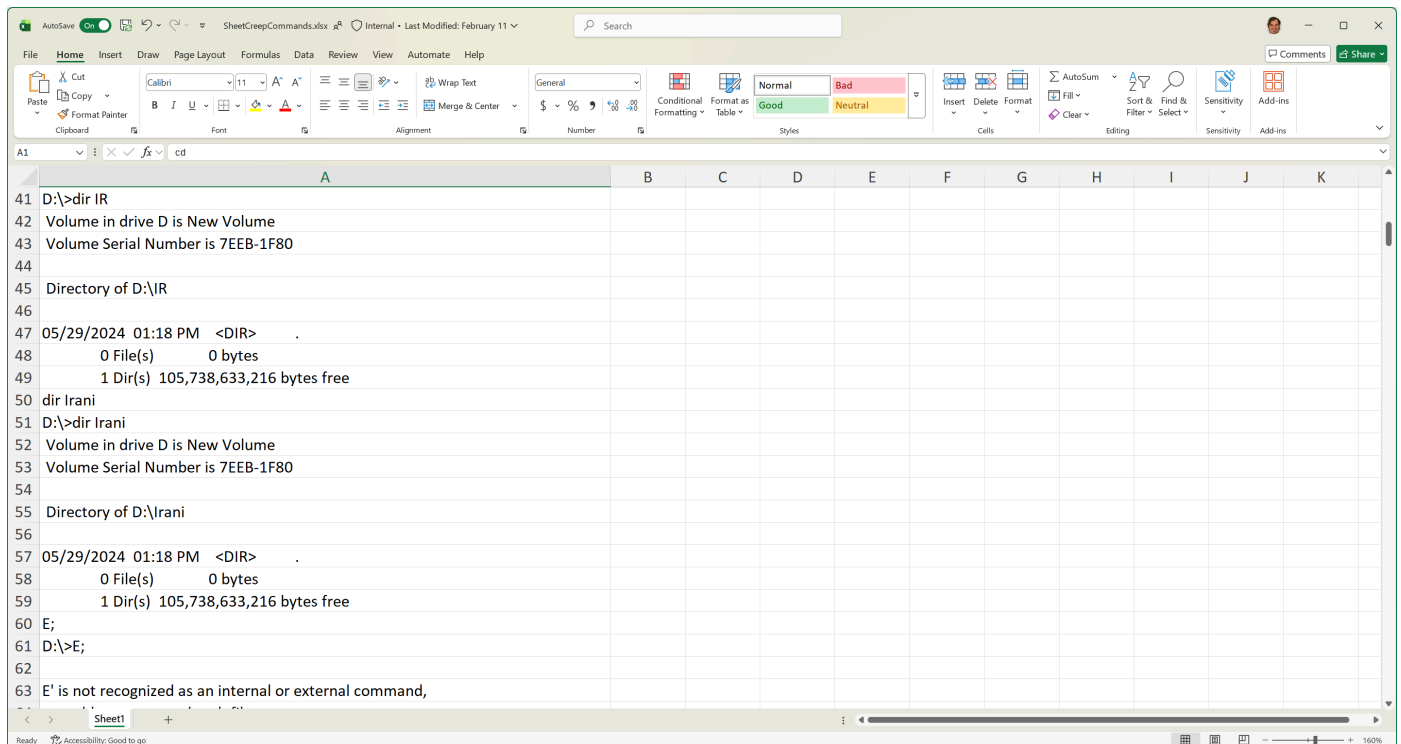
Our analysis of **MailCreep** aligns with this previous research. This Go-based infostealer uses the Microsoft Graph API to exfiltrate data. Its implementation and C2 mechanisms remain consistent with documented versions. We found no significant deviations or updates in this campaign. For a detailed breakdown of its routines, we recommend reading the original Zscaler analysis.

While the basic architecture of **SheetCreep** remains a C#-based backdoor utilizing Google Sheets for command retrieval, our investigation uncovered several new insights. The sample identified at the victim (compiled at 2025-10-28 09:58:46) used a simple file-detection bypass technique - the entire binary was written in reverse within the file. After mirroring the bytes from end to start, the file yields a valid MZ PE .NET assembly.

Typical executable files begin with the "MZ" header, represented by the Hex values 4D 5A. Static security scanners rely on this header to identify binary files for analysis. This signature is named after Mark Zbikowski, a lead architect of MS-DOS. Coincidentally, one of the writers of this research shares these same initials.

The primary module, named GServices.dll, implements a Google Sheets-based C2 technique similar to publicly available tools like [google_RAT](#) or [GC2-sheet](#). This logic transforms a Google Drive spreadsheet into a bidirectional C2 hub where instructions and outputs are exchanged as cell data. The backdoor routine follows a specific cryptographic flow:

1. It polls a specific spreadsheet for new entries.
2. It fetches and decrypts issued commands using Base64 followed by DES in ECB mode.
3. It executes the decrypted command on the host.
4. It re-encrypts the response and transmits it back to the spreadsheet using the Google Drive API.



Decrypted command sequence within the malicious C2 spreadsheet. The commands are presented here in their unencrypted state. Notice the "E;" typo as evidence of a manual, hands-on operation.

Our investigation also identified the following accounts associated with this C2 infrastructure:

- sasikapeeper@gmail[.]com
- service-india-api@service-india-471504[.]iam.gserviceaccount.com
- sasikapeeper@gmailapp-468713.iam[.]gserviceaccount.com
- **unenc_requests**: Used for polling incoming tasks.
- **unenc_outputs**: Used for posting command results.
- **unenc_heartbeats**: Used for session tracking.
- **unenc_systems**: Used for exfiltrating system fingerprint data.

