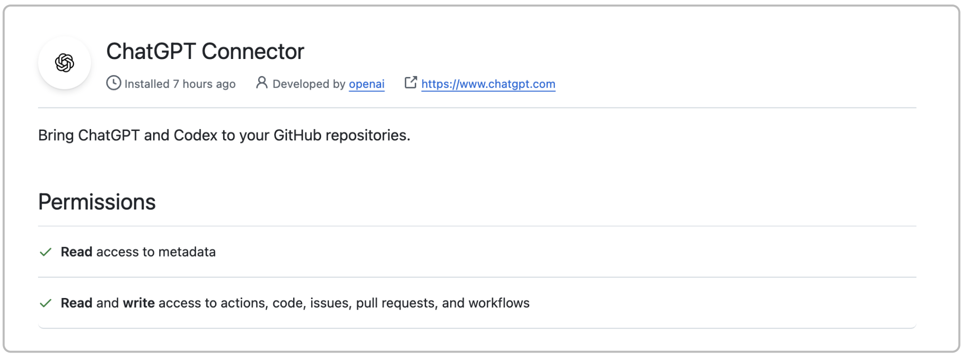
ChatGPT

Security Debrief: OpenAI's ChatGPT Connector GitHub App

Exact Permissions Review



ChatGPT Connector

Installed 7 hours ago · Developed by openai · https://www.chatgpt.com

Bring ChatGPT and Codex to your GitHub repositories.

Permissions

✓ **Read** access to metadata

✓ **Read** and **write** access to actions, code, issues, pull requests, and workflows

Figure: The ChatGPT Connector GitHub App (by OpenAI) and its permission summary. The **ChatGPT Connector** app requests broad repository access when installed. According to GitHub's installation screen, it is granted:

- **Read access to repository metadata** – This default permission allows reading basic repo information (metadata) but not sensitive content [1]. GitHub notes that the metadata API endpoints are read-only and do not leak private data.
- **Read and write access to code (repository contents)** – Full access to view and modify the code in your repositories [2]. In practice, this means the app's installation token can **read all files and push commits** to the repo. It is essentially equivalent to a user with write permissions to the repository's contents.
- **Read and write access to pull requests** – Permission to read PR data and perform PR actions via the API. This includes creating new pull request objects and modifying or merging PRs. (For example, OpenAI's app can open a PR with code changes. GitHub's docs note that write-level PR access is needed for an app to create or update pull requests [3].)
- **Read and write access to issues** – Ability to read issue data and also create or edit issues and comments. The connector could, for instance, open new issues or comment on existing ones via the GitHub API.
- **Read and write access to Actions and Workflows** – This grants control over GitHub Actions CI/CD workflows. "Actions" permission covers workflow runs and artifacts (e.g. the app could trigger or cancel runs, or read action logs/artifacts), while "Workflows" permission is required to **create or modify workflow files** in the repository. In fact, GitHub will reject any attempt by an app to add or change files under `.github/workflows/` unless it has the `workflows` permission [4]. With these permissions, the ChatGPT Connector can potentially enable/disable workflows or push new workflow files.

Overall, the connector's permission set is very expansive – essentially **full read/write for code and project data** on selected repositories (except for admin settings). This is comparable to an OAuth token with the **"repo" scope**, which GitHub defines as full read/write access to code, issues, pull requests, and more [2]. OpenAI presumably chose these broad scopes to ensure ChatGPT can not only read your code but also create branches, propose changes, and interact with development workflow as needed.

Actual Permissions vs. Minimal Desired Access

The current permission set **goes beyond just creating pull requests**. In an ideal least-privilege scenario, we might only give the app the ability to propose changes via PRs, without the ability to directly modify code in the main repository. For example, the minimal needed rights could be: **read-only code access** (so ChatGPT can read your repo) and permission to **write pull requests** (so it can open PRs with suggested changes). This would prevent the app from pushing commits directly to protected branches.

However, GitHub's model doesn't offer a separate "create PR without direct code write" permission – write access to code (contents) is a blanket permission that includes pushing commits to any branches the app has access to. In practice, if an app can write to the repo's contents, it can theoretically push to `main` (or any branch) unless blocked by branch protections. This is why the ChatGPT Connector having **"write" on code** is significant: it implies the app could commit changes outright. The safer workflow is for AI assistants to only open PRs that a human reviews and merges, rather than the bot altering the main codebase on its own. **The risk of granting write access to code is that the app (or an agent acting through it) could bypass the normal code review process and modify source code directly.** For instance, it might inadvertently push a change to `master` instead of creating a PR, if misconfigured or in response to a prompt – a scenario one would want to avoid.

By contrast, a pull-request-only approach confines the AI's changes to a new branch and requires maintainers to manually approve and merge. This greatly reduces risk, since maintainers can inspect AI-generated code before it affects production. It also ensures compliance with typical development workflows (tests, reviews, etc., on the PR). **Direct write access to code is inherently riskier** – e.g. an agent could alter critical files or bypass approvals. In summary, the ChatGPT Connector's actual permissions enable both PR creation *and* direct commits, whereas a least-privilege design would try to allow the former without the latter. Unfortunately, as discussed next, GitHub doesn't yet provide such fine-grained control.

