CI/CD Goat Security Assessment

DevSecOps Laboratory Report

Cybersecurity Engineering

TEK-UP
Private Higher School of Technologies & Engineering

Students	:	**Mohamed Yacine Riabi** & **Abdallah Dridi**
Instructor	:	Prof. Houssem Eddine Hermassi
Program	:	Cybersecurity Engineering
Institution	:	TEK-UP University
Lab Platform	:	CI/CD Goat — DevSecOps

This report documents a structured hands-on assessment of the CI/CD Goat training environment, mapping real-world DevSecOps attack patterns to specific pipeline weaknesses, exploited trust boundaries, and mitigating controls.

Contents

List of Figures

List of Tables

CHAPTER 1

Executive Summary

1.1 What Was Assessed

Eleven CI/CD Goat challenges were completed against a locally-deployed Jenkins, Gitea, GitLab, and LocalStack environment. Every challenge represented a distinct, validated trust boundary failure—not theoretical, not simulated: each flag was captured and submitted to CTFd.

The environment modelled a realistic internal DevSecOps stack. The attack surface included pipeline definitions, SCM tokens, runner scheduling, infrastructure-as-code deployments, internal package registries, and container image namespaces.

1.2 What Broke and Why

Core Pattern

Every compromised challenge shared one root cause: **a privileged pipeline context accepted input from an insufficiently-controlled source**. The trust model was always wider than the protection model.

The eleven challenges fell into four exploit classes:

- **Poisoned pipeline execution** (White Rabbit, Mad Hatter, Dormouse): Pipeline code, Makefiles, or remote helper scripts were attacker-controlled at the point where credentials were already bound.
- **Token escalation** (Caterpillar, Mock Turtle): PR jobs and auto-merge automation held SCM write tokens and exposed them to unreviewed fork content or weak diff checks. Caterpillar turned read-only repo access into trusted-branch modification in two build cycles.
- **Execution placement abuse** (Cheshire Cat, Hearts): Agent label selection and SSH launcher configuration were both reachable from low-privilege positions, giving attacker-controlled code access to the Jenkins controller filesystem and the system credential

store.

- **Supply-chain compromise** (Twiddledum, Dodo, Gryphon): Internal package versioning, Terraform apply-time provisioners, and mutable container image tags all allowed state to diverge from what any static scanner or reviewer had approved. Gryphon was a four-stage chain: package publish → scheduled token leak → registry push → deployment image poisoning.

1.3 Risk Summary

Two tasks scored **Critical** under CVSS 3.1—Cheshire Cat (9.1) and Gryphon (9.4). Eight scored **High**. The single Medium was Duchess, a Git history leak with no live pipeline component.

The practical impact across the set: Jenkins credential exposure, SCM write token escalation, Jenkins controller filesystem access, SSH credential replay to an attacker-controlled host, live cloud infrastructure mutation post-scanner, and full deployment-image supply-chain compromise.

CHAPTER 2

Why CI/CD Security Matters

2.1 CI/CD as an Attack Surface

The Privileged Hub Problem

A modern CI/CD system is not just a build tool. It is a privileged hub that holds source code, signing keys, registry credentials, deployment tokens, cloud access, and production-facing execution paths. Compromising the pipeline is often more impactful than compromising the application it builds.

Continuous integration and delivery pipelines sit at the intersection of every trust domain in a software organization: developer identity, reviewed code, packaged artifacts, infrastructure state, and running services. NIST SP 800-218 (SSDF) treats secure software development as a full lifecycle discipline, and CI/CD is the layer where most of those controls either hold or collapse.

2.2 The CI/CD Attack Surface Landscape

Figure 2.1 maps the five canonical injection points where an attacker can enter a typical CI/CD pipeline. All five appeared in this lab.