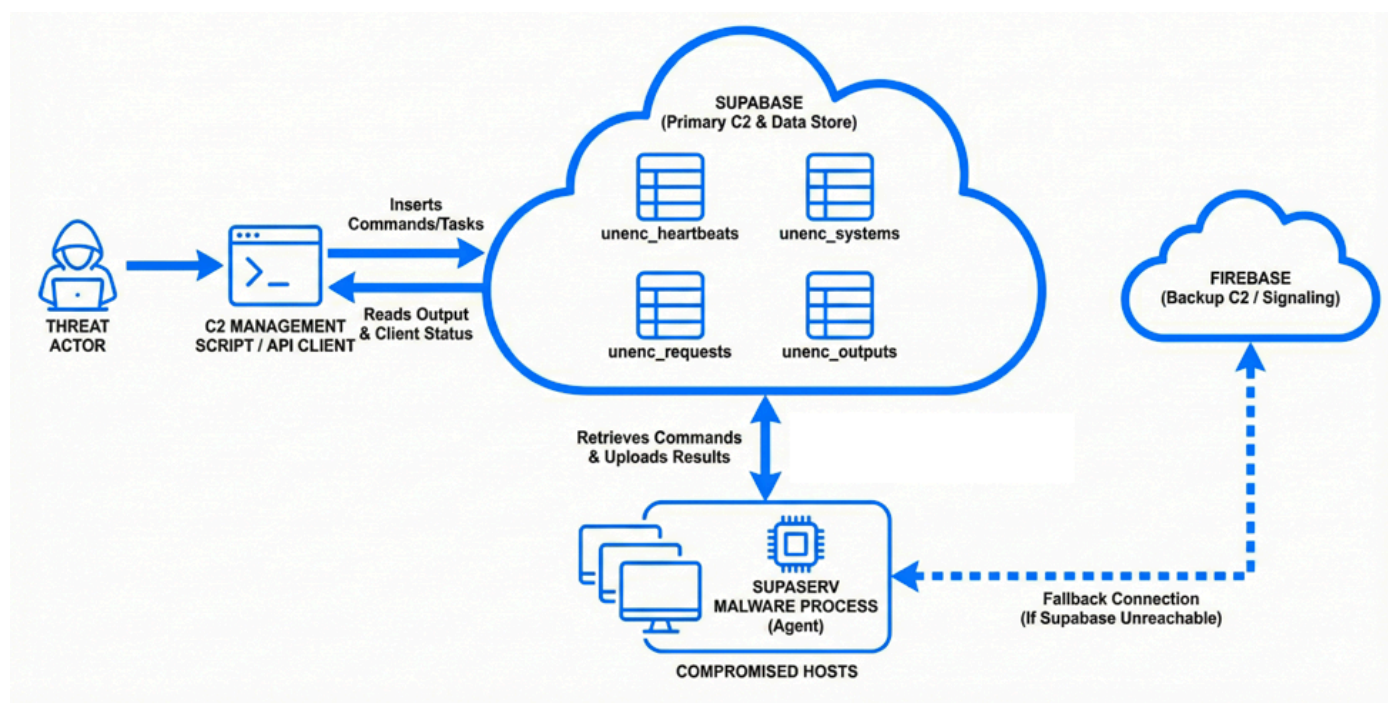
SupaServ

SupaServ is a backdoor written in Rust that exemplifies the transition from simple scripted routines to more complex, native implementations. The presence of Unicode emojis in the code, such as '🌐 Browser simulation enabled', '🔀 Random user agents', and '🕒 Completed cycle', is a hallmark of AI-assisted or AI-generated development.

SupaServ maintains its presence on the host by using the ITaskService COM interface to create scheduled tasks that trigger at a user login. We have detected task names such as DateAndTimeService and personalServiceTask.

The malware establishes a primary communication channel via the Supabase platform, with a Firebase backup. Supabase and Firebase are cloud platforms that provide managed database services to developers. Firebase is a proprietary Google service, whereas Supabase is an open-source alternative based on PostgreSQL, and traffic to these domains is often trusted by default in enterprise environments. Authentication tokens for both services are hardcoded directly within the binary.

Command and control tasks are managed through four database tables:



SupaServ implements two primary operations, open and download, where the path and URL parameters are provided in the request from the C2. Analysis reveals two distinct variations in how these operations are handled.

Older Version

- **MD5 Hashes:** bea885be7d436272111504dc51685db0e, 569bb4899de5759c32ea6df661c35d4c
- **Download:** Utilized curl -s -L -o to retrieve files.
- **Execution:** Ran commands by spawning cmd.exe and attaching pipes to stdin and stdout.

Newer Version

- **MD5 Hash:** ca2edac970d8afed99db4b5cda72a13e
- **Download:** Uses the native request library to download data.
- **Execution:** Loads the .NET CLR runtime into memory using COM object interfaces to execute the PowerShell cmdlet Start-Process.

To complicate network-level detection, the malware mimics legitimate browser traffic by modifying its HTTP requests. It rotates between a set of hardcoded origin URLs and common User-Agents.

