INTELLIGENCE BRIEFING
Security Command Center
TLP:CLEAR
2026-06-03 19:10 UTC

GitHub OAuth Tokens at Risk: VS Code Webview Flaw Enables Silent One-Click Exfiltration

CVE VULNERABILITY | HIGH | CVSS 7.5

SCC Item ID	SCC-CVE-2026-0262
Type	CVE Vulnerability
Severity	HIGH
CVSS Base Score	7.5
Affected Products	Microsoft Visual Studio Code (VS Code), GitHub.dev, GitHub OAuth token handling, no specific version pinned in available disclosure
Published	2026-06-03T08:58:22
Discovery Source	Rss

Executive Summary

A disclosed but unpatched vulnerability in GitHub.dev allows an attacker to steal a developer's GitHub OAuth token with a single user click, requiring no malware installation and leaving no visible trace. The stolen token carries full read/write access to every repository the victim can reach, including private codebases, secrets stored in code, and CI/CD pipeline configurations. Microsoft has acknowledged the issue via security advisories dated June 2026 and states a fix is in progress with an estimated 72-hour timeline; no patch is available as of disclosure.

Technical Analysis

The attack exploits three chained weaknesses in VS Code's webview architecture. First, VS Code's webview message-passing mechanism processes cross-origin postMessage events without sufficient origin validation (CWE-345), allowing a malicious or compromised webview to inject instructions into the trusted VS Code host process. Second, the attacker uses synthetic keypress injection to simulate authenticated user actions within the VS Code environment, triggering OAuth token issuance or transmission without user awareness (CWE-284). Third, the extension trust model can be bypassed to execute the attack in a privileged workspace context (CWE-269). CWE-79 (XSS) is applicable as a potential amplification or delivery vector for initial webview injection but is not a core component of the exploit chain. The resulting stolen OAuth token carries full read/write scope across all repositories accessible to the victim, not scoped to the originating repository. No patch exists as of disclosure. Microsoft acknowledged the report via security advisories (June 2026) and has indicated a fix is targeted for completion within 72 hours of initial disclosure. MITRE techniques: T1528 (Steal Application Access Token), T1552.001 (Credentials in Files), T1059.007 (JavaScript execution), T1204.001 (User Interaction -

One-Click), T1566.002 (Phishing - Spearphishing Link). Coverage corroborated by BleepingComputer and The Hacker News.

Action Checklist

1. Step 1: Containment, Immediately audit all GitHub OAuth tokens issued to developer accounts that use VS Code or GitHub.dev. Revoke tokens for any account where exposure cannot be ruled out. In GitHub: Settings > Developer settings > Personal access tokens. For organizations, audit via the GitHub org admin panel under Security > OAuth application access. Treat all active GitHub OAuth tokens on affected developer workstations as potentially compromised until a patch is confirmed deployed.
2. Step 2: Detection, Query your SIEM or GitHub audit log API for anomalous OAuth token usage: unexpected repository clones, cross-repo access patterns, access from unfamiliar IP addresses or user agents, or token activity outside business hours. GitHub audit log event types to monitor: 'oauth_access.create', 'repo.download', and 'git.clone' from unfamiliar sources. Check VS Code extension activity logs for unrecognized extensions with webview permissions. NIST AU-6 requires review of audit records for anomalous activity; CIS 8.2 requires audit log collection to be enabled across enterprise assets - verify both are in place for developer systems.
3. Step 3: Eradication, No vendor patch is available as of disclosure. Apply the following interim mitigations: (1) Disable GitHub.dev access at the network or browser policy level for developer workstations where not operationally required; (2) Remove or disable VS Code extensions that use webview APIs and are not from verified, trusted publishers, audit via 'code --list-extensions' and cross-reference against your approved extension inventory (CIS 2.1, CIS 2.3); (3) Enforce extension allowlisting via VS Code's 'extensions.allowedExtensionIDs' or equivalent MDM policy (NIST CM-7, least functionality); (4) Rotate all GitHub OAuth tokens for affected developers regardless of confirmed exploitation.
4. Step 4: Recovery, After token rotation, verify that revoked tokens no longer authenticate against the GitHub API. Re-issue tokens with the minimum required scopes (NIST AC-6, least privilege; CIS 3.3). Confirm no unauthorized commits, branch modifications, secret exposure, or pipeline changes occurred during the exposure window by reviewing GitHub repository audit logs and commit history. Monitor D3-CRO (Credential Rotation) and D3-UAP (User Account Permissions) controls for sustained enforcement. Re-enable GitHub.dev access only after Microsoft confirms patch availability and deployment is verified.
5. Step 5: Post-Incident, This incident exposes a control gap in developer workstation trust: VS Code extensions and webview-based environments are treated as implicitly trusted, without per-origin message validation or scoped token issuance. Remediation actions: (a) Formalize an extension vetting process aligned with CIS 2.1 and CIS 2.3, approved extension inventory with documented review cadence; (b) Enforce MFA on GitHub accounts (CIS 6.3, CIS 6.5; NIST IA-2) to limit the value of stolen tokens as a sole authentication factor where GitHub supports hardware key or app-based second factors; (c) Apply D3-CH (Credential Hardening) and D3-MFA (Multi-factor Authentication) countermeasures to developer identity pipelines; (d) Review CI/CD pipeline permissions to ensure OAuth tokens used in automation carry only required scopes, limiting blast radius of future token compromise.