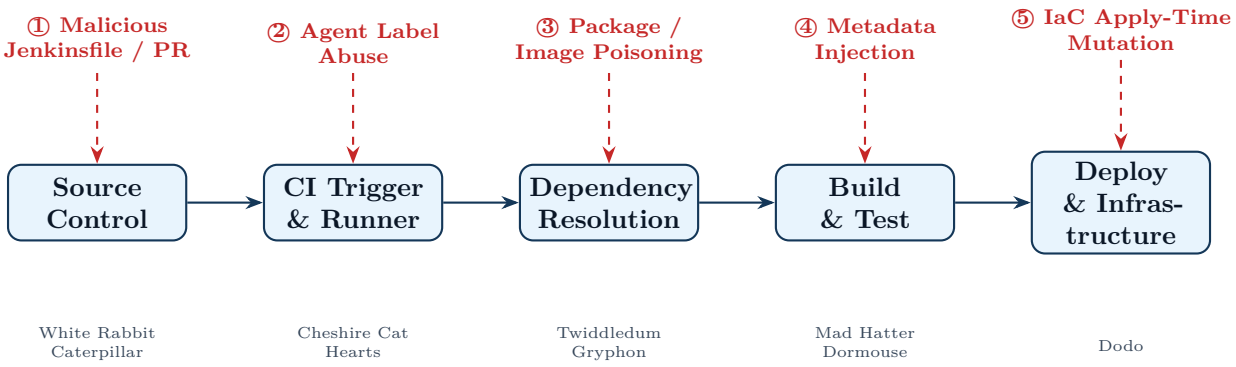

Figure 2.1: CI/CD pipeline stages with the five primary attack injection points and the corresponding lab challenges.

2.3 Mapping to Industry Risk Frameworks

OWASP CI/CD Risk	Description	Lab Challenges
CICD-SEC-1	Insufficient Flow Control	Mock Turtle
CICD-SEC-2	Inadequate Identity & Access	Cheshire Cat, Hearts
CICD-SEC-3	Dependency Chain Abuse	Twiddledum, Gryphon
CICD-SEC-4	Poisoned Pipeline Execution	White Rabbit, Mad Hatter, Dormouse
CICD-SEC-5	Insufficient PBAC	Caterpillar
CICD-SEC-6	Insufficient Credential Hygiene	Duchess, Caterpillar
CICD-SEC-7	Insecure System Config	Hearts, Cheshire Cat
CICD-SEC-8	Ungoverned 3rd-Party Services	Gryphon
CICD-SEC-9	Improper Artifact Integrity	Dodo, Gryphon
CICD-SEC-10	Insufficient Logging & Visibility	(Cross-cutting)

Figure 2.2: OWASP Top 10 CI/CD Security Risks (2022) mapped to lab challenge categories.

2.4 From Source Access to Pipeline Compromise

The lab evidence is consistent with what CISA and NSA documented in their joint 2023 guidance on defending CI/CD environments: the most dangerous CI/CD attack paths do not start with a remote code execution exploit against the build platform. They start with repository access, fork rights, or package upload permissions—all positions that organizations routinely grant to developers, contractors, and automation accounts.

CHAPTER 3

Background and Related Security Concepts

3.1 CI/CD Architecture in Brief

The CI/CD Goat lab models a realistic internal delivery stack: Gitea and GitLab for source control, a Jenkins orchestrator with its agents, LocalStack for simulated AWS services, Docker-in-Docker workers, a `prod` host for helper script deployment, and CTFd for challenge validation.

