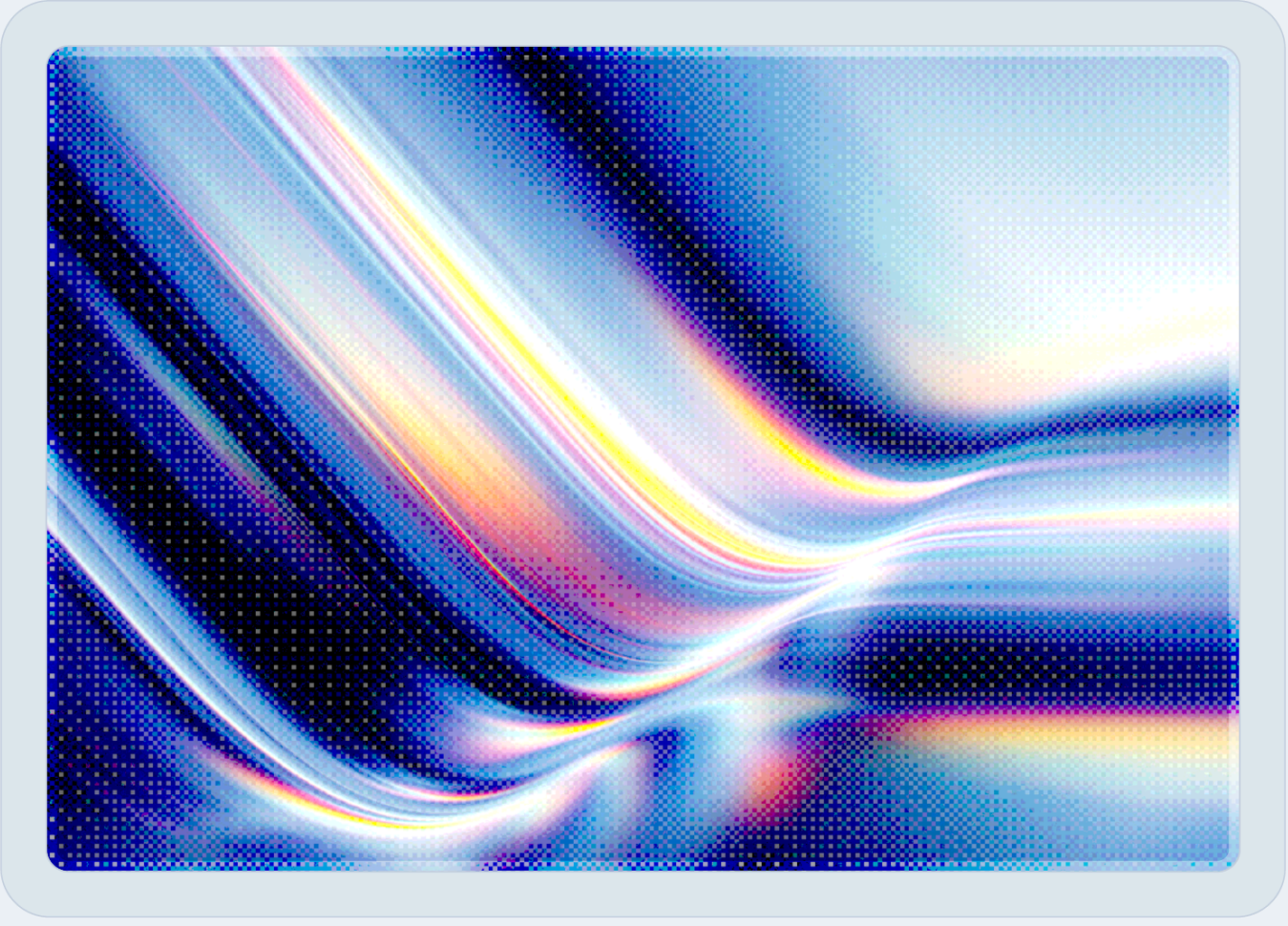
CSAI Foundation | Cloud Security Alliance

PolinRider: North Korea-Linked Campaign Poisons 7,600 Developer GitHub Repositories

Security Implications and Guidance for Forged-Commit Repository Worms

2026-11-29

AI-assisted Rapid Research



CSAI

CSA cloud security alliance®

© 2026 Cloud Security Alliance. Some rights reserved.

You may download, store, display, view, print, redistribute, and link to this document in its original, unmodified form, provided that attribution to the Cloud Security Alliance is maintained and all trademark and copyright notices remain intact.