IR / Forensic Enrichment

Triage Priority	IMMEDIATE
Escalation Criteria	Escalate to CISO and legal/privacy counsel immediately if GitHub audit logs confirm any stolen OAuth token was used to access repositories containing PII, PHI, payment card data, proprietary source code, or CI/CD pipeline secrets, as this transitions from a credential exposure event to a potential data breach requiring regulatory notification assessment under GDPR, CCPA, or applicable sector-specific frameworks.
Recovery Notes	After token rotation and scope reduction, maintain active monitoring of GitHub org audit logs via the API for a minimum of 30 days post-incident, specifically watching for 'oauth_access.create' events that could indicate re-exploitation if the VS Code webview flaw remains unpatched and developer access to github.dev is prematurely restored. Verify that all GitHub Actions workflows using previously compromised developer tokens have been re-keyed to newly issued, scope-restricted tokens and that no pipeline runs executed during the exposure window introduced unauthorized dependencies, modified build artifacts, or exfiltrated repository secrets via external HTTP calls logged in Actions run logs. Re-enable github.dev access only upon confirmed Microsoft patch deployment and validate by reviewing VS Code release notes and the associated security advisory for the specific postMessage origin-validation fix.
Forensic Artifacts	GitHub Org Audit Log API (GET /orgs/{org}/audit-log) — JSON export filtered to 'oauth_access.create', 'git.clone', 'repo.download', 'repo.push', and 'workflow_run.created' events during the exposure window, which would capture the initial stolen-token authentication and any subsequent read/write operations the attacker performed using the silently exfiltrated GitHub OAuth token. | VS Code Extension Host Logs at %APPDATA%\Code\logs\ (Windows) or ~/.config/Code/logs/ (macOS/Linux) — these logs record webview instantiation, inter-extension postMessage events, and extension activation sequences that the VS Code webview vulnerability exploitation would traverse during the single-click token exfiltration trigger. | VS Code GitHub Authentication Extension Storage at %APPDATA%\Code\User\globalStorage\github.vscode-pull-request-github\ (Windows) or ~/.config/Code/User/globalStorage/github.vscode-pull-request-github/ (macOS/Linux) — contains cached OAuth session state, token metadata, and authentication records that may retain evidence of the token present at time of exploitation. | Browser Network Traffic Logs or Enterprise Proxy Logs for github.dev — capture HTTP requests and responses to github.dev including any anomalous cross-origin postMessage-triggered outbound requests, since the token exfiltration mechanism operates through the browser-hosted github.dev webview environment and would produce a distinct outbound HTTP request carrying the stolen OAuth token to an attacker-controlled endpoint. | GitHub Actions Workflow Run Logs for all repositories accessible to compromised developer accounts — specifically the raw log output of any workflow runs executed during the exposure window, which would reveal whether the stolen token's write access was used to inject malicious steps, exfiltrate GITHUB_TOKEN or repository secrets via curl/wget to external endpoints, or modify protected branch rules.

