CSAI Foundation

Cloud Security Alliance AI Safety Initiative

prt-scan: GitHub Actions Supply Chain Campaign

AI-Assisted Exploitation of pull_request_target Misconfigurations

Unofficial AI-assisted Research

2026-04-14

Key Takeaways

- The prt-scan campaign is an AI-assisted supply chain attack that exploited a commonly misconfigured GitHub Actions workflow trigger — `pull_request_target` — to steal repository secrets, cloud credentials, and CI tokens from open-source projects at scale.
- Beginning March 11, 2026, a single threat actor cycled through six separate GitHub accounts and submitted more than 500 malicious pull requests in coordinated waves, achieving verified credential theft in fewer than 10% of attempts [1][2].
- The campaign's five-phase payload — spanning environment variable exfiltration, secret enumeration, workflow injection, label-gating bypass, and a persistent `/proc` scanner — represents a materially more sophisticated payload than prior CI/CD exploitation campaigns [1][3], incorporating five discrete stages and AI-assisted target adaptation.
- prt-scan is the second AI-assisted campaign in early 2026 to exploit `pull_request_target` misconfiguration at scale, following the hackerbot-claw operation in February; both suggest that automated CI/CD exploitation is maturing as an attack capability, with at least two distinct operations demonstrating scale and iteration in early 2026 [2][4].
- GitHub's November 2025 change enforcing default-branch workflow sourcing for `pull_request_target` reduces but does not eliminate attack surface; repositories must still apply least-privilege token permissions, require human approval before privileged workflow runs, and audit existing workflow configurations [5].

Background

The pull_request_target Misconfiguration

GitHub Actions introduced the `pull_request_target` event trigger to allow automated workflows — such as those that label or comment on pull requests from untrusted forks — to run with access to the base repository's secrets and a `GITHUB_TOKEN` scoped to the base repository. Unlike the standard `pull_request` trigger, which confines execution to the forked branch context with

minimal permissions, `pull_request_target` intentionally grants elevated privileges so that maintainers can automate reviewer assignment, continuous integration labeling, and contribution workflows across fork boundaries [6][7].

This design is secure in principle but dangerous in practice when workflows also check out or execute code from the incoming pull request. The GitHub Security Lab documented this class of vulnerability as early as 2021, coining the term "pwn request" to describe the pattern in which `pull_request_target` triggers code execution from an untrusted branch under elevated-privilege context [6]. Despite years of documentation and public advisories, open-source repositories have continued to introduce the misconfiguration — either by copying workflow templates that conflate the two triggers or by intentionally using `pull_request_target` for convenience while inadvertently checking out attacker-controlled code [7][8][9].

Wiz Research analysis of the prt-scan campaign found the misconfiguration present across a wide cross-section of GitHub's public repository ecosystem, from single-developer hobby projects to maintained repositories within established open-source foundations [1].

Preceding Campaign: hackerbot-claw

Between February 21 and March 2, 2026, a GitHub account named hackerbot-claw conducted a more targeted predecessor operation, systematically scanning public repositories for exploitable `pull_request_target` workflows and submitting malicious pull requests to a curated list of high-profile targets, including repositories linked to Microsoft, DataDog, and Cloud Native Computing Foundation projects [4][10]. The campaign attracted significant industry attention; GitHub's November 2025 platform-level mitigations followed shortly after [5], though the direct relationship between the two is not publicly documented. hackerbot-claw demonstrated that the attack class was feasible at scale; prt-scan demonstrated that it was repeatable and evolving — and may indicate the attack class is becoming accessible to a broader range of actors, given its iterative refinement across six weeks and reliance on AI-assisted payload generation [1][3], though the relationship between the two campaigns' actors is unconfirmed.

Security Analysis

Campaign Timeline and Actor Behavior

Wiz Research reconstructed six waves of activity starting March 11, 2026, attributing them to a single threat actor based on consistent tooling, branch-naming conventions, and payload structure [1]. The earliest wave used a now-deleted account and opened pull requests against a small set of repositories to test payload viability. Subsequent waves refined the tooling and gradually expanded targeting scope. The campaign's most visible phase began April 2, 2026 at approximately 20:59 UTC, when the account `ezmtebo` was created. Within two minutes of account creation, the first malicious pull request was opened, and over the next 26 hours the account submitted more than 475 pull requests across a wide range of repositories [1].

