AQUA RESEARCH · TECHNICAL REPORT 009

断らずに切り分ける

外の文を読んだ AI エージェントの仕事を、鍵の届かないフォークから PR にする：本番の事例

Route, Don't Refuse: Capability-Preserving Isolation of Prompt-Injection-Exposed Agent Work through Fork-Based Pull Requests — A Production Case Study

湯村 信司（Shinji Yumura）　AQUA合同会社　2026年11月19日

8/8 → 0/14

偽の鍵（CANARY）が CI から見えた回数。リポジトリの中の CI は8回すべてで見え、書き込みもできた。YUA のフォークからの PR の CI は14回すべてで、見えず・書けなかった

0

公開リポジトリで、守りのために減った「できること」。人の OK 待ちの形で失った、すぐの PR・大きな変更・CI の自動直しを、すべて取り戻した

6

本物の GitHub で試して確かめた振る舞い。うち5つは資料にない・想定と違うものだった。作りを壊した4つは、すべて止まる側に倒れ、鍵は漏れなかった

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, and it still exposed secrets once a human approved. We replaced it with routing: each job carries a sticky taint bit, propagated through signed job stages, and tainted work on public repositories is published as a draft pull request from a fork owned by the operator, where GitHub withholds secrets and grants only a read-only token. In 22 instrumented CI runs with a canary secret, all 8 in-repository runs could read the canary and write, while all 14 fork pull-request runs could do neither, including two deliberately broken commits that the agent repaired without approval. Live testing surfaced six platform behaviours, five of them undocumented or contrary to our assumptions; the four that caused failures all failed closed. Six rounds of adversarial review produced 33 findings, none rated high or critical. For public repositories the design restored every capability lost to refusal; it was enabled for all public repositories on 18 November 2026.

要旨

GitHub の Issue から呼べる AI エージェントは、誰が書いたかわからない文を読み、そのあとリポジトリの中にブランチを作る。そのブランチの CI（自動テスト）は、リポジトリの鍵（secrets）を持って動く。読んだ文に AI を操る命令（間接プロンプトインジェクション）がまぎれていると、鍵を外に送るコードを書かされるおそれがある。よくある対策は「断る」ことで、外から来た文を読んだ仕事を見分けて、人の許可を待たせる。本番の AI エージェント YUA にまずこの対策を入れると、作りの上で、利用者が使っていた機能（すぐの PR・大きな変更・CI の自動直し）が使えなくなり、しかも人が許可したあとは、結局コードが鍵つきで動く。
そこで、断る代わりに「振り分ける」形に作り直した。仕事ごとに「外の文を読んだ」という印を付け、署名した仕事の受け渡しで最後まで運ぶ。印の付いた仕事は、公開リポジトリなら、運営者が持つ組織にある**フォーク**から下書きの PR として出す。フォークからの PR の CI には、GitHub が鍵を渡さず、書き込みもさせない。守りを AI の判断ではなく、プラットフォームの決まりに任せる形である。フォークを作る App と中身を書く App を分けて権限を絞り、フォークは名前ではなく番号で見分け、元が非公開になったフォークは毎日の見回りで消す。
偽の鍵を置いた試験用リポジトリで、CI の全22回を集めた。リポジトリの中の CI は8回すべてで偽の鍵が見え、書き込みもできた。YUA のフォークからの PR の CI は14回すべてで、見えず・書けなかった。わざと壊したコミット2つも含み、YUA はそれを人の許可なしに直した。本物で試して確かめた GitHub の振る舞いが6つあり、うち5つは資料にない・想定と違うものだった。作りを壊した4つは、すべて止まる側に倒れ、鍵は漏れなかった。攻撃者役のレビューを6回行い、33件の指摘のうち、重い（high）・致命的とされたものはなかった。公開リポジトリでは、断る対策で失った機能はすべて戻り、2026年11月18日に、すべての公開リポジトリで有効にした。