3.2 The Trust Zone Model

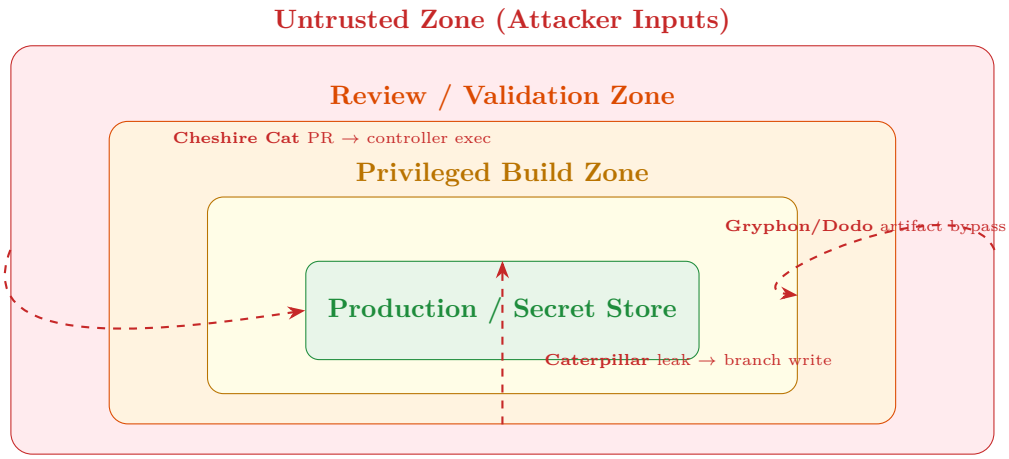

Figure 3.1: CI/CD trust zone model. Red arrows indicate the trust boundary violations observed in the lab.

3.3 Recurring Weakness Classes

- Direct poisoned pipeline execution through repository-controlled pipeline code

- Indirect pipeline execution through mutable helper scripts or Makefiles
- Secret exposure through logs, history, or over-scoped job credentials
- Runner and controller abuse through unsafe execution placement
- Dependency-chain abuse through internal packages and mutable base images
- Weak automation logic around webhook processing or branch merging
- IaC scanner bypass through deployment-time side effects

CHAPTER 4

Scope, Environment, and Methodology

4.1 Authorized Scope

The assessment was restricted to the local services explicitly provided by the lab:

- CTFd: `http://localhost:8000`
- Jenkins: `http://localhost:8080`
- Gitea: `http://localhost:3000`
- GitLab: `http://localhost:4000`

No systems outside localhost were targeted. No public CVEs against the platform itself were used.