Per-Action IR Details

Step 1: Containment — Immediately audit all GitHub OAuth tokens issued to developer accounts that use VS Code or GitHub.dev. Revoke tokens for any account where exposure cannot be ruled out. In GitHub: Settings > Developer settings > Personal access tokens. For organizations, audit via the GitHub org admin panel under Security > OAuth application access. Treat all active GitHub OAuth tokens on affected developer workstations as potentially compromised until a patch is confirmed deployed.

NIST Phase: Containment

Reference: NIST 800-61r3 §3.3 — Containment Strategy

Controls: NIST IR-4 (Incident Handling), NIST AC-2 (Account Management), NIST AC-6 (Least Privilege), CIS 5.1 (Establish and Maintain an Inventory of Accounts), CIS 6.2 (Establish an Access Revoking Process)

Compensating: Use the GitHub CLI to enumerate all active PATs and OAuth app authorizations without a SIEM: run 'gh auth status' on each developer workstation to confirm active token identity, then use the GitHub REST API endpoint GET /orgs/{org}/credential-authorizations (requires org:admin scope) scripted via curl or PowerShell Invoke-RestMethod to enumerate all OAuth tokens at the org level. A two-person team can assign one analyst to workstation enumeration and one to API token audit simultaneously. Revoke via DELETE /orgs/{org}/credential-authorizations/{credential_id}.

Evidence: Before revoking tokens, capture the full OAuth token authorization list from GitHub Settings > Developer settings > Personal access tokens including token name, creation date, last-used date, and granted scopes. For org-level audit, export the GitHub org audit log covering the 30-day window prior to disclosure via GET /orgs/{org}/audit-log?include=all&per_page=100 and preserve the raw JSON. Screenshot or export the GitHub.dev browser session history and any active VS Code authentication state from %APPDATA%\Code\User\globalStorage\github.vscode-pull-request-github\ (Windows) or ~/.config/Code/User/globalStorage/github.vscode-pull-request-github/ (Linux/macOS) before any tokens are rotated, as these directories may contain cached token state or session identifiers written by VS Code's GitHub authentication extension.

Step 2: Detection — Query your SIEM or GitHub audit log API for anomalous OAuth token usage: unexpected repository clones, cross-repo access patterns, access from unfamiliar IP addresses or user agents, or token activity outside business hours. GitHub audit log event types to monitor: 'oauth_access.create', 'repo.download', and 'git.clone' from unfamiliar sources. Check VS Code extension activity logs for unrecognized extensions with webview permissions. NIST AU-6 requires review of audit records for anomalous activity; CIS 8.2 requires audit log collection to be enabled across enterprise assets — verify both are in place for developer systems.

NIST Phase: Detection Analysis

Reference: NIST 800-61r3 §3.2 — Detection and Analysis

Controls: NIST AU-6 (Audit Record Review, Analysis, and Reporting), NIST AU-2 (Event Logging), NIST AU-3 (Content of Audit Records), CIS 8.2 (Collect Audit Logs), CIS 7.1 (Establish and Maintain a Vulnerability Management Process)

Compensating: Without a SIEM, query the GitHub audit log API directly using a paginated curl loop: 'curl -H "Authorization: Bearer {admin_token}" "https://api.github.com/orgs/{org}/audit-log?include=all&phrase=action:oauth_access" | jq .' and pipe through jq filters for actor_ip, user_agent, and @timestamp fields. For developer workstations, deploy Sysmon with Swift On Security's config and filter for Event ID 4688 (Process Creation) where ParentImage contains 'Code.exe' and CommandLine contains 'git clone' or 'curl' targeting github.com, which would indicate a webview-injected process spawn. Review VS Code extension host logs at %APPDATA%\Code\logs\ (Windows) or ~/.config/Code/logs/ (Linux/macOS) for entries referencing postMessage, webview, or iframe activity from non-Microsoft extension IDs.