Rotated Origin URLs:

Specific HTTP Headers Set:

- 'Sec-Fetch-Dest': 'empty'
- 'Sec-Fetch-Mode': 'cors'
- 'Sec-Fetch-Site': 'cross-site'
- 'Sec-Ch-Ua': '"Not_A Brand";v="8", "Chromium";v="120"'
- 'Sec-Ch-Ua-Mobile': '?0'
- 'Sec-Ch-Ua-Platform': '"Windows"'

LuminousStealer

LuminousStealer is a recently identified Rust-based infostealer (compiled 2026-01-07). Like SupaSrv, it is a vibe-coded tool containing multiple emoji-enriched logging strings:

- 📁 Sent folder metadata to Firebase
- 📊 Enumerating top-level folders and files in drive
- ✅ Folders stored in DB
- 📁 Sending folders to Firebase...
- 📁 No top-level files in drive
- 📀 Enumerating drive

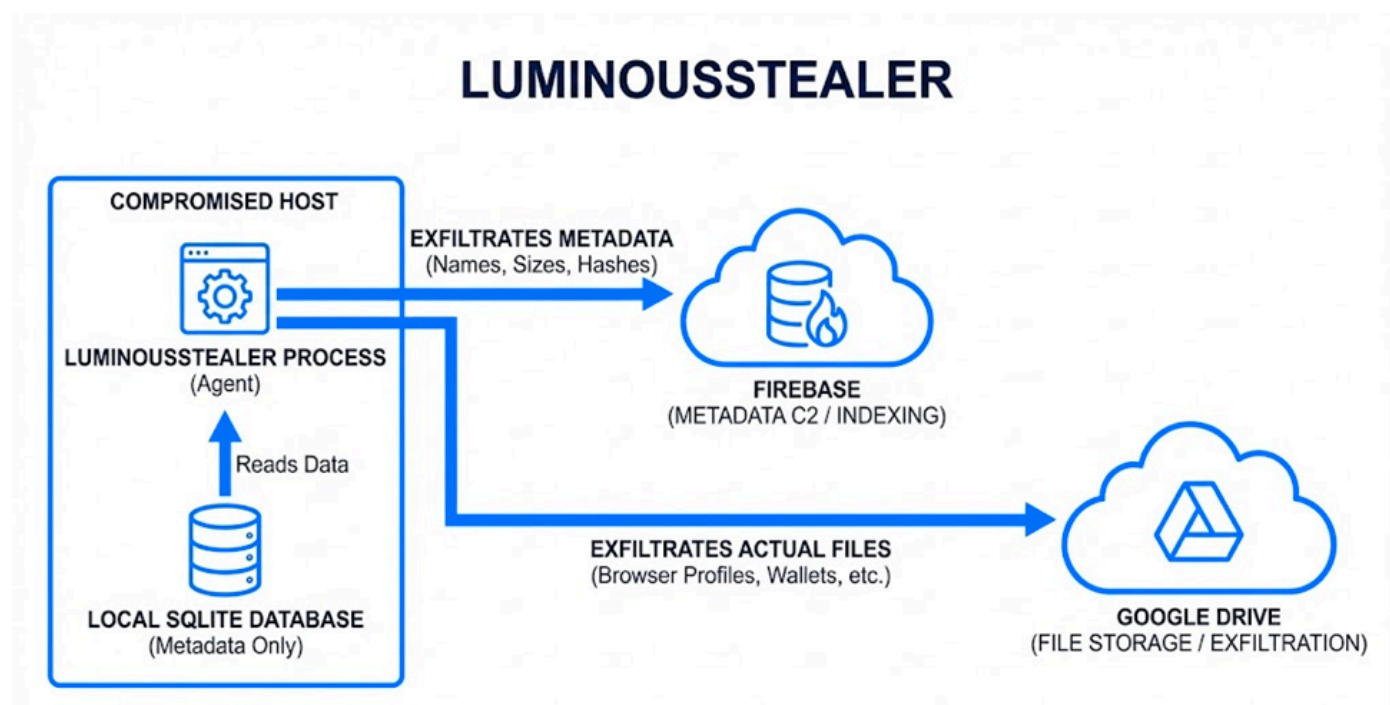
Persistence is maintained through a scheduled task created with high privileges named LuminousBackupService.

Unlike simpler harvesters, LuminousStealer implements more robust local staging using an SQLite database (%LOCALAPPDATA%\backup_database\backup.db) to manage five internal tables:

- **files**: Catalog of all identified files.
- **sessions**: Tracking for active exfiltration sessions.
- **folders**: Directory structure metadata.
- **command_queue**: Tasks received from the C2.
- **upload_queue**: Files staged and waiting for cloud exfiltration.

The malware performs recursive directory scans across all available drives, looking for files with the following extensions: .txt, .docx, .pdf, .png, .jpg, .xlsx, .pptx, .zip, .rar, .doc, and .xls. To minimize its operational footprint, LuminousStealer stores this metadata in the local SQLite database. By comparing scan results against existing records (diffing), it ensures that only new or modified files are cataloged for exfiltration.

For actual exfiltration, LuminousStealer uses split-strategy of dividing metadata and content of files. All file metadata and catalog records from the SQLite database are uploaded to Firebase, while the actual file contents are queued and uploaded to Google Drive. Both Firebase and Google Drive are authenticated using Google OAuth.



To complicate network-level detection, LuminousStealer uses the same browser simulation technique as SupaServ. The only difference is that it uses a more specific Sec-Ch-Ua string: 'Sec-Ch-Ua': "Not_A Brand";v="8", "Chromium";v="120", "Google Chrome";v="120"

Interestingly, the analysis of the binary revealed the embedded build path `C:\Users\kumar\.cargo\` (.cargo is the global directory for the Rust package manager). "Kumar" is a common Hindu name and this likely represents a deliberate attempt at confusion. Another example of this tactic is the CrystalShell C2 infrastructure which uses a Discord server named "Jinwoo's Server". These are very common tactics by APTs to mislead investigators and complicate geopolitical attribution – or maybe the developer just likes the main protagonist from Solo Leveling.

CrystalShell

CrystallShell is a backdoor developed in the Crystal language, designed for cross-platform deployments (Windows/Linux/macOS). The samples identified between late December 2025 and late January 2026 reveal improvements in C2 credential management, but also several notable implementation failures.

Most of the samples we've analyzed use Discord as the primary C2 channel, though a single variant was identified using Slack. The actor's method for securing the Discord bot tokens evolved across the samples.