4.2 Environment Overview

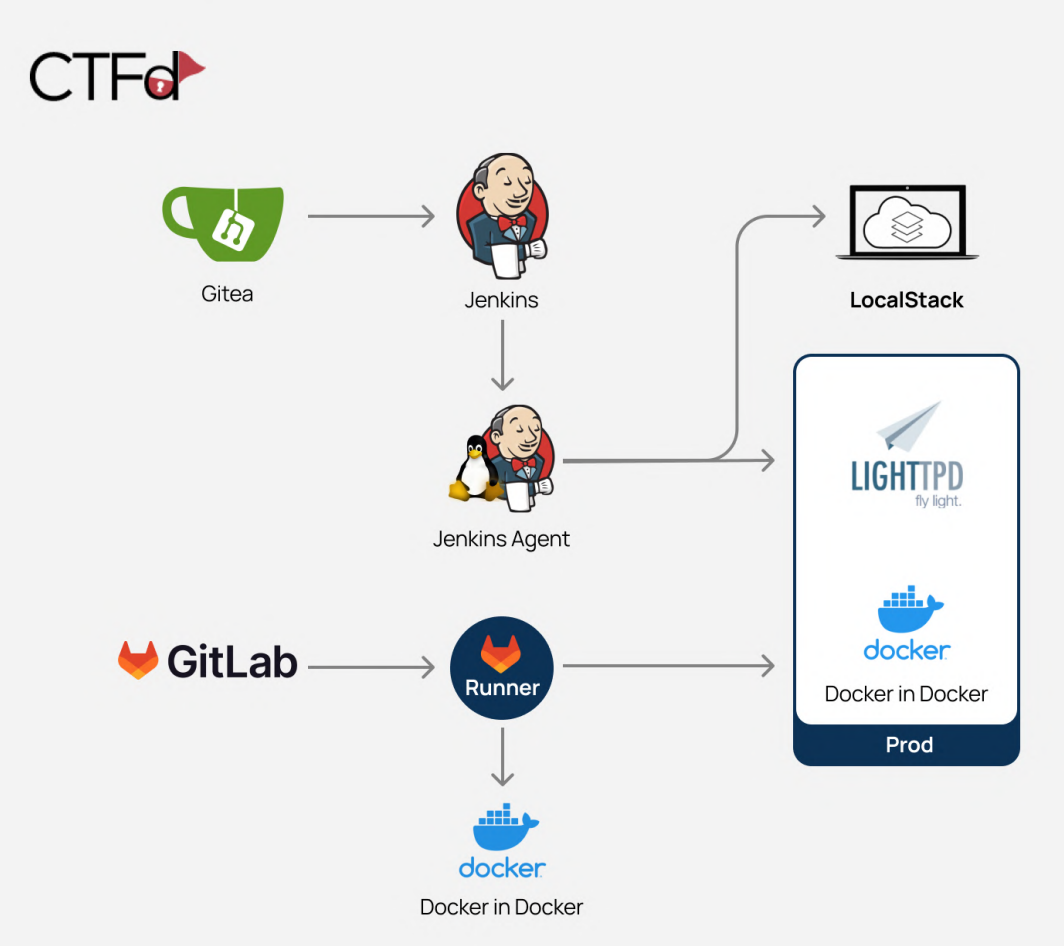

Figure 4.1: CI/CD Goat architecture diagram showing the service relationships relevant to pipeline and deployment abuse.

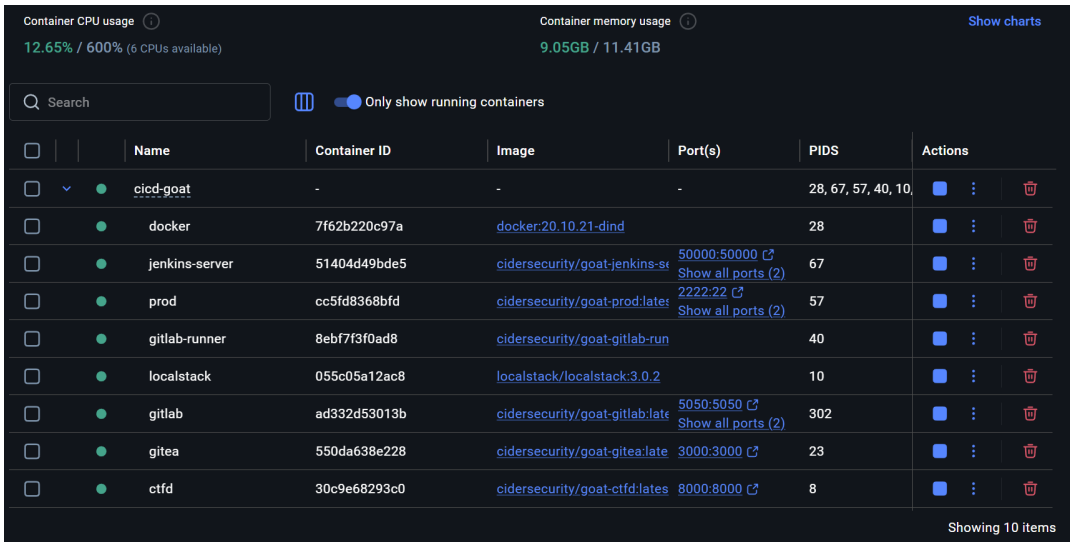

Figure 4.2: Running CI/CD Goat containers in Docker Desktop confirming all local services were active.