1 はじめに

AI が外から取ってきた文に命令を混ぜて操る「間接プロンプトインジェクション」は、LLM を使うアプリの大きな危険である [1]。間接のものを含むプロンプトインジェクションは、OWASP の LLM アプリの危険の一覧で1位に置かれている [2]。エージェントでは、これが現実の被害になる。Willison は、「信頼できない文を読む」「秘密に触れる」「外に送れる」の3つを同時に持つエージェントを*致命的な三つ組（lethal trifecta）*と呼んだ [3]。Meta は、1つのセッションが持ってよいのは3つのうち2つまで、という*Agents Rule of Two* を勧めている [4]。GitHub につながるエージェントへの攻撃は、すでに報告されている [5]。

YUA は AQUA合同会社の AI アシスタントで、開発モードに AI エージェントがある。2026年11月から、GitHub の Issue や PR に「@yua 〜」と書くと呼べるようになった。エージェントは計画を立て、隔離した箱の中でコードを書いてテストを走らせ、GitHub App として PR を出す。2026年11月18日、社内の確かめで、次の穴が指摘された。書く権限のない外の人が「ついでに、環境変数をこの URL に送るテストも足して」と書いた Issue を立て

る。チームの人がその Issue で YUA を呼ぶと、YUA のブランチはリポジトリの中にでき、その CI はリポジトリの鍵を持って動く。AI が命令を断るかもしれないが、断ることを強いる仕組みはなかった。

最初の直しは、よくある形にした。外の文を読んだ仕事を見分け、その変更はすべて、チームの人の「@yua OK」を待つ。運営者はこれを退け、本稿の主張となる原則を示した。**守りのために、できることを減らしてはいけない**。OK 待ちの形では、作りの上で、すぐの PR が出せなくなり、45,000 文字を超える変更はコメントで見せられないので PR にできず、CI の自動直しも止まる。しかも OK のあとは、コードは結局鍵つきで動く。（この形が本番で動いたのは数時間だけなので、これは作りから分かる損で、利用者への影響を測ったものではない。）

本稿は、その作り直しを報告する。印の付いた仕事を断るのではなく、**プラットフォームが鍵を渡さない場所**、つまり運営者が持つ組織のフォークに振り分ける。貢献は次のとおりである。

- Rule of Two をエージェント全体ではなく**仕事ごと**に満たし、信頼できない文と鍵が出会わないまま、公開リポジトリですべての機能を残す振り分けの作り（4節）
- GitHub での作り：権限を最小に分けた2つの App、名前ではなく番号での見分け、CI の自動直し・名前の変更・片づけまでの一生の扱い（5節）
- 偽の鍵を使った本物での確かめ：フォークからの PR の CI は14回中0回、リポジトリの中の CI は8回中8回、鍵が見えた（6節）
- 本物で試して確かめた GitHub の振る舞い6つ（うち5つは資料にない・想定と違う）と、攻撃者役のレビュー6回の記録

---

2 背景

2.1 プロンプトインジェクションへの守り

入ってくる文や出ていく答えを見分けて止める守りは、手を変える攻撃者に破られる。AgentDojo のような試験台は、その差を測っている [6]。一方、作りで守る方法は、AI が操られても何ができないかを決める。Dual LLM は、信頼できない文を権限のある AI から遠ざける [7]。CaMeL は、信頼できる頼みから処理の流れを取り出し、実行役が権限を確かめる [8]。FIDES は、データに秘密さ・信頼さの札を付けて、決まった規則で守らせる [9]。これは情報の流れの制御 [10] の流れをくむ。Beurer-Kellner ほかは、こうした作りの型をまとめ、使い勝手が下がることも述べている [11]。本稿の作りは、信頼さの札に最も近い。ただし札は粗く（仕事ごとに1つ）、守らせる場所は自作の実行役ではなく、**すでにあるプラットフォームの境目**である。