1. **Initial Stage (Dec 2025):** Tokens were hardcoded directly within the binary.
2. **Intermediate Stage:** The malware started fetching tokens from a public, unauthenticated Firebase Realtime Database endpoint.
3. **Current Stage (Jan 2026):** The most recent samples utilize Supabase with authentication enabled to retrieve C2 credentials.

The malware uses hardcoded Discord channel IDs for command and control and implements a rudimentary authorization check. It validates the Discord User ID of the sender. If a command originates from an unauthorized user, the bot responds with "*You are not authorized.*"

All communications within the C2 channel are Base64-encoded to obscure the traffic from simple string-based filtering. Upon initialization, the agent announces its presence by posting a message in the format Users:/.

All interactions between bots and operator follow the same command and response format:

- **Targeting:** The protocol supports targeted execution using a syntax. The code accepts targeting strings in three formats, //, /, and , but the filtering routine specifically extracts and evaluates the filter. Commands issued without a target prefix are executed by all connected agents.
- **Command Format:** Direct commands must be prefixed with a specific tag and Base64 encoded: ! en
 - **users:** Returns the agent identifier in the format Users:/ (e.g., Users:administrator /7456).
 - **lists:** Returns detailed metadata: / PID: LastSeen:. Example: win10-vm/administrator PID:7456 LastSeen:2026-01-26 09:56:25 -08:00.
 - **host / hosts / hostname:** Returns the host and user information: /. Example: win10-vm/administrator.
 - **ls | dir:** Both forms are accepted to list the contents of a directory.
 - **pwd | cd:** Changes or displays the current working directory, which is tracked internally by the malware.
 - **status:** Returns the connectivity state: // - online|offline.
 - Any other command is executed via /bin/sh -c on Linux/macOS. On Windows, it uses cmd.exe /c > %TEMP%\<8-random-chars>cr_shell_output 2>&1. Notably, the current implementation fails to delete this file afterwards.
- **Results Output:** Results from built-in commands like ls, pwd, or status are returned as a standard Base64 string: base64(). Results from commands passed to cmd.exe or /bin/sh are returned with a leading ! marker: ! base64().

CrystalShell includes a translation layer that detects the host operating system and rewrites Linux-style commands into their Windows equivalents.

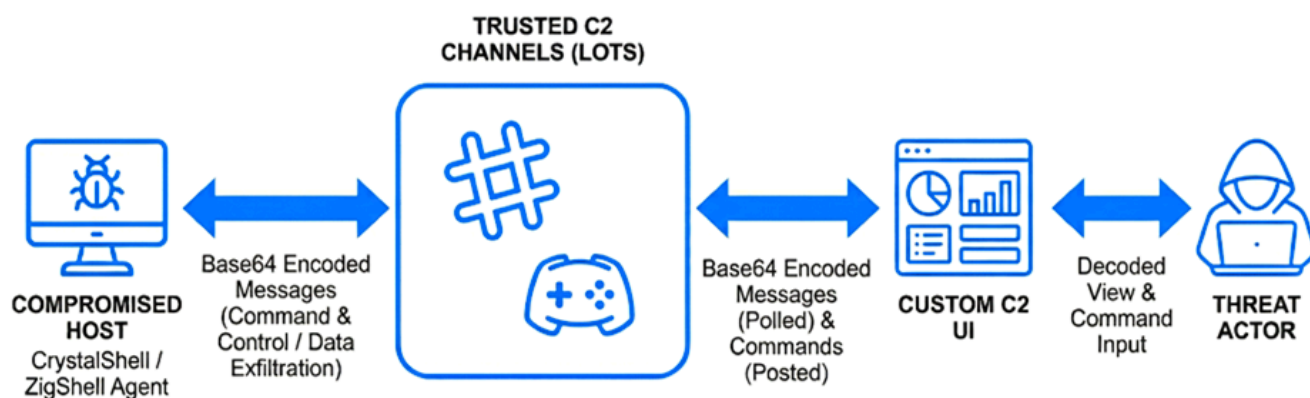
- ls -> dir

- cat -> type
- rm -> del
- pwd -> cd
- mv -> move
- cp -> copy

Commands issued through the Slack and Discord management interfaces show the operators performing a wide range of tasks, following a standard progression from initial discovery to final exfiltration. Messages related to CrystallShell (both Discord and Slack variants) and ZigShell started on 2025-12-08 and remain ongoing as of the time of writing this report.

- Frequent use of curl, ipinfo, ipconfig, net view, and arp to map the network.
- Regular listing of scheduled tasks/processes/directories.
- Reliance on PowerShell for advanced tasks, mostly using iex (Invoke-Expression) to run code directly from remote servers and loading malicious .NET assemblies into memory. PowerShell was also used for capturing screenshots and recording audio.
- Deployment of additional tools and malware, mostly hosted on the file-sharing service tmpfiles.org.
- Systematic staging of sensitive data followed by exfiltration to attacker-controlled cloud storage.

Notably, we observed frequent command-line errors and malformed commands.