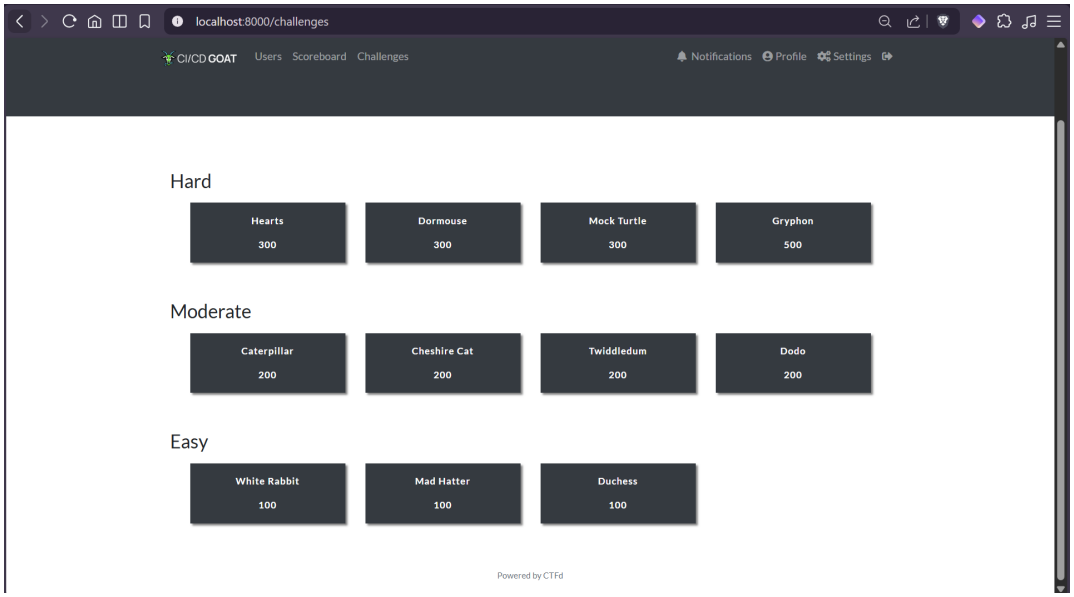

Figure 4.3: CTFd challenge interface after environment startup.

4.3 Service Access

Service	URL	Username	Password
CTFd	`http://localhost:8000`	`alice`	`alice`
Jenkins	`http://localhost:8080`	`alice`	`alice`
Gitea	`http://localhost:3000`	`thealice`	`thealice`
GitLab	`http://localhost:4000`	`alice`	`ali12345`

Table 4.1: Default service credentials used during the lab assessment.

4.4 Methodological Flow

Each challenge followed a repeatable workflow: CTFd prompt review → trust-boundary hypothesis → repository and job inspection → narrow exploitation to confirm the weakness → flag capture and CTFd validation → risk mapping and controls derivation.

4.5 Challenge Status

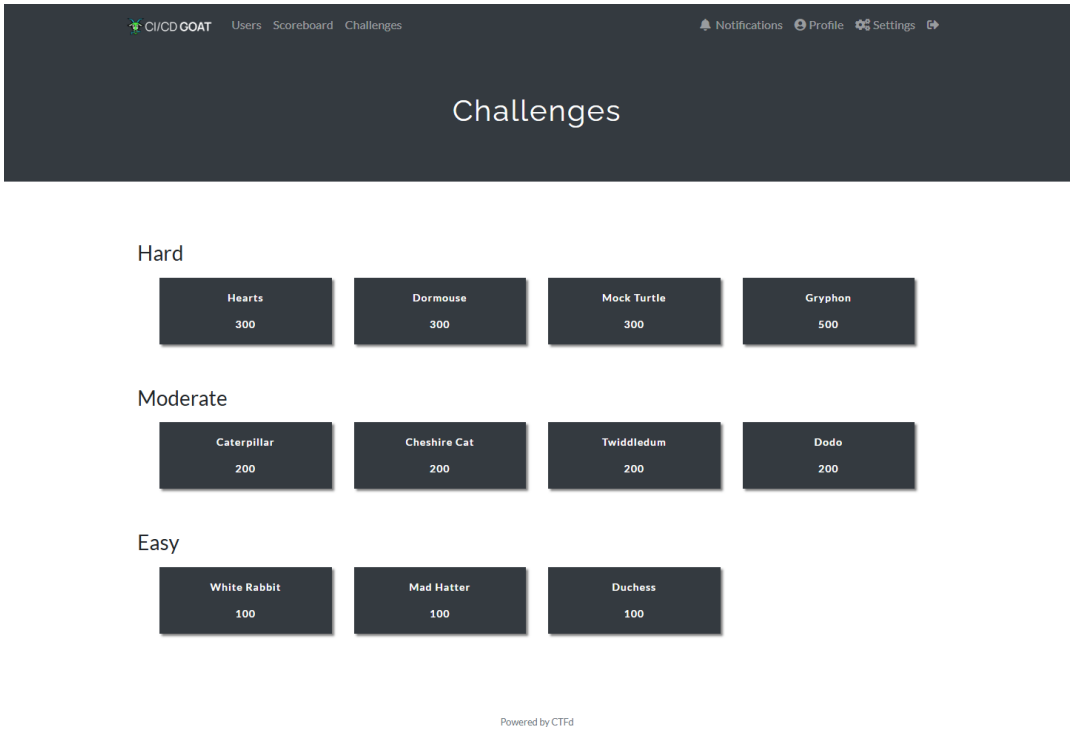

Figure 4.4: CTFd challenge list after completion.