Evidence: Preserve the GitHub org audit log JSON export filtered to 'oauth_access.create', 'git.clone', and 'repo.download' events for the 30 days preceding disclosure — the token theft via VS Code webview postMessage injection would manifest as OAuth token creation or first-use events originating from an IP or user-agent inconsistent with the developer's normal GitHub Desktop or CLI activity. Capture VS Code extension log files from %APPDATA%\Code\logs\ (Windows) or ~/.config/Code/logs/ (macOS/Linux) which record extension activation, webview instantiation, and inter-process messaging events that the malicious webview exploitation would traverse. Export browser developer console logs from GitHub.dev sessions if available via browser history or enterprise proxy logs, filtering on postMessage events or unexpected cross-origin communications to identify the single-click trigger event.

Step 3: Eradication — No vendor patch is available as of disclosure. Apply the following interim mitigations: (1) Disable GitHub.dev access at the network or browser policy level for developer workstations where not operationally required; (2) Remove or disable VS Code extensions that use webview APIs and are not from

verified, trusted publishers — audit via 'code --list-extensions' and cross-reference against your approved extension inventory (CIS 2.1, CIS 2.3); (3) Enforce extension allowlisting via VS Code's 'extensions.allowedExtensionIDs' or equivalent MDM policy (NIST CM-7, least functionality); (4) Rotate all GitHub OAuth tokens for affected developers regardless of confirmed exploitation.

NIST Phase: Eradication

Reference: NIST 800-61r3 §3.4 — Eradication

Controls: NIST CM-7 (Least Functionality), NIST SI-2 (Flaw Remediation), NIST AC-6 (Least Privilege), CIS 2.1 (Establish and Maintain a Software Inventory), CIS 2.3 (Address Unauthorized Software), CIS 4.6 (Securely Manage Enterprise Assets and Software)

Compensating: Run 'code --list-extensions --show-versions' on all developer workstations and diff the output against your approved extension inventory. For extensions with webview API usage, cross-reference against the VS Code Marketplace API: GET https://marketplace.visualstudio.com/items?itemName={publisher}.{extension} to check publisher verification status. Block github.dev at the DNS or proxy layer using Pi-hole, Windows Firewall GPO, or /etc/hosts entries pushed via a simple bash or PowerShell script across developer machines — this directly removes the single-click attack surface while the patch is pending. Set 'extensions.allowedExtensionIDs' in VS Code's settings.json deployed via policy to restrict the webview attack surface to only allowlisted extension IDs.

Evidence: Before disabling or removing suspicious VS Code extensions, preserve the extension installation directory — %USERPROFILE%\.vscode\extensions\ (Windows) or ~/.vscode/extensions/ (macOS/Linux) — as a forensic snapshot, since a malicious or compromised extension exploiting the webview postMessage vulnerability would reside here with its package.json declaring webview activation events. Capture the output of 'code --list-extensions --show-versions' before any removals as a timestamped baseline. Preserve VS Code workspace and user settings files (settings.json, keybindings.json) from %APPDATA%\Code\User\ (Windows) or ~/.config/Code/User/ (macOS/Linux) which may reflect unauthorized configuration changes introduced by a webview-exploiting extension.

Step 4: Recovery — After token rotation, verify that revoked tokens no longer authenticate against the GitHub API. Re-issue tokens with the minimum required scopes (NIST AC-6, least privilege; CIS 3.3). Confirm no unauthorized commits, branch modifications, secret exposure, or pipeline changes occurred during the exposure window by reviewing GitHub repository audit logs and commit history. Monitor D3-CRO (Credential Rotation) and D3-UAP (User Account Permissions) controls for sustained enforcement. Re-enable GitHub.dev access only after Microsoft confirms patch availability and deployment is verified.

NIST Phase: Recovery

Reference: NIST 800-61r3 §3.5 — Recovery

Controls: NIST AC-2 (Account Management), NIST AC-6 (Least Privilege), NIST IR-4 (Incident Handling), CIS 3.3 (Configure Data Access Control Lists), CIS 6.1 (Establish an Access Granting Process), CIS 6.2 (Establish an Access Revoking Process)

Compensating: Verify token revocation without enterprise tooling by running: 'curl -H "Authorization: token {old_token}" https://api.github.com/user' — a 401 response confirms revocation. For commit and branch integrity verification, run 'git log --all --oneline --author-date-is-committer-date' against each critical repository and filter commits from the exposure window, cross-referencing committer email and GPG signing status. Use 'git verify-commit {hash}' to check for unsigned commits that should have been signed under your policy, which would indicate commits made with the stolen OAuth token rather than the developer's local key. For CI/CD pipeline integrity, audit GitHub Actions workflow files (.github/workflows/) via 'git diff {date_before_exposure}..HEAD -- .github/workflows/' to detect unauthorized pipeline modifications made using the stolen token's write access.