This document may not be modified or altered. You may quote portions of the document as permitted by the Fair Use provisions of the United States Copyright Act, provided that attribution is given to the Cloud Security Alliance.

This document may be shared on professional and social media platforms in its original form with attribution.

This document was generated with AI assistance and has not undergone official CSA review and approval processes.

Key Takeaways

PolinRider is a software supply chain campaign attributed by several research firms to North Korean operators. It has turned the GitHub repositories of ordinary, long-established developers into malware distribution points. In November 2026, The Hacker News reported Apiiro's estimate that at least 7,609 GitHub repositories owned by 2,330 accounts were infected. By Apiiro's analysis, 86% of those repositories (6,541) belong to personal accounts more than a year old with genuine activity, and only 16 owners look like throwaway accounts [1]. The victims, in other words, are real developers rather than attacker-created lures. The worm spreads from a compromised workstation into every repository the developer can push to, rewriting existing commits while keeping the original author, message, and date so that the poisoned history looks legitimate [1][6].

The campaign is dangerous because it does not wait for a developer to install anything. Payloads hide in build configuration files, test fixtures such as `conftest.py`, fake font files, and `.vscode/tasks.json` entries set to `runOn: folderOpen`, so building, testing, or simply opening a project can run the code. In the newest wave, opening the folder in VS Code (or a VS Code-compatible editor) and accepting the workspace-trust prompt, which many developers do by reflex, is enough to run the payload [1]. Once running, the loaders read the current command-and-control (C2) address from Ethereum and other blockchain transactions using the "EtherHiding" and "NullReceiver" techniques. Because those records cannot be deleted, a single new transaction redirects every deployed loader [1][2].

PolinRider also spans ecosystems and keeps evolving. Researchers have tracked it across npm, Packagist, Go modules, and Chrome extensions [3][4], and most recently through Python and TypeScript loaders hosted directly in repositories [2]. The earliest known activity dates to 2025 [6], and Socket considers a major disruption of the operators unlikely in the near term [3]. For defenders, the most important lesson is that cleaning the repository does not end the incident. Any machine that opened or built an affected repository should be treated as compromised, and credentials should be rotated from a separate, clean host [3][5].

Background

PolinRider first came to public attention in March 2026, when OpenSourceMalware flagged a cluster of GitHub repositories carrying obfuscated JavaScript appended to otherwise normal configuration files [4]. By April 11, 2026, The Hacker News reported, the activity had reached 1,951 public GitHub repositories associated with 1,047 unique owners [4]. Wiz's threat catalog, published around the same time, described the campaign as compromising more than 1,900 repositories through malicious npm packages, VS Code artifacts, and injected

JavaScript, with a final-stage payload consistent with the BeaverTail malware family [7]. At that stage the campaign looked like one more entry in a long line of North Korean operations aimed at developers. Its most notable feature was the way it reused victims' own repositories.

Over the summer, the campaign's reach across ecosystems became clearer. In July 2026, Socket reported 162 malicious release artifacts across 108 unique packages and browser extensions spanning npm, Packagist (Composer), Go modules, and the Chrome Web Store [4][5]. Socket observed that the JavaScript loaders led to the DEV#POPPER remote access trojan and the OmniStealer information stealer [5]. In September, Socket documented the compromise of the Visanduma GitHub organization, which had begun in mid-June 2026. The malicious code reached the development branches (not tagged releases) of the Packagist package `visanduma/nova-two-factor`, a package with more than 700,000 cumulative downloads across all versions [3]. Socket's central observation was that PolinRider spreads mainly through Git collaboration rather than through package registries. Ecosystems such as Go and Packagist resolve code directly from Git, so a poisoned repository can reach downstream users without the attacker ever stealing a registry token [3].

In late October, OpenSourceMalware pushed the campaign's history back further. It identified nine npm packages from a single maintainer that were published within a 90-minute window on May 4, 2025, which it called the earliest known PolinRider activity, eight months before the earliest documented victims [6]. Those earliest victim infections date to around the turn of the year; SecurityWeek described the campaign as ongoing since December 2025 [5], although public disclosure did not follow until March 2026 [4]. The on-chain stages used in the May 2025 attack had been written to the BNB Smart Chain and Ethereum on February 7, 2025, 86 days before the npm publication [6]. The same maintainer's GitHub repositories were then reinfected in four separate windows during 2026. OpenSourceMalware assesses that the infection persisted on the maintainer's own machine and that the later reinfections were automated force-pushes from it [6].