Challenge	Platform	Primary Weakness	Risk Class	Status
White Rabbit	Jenkins + Gitea	Repo-controlled Jenkinsfile	Direct PPE	Done
Caterpillar	Jenkins + Gitea	PR pipeline leaks write token	Token escalation	Done
Mad Hatter	Jenkins + Gitea	Makefile bypass	Indirect PPE	Done
Duchess	Gitea	Secret in Git history	Secret hygiene	Done
Cheshire Cat	Jenkins + Gitea	Controller node execution	Runner isolation	Done
Twiddledum	Jenkins + Gitea	Trusted package abuse	Dep-chain abuse	Done
Dodo	Jenkins + Terraform	Scanner bypass + public bucket	IaC bypass	Done
Hearts	Jenkins	Agent admin credential abuse	Credential replay	Done
Dormouse	Jenkins + Gitea	External script poisoning	Indirect PPE / webhook	Done
Mock Turtle	Jenkins + Gitea	Weak auto-merge control	Flow-control failure	Done
Gryphon	GitLab	Package + image supply chain	Supply-chain abuse	Done

Table 4.2: Challenge summary with weakness class and status.

CHAPTER 5

Challenge Assessments

5.1 White Rabbit — Direct Poisoned Pipeline Execution

▷ Context

White Rabbit tested the trust relationship between Gitea and Jenkins. The repository exposed a root `Jenkinsfile`, and the corresponding multibranch job executed it during pull request builds.

▷ Observed Behavior

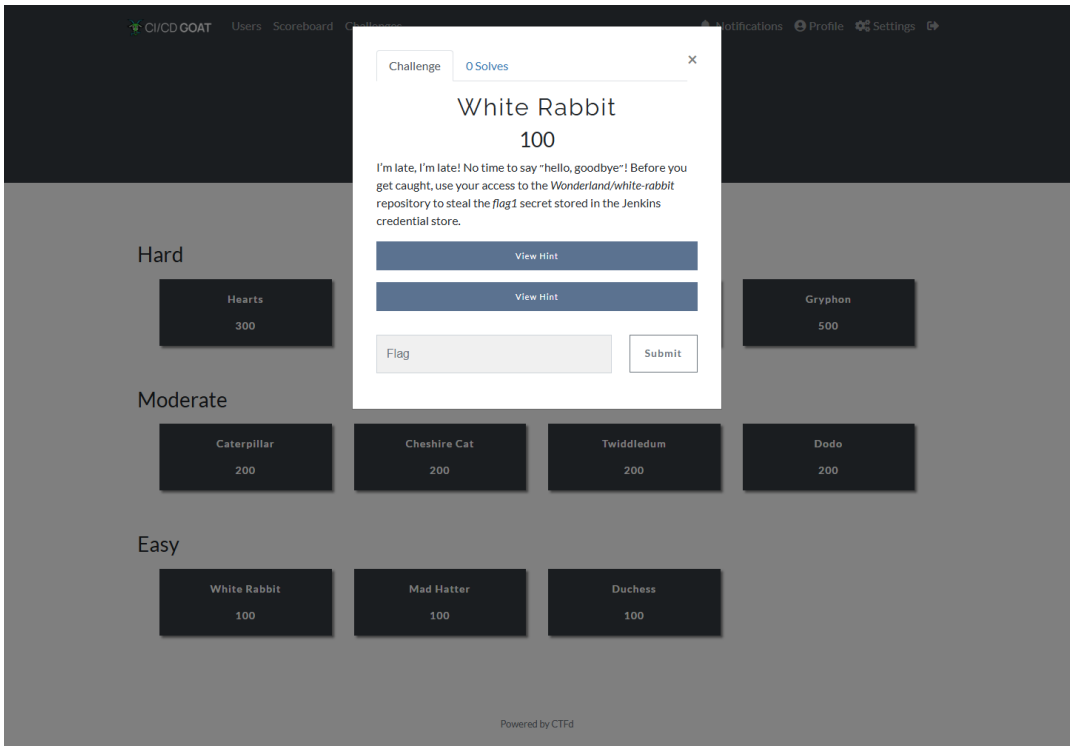

Figure 5.1: White Rabbit challenge prompt.