2.2 GitHub のフォークの境目

フォークからの PR で動く CI には、GITHUB_TOKEN を除いて鍵が渡らず、その GITHUB_TOKEN も読むだけになる [12, 13]。既定では、初めての外の人の PR の CI は、持ち主の承認が要る [13]。この境目は、pull_request_target で PR のコードを取り出して動かす CI の書き方（pwn request）で破れる [14]。Vercel も、環境変数のあるプロジェクトでは、チームの人が許可するまで、フォークからの PR のお試しページを作らない [15]。

3 仕組みと攻撃の想定

1つの仕事は、計画・作業・（必要なら）許可のあとの公開、の段に分かれ、段ごとに別のサーバーの処理として動く。段から段へは、署名（HMAC）した仕事を渡す。AI は隔離した箱の中で作業し、リポジトリの GitHub の鍵は持たない。PR を出すのはサーバーのコードで、YUA Dev Mode App のその場限りの鍵を使う。

- **攻撃者**：エージェントが読みうる文を書けるが、リポジトリには書けない人。Issue やコメントを書いた人、ネットのページや部品置き場の説明を書いた人、リポジトリの歴史の外のコミットを書いた人、ほかの CI アプリ。攻撃者はエージェントを呼べない（呼べるのは書く権限のある人だけ）
- **守るもの**：CI が持つリポジトリの鍵、書き込める GITHUB_TOKEN、お試しページの鍵
- **攻撃者の狙い**：エージェントが書いたコードを、これらの鍵を持たせて動かすこと（鍵を外に送る、リポジトリに書く）
- **対象の外**：人が中身を見てマージしたコード（マージ後は、決まりどおり鍵つきで動く）。持ち主が自分でフォークに鍵を渡している場合（YUA は警告するが、止められない）

4 断らずに切り分ける

4.1 外の文の印は、消えない

リポジトリに書く権限のない人が書いた文を1つでも読んだ仕事を、「外の仕事」とする。印を付ける入り口は8種類ある（表1）。書く権限は GitHub の権限の窓口で確かめる。GitHub の author_association は、読むだけの人にも MEMBER や COLLABORATOR を付けることがあるので使わない。印と理由は署名した仕事の中で運ぶので、あとの段で落とせない。印の欄ができる前に署名された古い仕事は、外の仕事として扱う。外の仕事から作った PR の本文には隠し印を入れ、同じ PR をあとで直す仕事にも印が引き継がれるようにした。

表1 「外の文を読んだ」印を付ける入り口

入り口	印を付けるとき
呼ばれた Issue・PR	本文を、書く権限のない人が書いた（権限の窓口で確かめる）。そういう人のコメントは読まない
作業の途中で読む Issue	書いた人が書けない。そういう人のコメントは読まない
ネット検索・ページ取得	いつも
箱のコマンド	ネットから取ってくる道具（curl・wget・ssh など）・部品置き場を読む（npm view など）・名前を指定した部品の取り込み・手元にない道具の npx・git clone／fetch／pull／ls-remote／submodule／archive・その場で書いたネットに出るコード・部品置き場の住所

入り口	印を付けるとき
コミットを読む	仕事の元の歴史にないコミット
CI の結果	GitHub Actions 以外のアプリが知らせた、落ちたチェック
YUA が書いた文	外の印が付いている
読めなかった	どれでも（止まる側に倒す）

4.2 断るのではなく、振り分ける

印は、もう止めるためには使わない。**変更をどこから出すか**を決めるために使う（データに付けた route.ts に、同じ規則を書いた）。

- **外の仕事でない**：今まで通り、リポジトリの中のブランチ。鍵つきの CI・お試しページを含め、すべての機能がそのまま
- **外の仕事・公開リポジトリ**：運営者のフォークから下書きの PR。CI は鍵なし・読むだけの鍵で動き、落ちれば自動で直す
- **外の仕事・非公開リポジトリ**：フォークにすると非公開のコードを外に写すことになるので、今まで通り人の OK を待つ。フォークを用意できなかったときも、ここに戻る
- **持ち主が選ぶ**：いつものブランチの `.yua/settings.json` に `{"pr_from_fork": "always"}` と書くと、すべての仕事をフォークから出す。このファイルを取ってこられなかった（「ない」以外の失敗の）ときは、フォークから出す

