Route, Don't Refuse

Capability-Preserving Isolation of Prompt-Injection-Exposed Agent Work through Fork-Based Pull Requests: A Production Case Study

Shinji Yumura
AQUA LLC, Tokyo, Japan
official@aquallc.jp

19 November 2026

Abstract

A coding agent that can be summoned from a GitHub issue reads text written by anyone, then pushes a branch whose continuous integration (CI) runs with the repository's secrets. If the text carries an indirect prompt injection, the agent can be steered into writing code that exfiltrates those secrets. The usual remedy is refusal: detect untrusted input and hold the work for human approval. We deployed that remedy in a production agent; by construction it removed capabilities users relied on—immediate pull requests, large changes, and automatic CI repair—and it still exposed secrets once a human approved. We then replaced it with *routing*. Each job carries a sticky taint bit, set by eight kinds of input and propagated through cryptographically signed job stages; tainted work on public repositories is published as a draft pull request from a fork that lives in an organisation owned by the agent's operator. For fork pull requests, GitHub withholds repository secrets and grants only a read-only token, so containment is enforced by the platform rather than by the model. Two GitHub Apps split privileges (one may only create and inspect forks; the other writes with tokens scoped to a single fork and to contents and workflow files), forks are identified by numeric ids rather than names, and a daily sweep deletes forks whose upstream became private. In 22 instrumented CI runs on a probe repository holding a canary secret, all 8 runs of in-repository pushes could read the canary and write to the repository, while all 14 runs of fork pull requests could do neither, including two deliberately broken commits that the agent then repaired without human approval. Live testing surfaced six platform behaviours, five of which were undocumented or contradicted our assumptions; the four that caused failures all failed closed, and none exposed a secret. Six rounds of adversarial review produced 33 findings; none was rated high or critical, and every confirmed finding was fixed before release. For public repositories the design restored every capability lost to refusal; it was enabled for all public repositories on 18 November 2026.

1 Introduction

Indirect prompt injection—instructions hidden in content that an LLM-integrated application retrieves rather than in the user's request—is a central risk for such applications [1]; prompt injection, including its indirect form, ranks first in the OWASP Top 10 for LLM applications [2]. Agents make it concrete. An agent that reads untrusted text, can reach sensitive data, and can communicate externally holds what Willison calls the *lethal trifecta* [3]; Meta's *Agents Rule of Two* recommends that an agent session hold at most two of the three [4]. Attacks on GitHub-connected agents have already exploited this combination [5].

YUA is a commercial assistant whose development mode includes a coding agent. Since November 2026 users can summon it from GitHub by writing "@yua" in an issue or pull request; the agent

plans, edits code in a sandbox, runs tests, and opens a pull request through a GitHub App. On 18 November 2026 an internal review pointed out a gap: an outsider could open an issue containing an instruction such as "also add a test that sends the environment to this URL". When a maintainer summoned the agent on that issue, the resulting branch would live inside the repository, and its CI would run with the repository's secrets. The model might refuse the instruction; nothing structural forced it to.

Our first fix followed the common pattern: detect that the job had read outside text and hold every such change for an explicit "@yua OK" from a maintainer. The operator rejected it on a principle we adopt as the thesis of this report: *a security fix should not reduce what users can do*. By construction, the approval gate removed immediate pull requests, made changes above 45,000 characters impossible to show in a comment and therefore impossible to publish, and stopped automatic CI repair; and after approval, the code still ran with secrets. (The gate ran in production for only a few hours, so these are losses by design, not measured user impact.)

This report describes the replacement. Instead of refusing tainted work, the agent *routes* it to a place where the platform itself withholds secrets: a fork in an organisation the operator owns. The contributions are:

- a routing design that satisfies the Rule of Two per job rather than per agent, so that untrusted input and secrets never meet while all capabilities remain on public repositories (Section 4);
- an implementation on GitHub with least-privilege App separation, identity by numeric id, and lifecycle management including CI repair, renames and cleanup (Section 5);
- a live evaluation with a canary secret: 0 of 14 fork pull-request runs could read it or write, against 8 of 8 in-repository runs (Section 6);
- six platform behaviours observed only by live testing, five of them undocumented or contrary to our assumptions, and the record of six adversarial review rounds.