All pull requests carried identical surface-level metadata: a branch name beginning with the `prt-scan-` prefix and the PR title "ci: update build configuration" — a plausible string that closely mimics routine CI maintenance PR titles [1][3]. The consistency of this metadata across hundreds of submissions ultimately made the campaign trivial to identify in aggregate, but individual repository maintainers encountering a single unfamiliar PR with a routine-sounding title remained susceptible to accidental triggering.

Five-Phase Payload Architecture

The prt-scan payloads became substantially more sophisticated over the campaign's six weeks. By the `ezmtebo` phase, the attacker had moved beyond static shell scripts to payloads showing evidence of AI-assisted or programmatically adaptive generation [1] — producing tech-stack-native wrappers that adapted their outer disguise to each target's primary technology stack: Go test files for Go repositories, npm lifecycle scripts for JavaScript projects, and `conftest.py` fixtures for Python projects. The malicious logic was embedded within these containers, reducing the likelihood that a cursory code review would identify the payload as anomalous.

The core payload operated in five stages, which SafeDep and Wiz researchers documented through analysis of recovered workflow logs and PR comment artifacts [1][3]:

The first stage, **EXFIL**, executed immediately upon workflow launch. It dumped all environment variables and extracted the `GITHUB_TOKEN` from git configuration, then gzip-compressed and base64-encoded the output before printing it to workflow logs under a predictable marker string. This exfiltration

channel is notable because it requires no outbound network connection to attacker-controlled infrastructure; all traffic flows to `api.github.com` or link-local cloud metadata endpoints, making network-layer detection difficult [1][3].

The second stage, **RECON**, used the stolen token to enumerate the repository's secret names via the GitHub REST API, list deployment environments, catalog existing workflow files, and probe AWS, Azure, and GCP metadata endpoints for instance metadata service (IMDS) credentials. Successful IMDS access would have yielded temporary cloud credentials independent of repository-configured secrets [1].

The third stage attempted **workflow injection**. If the stolen `GITHUB_TOKEN` carried write permissions to the repository — a condition present when repositories have not restricted the default token permissions — embedded Python created a temporary workflow file at `.github/workflows/.prt_tmp_<nonce>.yml` on the default branch and dispatched it. This injected workflow ran in the context of the base repository and could exfiltrate the actual values of every configured secret, not merely their names [1][3].

The fourth stage addressed a common `pull_request_target` hardening technique: gating privileged workflow execution behind a required pull request label. Many repository maintainers configure their workflows to require a maintainer-assigned label (such as "safe to test") before running CI against a fork PR, under the assumption that this human review step prevents untrusted execution. The prt-scan payload included a base64-encoded Python script that parsed all workflow YAML files in the repository for `pull_request_target` label gates, extracted the required label name (defaulting to "safe to test" when the label name was not determinable), created the label via the GitHub API using the stolen token, and applied it to the attacker's own pull request — effectively auto-approving the PR for elevated execution [1][3].

The fifth stage, **DELAYED**, launched a background Python daemon that polled `/proc/*/environ` every two seconds for ten minutes after the main script exited. This scanner targeted secrets that might be loaded into the process environment by later job steps that had not yet executed when the main payload ran. Captured values were exfiltrated through PR comments, which persist even after workflow log retention windows expire or logs are manually deleted [1][3].

Exfiltration Channel Design and Detection Resistance

The choice to exfiltrate through GitHub's own infrastructure — workflow logs and PR comments — was deliberate and consequential. Standard network-layer security controls such as egress filtering and SIEM rules tuned for connections to unusual IP addresses would not fire on traffic that consists entirely of

legitimate GitHub API calls. Detection required either content inspection of PR comments and workflow logs for base64-encoded exfiltration markers or behavioral analysis of the GitHub API calls made during workflow execution [1][3].

This approach also created persistence advantages for the attacker. Workflow log data is retained according to repository-level log retention settings, but PR comments are not subject to the same automatic deletion. Exfiltrated credential material embedded in PR comments on archived or neglected repositories may remain accessible to the attacker indefinitely.

Impact Assessment

The campaign's fewer-than-10% credential theft success rate, applied across more than 500 pull requests, yielded dozens of confirmed compromises [2][4]. The majority of successful attacks were against small or inactive repositories, and the most commonly exfiltrated material was the ephemeral `GITHUB_TOKEN` for the workflow run — a time-limited credential that expires when the workflow completes. However, verified credential theft of higher-value material was also observed, including AWS access keys, Cloudflare API tokens, and Netlify authentication tokens in repositories that had stored such credentials as repository or environment secrets [2][4]. These credentials carry no automatic expiration and may enable persistent access to cloud infrastructure, content delivery networks, or deployment pipelines.