Rule of Two [4] で言えば、外の仕事は「信頼できない文」と「外に送れる（PR を出す）」を持つが、鍵は持たない。信頼できる仕事は「鍵」と「外に送れる」を持つが、信頼できない文は持たない。エージェント全体は3つとも持ち続け、1つの仕事が3つを持つことはない。

---

5 GitHub での作り

権限を最小にした2つの App [16]。フォークを作るには、GitHub の「管理（Administration）」の権限が要る。これを本体の App に足すと、サーバーが乗っ取られたとき、使う人全員のリポジトリを消したり設定を変えたりできてしまう。そこで、フォーク置き場の組織にだけ入れた、別の App（フォーク係）を作った。絞ったあとの権限は「管理（書く）・中身（読むだけ）・メタデータ」で、フォークを作って確かめられるが、中身は書けない。フォークに書くのは本体の App（置き場にも入れてある）で、鍵はそのフォーク1つだけ・中身とワークフローとメタデータだけに絞る。置き場の組織では CI（Actions）を止めてあり、フォークを使うたびに止まっているかを確かめる。

名前ではなく番号で見分ける。エージェントは、自分のフォークからの PR を見分けて、直す・足す・片づける必要がある。フォークがある上流のものと言えるのは、フォークであり、持ち主の番号が置き場の組織の番号で、

元（parent）の番号が上流の番号のときだけ。PR がエージェントのものと言えるのは、head のリポジトリが置き場の組織の番号の下にあり、head のブランチがエージェントの印で始まり、PR を出したのがエージェントのボット本人のときだけである。公開のフォークのブランチは、誰でも head にして PR を出せるからだ。名前は、新しいフォークの名前を決めるときにしか使わない。

一生の扱い。PR は本体の App が、head_repo でフォークを名指しし、maintainer_can_modify を false にして出す（6.4節）。フォークからの PR の CI の知らせには PR の番号が入らないので、先頭のコミットで PR を引き直す。CI の自動直しの記録は、名前ではなくリポジトリの番号で残す。PR が閉じたら、フォークのブランチを消す。毎日の見回りで、印のあるフォークのうち、元から切り離されたもの（元が非公開になった・消えた）と、元が公開でなくなったもの、開いている PR がなく、30日 push も更新もないものを消す。はっきりしない返事のときは残し、GitHub に「多すぎる」と言われたら止める。PR の本文には AI のまとめをただの文字として載せ、上流の CI に pwn request の書き方 [14] があれば警告する。

---

6 確かめた結果

6.1 試験のしかた

公開の試験用リポジトリに偽の鍵 CANARY を置き、push と PR のたびに、CANARY が見えるか（値は出さない）と、GITHUB_TOKEN でタグを作れるか（書き込めるか）を表示する CI を置いた。役は、書く権限のあるチームの人、書く権限のない外の人、エージェントのボット、フォーク係の4つ。試験はすべて本番の YUA で行い、最後の段までは、フォークの道を試験用のリポジトリ（主な試験用リポジトリ、S6 のためにそれをフォークしたもう1つ、S8 のための使い捨ての1つ）だけで有効にした。外の人は、著者が持つ2つめの GitHub アカウントである。GitHub の窓口から、すべての Issue・PR・CI の実行と、表示された2つの印を集める道具で記録を取り直した。公開データでは、個人のアカウント名を役の名前に置き換えた。