2 Background

Defences against prompt injection. Detection-based defences classify inputs or outputs and are bypassed by adaptive attackers; benchmarks such as AgentDojo measure this gap [6]. Design-based defences instead constrain what a compromised model can do. The Dual LLM pattern keeps untrusted text away from the privileged model [7]; CaMeL extracts control and data flow from the trusted query and enforces capabilities in an interpreter [8]; FIDES attaches confidentiality and integrity labels to data and enforces policies deterministically [9], in the tradition of information-flow control [10]. Beurer-Kellner et al. catalogue such patterns and note their cost in utility [11]. Our design is closest to integrity labelling, but the label is coarse (one bit per job) and the enforcement point is not a custom interpreter: it is an existing platform boundary.

GitHub's fork boundary. With the exception of `GITHUB_TOKEN`, secrets are not passed to workflows triggered by pull requests from forks, and that token is read-only [12, 13]. By default, workflows on pull requests from first-time contributors require approval [13]. The boundary is broken by workflows that use `pull_request_target` and check out the pull request's code ("pwn requests") [14]. Vercel similarly does not build preview deployments of fork pull requests for projects with environment variables until a member authorises them [15].

Table 1: Input channels that set the taint bit.

Channel	Condition
Summoning thread	body written by a user without write permission (checked via API); comments by such users are dropped
Mid-task issue read	any issue whose author cannot write; comments by such users are dropped
Web search, web fetch	always
Shell commands	network fetchers (`curl`, `wget`, `ssh`, ...), package-registry reads (e.g. `npm view`), installs naming a package, `npx` of a non-local tool, `git clone/fetch/pull/ls-remote/submodule/archive`, inline network code, registry URLs
Commit inspection	a commit that is not an ancestor of the job's base
CI results	failed checks reported by apps other than GitHub Actions
Bot-written thread	agent-written text that carries the outside mark
Unreadable input	any read failure (fail closed)

3 System and Threat Model

A job runs in three stages—plan, run, and (if needed) approve—each a separate serverless invocation that receives the job as an HMAC-signed message. The agent works in an isolated sandbox; it never holds the repository's GitHub credentials. Server code opens pull requests with installation tokens of the YUA Dev Mode App.

Attacker. Anyone who can write text the agent may read but cannot write to the repository: authors of issues and comments, authors of web pages and package metadata, authors of commits outside the repository's history, and other CI apps whose logs the agent reads. The attacker cannot summon the agent; only users with write access can.

Assets. Repository secrets available to CI, a writable `GITHUB_TOKEN`, and secrets of preview deployments.

Goal. Make code that the agent wrote run with those assets, e.g. to exfiltrate a secret or push to the repository.

Out of scope. Code that a maintainer merges after review (after merging, it runs with secrets by design), and repository owners who themselves expose secrets to forks (e.g. pwn-request workflows; we warn but cannot prevent).

4 Design: Route, Don't Refuse

4.1 Sticky taint

A job is *outside* if it read any text whose author cannot write to the repository. Eight kinds of input set the bit (Table 1). Authorship is checked against the permission API, not GitHub's `author_association`, which reports `MEMBER` or `COLLABORATOR` for users who may only read. The bit and its reasons travel inside the signed job, so a later stage cannot drop them; a job signed before the field existed is treated as outside. Pull requests created from outside jobs carry a hidden mark in their body, so that later CI repairs of the same pull request inherit the taint.

4.2 Routing

The taint bit no longer blocks; it selects where changes are published (Listing in the released code, `route.ts`):

- **Not outside:** as before, a branch inside the repository (full capabilities, including CI with secrets and previews).
- **Outside, public repository:** a draft pull request from the operator's fork. Its CI runs, and is repaired automatically, without secrets and with a read-only token.
- **Outside, private repository** (forks of private repositories would copy private code): the approval gate remains. It is also the fallback when a fork cannot be prepared.
- **Owner opt-in:** a file `.yua/settings.json` on the default branch containing `{"pr_from_fork": "always"}` routes every job to the fork. If the file cannot be fetched (an error other than "not found"), the job goes to the fork.