Limitations of GitHub's OAuth/App Permission Model

GitHub's current app permission model lacks the granularity to allow "PR-only" write access. **Repository permissions are coarse** – e.g. an app can have read or read/write for "Contents" (code), but there is no built-in way to permit "write but only in pull requests" or "disallow direct pushes." As the OpsLevel team noted, *"even though we only need to perform a limited set of actions, the GitHub permissions model is not granular enough to restrict us to only these required actions."* [5] In their case (creating repos and PRs), they had to request broad write scopes because more restrictive scopes simply don't exist. This is the same core issue with ChatGPT's connector: to let it open PRs with code changes, we must give it broad code write access.

For OAuth apps, the situation is similar or worse – the classic `repo` scope grants full control over code, issues, and more [2]. GitHub **fine-grained PATs and GitHub Apps improved granularity** compared to a single monolithic scope, but they still tie together permissions in chunks that may exceed a specific app's minimal needs. For instance, "Contents: Read/Write" covers everything from creating branches and commits to deleting files. There's no way for a repository owner to configure an installed GitHub App to only allow certain write operations but not others. It's essentially an all-or-nothing choice for that category of data.

One partial safeguard is the use of **branch protection rules** on important branches (like `main`). If branch protections require pull requests and reviews before merge, then even an app with write permissions cannot push to protected branches unless explicitly exempted. GitHub has updated branch protection settings to allow or deny exceptions for GitHub Apps as well [6]. By default, an app without the **“ bypass branch protection”** ability would be forced to go through PRs on a protected branch. In other words, repository maintainers can configure that “ChatGPT Connector cannot directly push to main; it must open a PR and have it approved,” leveraging branch protection. This is a **crucial mitigation**, but it is external to the app’s permission grant. It requires admins to correctly set up protections. And if those settings are relaxed (or the app is added as an exception), the app’s token could push straight to main.

The lack of built-in granular scopes (like a hypothetical “PR creation” scope) is concerning when we imagine more autonomous coding agents in the future. As these AI agents become capable of writing and deploying code autonomously, the platform’s coarse permission model could lead to over-privileged apps. A future “AI Dev Agent” might only need to propose changes, but if given full write rights, a logic error or prompt injection could cause it to modify or delete code in ways the owners didn’t intend. GitHub’s current OAuth/App model doesn’t provide a way to *natively restrict* such an app to a safe subset of actions. This means **repository owners must carefully scope which repos an app can access and rely on external safeguards** (like branch rules and monitored workflows) to limit damage. The need for more fine-grained permission controls is increasingly evident as automated agents play a bigger role in codebases [5].

Security Threat Model and Potential Abuse Scenarios

Given the **high level of access** the ChatGPT Connector has, it’s important to consider what could go wrong if the app or its credentials were compromised. This could happen via a malicious update in OpenAI’s code, a supply-chain attack, or even an indirect prompt injection that causes ChatGPT to misuse its access. **If an attacker obtained the connector’s installation token (or could impersonate the app), they would essentially have the privileges of a repo collaborator with write access.** Some of the key threats include:

- **Unauthorized Code Changes (Backdoors or Destructive Commits):** With write access to code, a compromised app could insert malicious code into your repository – for example, adding a subtle vulnerability/backdoor in the source or outright replacing files with corrupted versions. It could push commits directly to existing branches (if not protected) or create new branches with harmful code. In the worst case, it might push to the default branch and deploy malware or destructive changes without code review. This kind of **malicious code injection** could go unnoticed if not actively monitored, potentially introducing serious security holes into the project.

- **Malicious Pull Requests and Merges:** Even if direct pushes are limited by policy, an attacker could use the app to craft a **malicious pull request**. For instance, they might open a PR that appears to make innocuous changes but actually introduces a vulnerability. If the organization has any automated merge or CI process that isn’t airtight, the attacker might manipulate it. Clutch Security notes that threat actors can abuse PR workflows – if CI/CD automatically runs on PRs and the code isn’t thoroughly reviewed, an attacker’s code could slip through and, say, extract secrets during the build [7]. In the context of ChatGPT Connector, an attacker could theoretically have the app approve or merge its own PR (if the process allows) or simply rely on maintainers to merge a seemingly benign PR generated by “ChatGPT”. The key risk is that the normal safety net of human code review could be subverted or trusting maintainers might merge changes coming from a trusted app identity.