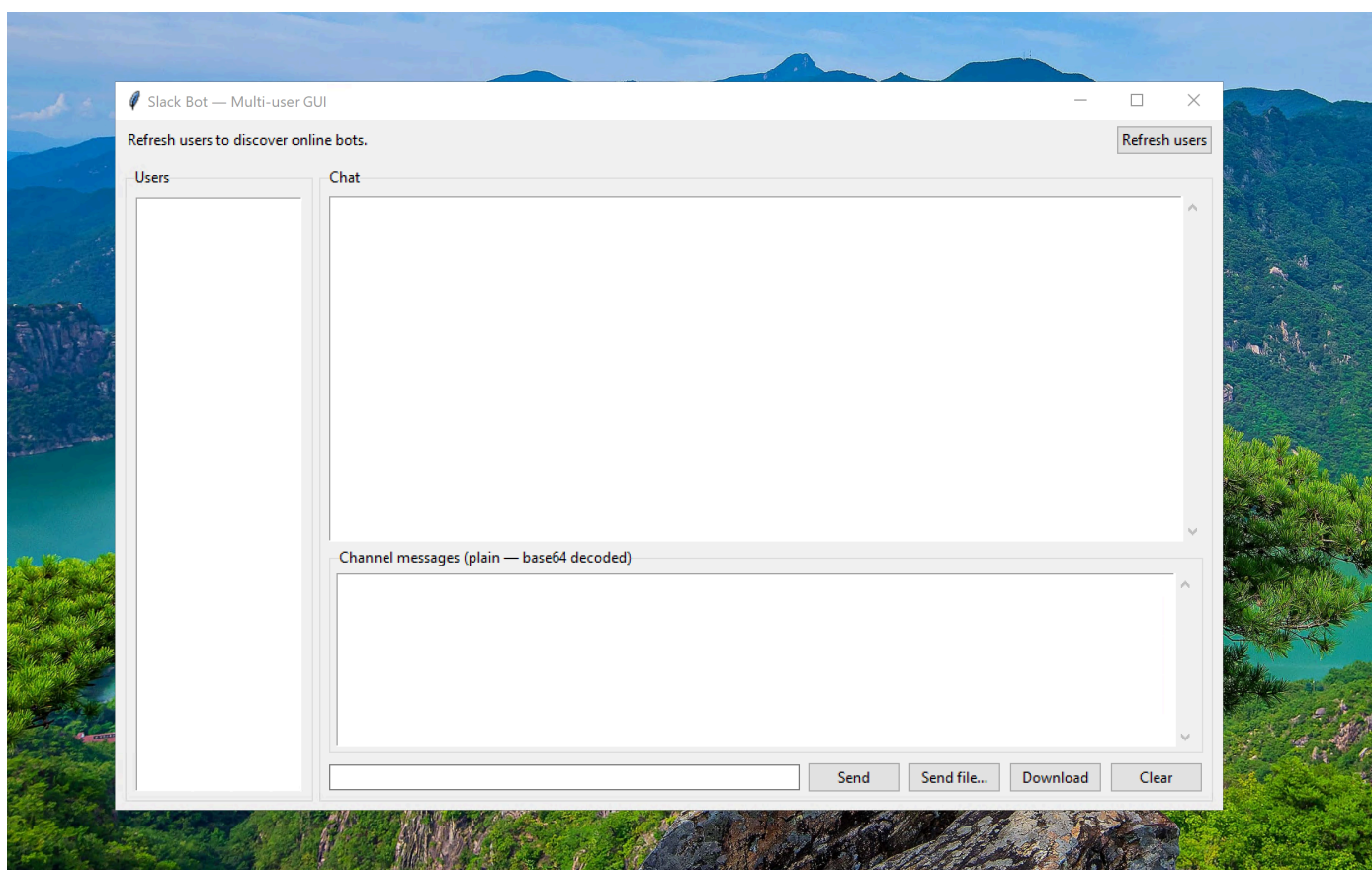
There are some notable limitations of vibe-coding visible in the CrystalShell code, for example, the implementation of status and lists commands. The developer appears to have intended to create a centralized state-management system where a primary agent would maintain a hashtable of all active bots. This hashtable was meant to store status entries for every compromised host, allowing an operator to query a single point for the health of the entire network. However, there is no protocol for bots to share their status or elect a master node to handle reporting. This oversight would lead to a broadcast storm where every bot attempts to reply to everyone else but the hashtable is not even populated with data about other bots.

Another example is the status command. Instead of merely querying the last_seen attribute to calculate the time delta, the malware refreshes this timestamp to the current time during the reporting process itself. This logic error ensures that the host will always appear "online" to itself during the check, because querying the state resets the very metric used for evaluation.

ZigShell

ZigShell is a functional counterpart to CrystalShell but utilizes Slack as its primary C2 infrastructure and is written in Zig language. Two identified samples carry a compile timestamp of 2026-02-09, placing them later in the campaign timeline than most CrystalShell variants (the Slack variant of CrystalShell was compiled 2026-01-01 08:54:53).

Despite the platform change from Discord to Slack, the communication protocol remains nearly identical to the CrystalShell implementation. The only difference is that while CrystalShell used !en command prefixes, ZigShell simplifies this to a standard ! prefix.



To automate the Base64 encoding and decoding process, attacker used a custom GUI wrapper. While we've been only able to locate this UI for ZigShell, we can safely assume that equivalent tool was developed for CrystalShell.

From functionality perspective, ZigShell added built-in support for file transfers:

- **send**: File upload to move files from the victim machine to the Slack channel. It includes a usage string: Usage: send .
- **drop**: File download to pull files from Slack to the victim machine. It supports both filenames and Slack file IDs: Usage: drop or drop (download from Slack to current directory). Or upload a file and say '! drop'.
- **get**: A command supporting both upload and download. Usage: get (from current directory -> Slack) or get (from Slack -> current directory).
- Any other command is executed via powershell.exe spawning cmd.exe

ZigShell appears to be a direct port of the CrystalShell logic into the Zig language. The nearly identical command structures and announcement formats suggest the use of a common design template.

CrystalFile

CrystalFile is a simple command interpreter written in Crystal language. Unlike the other tools in this suite that communicate directly with cloud APIs, CrystalFile operates as a passive listener that relies on local file state.

Operational Workflow:

1. **Polling**: The malware continuously monitors file C:\Users\Public\AccountPictures\input.txt.
2. **Parsing**: When the file is non-empty, the malware reads it line by line, ignoring lines that start with # (comments).
3. **Execution**: It processes each line through a built-in logic handler.
 1. cd: Change the current working directory.
 2. ls: List the content of the current directory.
 3. copy-item: Transfer file using PowerShell-like parameters like -path, -destination, and -recurse.
 4. In all other cases: Execute the command by invoking cmd.exe.
4. **Output**: Results are concatenated and written to output.txt in the same directory.
5. **Cleanup**: Once the cycle is complete, the malware clears input.txt and returns to its polling state.