November 2026 brought the largest wave to date. Apiiro reported that the loader was pushed to 2,040 repositories between October 27 and November 6, 2026, part of the 7,609-repository total cited above [1]. On November 24, SafeDep published an analysis of a new 58-line Python loader found in 375 public repositories under 260 accounts, and a November 28 update added a TypeScript variant found in 528 further repositories [2]. Meanwhile, researchers documented a GitHub account, "Eternity0207," that had opened more than 200 pull requests against open-source projects before GitHub suspended it [1]. Targeted projects included the AsyncAPI Generator, Open Food Facts, and CircuitVerse, and named victims included Neutralinojs, InstaWP Connect, and fxhash [1].

On attribution, researchers link PolinRider to North Korea's Contagious Interview operation (tracked by MITRE ATT&CK as G1052), in which fake recruiters lure developers into running malicious take-home assessments [4] [15]. PolinRider is assessed to be a sub-cluster operating in parallel with Contagious Interview [1]. In this reading, the fake-job lure provides initial access to a developer's machine, and PolinRider converts that access into distribution through repositories. Recent lures cited in reporting include fake hiring portals branded BlockNovas, AccuraSoftHire, and HuntRole, along with a malicious VS Code Marketplace extension named

`huntrole.assessment-tool` [1]. The attribution rests on shared code, encryption keys, and infrastructure observed by private researchers. It should be treated as a well-supported assessment rather than a confirmed government finding.

Security Analysis

The Propagation Model: Trust Inherited from Real Developers

PolinRider differs from most repository-based malware campaigns in where its trust comes from. Campaigns such as fake-repository floods typically depend on newly created accounts that defenders can identify by age, activity patterns, or star inflation. PolinRider instead inherits the reputation of established developers. Once a workstation is infected, a propagation script (documented in earlier waves as `temp_auto_push.bat`) amends the most recent commit in each reachable repository. It folds the payload into that commit, rolls back the timestamp, and force-pushes the result using the developer's own credentials or authenticated session [6] [7]. Apiiro found that the rewritten commits keep the original author, message, and date [1]. Socket's Karlo Zanki concluded that, as a result, a repository's GitHub landing page and visible commit history are "unreliable indicators of compromise" [4].

The access model also explains the worm's speed, although the precise mechanism remains unsettled. Early in the campaign, OpenSourceMalware stated that the actor was not using stolen GitHub credentials and that victims had instead been compromised via a malicious VS Code extension or npm package [4]. Later reporting describes stolen GitHub tokens being used for force-pushes [1]. It is not yet clear whether this reflects a change in tradecraft or differing observations across researchers. In either case, the attacker never needs to defeat GitHub authentication directly, because the victim's workstation already holds all the access required. Every collaborator relationship, organization membership, and personal access token on that machine becomes a path to the next set of repositories. This pattern resembles the self-propagating npm worms of 2025 and 2026, but it operates at the Git layer, beneath package registries.

Execution Triggers Built into Developer Workflow

The campaign's choice of execution triggers reflects familiarity with routine developer workflows. Its loaders do not rely on install hooks, which security tools increasingly inspect. Instead, they ride on actions developers take routinely, such as building, testing, or opening a project. The table below summarizes the triggers documented across research reports.

Trigger	Mechanism	Example observed	Source
Build	Obfuscated JavaScript appended after hundreds of whitespace characters in `tailwind.config.js`, `postcss.config.mjs`, `eslint.config.mjs`, `next.config.mjs`, `babel.config.js`, `jest.config.js`	`npm run build` in AsyncAPI Generator pull requests	[1][2] [4]
Folder open	`.vscode/tasks.json` with `"runOn": "folderOpen"` in VS Code-compatible editors	CircuitVerse; TaskJacker cluster merged into PolinRider	[1][3] [4]
Test run	Planted `utils` package or `conftest.py` imported by `pytest`	Open Food Facts; SafeDep Python loader	[1][2]
Package import	Payload appended to `index.js` or `webpack.config.js` of a published npm package	`itsa-*` packages (May 2025)	[6]
Server-side PHP	Obfuscated JavaScript in `index.php` executed via `shell_exec`	Visanduma repositories	[3]
Disguised assets	Executable content in fake `.woff2` / `.llf` font files	`fa-solid-400.woff2`, `fa-solid-500.woff2`	[1][3]

Several of these triggers fire before a developer has read any code. The folder-open trigger may be particularly relevant to teams using AI coding assistants and agent-driven IDEs, which can clone and open repositories on a user's behalf; whether a given tool honors `runOn: folderOpen` without a trust prompt depends on its configuration. Separately, reporting indicates that the second-stage payload looks for developer applications including VS Code, Cursor, GitHub Desktop, and Google Antigravity. Apiiro also found the November remote-access payload persisting in VS Code, Cursor, Discord, and GitHub Desktop [1].