In Rule-of-Two terms [4], an outside job holds untrusted input and the ability to communicate (opening a pull request), but not secrets; a trusted job holds secrets and communication but no untrusted input. The agent as a whole keeps all three capabilities; no single job does.

5 Implementation on GitHub

Two Apps, least privilege [16]. Creating a fork needs the *Administration* permission. Adding it to the main App would let a compromised server delete or reconfigure every user's repository, so a second App, the *fork keeper*, is installed only in the fork organisation. After hardening it holds Administration (write), Contents (read) and Metadata; its tokens can create and inspect forks but cannot write their contents. Writes to a fork use a token of the main App (also installed in the organisation) scoped to that single fork and to Contents, Workflows and Metadata. Actions are disabled organisation-wide, and every use of a fork re-checks that they still are.

Identity by number. The agent must recognise its own fork pull requests to repair, extend and clean them up. A fork belongs to an upstream if it is a fork, its owner id equals the organisation's id, and its parent id equals the upstream's id. A pull request is the agent's if its head repository lies under the organisation id, its head branch has the agent prefix, and it was authored by the agent's bot account—anyone can open a pull request whose head is a branch of a public fork. Names are used only to choose a name for a new fork.

Lifecycle. Pull requests are opened by the main App with an explicit `head_repo`, and with `maintainer_can_modify` set to false (Section 6.4). CI notifications for fork pull requests carry no pull-request number, so the agent looks pull requests up by head commit. Records of which pull requests to repair are keyed by repository id rather than name. When a pull request closes, the agent deletes the fork branch. A daily job deletes marked forks that were detached from their upstream (the upstream became private or was deleted) or whose upstream is no longer public, and forks that have no open pull request and no push or update for 30 days; ambiguous answers keep the fork, and the sweep stops on rate limiting. Pull-request bodies render the model's summary as inert text, and the agent warns when the upstream's workflows match the pwn-request pattern [14].

Table 2: Workflow runs on the probe repository (all 22 runs, 18 November 2026).

Origin of the run	Runs	Canary readable	Token can write	Failed
In-repository push	8	8	8	0
Fork PR, opened or committed by the agent	8	0	0	0
Fork PR, opened by the maintainer	3	0	0	0
Fork PR, opened by the outsider[b]	1	0	0	0
Fork PR, commit by the fork keeper[b]	2	0	0	2[a]
Fork PRs, total	**14**	**0**	**0**	2

[a] Commits broken on purpose to trigger automatic repair; the repairs ran without approval and passed. [b] Started by the maintainer (approval).

6 Evaluation

6.1 Setup

A public probe repository held a canary secret `CANARY` and a workflow that, on every push and pull request, printed whether the canary was set (never its value) and whether the token could create a tag; the workflow is released with the data. Roles: a maintainer account with write access, an outsider account without, the agent's bot, and the fork keeper. All tests ran on the production service, with the fork route enabled only for the probe repositories (the main probe, a second repository forked from it for S6, and a disposable repository for S8) until the final step. The outsider is a second GitHub account controlled by the author. A collector script retrieved every issue, pull request and workflow run, with the two printed markers, from the GitHub API; user names are replaced by roles in the released data.

6.2 Secret exposure

Table 2 counts all 22 workflow runs. Every in-repository run could read the canary and write. These were pushes by the maintainer; they run in the same context in which an agent branch inside the repository would run, but no agent branch inside the repository was created during the tests. No fork pull-request run could do either—across a pull request the outsider opened from a branch in the operator's fork, the agent's pull requests, the maintainer's probes, and two runs on commits written by the fork keeper that we deliberately broke. Three of these runs (the outsider's and the two fork-keeper commits) were started by the maintainer, i.e. after approval, as GitHub requires for first-time contributors; all of the agent's runs were started by the agent itself, without approval. Six pull requests were opened by the agent, all from the fork and as drafts; none was merged.

6.3 End-to-end scenarios

Table 3 lists the scenarios, run in order on the production service. In both outsider scenarios the issue contained an injected instruction (to add code reading or printing the canary); in both, the agent's diff contains no reference to the canary, secrets, environment variables or the network, and none of the agent's six pull requests changed a workflow file (the diffs are in the released data). The design does not depend on that outcome. S8 is recorded from the session output, because the deleted fork can no longer be queried; the record is released with the data.