LuminousCookies

LuminousCookies is a specialized injector to exfiltrate cookies, passwords, and payment information from Chromium-based browsers, specifically targeting the bypass of App-Bound Encryption (ABE). Before we start with LuminousCookies analysis, it's useful to look at ABE and how it's being decrypted.

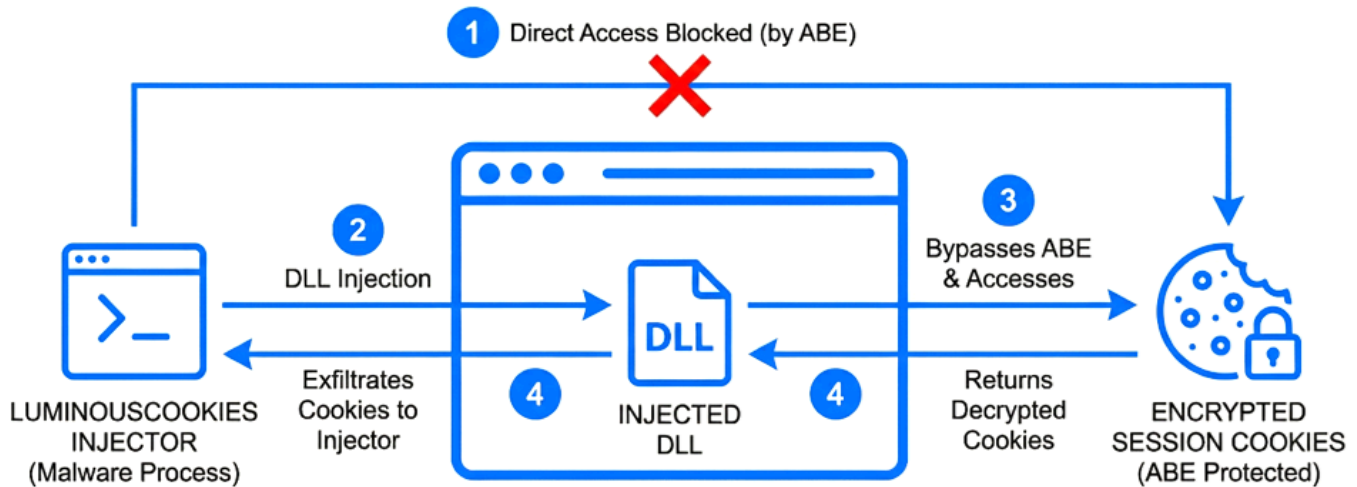
What is App-Bound Encryption (ABE)?

App-Bound Encryption is a defense-in-depth mechanism introduced in Chromium version 127 to secure sensitive assets like session cookies and stored credentials. Historically, these secrets were protected by the Windows Data Protection API (DPAPI), which only verified the identity of the user, which allowed any malicious process running with user privileges to decrypt the browser master key. ABE mitigates this by binding the encryption key to the cryptographic identity of the browser application itself. Decryption requests must be processed through a verified system service that confirms the request originates from a legitimate, signed browser binary. This shift forces attackers to move beyond simple file harvesting to more invasive techniques, such as process injection, to access the decrypted data.

ABE Decryptor Foundation

Under normal circumstances, only the signed browser binary is authorized to decrypt the `app_bound_encrypted_key` value. To bypass this restriction, an injector must force the browser to load a malicious DLL into its own memory space. Once inside, the DLL can retrieve sensitive data by masquerading as a legitimate part of the browser process. With this key, all stored browser secrets, including session cookies and passwords, can be decrypted using standard AES algorithms.

The public proof of concept [xaitax/Chrome-App-Bound-Encryption-Decryption](https://github.com/xaitax/Chrome-App-Bound-Encryption-Decryption) hosted on GitHub demonstrated how App-Bound Encryption can be bypassed without administrative privileges by subverting legitimate browser components. Analysis of the fleet revealed multiple variations of malicious DLLs and injectors, but all identified samples were based on this original research (or its forks). The predecessor to LuminousCookies was a standalone C++ utility directly compiled from modified versions of the public GitHub repositories. This version was typically delivered as a DLL for sideloading and was characterized by a visible forensic footprint, including clear-text logging in `%TEMP%/chrome_decrypt.log` and output folders such as `output_dll_default`. We have found various versions of both an injector and injected library.



STEPS:

1. LuminousCookies Injector attempts direct access to ABE-protected cookies, which is blocked by the OS.
2. The Injector performs DLL injection, loading its malicious DLL into the Legitimate Browser Process.
3. Running within the browser's trusted process space, the Injected DLL bypasses ABE to access and decrypt the session.
4. The Injected DLL then exfiltrates the decrypted cookies back to the original Injector process.

LuminousCookies Injector

LuminousCookies (named by threat actor) is a modernized injector written in Rust. Two analyzed samples show compile timestamps of 2026.01.07 08:06:04 and 2026.01.27 11:46:05. The samples contain the hardcoded build path C:\Users\kumar\, providing a direct link to the development environment used for SupaServ.

The tool utilizes a standard injection method to deploy its payload into chrome.exe, brave.exe, or edge.exe:

1. Creating a suspended browser process.
2. Using VirtualAllocEx and WriteProcessMemory to write the path of payload.dll into the target process memory.
3. Resolving the address of LoadLibrary within kernel32.dll.
4. Using CreateRemoteThread to call LoadLibrary, forcing the browser process to load the malicious payload.