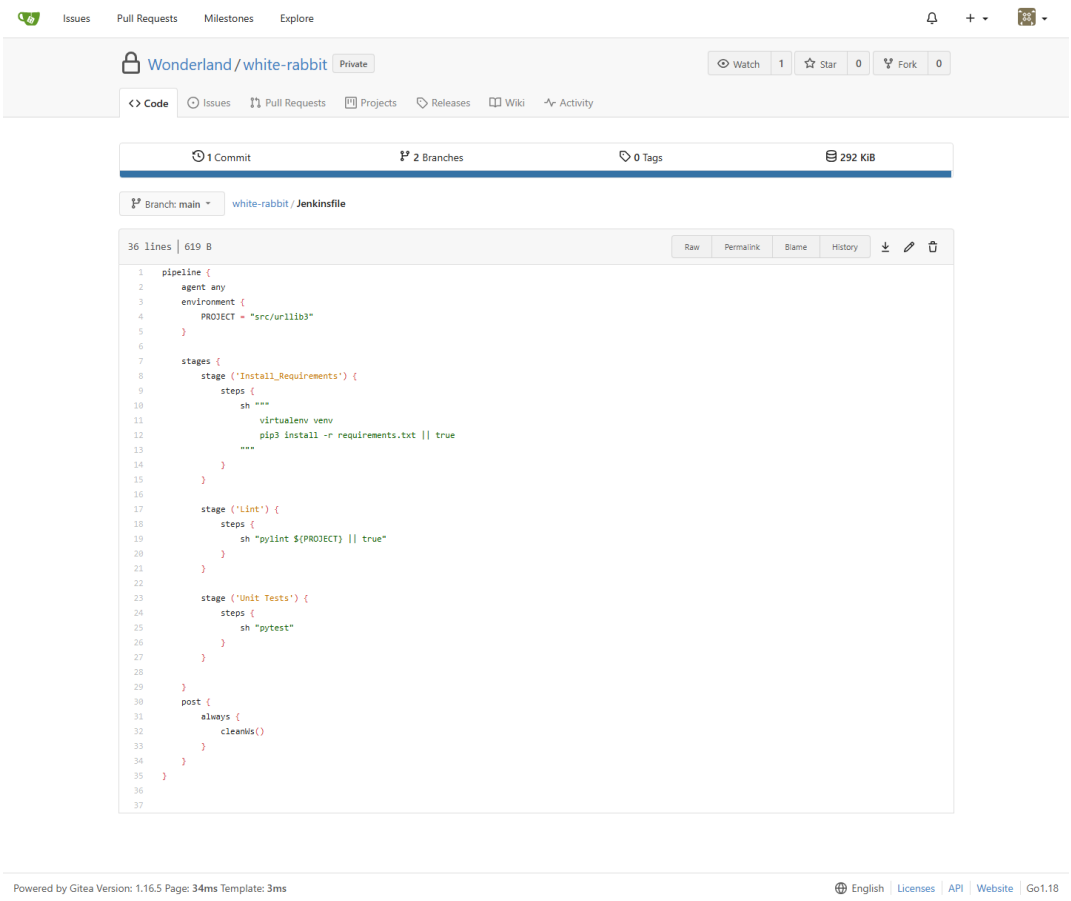

Figure 5.2: Repository Jenkinsfile reviewed from Gitea.