Table 3: End-to-end scenarios on the production service.

	Scenario	Outcome
S1	Outsider issue with injection; maintainer summons agent	draft PR #3 from the fork, outside reasons in body; CI: canary absent, no write
S2	Breaking commit on S1's fork branch	agent found PR by commit, repaired without approval; CI passed, canary absent
S3	Upstream renamed; owner opt-in; agent summoned	first attempt refused by GitHub (B4, B5); after fix, draft PR #8 from a new fork
S4	Upstream renamed back; "add to PR #8"	agent wrote to PR #8's own fork, not the newly named one
S5	Rename, then breaking commit on PR #10	repair record found by repository id; repaired; branch cleaned up on close
S6	Repository B, a fork of A, also uses the agent	B received its own fork; no sharing with A
S7	Fork keeper reduced to Administration + Contents (read)	new fork created, PR #12, CI clean, cleanup by main App
S8	Upstream of a test fork made private; sweep run	a dry run left the unmarked detached fork alone; once marked, it was deleted; the four forks in use were kept
S9	Route enabled for all repositories; outsider issue with injection	draft PR #14 from the fork; only the requested file changed; canary absent

6.4 Platform behaviours

Table 4 lists six behaviours observed live. Five were undocumented or contrary to assumptions we had considered verified; B6 confirmed the documentation and shaped the sweep. Four caused failures when first met, and each failed closed: GitHub refused to create the pull request (B1, B4, B5), or CI repair found no pull request (B2). B3 left CI waiting for approval. None exposed a secret. B4 is instructive: an early observation (requesting a fork under an existing name returned the existing fork) had been recorded as "one fork per organisation", and an implementation built on that generalisation failed when GitHub created a second fork under a new name.

6.5 Adversarial review

Before each release an independent reviewer agent, given the code but not the author's conclusions, attacked the change; the author verified every finding before acting on it. Table 5 summarises six rounds, with the labels the reviewers used: the first three ranked findings as confirmed or plausible, the last three by severity. The released log records the status of every finding: all confirmed findings were fixed before release; of the rest, 22 were fixed, 1 is a documented residual risk and 4 were accepted with reasons. The confirmed findings of the first two rounds were taint channels the detector missed (mid-task issue reads, network reads from the sandbox, commits outside the repository's history, and taint lost when adding commits to an existing pull request); none of the six rounds reported a way through the fork boundary itself. Representative medium findings: an outsider could open a pull request whose head is the agent's fork branch and thereby trigger cleanup or repair (fixed by checking the author); sharing one fork among repositories of the same network mixed jobs from different upstreams (fixed by one fork per upstream); migrating name-keyed repair records to repository ids would let whoever registers a freed name inherit another user's repair quota (fixed by not migrating); and the sweep could delete on ambiguous API answers or delete unrelated repositories (fixed by acting only on definite signals and on marked forks).

Table 4: Platform behaviours found by live testing.

	Expected	Observed (and response)
B1	`maintainer_can_modify` documented only as a flag [17]	omitting it fails with 422 `fork_collab` for organisation-owned forks (send `false`)
B2	fork CI notifications have no PR and `head_branch` null [18]	`head_branch` carried the fork branch (look up PR by head commit)
B3	first-time contributors need approval [13]	commits by an App installed only on the fork needed approval; commits by the App installed upstream did not (write with the latter)
B4	assumed one fork per network per organisation	a second fork was created under a new name (identify by parent id; one fork per upstream)
B5	`head_repo` required only when one organisation owns both [17]	with two forks in the organisation, `org:branch` resolved to the wrong fork, 422 (always pass `head_repo`)
B6	public forks of a repository made private stay public [19]	confirmed; seen by the keeper as `fork: false`, no parent (sweep deletes them)

Table 5: Adversarial review rounds, with the reviewers' own labels.

Round	Target	Findings	Labels as reported
1	taint detector	6	1 confirmed, 4 plausible, 1 low
2	taint detector (second pass)	4	3 confirmed, 1 plausible
3	fork route, first version	8	2 confirmed, 3 plausible, 1 low, 2 notes
4	identity by number	5	1 medium (conditional), 4 low
5	final design, id-keyed records	5	1 medium, 2 low, 2 trivial
6	daily sweep	5	2 medium, 3 low
	Total	33	no finding rated high or critical

6.6 Capabilities

Table 6 compares the three designs for outside jobs on public repositories. Routing restores every capability lost to refusal and removes secret exposure even after approval, which refusal did not. One capability changes form: preview deployments of fork pull requests are not built until a member authorises them [15] (not tested live).