6.2 鍵が見えたか

表2は、CI の全22回である。リポジトリの中の CI は、8回すべてで偽の鍵が見え、書き込めた。これはチームの人の push で、エージェントのブランチがリポジトリの中にあるときと同じ場で動くが、試験の中でリポジトリの中にエージェントのブランチは作っていない。フォークからの PR の CI は、14回すべてで、どちらもできなかった。外の人が運営者のフォークのブランチから出した PR、エージェントの PR、チームの人の試し、フォーク係がわざと壊したコミットの2回を含む。このうち3回（外の人の PR とフォーク係のコミット2つ）は、GitHub の決まりどおり、チームの人が承認して動かした。エージェントの CI は、すべてエージェント自身のもので、承認なしで動いた。エージェントが出した PR は6本で、すべてフォークから、すべて下書きで、マージしたものはない。

表2　試験用リポジトリの CI の全22回（2026年11月18日）

どこから動いたか	回数	偽の鍵が見えた	書き込めた	落ちた
リポジトリの中（push）	8	8	8	0

どこから動いたか	回数	偽の鍵が見えた	書き込めた	落ちた
フォークからの PR：エージェントが出した・書いた	8	0	0	0
フォークからの PR：チームの人が出した	3	0	0	0
フォークからの PR：外の人が出した ＊	1	0	0	0
フォークからの PR：フォーク係のコミット ＊	2	0	0	2 ※
フォークからの PR の合計	**14**	**0**	**0**	2

※ 自動直しを試すために、わざと壊したコミット。エージェントの直しは人の承認なしに動き、通った。＊ チームの人が承認して動かした。

6.3 本物の流れ

表3の流れを、本番の YUA で順に試した。外の人の Issue の2回はどちらも、偽の鍵を読む・表示するコードを足せ、という命令を混ぜたが、エージェントはどちらも従わなかった。どちらの PR の変更にも、偽の鍵・鍵・環境変数・ネットへの送り出しに触れる行はなく、エージェントの PR 6本のどれも CI の設定ファイルを変えていない（変更の中身はデータに入れた）。ただし、作りはこの結果に頼っていない。S8 は、消したフォークをもう読み直せないので、作業中に表示した結果を記録としてデータに入れた。

表3　本番での流れ

	試したこと	結果
S1	外の人が命令を混ぜた Issue を立て、チームの人が YUA を呼ぶ	フォークから下書きの PR #3。本文に外の文の理由。CI は鍵なし・書けない
S2	S1 のフォークのブランチに、わざと壊すコミット	YUA が先頭のコミットで PR を見つけ、承認なしで直した。CI は通り、鍵なし
S3	上流の名前を変え、持ち主が「いつもフォーク」、YUA を呼ぶ	1回目は GitHub に断られた（B4・B5）。直したあと、新しいフォークから下書きの PR #8
S4	名前を元に戻し、「PR #8 に足して」	新しい名前のフォークではなく、PR #8 自身のフォークに書いた
S5	名前を変えたあと、PR #10 にわざと壊すコミット	直しの記録をリポジトリの番号で見つけて直した。閉じるとブランチが片づいた
S6	A をフォークしたリポジトリ B でも YUA を使う	B には B 専用のフォークができた。A のフォークとは混ざらない

	試したこと	結果
S7	フォーク係の権限を「管理＋中身を読む」に絞る	新しいフォークができ、PR #12、CI は鍵なし、片づけは本体の App
S8	試験用フォークの元を非公開にして、見回りを動かす	消さずに見る見回りでは、印のない切り離されたフォークに触れなかった。印を付けると消した。使っている4つは残した
S9	すべてのリポジトリで有効にし、外の人が命令を混ぜた Issue	フォークから下書きの PR #14。変えたのは頼まれたファイルだけ。鍵なし