The malware includes a help menu:

Usage:

```
luminous_cookies [options] <chrome|brave|edge|all>
```

Options:

```
--output-path|-o <path>  Directory for output files (default:  
.\output\  
--verbose|-v              Enable verbose debug output from the  
injector  
--fingerprint|-f          Extract browser fingerprinting data  
--help|-h                 Show this help message
```

Browser targets:

```
chrome  - Extract from Google Chrome  
brave    - Extract from Brave Browser  
edge     - Extract from Microsoft Edge  
all      - Extract from all installed browsers
```

After execution, the LuminousCookies injector first attempts to locate and load a file named `payload.dll` from its current working directory. The tool then scans the system to identify active target processes for Google Chrome, Microsoft Edge, or Brave. Once a target process is found, the injector initiates the LoadLibrary injection sequence to force the browser to load the payload. For data transfer, the injector creates a named communication pipe and enters a wait state until it receives a completion signal from the injected decryptor module.

```
[*] [*] Processing 3 browser(s):  
[*] Chrome  
[#] Searching Registry for: chrome.exe
```

```
[#] Found at: "C:\\Program
Files\\Google\\Chrome\\Application\\chrome.exe"
[#] Creating suspended process: "C:\\Program
Files\\Google\\Chrome\\Application\\chrome.exe"
[#] Process created. PID: 6100
[#] Architecture check passed (x64).
[#] Pipe name: \\.\\pipe\\chrome.nacl.6100.decryption
[#] Named pipe server created: \\.\\pipe\\chrome.nacl.6100.decryption
[#] Injecting DLL: C:\\Users\\<REDACTED>\\Desktop\\payload.dll
[#] Allocated memory at 0x80000
[#] Remote thread created for LoadLibrary
[#] Waiting for payload connection...
```

A technical conflict exists between the known ABE decryptor DLLs and the LuminousCookies injector. Standard ABE decryptor DLLs expect a communication pipe name to be passed as a parameter to DllMain, typically accomplished via process hollowing. However, the LoadLibrary injection method used by LuminousCookies does not support passing arguments to the entry point. This suggests the existence of a modified variant of the payload.dll that likely contains hardcoded pipe names, which we were unable to obtain.

BackupSpy

BackupSpy is a Rust-based utility (compiled 2026-01-12) designed to monitor the local file system and external media for high-value data. Internally identified by the string "BACKGROUND WATCHER STARTED - VERSION 2", this tool functions as a specialized collector that stages data for exfiltration by other components in the suite, storing everything in a single location C:\\Users\\Public\\systemTemp.

BackupSpy uses a hardcoded custom configuration stored at C:\\Users\\Public\\systemTemp\\config.toml. If the configuration file is not found, it drops a default version. The default configuration includes an extension whitelist (16 file types) and a 50MB maximum file size threshold.

```
# enable_copy = true (by default) : set to false manually if you don't want copying
enable_copy = true
```

```
# copy_destination : where files will be copied when enable_copy = true
copy_destination = "C:\\Users\\Public\\systemTemp\\copied"
```

```
# Extension whitelist: only files with these extensions will be considered for copying/manifest
allowed_extensions =
```

```
["txt","md","pdf","doc","docx","xls","xlsx","csv","json","xml","html","htm","jpg","jpeg","png","ppt","pptx"]
```

max_copy_size_bytes : any file larger than this will be skipped during copy (safety)
max_copy_size_bytes = 52428800

BackupSpy generates a detailed manifests of exfiltrated files in JSON format (manifest.json and drive_manifest__%Y%m%dT%H%M%S.json). The tool also maintains an operation log at C:\Users\Public\systemTemp\log.txt. This log records scan starts, file matches, and errors.

C:\Users\Public\systemtemp		
n	Name	Size
..		Up
copied		Folder
config	toml	602
drive_manifest_D_20260224T043048	json	89
log	txt	920
manifest	json	82

Manifest files include the following attributes:

- manifest.json
 - tracked_files
 - path
 - size
 - filename
 - last_modified
- drive_manifest.json
 - drive
 - detected_at
 - files
 - path
 - size

ZigLoader

ZigLoader is a specialized loader written in the Zig programming language. Its primary function is the decryption and execution of arbitrary shellcode in memory. The analyzed sample (c0218ece73d9046e25293fbef71bc70c) shows signs of active development and anti-forensic techniques, including timestomping the PE header's timestamp to date 1981-07-24 18:17:51. In the analyzed sample, the shellcode only executes calc.exe, indicating that ZigLoader is likely a proof-of-concept project.

ZigLoader uses a rolling XOR scheme to decrypt its 368-byte internal payload. The routine starts with a base key (0xAA) that is modified after processing each byte.

```
key = 0xAA
for byte in encrypted_data:
    payload += bytes([byte ^ (key & 0xFF)])
    key = (key + 0x37) & 0xFF
```

The decrypted output is a Base64 string that, once decoded, contains x64 shellcode. This shellcode does not call Windows APIs directly by name - instead, it uses a custom hashing algorithm to resolve function addresses at runtime.

Identified API Hashes:

- **0x876F8B31**: kernel32.dll!WinExec
- **0x9DBD95A6**: kernel32.dll!GetVersion
- **0x56A2B5F0**: kernel32.dll!ExitProcess
- **0x6F721347**: ntdll.dll!RtlExitUserThread

The loader leverages low-level system calls for memory manipulation, including `NtAllocateVirtualMemory` and `NtFreeVirtualMemory`, to stage the shellcode before execution via `CreateThread`.

Gate Sentinel Beacon

Gate Sentinel Beacon is a customized version of the open-source [GateSentinel project](#). This tool is a modern C2 framework, combining a Go-based server and C-based client.