The observed success rate may understate the campaign's aggregate risk in two respects. First, even ephemeral `GITHUB_TOKEN` theft enables an attacker who can act quickly to push code to the default branch, create releases, or modify workflow files before the token expires — consequences significantly more severe than the token's transience might suggest. Second, the AI-assisted targeting methodology is iterative: each wave of the prt-scan campaign showed improved payload obfuscation and success rates compared to the prior wave, indicating an active optimization loop [1].

Platform-Level Response

GitHub's November 2025 update to `pull_request_target` behavior, effective December 8, 2025, changed the event to always source workflow files from the repository's default branch rather than from any branch designated as the pull request base [5]. This eliminates an earlier vulnerability class in which maintainers might fix a `pull_request_target` misconfiguration in the default branch while older, vulnerable workflow versions remained on long-lived feature or release branches and continued to execute for PRs targeting those branches. The December 2025 change ensures that only the current default-branch workflow definition governs `pull_request_target` execution [5].

This platform change is meaningful but does not address the core risk: a misconfigured workflow on the default branch remains misconfigured regardless of where GitHub sources the workflow file from. Repositories that have never audited their `pull_request_target` workflows for unsafe code checkout remain exposed.

Recommendations

Immediate Actions

Organizations should audit all repositories for `pull_request_target` workflow triggers and assess whether those workflows check out or execute code from the pull request branch. The security risk exists specifically at the intersection of elevated permissions and untrusted code execution; workflows that use `pull_request_target` but do not touch attacker-controlled code are not vulnerable to this attack class. GitHub's official security hardening documentation provides pattern-matching guidance for identifying unsafe workflows [11].

Repositories must also audit and restrict the permissions granted to the `GITHUB_TOKEN` for `pull_request_target` jobs. GitHub's default `GITHUB_TOKEN` permission model historically granted write access to most repository scopes unless explicitly restricted — a default that GitHub began tightening in 2023 but which many repositories inherited before the change [11]. Restricting the token to the minimum required permissions — typically `pull-requests: write` for labeling or commenting workflows — eliminates the workflow injection stage of the prt-scan attack chain entirely [11].

Any repository that received a pull request from an account using the `prt-scan-` branch prefix, particularly between March 11 and April 10, 2026, should treat its repository secrets and any cloud credentials stored as CI secrets as potentially compromised. Rotation of affected credentials is warranted regardless of whether the pull request was merged, because the `pull_request_target` trigger fires on PR creation, not on merge.

Short-Term Mitigations

A highly effective architectural mitigation, and the one recommended by GitHub Security Lab [6][11], is to separate workflows that require secrets from workflows that process untrusted pull request content. CSA recommends a two-workflow pattern: a first workflow triggered by `pull_request` that builds and tests code from the fork with no secrets access, followed by a second workflow triggered by

`workflow_run` that consumes the artifacts from the first run and then performs privileged operations (such as posting results or deploying to staging). This pattern prevents attacker-controlled code from ever executing in a context where secrets are available [6][11].

For repositories that must use `pull_request_target` for functional reasons, the required-approval-before-first-run GitHub feature — available for repositories that have not previously received pull requests from a given contributor — provides a meaningful gating control. However, the prt-scan campaign's auto-label bypass demonstrates that label-based gates alone are insufficient if the workflow token carries sufficient permissions to create and apply labels [1][3].

Organizations should implement monitoring for anomalous GitHub API activity patterns, including rapid PR submissions from newly created accounts, PR branches prefixed with scanning tool names, and workflow runs that make unexpected GitHub API calls during execution. GitHub's audit log API and third-party CI/CD security platforms can provide the visibility needed to detect these patterns in near real time [1].

Strategic Considerations

The evolution from hackerbot-claw to prt-scan in under two months suggests that organizations that do not treat CI/CD security as an ongoing operational discipline face an expanding attack surface as tooling matures [1][4]. The progression across these two campaigns demonstrates that the threat actor community is actively iterating on this attack class: expanding targeting scope, improving payload obfuscation, and incorporating AI-assisted payload generation to lower the skill threshold required for execution [1][2][4].

Longer term, the structural vulnerability enabling this attack class — the existence of a workflow trigger that runs with elevated privileges against untrusted code — warrants review at the platform level beyond GitHub's December 2025 default-branch sourcing change. The security community has raised proposals in GitHub Community Discussions for features including mandatory first-contribution approval gates, per-job secret scoping, and runtime anomaly detection within GitHub Actions that would reduce the blast radius of any single workflow misconfiguration [5][12]. Organizations should engage their GitHub enterprise account teams to understand the roadmap for these platform-level controls and plan mitigations accordingly.