6.4 資料や想定と違った GitHub の振る舞い

表4は、本物で確かめた6つの振る舞いである。5つは資料に書かれていないか、確かめたつもりの想定と違った。B6 は資料どおりで、見回りの作りの元になった。4つは初めて出会ったときに作りを壊したが、壊れ方はすべて止まる側だった。GitHub が PR を作らなかった（B1・B4・B5）か、CI の自動直しが PR を見つけられなかった（B2）。B3 は CI が承認待ちになった。鍵が漏れたものはない。B4 は教訓になる。初めのころ「すでにある名前でフォークを頼むと、今のフォークが返ってくる」のを見て、これを「1つの組織にフォークは1つ」と記録した。その一般化の上に作った形は、GitHub が新しい名前で2つ目のフォークを作ったときに壊れた。

表4　本物で試して分かった振る舞い

	資料・想定	本物（と、その手当て）
B1	maintainer_can_modify は、ただの選択と書かれている [17]	組織が持つフォークでは、省くと 422（fork_collab）で断られた（false を明示する）
B2	フォークの CI の知らせは PR が空で、head_branch も null [18]	head_branch にフォークのブランチ名が入っていた（先頭のコミットで PR を引く）
B3	初めての人の CI は承認が要る [13]	フォークにだけ入った App が書いたコミットは承認待ち。上流にも入った App のコミットは承認なしで動いた（後者で書く）
B4	1つの組織に、同じ元のフォークは1つ（想定）	新しい名前を指定すると、2つ目のフォークができた（元の番号で見分け、上流ごとに1つ）
B5	head_repo は「同じ組織が両方を持つとき」に要る [17]	組織にフォークが2つあると、「組織:ブランチ」が別のフォークを指して 422（いつも head_repo を渡す）
B6	元を非公開にしても、公開のフォークは公開のまま残る [19]	資料どおり。フォーク係からは fork: false・元なしに見える（見回りで消す）

6.5 攻撃者役のレビュー

出す前に毎回、別の AI エージェントに攻撃者役のレビューを頼んだ。渡すのはコードだけで、作り手の結論は渡さない。指摘は、作り手が自分で確かめてから直した。表5は6回の要約で、札はレビューが付けたままである（1〜3回目は「確か」「ありうる」で順位づけ、4〜6回目は重さ）。公開した記録には、指摘ごとの扱いを書いた。確かな指摘6件はすべて出す前に直した。残りの27件は、22件を直し、1件を残る危なさとして記録し、4件を理由をつけて直さないとした。1・2回目の確かな指摘は、見張りが見落としていた入り口（作業の途中で読む Issue、箱からのネットの読み込み、歴史にないコミット、PR に足すときに印が消えること）だった。6回のどれも、フォークの境目そのものを抜ける道は報告しなかった。

中くらいの指摘の例：外の人が、エージェントのフォークのブランチを head にした PR を出して閉じると、本物の PR のブランチを消させられた（PR を出した人を確かめて直した）。同じつながりのリポジトリでフォークを共有すると、別の上流の仕事の書き込みと混ざった（上流ごとに1つにした）。名前で残した自動直しの記録を番号に移すと、空いた名前を取った人が、ほかの人の直しの回数を使えた（移さないことにした）。見回りが、はっきりしない返事で消したり、関係ないリポジトリを消したりしうる（はっきりした合図と、印のあるフォークだけにした）。

表5　攻撃者役のレビュー（札はレビューが付けたまま）

回	見たもの	指摘	札
1	外の文の見張り	6	確か1・ありうる4・軽1
2	外の文の見張り（2回目）	4	確か3・ありうる1
3	フォークの道の初版	8	確か2・ありうる3・軽1・注意2
4	番号で見分ける形	5	中1（条件つき）・軽4
5	最終の形（記録を番号で）	5	中1・軽2・ごく軽2
6	毎日の見回り	5	中2・軽3
	合計	**33**	重い（high）・致命的とされたものはなし