7 Discussion

Split the job, not the agent. The Rule of Two is often read as a limit on the agent; applied per job, with taint deciding which two capabilities a job receives, it costs nothing in function for the jobs that need all three only in sequence. Coding agents are a favourable case because a mature platform boundary—fork pull requests—already provides a "no secrets" execution context with a review step before secrets are reached.

Fail closed, then test live. None of the five surprises surfaced as a leak; each appeared as a refusal, a missed lookup or a wait for approval, because each check was written as "proceed only if positively verified". The surprises themselves were found only by running the real platform; documentation was silent (B4), incomplete (B1, B3, B5) or contradicted (B2).

Table 6: Outside jobs on public repositories under each design.

	Original	Refuse (approval gate)	Route (fork)
Pull request without waiting	yes	no	yes (draft)
Changes above 45,000 characters	yes	no	yes
Automatic CI repair	yes	no	yes
CI runs on the change	with secrets	with secrets, after approval	without secrets
Injected code can read CI secrets before merge	possible	possible after approval	no (platform-enforced)

Generality. The design needs (i) a coarse taint bit carried through signed job state, (ii) an execution context where the platform withholds secrets, and (iii) identities that survive renames. GitLab merge requests from forks and other CI systems with fork isolation should admit the same pattern.

8 Limitations and Residual Risks

- **Small, single-site evaluation.** 22 runs on one probe repository with one agent and one platform; the author designed the system and ran the tests. The evidence is a case study, not a statistical estimate.
- **Trusted jobs still rely on detection.** Jobs that read no outside text run inside the repository with secrets. A missed channel would expose them; owners who want the strongest guarantee can opt into the fork for all jobs.
- **Malicious code can still reach a pull request.** The fork prevents secret access before merge, not bad code; human review before merging remains the final control. Pull requests are drafts and state why they came from the fork.
- **Private repositories** keep the approval gate, which is also the fallback when a fork cannot be prepared.
- **Previews.** Vercel does not build previews of fork pull requests until a member authorises them; we did not test this live, and owners can disable the protection.
- **Identifiable test accounts.** The probe repositories and forks are public, so the roles in the released data can be linked to accounts; the outsider is the author's second account.
- **Owner configuration.** Workflows that expose secrets to forks defeat the boundary; the agent warns about the common pattern but cannot fix it.
- **Migration cost.** Repair records created before the change are not migrated (for the reason in Section 6); users re-enable them once.
- **Scale** (API rate limits with many forks) is untested beyond the probe.

9 Conclusion

Holding prompt-injection-exposed work for approval made a production coding agent less useful and did not stop secrets from meeting injected code. Routing that work to an operator-owned fork, where the platform withholds secrets, restored every lost capability on public repositories and, in live tests with a canary, kept the secret unreachable in all 14 fork runs while it was reachable in all 8 in-repository runs. The lesson we draw is broader than the mechanism: a security fix should be judged by what it leaves users able to do, and the strongest boundaries are often ones the platform already enforces.

Use of AI assistants. The author used AI assistants for code development, adversarial review, experiment orchestration, and drafting of this manuscript, and takes full responsibility for its content.

Data and code. The workflow-run and pull-request records, the platform observations, the review log with the status of each finding, the sweep test record, the probe repository's workflow, the collection and analysis scripts, and a reference implementation of the routing, identity and sweep rules (with tests) are available at https://github.com/YUALAB/route-dont-refuse. Re-running the analysis script on the released records reproduces Tables 2 and the pull-request counts. The production source code of the agent is not released. A Japanese version of this report is available at https://www.aquallc.jp/research/route-dont-refuse/.

References

[1] Kai Greshake, Sahar Abdelnabi, Shailesh Mishra, Christoph Endres, Thorsten Holz, and Mario Fritz. Not what you've signed up for: Compromising real-world LLM-integrated applications with indirect prompt injection. In *Proceedings of the 16th ACM Workshop on Artificial Intelligence and Security (AISec)*, 2023. URL https://arxiv.org/abs/2302.12173. arXiv:2302.12173.