The most significant deviation from the original source is the implementation of a additional persistence mechanism. Rather than relying solely on services or registry keys, the attacker modifies legitimate application shortcuts (.lnk files). In addition to opening the intended application, the modified shortcut also executes the malware in the background.

The investigation identified five shortcuts that were modified:

- **Google Chrome**: C:\Users\%USERNAME%\Desktop\Google Chrome.lnk
- **Microsoft Edge (Public)**: C:\ProgramData\Microsoft\Windows\Start Menu\Programs\Microsoft Edge.lnk
- **Microsoft Edge (User)**:
C:\Users\%USERNAME%\AppData\Roaming\Microsoft\Windows\Start Menu\Programs\Microsoft Edge.lnk

- **Microsoft Edge (User-Pinned Taskbar):**
C:\Users\%USERNAME%\AppData\Roaming\Microsoft\Internet Explorer\Quick Launch\User Pinned\TaskBar\Microsoft Edge.Ink
- **Microsoft Edge (OS-Pinned Taskbar):**
C:\Users\%USERNAME%\AppData\Roaming\Microsoft\Internet Explorer\Quick Launch\User Pinned\ImplicitAppShortcuts*\Microsoft Edge.Ink

Conclusion and Recommendations

The transition of APT36 toward vibeware represents a technical regression. While AI assisted development increases sample volume, the resulting tools are often unstable and riddled with logical errors. The actor's strategy incorrectly targets signature-based detection, which has long been superseded by modern endpoint security.

Advanced defense stacks include behavioral analysis, memory scanning, and process monitoring to identify malicious activity, meaning a niche binary remains detectable the moment it attempts to inject into a process or communicate with a cloud API.

The threat remains a hybrid model. AI produces the disposable front line of the malware fleet, but the subsequent hacking operations remain manual.

1. Prioritize Behavioral Detections

Prioritize detections [based on process behavior](#) rather than file runtimes. Monitor for the execution of unsigned binaries in user writable directories like %APPDATA%. EDR platforms should alert on unusual API calls or process hollowing, which remain consistent across the vibeware fleet regardless of the language used.

2. Audit and Control Trusted Cloud Services (LOTS)

The reliance on Discord, Slack, and Google Sheets for command and control is a central pillar of this campaign. Organizations must implement granular monitoring for these services. Any persistent outbound connection to these platforms originating from unverified binaries should be treated as a potential indicator of compromise.

3. Hostile Network Environments

To protect against the manual hacking phase, turn your network into a hostile and unpredictable environment for attackers by implementing [dynamic attack surface reduction](#), combined with an actionable EDR/XDR platform such as [Bitdefender GravityZone](#) and mature SOC/MDR operations. These measures introduce friction and uncertainty for the human operator, forcing them to reveal their presence during the lateral movement and discovery phases.

IOCs and How to Follow Our Research

For our OEM partners and integrations, access to our [threat intelligence](#) data is primarily provided programmatically. We also offer a user interface, [IntelliZone Portal](#). This is where partners get more ways to interact with our data, like an operational dashboard of threats targeting their industry. A full breakdown of this research can be found on the platform under ThreatID BD9a057rfr:

<https://intellizone.bitdefender.com/en/threat-search/threats/BD9a057rfr>

Beyond our core TI platform, here are additional ways to stay current with our research.

Explore Public IOCs on GitHub

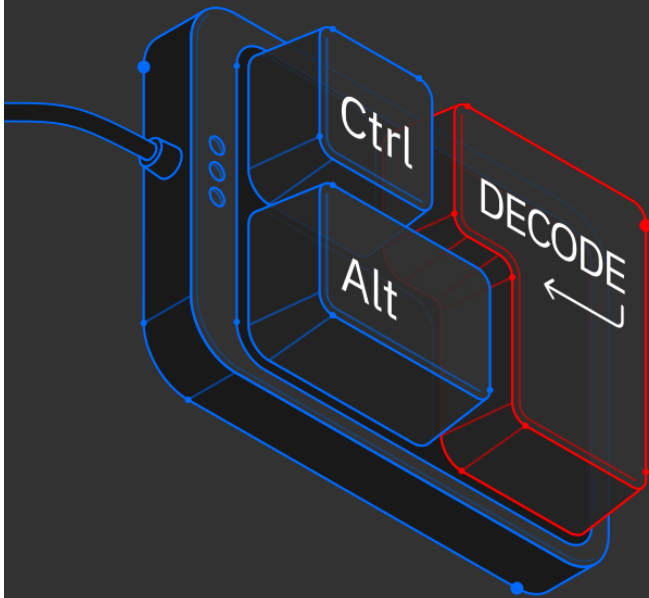
We are hosting all Indicators of Compromise (IOCs) from this and all future research on a public GitHub repository to improve accessibility and collaboration for the entire security community:

https://github.com/bitdefender/malware-ioc/blob/master/2026_03_05-apt36-iocs.csv

Watch & Read Ctrl-Alt-DECODE

This research is part of Ctrl-Alt-DECODE, Bitdefender's newly established threat intelligence initiative.

1. **Subscribe to the Newsletter:** Get exclusive threat intelligence, original research, and actionable advisories directly from Bitdefender Labs and MDR teams:
<https://www.linkedin.com/newsletters/7371216616015036416/>
2. **Watch the Live Series:** See the expert analysis on the APT36's vibeware on our [next Ctrl-Alt-DECODE episode](#) (or catch up with [our previous episodes](#)). For this session, we're excited to welcome the Bitdefender Labs researcher who led the forensic analysis who will join us in the chat, giving you a rare opportunity to ask technical questions about the research, or even what it's really like to work in cybersecurity forensics.



LIVE DISCUSSION

→ Episode 07

Vibeware Analysis: AI-Generated Toolkit by APT36

Join us for *Vibeware Analysis: [Is AI-Generated Malware by APT36 a Real Threat or Just Noise?](#)*