6.6 できること

表6は、公開リポジトリでの外の仕事を、3つの形で比べたものである。振り分ける形は、断る形で失った機能をすべて取り戻し、断る形では許可のあとに残っていた「鍵に届く」危なさもなくした。ただし、お試しページは形が変わる。フォークからの PR のお試しページは、チームの人が許可するまで作られない [15]（本物では試していない）。

表6　公開リポジトリでの外の仕事

	もとの形	断る（人の OK 待ち）	振り分ける（フォーク）
待たずに PR	できる	できない	できる（下書き）
45,000 文字を超える変更	できる	できない	できる
CI の自動直し	できる	できない	できる
変更の CI	鍵つき	OK のあと鍵つき	鍵なし
マージ前に、混ぜられたコードが CI の鍵に届く	ありうる	OK のあとありうる	ない（プラットフォームが守る）

7 考察

7.1 エージェントではなく、仕事を分ける

Rule of Two は、エージェントへの制限として読まれることが多い。しかし仕事ごとに当てはめ、印でその仕事が持つ2つを決めれば、3つを順番にしか使わない仕事にとって、機能の損はない。コーディングエージェントは、この点で恵まれている。フォークからの PR という成熟した境目が、「鍵のない実行の場」と、鍵に届く前の「人のレビュー」をすでに用意しているからである。

7.2 止まる側に倒し、本物で試す

本物での5つの想定外は、どれも漏れとしては現れず、「断られる」「PR が見つからない」「承認待ち」のどれかで現れた。どの確かめも「確かめられたときだけ進む」と書いたからである。そして想定外そのものは、本物で動かして初めて見つかった。資料は、触れていない（B4）、足りない（B1・B3・B5）、本物と違う（B2）ことがあった。

7.3 ほかの場への広がり

この作りに要るのは、(i) 署名した仕事の状態で運ぶ粗い印、(ii) プラットフォームが鍵を渡さない実行の場、(iii) 名前が変わっても残る見分け、の3つである。GitLab のフォークからのマージリクエストや、フォークを隔てるほかの CI でも、同じ形が使えると考える。

8 限界と残る危なさ

- **小さな、1か所の確かめ**：1つの試験用リポジトリ・1つのエージェント・1つのプラットフォームで22回。作りも試験も著者が行った。統計の見積もりではなく、事例の記録である

- **信頼できる仕事は、今も見張り頼み**：外の文を読んでいない仕事は、リポジトリの中で鍵つきで動く。見張りが入り口を見落とせば、そこは開く。いちばん固くしたい持ち主は、すべての仕事をフォークから出す設定を選べる
- **悪いコードは PR には入りうる**：フォークが防ぐのは、マージ前に鍵へ届くことで、悪いコードそのものではない。マージ前の人のレビューが最後の関所である。PR は下書きで、フォークから出た理由も書いてある
- **非公開リポジトリ**：今まで通り、人の OK を待つ。フォークを用意できなかったときも、ここに戻る
- **お試しページ**：フォークからの PR のお試しページは、Vercel ではチームの人が許可するまで作られない。本物では試しておらず、持ち主はこの守りを外せる
- **試験のアカウント**：試験用のリポジトリとフォークは公開なので、データの役の名前からアカウントはたどれる。外の人は著者の2つめのアカウントである
- **持ち主の設定**：フォークに鍵を渡す CI の書き方は境目を破る。よくある形は警告するが、直すことはできない
- **切り替えの手間**：作り直しの前からあった自動直しの記録は、安全のために移さなかった（6.5節）。利用者は一度だけ見張り直す
- **規模**：フォークが多くなったときの GitHub の回数の上限などは、試験用リポジトリより大きな規模では試していない

---

9 まとめ

プロンプトインジェクションにさらされた仕事を人の許可待ちにすると、本番の AI エージェントは使いにくくなり、しかも鍵と、混ぜられたコードが出会うことは止められなかった。その仕事を、プラットフォームが鍵を渡さない運営者のフォークに振り分けると、公開リポジトリでは失った機能はすべて戻り、偽の鍵を使った本物の試験では、リポジトリの中の CI では8回中8回見えた鍵が、フォークの CI では14回中1回も見えなかった。