- **CI/CD Workflow Tampering:** With the **Actions and Workflows permissions**, a compromised connector could alter the repository's continuous integration pipeline. It might inject a new GitHub Actions workflow file that runs on certain triggers. For example, an attacker could use the token to **add a malicious Actions workflow** that executes on every push or on a schedule. This workflow could exfiltrate secrets or modify build artifacts. A real-world example of this occurred in the Grafana incident, where attackers who stole a GitHub App token used it to push a malicious GitHub Actions workflow into the repo – that workflow collected all secret credentials from the environment and sent them out [8]. The ChatGPT Connector could similarly be instructed (or forced) to add a workflow that, say, runs `env` and sends environment variables to an external server. In addition, the app could modify or disable existing workflows: e.g. it could turn off tests or security scans (by disabling the workflow or changing its triggers), thereby **disabling CI/CD safeguards**. This would let malicious changes avoid detection. In short, control over workflows means the attacker can **undermine the integrity of the development and deployment process**.

- **Exfiltration of Source Code and Data:** The connector has read access to all repository content, so an attacker could use it as a vehicle to steal sensitive information. This could be proprietary source code, configuration files, or even data files stored in the repo. The app could systematically read the entire repository and send it to an external server controlled by the attacker. Because ChatGPT is meant to read code, such actions might not be immediately suspicious. **High-privilege tokens pose a significant data leakage risk if compromised [9] .** In the case of ChatGPT Connector, any private code it has access to could be harvested. This is especially dangerous if the repositories contain secrets (API keys, credentials) in the code – the attacker could search for and extract those as well. Essentially, the organization's intellectual property and any sensitive info in the repo could be exposed.

- **Credential Theft via CI Secrets:** Beyond just reading the code, an attacker might go after **secrets stored in the repository's CI/CD**. Many projects have CI secrets (deploy keys, cloud credentials, etc.) that are accessible to workflows. By abusing the Actions permission, the attacker can run a job that dumps these secrets. For instance, the malicious workflow in the Grafana case serialized all GitHub Action secrets and exfiltrated them [10]. If ChatGPT Connector were compromised, it could be instructed to perform a similar action: trigger a custom workflow run that prints all secret variables or database passwords, etc., and then capture that output. This could lead to broader compromise (e.g. leaking cloud infrastructure credentials). It's a form of supply-chain attack using the CI pipeline.

- **Project Integrity and Availability Attacks:** With issue and PR privileges, a malicious actor could also disrupt the project's workflow in less subtle ways – for example, mass-closing all issues or open pull requests, vandalizing issue comments with spam or false information, or opening a flood of new issues to overwhelm maintainers. They might modify wiki pages or release notes (if those are stored in the repo) to mislead users. While these actions may be noticed and rolled back, they could impede the development process and trust. Also, since the app can modify workflows and code, an attacker could attempt to **sabotage the repository** – e.g. insert code that intentionally breaks the build or delete critical files (ransomware style). Although the Git history could restore things, it would cause downtime and chaos.

In summary, the **attack surface is significant** whenever an app like ChatGPT Connector has wide write access. A compromised token would let an adversary do almost anything the repository owner can do short of changing settings: publish malicious code, alter CI pipelines, steal confidential data, and generally wreak havoc in the codebase. These scenarios underscore the importance of limiting app permissions and using defense-in-depth: monitor the app's activity, enforce branch protections (so even a rogue ChatGPT PR can't auto-merge), and perhaps restrict the app to only non-production repositories or a sandbox until it's fully trusted. As one security analysis succinctly put it, *"tokens with high privileges pose significant risks if compromised."* [9] This absolutely applies to the ChatGPT Connector's token – it must be handled as a sensitive credential, because in the wrong hands it could facilitate a serious supply chain attack on your code.

Sources: The permission scopes and their implications are based on GitHub's official documentation and installation screens [1] [2]. Observations about limited permission granularity reference GitHub's own model and expert commentary [5]. Security scenarios are informed by real incidents (e.g. Grafana's breach via a GitHub App token [8]) and industry analysis of risks in CI/CD integrations [7] [9].

[1] GitHub | Buildkite Documentation
https://buildkite.com/docs/pipelines/source-control/github

[2] Scopes for OAuth apps - GitHub Docs
https://docs.github.com/en/apps/oauth-apps/building-oauth-apps/scopes-for-oauth-apps

[3] [5] Updated OpsLevel GitHub App Permissions as Part of Service Creation
https://www.opslevel.com/resources/updated-opslevel-github-app-permissions-as-part-of-service-creation

[4] refusing to allow a GitHub App to create or update workflow `.github ...
https://github.com/orgs/community/discussions/51520

[6] jenkins - In GitHub how can I grant an app permissions to commit to a branch with protections? - Stack Overflow
https://stackoverflow.com/questions/65348124/in-github-how-can-i-grant-an-app-permissions-to-commit-to-a-branch-with-protecti

[7] The New York Times Exposed GitHub Token Breach
https://www.clutch.security/blog/the-new-york-times-exposed-github-token-breach

[8] [10] Grafana GitHub Actions Security Incident - StepSecurity
https://www.stepsecurity.io/blog/grafana-github-actions-security-incident

[9] CodeQLEAKED: GitHub Supply Chain Attack Enables Code Execution
https://gbhackers.com/codeqleaked-github-supply-chain-attack/