Evidence: Before re-enabling github.dev access, capture a full export of GitHub repository audit events covering the exposure window for every repository accessible by the affected developer accounts — specifically filtering for 'repo.push', 'protected_branch.update', 'secret_scanning_alert.dismissed', and 'workflow_run.created' events, which would represent malicious use of the stolen OAuth token's read/write scope against private codebases and CI/CD pipelines. Preserve a snapshot of GitHub Actions secrets configuration and environment variable listings accessible to the compromised token scope, as exfiltration of CI/CD secrets via the stolen token's API access would leave no commit trail but would appear in audit log API calls to /repos/{owner}/{repo}/actions/secrets.

Step 5: Post-Incident — This incident exposes a control gap in developer workstation trust: VS Code extensions and webview-based environments are treated as implicitly trusted, without per-origin message validation or scoped token issuance. Remediation actions: (a) Formalize an extension vetting process aligned with CIS 2.1 and CIS 2.3 — approved extension inventory with documented review cadence; (b) Enforce MFA on GitHub accounts (CIS 6.3, CIS 6.5; NIST IA-2) to limit the value of stolen tokens as a sole authentication factor where GitHub supports hardware key or app-based second factors; (c) Apply D3-CH (Credential Hardening) and D3-MFA (Multi-factor Authentication) countermeasures to developer identity pipelines; (d) Review CI/CD pipeline permissions to ensure OAuth tokens used in automation carry only required scopes, limiting blast radius of future token compromise.

NIST Phase: Post Incident

Reference: NIST 800-61r3 §4 — Post-Incident Activity

Controls: NIST IA-2 (Identification and Authentication — Organizational Users), NIST CM-7 (Least Functionality), NIST IR-4 (Incident Handling), NIST AU-6 (Audit Record Review, Analysis, and Reporting), CIS 2.1 (Establish and Maintain a Software Inventory), CIS 2.3 (Address Unauthorized Software), CIS 6.3 (Require MFA for Externally-Exposed Applications), CIS 6.5 (Require MFA for Administrative Access), CIS 7.1 (Establish and Maintain a Vulnerability Management Process), CIS 7.2 (Establish and Maintain a Remediation Process)

Compensating: Formalize the VS Code extension vetting process using a lightweight inventory: maintain a CSV or Git-tracked JSON file listing approved extension IDs, publisher verification status, webview API usage (detectable from each extension's package.json 'contributes.views' and 'enabledApiProposals' fields), and last-review date. For CI/CD token scoping without an enterprise secrets manager, audit GitHub Actions workflow YAML files for hardcoded or over-scoped GITHUB_TOKEN permissions blocks — set 'permissions: read-all' or granular per-job permissions in each workflow file as a free, immediate blast-radius reduction. Enable GitHub's built-in token expiration (fine-grained PATs support expiration dates) and document a 90-day maximum rotation policy enforced via a shared calendar or GitHub-native reminder.

Evidence: For the lessons-learned record, preserve the final GitHub audit log export covering the full incident window, the timestamped extension inventory snapshots taken during eradication, and the before/after token scope comparison showing the reduction in OAuth permissions applied during recovery — these collectively document the control gap (implicit webview trust, over-scoped tokens) and the specific remediation applied to VS Code and GitHub.dev developer environments affected by this postMessage-based OAuth token exfiltration vulnerability.