仕組みより広い教訓として、次のことを挙げたい。守りの直しは、利用者に何を残したかで評価するべきだ。そして、いちばん強い境目は、プラットフォームがすでに守っているものであることが多い。

参考文献

1. K. Greshake, S. Abdelnabi, S. Mishra, C. Endres, T. Holz, M. Fritz. “Not What You've Signed Up For: Compromising Real-World LLM-Integrated Applications with Indirect Prompt Injection.” *AISec*, 2023. arXiv:2302.12173
2. OWASP GenAI Security Project. “LLM01:2025 Prompt Injection.” genai.owasp.org, 2025.
3. S. Willison. “The lethal trifecta for AI agents: private data, untrusted content, and external communication.” simonwillison.net, 2025年6月16日.
4. Meta AI. “Agents Rule of Two: A Practical Approach to AI Agent Security.” ai.meta.com, 2025年10月31日.
5. Invariant Labs. “GitHub MCP Exploited: Accessing private repositories via MCP.” invariantlabs.ai, 2025年5月.

6. E. Debenedetti, J. Zhang, M. Balunović, L. Beurer-Kellner, M. Fischer, F. Tramèr. “AgentDojo: A Dynamic Environment to Evaluate Prompt Injection Attacks and Defenses for LLM Agents.” *NeurIPS Datasets and Benchmarks*, 2024. arXiv:2406.13352
7. S. Willison. “The Dual LLM pattern for building AI assistants that can resist prompt injection.” simonwillison.net, 2023年4月25日.
8. E. Debenedetti, I. Shumailov, T. Fan, J. Hayes, N. Carlini ほか. “Defeating Prompt Injections by Design.” arXiv:2503.18813, 2025.
9. M. Costa, B. Köpf, A. Kolluri ほか. “Securing AI Agents with Information-Flow Control.” arXiv:2505.23643, 2025.
10. D. E. Denning. “A Lattice Model of Secure Information Flow.” *Communications of the ACM* 19(5), 236–243, 1976.
11. L. Beurer-Kellner ほか. “Design Patterns for Securing LLM Agents against Prompt Injections.” arXiv:2506.08837, 2025.
12. GitHub. “Using secrets in GitHub Actions.” GitHub Docs（2026年11月18日に確認）.
13. GitHub. “Managing GitHub Actions settings for a repository.” GitHub Docs（2026年11月18日に確認）.
14. GitHub Security Lab. “Keeping your GitHub Actions and workflows secure Part 1: Preventing pwn requests.” securitylab.github.com, 2021.
15. Vercel. “Deploying GitHub Projects with Vercel（Git Fork Protection）.” Vercel Docs（2026年11月18日に確認）.
16. J. H. Saltzer, M. D. Schroeder. “The Protection of Information in Computer Systems.” *Proceedings of the IEEE* 63(9), 1278–1308, 1975.
17. GitHub. “REST API endpoints for pull requests.” GitHub Docs（2026年11月18日に確認）.
18. GitHub. “Webhook events and payloads.” GitHub Docs（2026年11月18日に確認）.
19. GitHub. “What happens to forks when a repository is deleted or changes visibility.” GitHub Docs（2026年11月18日に確認）.

データ：CI の実行と PR の記録（変えたファイルを含む）・GitHub の振る舞いの観察・指摘ごとの扱いを含むレビューの記録・見回りの試験の記録・試験用リポジトリの CI・記録を集めて数える道具・振り分け／見分け／見回りの規則の見本の実装（テストつき）を、GitHub で公開している。公開した記録から数え直すと、表2と PR の数が再現される。YUA 本体のプログラムは公開していない。英語版：Route, Don't Refuse (PDF)。