In organizations that have incorporated AI-augmented software delivery pipelines, the workflows targeted by prt-scan may also govern not only deployment but also model training, dataset preparation, and AI system evaluation. A compromised CI pipeline in an AI-augmented delivery context represents a

broader risk surface than traditional software supply chain attacks, as stolen credentials may provide access to model registries, training data, or inference endpoints that sit outside the conventional software deployment footprint.

CSA Resource Alignment

The prt-scan campaign is directly relevant to several CSA frameworks and active research areas.

MAESTRO (Multi-Agent Environment, Security, Threat, Risk, and Outcome) provides the most directly applicable threat modeling lens for this campaign. MAESTRO's Layer 3, Deployment Infrastructure, explicitly addresses the CI/CD pipeline as a security-critical component of agentic AI systems — the same pipeline layer that prt-scan targeted. As organizations integrate LLM-assisted code generation and AI-driven DevOps automation into GitHub Actions workflows, CI/CD pipelines become not merely software delivery infrastructure but the control plane for AI system behavior. MAESTRO's February 2026 applied threat modeling work on CI/CD pipelines identified `pull_request_target` misconfiguration as a concrete threat path within the agentic deployment surface [13].

CSA AI Controls Matrix (AICM) addresses supply chain controls for AI systems, including requirements for integrity verification of pipeline components and access controls governing model and artifact deployment. The workflow injection phase of prt-scan — in which stolen tokens were used to create new workflow files on the default branch — represents exactly the class of unauthorized pipeline modification that AICM supply chain controls are designed to prevent [14].

CSA Cloud Controls Matrix (CCM) v4.1 provides applicable controls within its Supply Chain Management, Transparency, and Accountability (STA) domain. The STA control family's requirements for supplier trust, software bill of materials transparency, and pipeline integrity verification extend naturally to CI/CD infrastructure. CCM's Application Security domain additionally addresses the security of software development workflows, including requirements for access control and secret management in automated build and test systems [15].

CSA Zero Trust guidance is relevant to the token permission architecture exposed by prt-scan. The attack chain's workflow injection and auto-label bypass stages both depended on `GITHUB_TOKEN` permissions that exceeded the minimum needed for the workflow's legitimate function — a direct violation of least-privilege principles that underpin Zero Trust architecture. Applying Zero Trust token scoping to CI/CD workflows would have broken the attack chain at stage three.

References

[1] Wiz Research. "prt-scan: AI-Powered GitHub Actions Supply Chain Attack." Wiz Blog, April 2026.

[2] Dark Reading. "AI-Assisted Supply Chain Attack Targets GitHub." Dark Reading, April 2026.

[3] SafeDep. "prt-scan: A 5-Phase GitHub Actions Credential Theft Campaign." SafeDep Security Blog, April 2026.

[4] Orca Security. "HackerBot-Claw: An AI-Assisted Campaign Targeting GitHub Actions Pipelines." Orca Security Blog, March 2026.

[5] GitHub. "Actions pull_request_target and environment branch protections changes." GitHub Changelog, November 7, 2025.

[6] GitHub Security Lab. "Keeping your GitHub Actions and workflows secure Part 1: Preventing pwn requests." GitHub Security Lab, 2021.

[7] Orca Security. "pull_request_nightmare Part 1: Exploiting GitHub Actions for RCE and Supply Chain Attacks." Orca Security Blog, 2024.

[8] Sysdig. "Dangerous by default: Insecure GitHub Actions found in MITRE, Splunk, and other open source repositories." Sysdig Blog, 2024.

[9] JFrog Security Research. "pull_request_target Exploitation - Part 1." JFrog Security Research Blog, 2024.

[10] StepSecurity. "hackerbot-claw: An AI-Powered Bot Actively Exploiting GitHub Actions." StepSecurity Blog, March 2026.

[11] GitHub. "Security hardening for GitHub Actions." GitHub Documentation, 2025.

[12] GitHub Community. "Towards a secure by default GitHub Actions." GitHub Community Discussions, 2025.

[13] Cloud Security Alliance. "Applying MAESTRO to Real-World Agentic AI Threats: From Framework to CI/CD Pipeline." CSA Blog, February 11, 2026.

[14] Cloud Security Alliance. "Introducing the CSA AI Controls Matrix." CSA Blog, July 2025.

[15] Cloud Security Alliance. "Cloud Controls Matrix and CAIQ v4.1." CSA Research, 2026.