[2] OWASP GenAI Security Project. LLM01:2025 prompt injection. OWASP Top 10 for LLM Applications 2025, 2025. URL https://genai.owasp.org/llmrisk/llm01-prompt-injection/.

[3] Simon Willison. The lethal trifecta for AI agents: private data, untrusted content, and external communication. Blog post, 16 June 2025, June 2025. URL https://simonwillison.net/2025/Jun/16/the-lethal-trifecta/.

[4] Meta AI. Agents rule of two: A practical approach to AI agent security. Blog post, 31 October 2025, October 2025. URL https://ai.meta.com/blog/practical-ai-agent-security/.

[5] Invariant Labs. GitHub MCP exploited: Accessing private repositories via MCP. Blog post, May 2025. URL https://invariantlabs.ai/blog/mcp-github-vulnerability.

[6] Edoardo Debenedetti, Jie Zhang, Mislav Balunović, Luca Beurer-Kellner, Marc Fischer, and Florian Tramèr. AgentDojo: A dynamic environment to evaluate prompt injection attacks and defenses for LLM agents. In *Advances in Neural Information Processing Systems 37, Datasets and Benchmarks Track*, 2024. URL https://arxiv.org/abs/2406.13352. arXiv:2406.13352.

[7] Simon Willison. The dual LLM pattern for building AI assistants that can resist prompt injection. Blog post, 25 April 2023, 2023. URL https://simonwillison.net/2023/Apr/25/dual-llm-pattern/.

[8] Edoardo Debenedetti, Ilia Shumailov, Tianqi Fan, Jamie Hayes, Nicholas Carlini, et al. Defeating prompt injections by design, 2025. URL https://arxiv.org/abs/2503.18813.

[9] Manuel Costa, Boris Köpf, Aashish Kolluri, et al. Securing AI agents with information-flow control, 2025. URL https://arxiv.org/abs/2505.23643.

[10] Dorothy E. Denning. A lattice model of secure information flow. *Communications of the ACM*, 19(5): 236–243, 1976. doi: 10.1145/360051.360056.

[11] Luca Beurer-Kellner et al. Design patterns for securing LLM agents against prompt injections, 2025. URL https://arxiv.org/abs/2506.08837.

[12] GitHub. Using secrets in GitHub Actions. GitHub Docs, accessed 18 November 2026, 2026. URL https://docs.github.com/en/actions/security-for-github-actions/security-guides/using-secrets-in-github-actions.

[13] GitHub. Managing GitHub Actions settings for a repository. GitHub Docs, accessed 18 November 2026, 2026. URL https://docs.github.com/en/repositories/managing-your-repositorys-settings-and-features/enabling-features-for-your-repository/managing-github-actions-settings-for-a-repository.

[14] GitHub Security Lab. Keeping your GitHub Actions and workflows secure part 1: Preventing pwn requests. GitHub Security Lab research, 2021. URL https://securitylab.github.com/resources/github-actions-preventing-pwn-requests/.

[15] Vercel. Deploying GitHub projects with Vercel: Git fork protection. Vercel Docs, accessed 18 November 2026, 2026. URL https://vercel.com/docs/git/vercel-for-github.

[16] Jerome H. Saltzer and Michael D. Schroeder. The protection of information in computer systems. *Proceedings of the IEEE*, 63(9):1278–1308, 1975. doi: 10.1109/PROC.1975.9939.

[17] GitHub. REST API endpoints for pull requests. GitHub Docs, accessed 18 November 2026, 2026. URL https://docs.github.com/en/rest/pulls/pulls.

[18] GitHub. Webhook events and payloads. GitHub Docs, accessed 18 November 2026, 2026. URL https://docs.github.com/en/webhooks/webhook-events-and-payloads.

[19] GitHub. What happens to forks when a repository is deleted or changes visibility. GitHub Docs, accessed 18 November 2026, 2026. URL https://docs.github.com/en/pull-requests/collaborating-with-pull-requests/working-with-forks/what-happens-to-forks-when-a-repository-is-deleted-or-changes-visibility.