The whitespace-padding technique serves a similar concealment purpose. A reviewer looking at a diff or file in a typical editor sees a clean configuration file, while the payload sits hundreds of columns to the right [1][6]. Pull-request timing compounds the problem. An AsyncAPI Generator maintainer reported that each payload stayed

on a pull-request branch for only "30-40 minutes" before being force-pushed away, "so the final diff looked clean" [1]. Taken together, these techniques defeat the casual inspection on which most open-source review depends.

Takedown-Resistant Command and Control

PolinRider's command-and-control design makes conventional infrastructure takedowns of limited value. The specific blockchains have varied by wave: the 2025 npm packages staged code on BNB Smart Chain and Ethereum [6], the early 2026 EtherHiding stages stored encrypted payloads in transaction data on the TRON, Aptos, and BNB Smart Chain networks [4][7], and the November 2026 loaders read from Ethereum [1][2]. The newer NullReceiver technique encodes the C2 server's IP address in the first bytes of a fabricated destination wallet address, followed by the marker string that decodes to "helloipbot!!" [1].

SafeDep's analysis of the November Python loader traced the full chain. The loader queries the newest self-sent transaction from wallet `0x3b1574417524c11a7403c4b232CAcBb320CceBC6`, decodes its input data into a 31 KB JavaScript stage, and runs it with Node.js. That stage reads a second "heartbeat" wallet, `0x1eEf0849EDa15d87f64e51C73C8442Cf3c8bCEF3`, to find the current C2 server [2]. SafeDep counted 366 heartbeat transactions from that wallet between September 4 and November 24, 2026, pointing to five C2 servers over time [2]. Final stages are retrieved from the `/0x/cls` and `/0x/ls` paths and run in detached `node -i` processes so their code does not appear in process listings [2].

Because blockchain records cannot be deleted, defenders cannot remove the dead-drop. Blocking known C2 IP addresses helps only until the operators publish a new transaction, and Apiiro noted that one transaction is enough to redirect every deployed loader [1]. Network defenses therefore work better against the loader's behavior. Examples include outbound queries to public blockchain RPC endpoints and explorer APIs from build or developer processes, followed by `node -e` or `node -i` execution, as OpenSourceMalware's hunting guidance describes [6].

Payload Objectives and Downstream Exposure

The payloads point to two objectives: financial theft and broad credential collection. Socket assesses cryptocurrency theft as the primary goal, with wider infostealer capability covering credentials and source code [3]. The 2025 stealer analyzed by OpenSourceMalware targeted more than 80 browser extensions and exfiltrated data through Telegram [6]. Apiiro's analysis of the November `/0x/cls` payload describes a likely remote-access tool. It avoids CI machines, runs operator-supplied JavaScript and shell commands, reads the clipboard, uploads files, and establishes persistence in developer applications [1]. Skipping CI environments suggests the operators want interactive, credential-rich developer workstations rather than short-lived build runners. That reading is consistent with the worm's need for fresh push access to keep spreading.

For enterprises, the exposure extends well beyond developers who were directly lured. Any engineer who cloned, built, tested, or opened an affected public repository may have run the loader. Any organization that pulls dependencies straight from Git branches, as Go modules and Packagist `dev-*` versions do, may have consumed poisoned code without a registry compromise [3]. SafeDep found that most of the planted Python files sat in a `utils` folder inside the project and that 370 of the 375 affected repositories had a push in November 2026 [2]. This indicates an active, recent wave rather than a historical artifact.

Recommendations

Immediate Actions

Organizations should first determine whether any developer or build system has interacted with affected repositories or packages. Security teams can search internal mirrors, monorepos, and developer workstations for the published indicators summarized in the table below, along with two structural signs: unusually long whitespace-padded lines in configuration files [1][6] and `.vscode/tasks.json` files containing `folderOpen` [1][3]. C2 IP addresses are likely to rotate with each new blockchain transaction, so teams should prioritize behavioral and wallet-based detections over IP blocklists.

