Poisoned Pipeline Execution, Technique T1677 - Enterprise

Archived: 2026-04-05 12:57:42 UTC

Adversaries may manipulate continuous integration / continuous development (CI/CD) processes by injecting malicious code into the build process. There are several mechanisms for poisoning pipelines:

- In a **Direct Pipeline Execution** scenario, the threat actor directly modifies the CI configuration file (e.g., `gitlab-ci.yml` in GitLab). They may include a command to exfiltrate credentials leveraged in the build process to a remote server, or to export them as a workflow artifact.[1][2]
- In an **Indirect Pipeline Execution** scenario, the threat actor injects malicious code into files referenced by the CI configuration file. These may include makefiles, scripts, unit tests, and linters.[2]
- In a **Public Pipeline Execution** scenario, the threat actor does not have direct access to the repository but instead creates a malicious pull request from a fork that triggers a part of the CI/CD pipeline. For example, in GitHub Actions, the `pull_request_target` trigger allows workflows running from forked repositories to access secrets. If this trigger is combined with an explicit pull request checkout and a location for a threat actor to insert malicious code (e.g., an `npm build` command), a threat actor may be able to leak pipeline credentials.[1][3] Similarly, threat actors may craft pull requests with malicious inputs (such as branch names) if the build pipeline treats those inputs as trusted.[4][5][6] Finally, if a pipeline leverages a self-hosted runner, a threat actor may be able to execute arbitrary code on a host inside the organization's network.[7]

By poisoning CI/CD pipelines, threat actors may be able to gain access to credentials, laterally move to additional hosts, or input malicious components to be shipped further down the pipeline (i.e., Supply Chain Compromise).

Source: https://attack.mitre.org/techniques/T1677