Detection Guidance

Primary detection surface is the GitHub audit log. Query for these events via the GitHub Audit Log API or your SIEM integration: 'oauth_access.create' (new token issuance), 'git.clone' and 'repo.download' from IP addresses not matching the developer's known geolocation or corporate egress range, repository access events spanning multiple repos in a short window (lateral spread pattern), and any API calls using a token after that token was issued during a GitHub.dev session. Secondary detection: on developer endpoints, monitor VS Code extension installs and webview process activity. Look for extensions spawning network connections or accessing the VS Code secret storage API (keytar). If endpoint detection tooling is present, alert on VS Code renderer processes making outbound HTTP calls to non-Microsoft, non-GitHub domains. Behavioral indicator: a developer reports no knowledge of a commit, clone, or repository access event that appears under their identity in GitHub logs. No public IOCs (hashes, domains, IPs) have been confirmed as of disclosure. D3-LAM (Local Account Monitoring) and D3-SFA (System File Analysis) apply for endpoint-level hunting. NIST AU-6 and CIS 8.2 provide the control baseline for audit log review and collection requirements.

Framework Mappings

MITRE-ATTACK

- **T1528** — Steal Application Access Token
- **T1195.002** — Compromise Software Supply Chain
- **T1552.001** — Credentials In Files
- **T1059.007** — JavaScript
- **T1204.001** — Malicious Link
- **T1195.001** — Compromise Software Dependencies and Development Tools
- **T1059** — Command and Scripting Interpreter
- **T1566.002** — Spearphishing Link

NIST-800-53R5

- **CM-7** — Least Functionality
- **SA-9** — External System Services
- **SR-3** — Supply Chain Controls and Processes
- **SI-7** — Software, Firmware, and Information Integrity
- **SI-3** — Malicious Code Protection
- **SI-4** — System Monitoring
- **AT-2** — Literacy Training and Awareness
- **SC-7** — Boundary Protection
- **SI-8** — Spam Protection
- **AC-3** — Access Enforcement
- **SI-10** — Information Input Validation
- **AC-6** — Least Privilege

OWASP-TOP10-2021

- **A01:2021** — Broken Access Control
- **A08:2021** — Software and Data Integrity Failures
- **A03:2021** — Injection

CIS-V8

- **6.1** — Establish an Access Granting Process
- **6.2** — Establish an Access Revoking Process
- **2.5** — Allowlist Authorized Software
- **16.10** — Apply Secure Design Principles in Application Architectures
- **5.4** — Restrict Administrator Privileges to Dedicated Administrator Accounts
- **6.8** — Define and Maintain Role-Based Access Control
- **7.3** — Perform Automated Operating System Patch Management
- **7.4** — Perform Automated Application Patch Management

SOC2-TSC

- **CC6.1** — The entity implements logical access security software, infrastructure, and architectures over protected information assets

HIPAA-SECURITY

- **164.312(a)(1)** — Access Control

ISO-27001-2022

- **A.8.28** — Secure coding
- **A.8.8** — Management of technical vulnerabilities

MITRE ATT&CK Mapping

Technique ID	Technique Name	Tactic
T1528	Steal Application Access Token	Credential-Access
T1195.002	Compromise Software Supply Chain	Initial-Access
T1552.001	Credentials In Files	Credential-Access
T1059.007	JavaScript	Execution
T1204.001	Malicious Link	Execution
T1195.001	Compromise Software Dependencies and Development Tools	Initial-Access
T1059	Command and Scripting Interpreter	Execution
T1566.002	Spearphishing Link	Initial-Access

Sources

Source	URL	Tier
Security News	https://thehackernews.com/2026/06/one-click-github-dev-attack-lets....	T3
VS Code zero-day lets hackers steal GitHub tokens in one click	https://www.bleepingcomputer.com/news/security/vs-code-zero-day-let...	T3
1-Click GitHub Token Vulnerability Lets Attackers Steal Users' OAuth ...	https://x.com/The_Cyber_News/status/2061994742632829176	T3
GitHub hit by a compromised VSCode extension : r/netsec - Reddit	https://www.reddit.com/r/netsec/comments/1tiiyxq/github_hit_by_a_co...	T3

Source	URL	Tier
The Wild West of VS Code extensions and how a poisoned ...	https://www.aikido.dev/blog/vs-code-extension-github-breach	T3

Generated 2026-06-03 19:10 UTC by TJS Security Command Center