Indicator	Type	Role	Source
`0x3b1574417524c11a7403c4b232CAcBb320CceBC6`	Ethereum wallet	Python loader dead-drop; self-sent transactions carry the JavaScript stage	[2]
`0x1eEf0849EDa15d87f64e51C73C8442Cf3c8bCEF3`	Ethereum wallet	"Heartbeat" wallet pointing to the current C2 server	[2]
`0xa322E5f3D311D3080e6f0121063e9aDC2490Ef1a`	Wallet	Dead-drop sender	[1]

Indicator	Type	Role	Source
`0xa56f613D8E5339F799B38E2b2c4B2D07f811e019`	Wallet	TypeScript-variant wallet	[2]
`/0x/cls`, `/0x/ls`	URL paths	Final-stage retrieval from C2	[2]
`95.135.181[.]84`, `95.135.181[.]151`	IP addresses	C2 servers (time-bound)	[1][2]

SafeDep's post provides `grep`, `find`, and `git log -S` commands suited to this sweep [2]. Searching Git history matters because, as noted earlier, the visible tip of a branch may have been restored to a clean state.

Where a trigger has run, and particularly on any machine with Node.js installed (which all documented loader chains require), the machine should be treated as compromised [2]. CSA recommends reimaging such machines rather than attempting to clean them. Credential rotation should be carried out from a separate, known-clean host [5]. It should cover GitHub personal access tokens and SSH keys, package registry tokens, cloud provider keys, CI/CD secrets, browser-stored passwords and session cookies, and any cryptocurrency wallet material [2] [3]. Organizations should also review GitHub audit logs for force-pushes, branch-protection changes, new tokens or sessions, and collaborator changes during the exposure window [3]. OpenSourceMalware advises investigators to "trust GitHub's push times over commit dates," since commit dates are forged [6].

Short-Term Mitigations

The most direct control against PolinRider's propagation method is to prevent force-pushes to protected branches. GitHub rulesets can block force-pushes, require pull requests, and require signed commits on default and release branches across an organization [8]. Requiring verified signatures is especially useful here, because the worm rewrites commits under the original author's identity, and enabling GitHub's vigilant mode flags unsigned commits that claim a user's identity [9]. Signing does not help if the signing key sits unprotected on the compromised workstation. Hardware-backed keys that require user presence (a touch) for each signature, or short-lived signing certificates, provide stronger assurance. Teams should also recognize that rulesets protect designated branches only; pull-request branches from contributor forks remain mutable, which is where the AsyncAPI force-push pattern occurred.

Editor and build behavior should be hardened in parallel. VS Code's Workspace Trust feature, kept in restricted mode for newly cloned repositories, prevents tasks and other automatic execution from running until a user explicitly trusts the folder [10]. Because the newest wave depends on developers accepting that prompt,

organizations should configure restricted mode as the default and train developers not to grant trust reflexively to unfamiliar repositories. Teams should treat any change to `.vscode/`, build configuration, or test configuration files as security-relevant during code review, and linters or pre-commit checks can flag lines longer than a reasonable threshold or containing large whitespace runs. Dependency consumers should stop resolving packages from mutable development branches such as `dev-main` and instead pin to reviewed commits or stable releases [3]. Finally, security operations teams can add detections for developer and build processes that contact blockchain RPC endpoints or explorer APIs and then spawn Node.js with inline or interactive arguments [6].

Strategic Considerations

PolinRider strengthens a conclusion CSA has drawn from earlier campaigns: developer workstations are production-adjacent systems and should be governed as such. A developer's laptop holds push rights to many repositories, tokens for registries and clouds, and sometimes personal cryptocurrency wallets, the asset class this campaign primarily targets [3]. That combination makes it among the most valuable footholds in the software supply chain. Organizations should consider separating day-to-day browsing and recruitment activity from development environments. They can also use isolated, ephemeral development containers for evaluating untrusted code, including take-home assessments and AI-agent-cloned repositories. Short-lived, narrowly scoped credentials, such as GitHub App tokens and OIDC-based publishing, limit how far a single compromised machine can spread a worm.

The campaign also exposes a structural weakness in how the industry judges repository trust. Account age, contribution history, and commit authorship, the signals most often used to separate legitimate projects from malicious ones, are exactly what PolinRider borrows from its victims. Provenance controls that tie code to verified build and push events, rather than to self-asserted commit metadata, deserve investment. So do platform-level capabilities that make force-pushes and history rewrites visible to downstream consumers. Finally, because researchers expect the operators to persist [3], organizations with significant exposure to North Korean targeting, particularly in cryptocurrency, Web3, and AI development, should include Contagious Interview-style recruitment lures in security awareness programs for engineering staff.

CSA Resource Alignment

CSA has tracked the expansion of North Korean developer-targeting operations throughout 2026, and PolinRider fits within that body of work. CSA Labs' research note *Contagious Interview Expands to Five Package Ecosystems* [11] documented how the Contagious Interview operation moved beyond npm into multiple package ecosystems. That analysis anticipated the cross-ecosystem reach PolinRider now shows across npm, Packagist, Go, and Python. PolinRider takes the trend a step further by shifting from registries to Git itself, which strengthens that note's recommendation to monitor all ecosystems a development team consumes rather than npm alone.

CSA's analysis of the Sapphire Sleet compromise of the Mastra AI npm ecosystem [12] examined a separate North Korean operation in which a hijacked maintainer account was used to backdoor AI developer tooling within minutes. Taken together, the Mastra and PolinRider incidents point toward a common DPRK-linked strategy: harvest the trust of legitimate maintainers and developers rather than build trust from scratch. The guidance in that note on maintainer account hardening and rapid credential rotation applies equally here.

CSA Labs' *GlassWorm Takedown: Developer Supply Chain Attack Campaigns* [13] established the developer workstation, and in particular the IDE extension and task surface, as a primary supply chain attack vector. PolinRider's use of `.vscode/tasks.json` folder-open triggers and its persistence in VS Code, Cursor, and GitHub Desktop are consistent with that framing. The editor allowlisting, workspace-trust, and detection recommendations in the GlassWorm analysis transfer directly to this campaign.

At the controls level, the CSA AI Controls Matrix (AICM) v1.1 [14] provides a framework for operationalizing these recommendations. The most relevant domains are Threat and Vulnerability Management (TVM); Application and Interface Security (AIS); Identity and Access Management (IAM); Supply Chain Management, Transparency, and Accountability (STA); Change Control and Configuration Management (CCC); and Universal Endpoint Management (UEM). Organizations can map force-push restrictions and commit-signing requirements to the CCC domain's change-control and integrity controls. Short-lived credential practices map to IAM, and developer-workstation isolation for AI development environments maps to UEM and STA.

References

[1] Lakshmanan, Ravie. "North Korea-Linked PolinRider Poisons Over 7,600 GitHub Repositories to Spread Malware." The Hacker News, November 26, 2026.

[2] SafeDep Team. "PolinRider Moves to Python with an Ethereum Calldata Loader." SafeDep, November 24, 2026 (updated November 28, 2026).

[3] Zanki, Karlo. "PolinRider Spreads Through Compromised GitHub Accounts and Packagist." Socket Threat Research Team, September 17, 2026.

[4] Lakshmanan, Ravie. "North Korean Hackers Publish 108 Malicious Packages and Extensions in PolinRider Campaign." The Hacker News, July 4, 2026.

[5] Arghire, Ionut. "North Korean Hackers Target Open Source Developers in Supply Chain Attacks." SecurityWeek, July 6, 2026.

[6] OpenSourceMalware. "Earliest Known PolinRider Malware Found in 2025 npm Packages." OpenSourceMalware, October 30, 2026.

[7] Wiz. "PolinRider Campaign: DPRK-Linked Supply Chain Attack Infects GitHub Repositories." Wiz Cloud Threat Landscape, April 9, 2026.

[8] GitHub. "About Rulesets." GitHub Docs, accessed November 29, 2026.

[9] GitHub. "Displaying Verification Statuses for All of Your Commits." GitHub Docs, accessed November 29, 2026.

[10] Microsoft. "Workspace Trust." Visual Studio Code Documentation, accessed November 29, 2026.

[11] Cloud Security Alliance. "Contagious Interview Expands to Five Package Ecosystems." CSA Labs, 2026.

[12] Cloud Security Alliance. "Sapphire Sleet Poisons Mastra AI npm Supply Chain." CSA Labs, June 2026.

[13] Cloud Security Alliance. "GlassWorm Takedown: Developer Supply Chain Attack Campaigns." CSA Labs, May 2026.

[14] Cloud Security Alliance. "AI Controls Matrix (AICM) v1.1." Cloud Security Alliance, 2026.

[15] MITRE ATT&CK. "Contagious Interview, Group G1052." MITRE Corporation, accessed November 29, 2026.