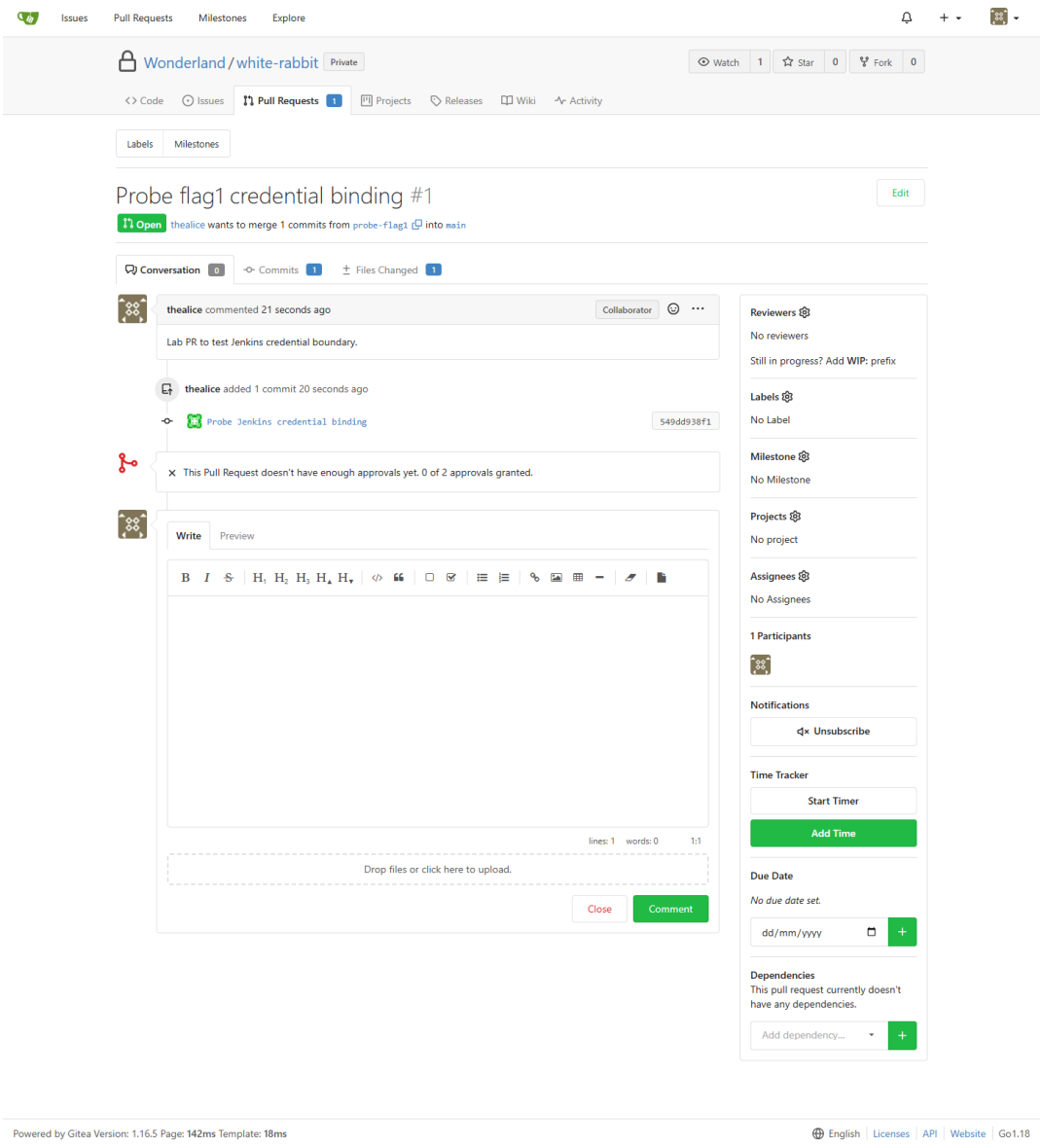

Figure 5.3: Pull request delivering the modified pipeline to Jenkins.

Direct pushes to `main` were blocked, but pull request execution ran in the same credential-bearing context.

Weakness: Direct PPE / Credential Exposure

Jenkins treated repository-controlled Groovy as trusted while binding `flag1` to the job. A pull request author had full control over what that binding executed.

Exploitation Path

Modify Jenkinsfile to bind `flag1`, base64-encode before print to bypass console masking, trigger via pull request. One build cycle to flag disclosure.

▷ Evidence

Figure 5.4: Pull request build disclosing the encoded secret.

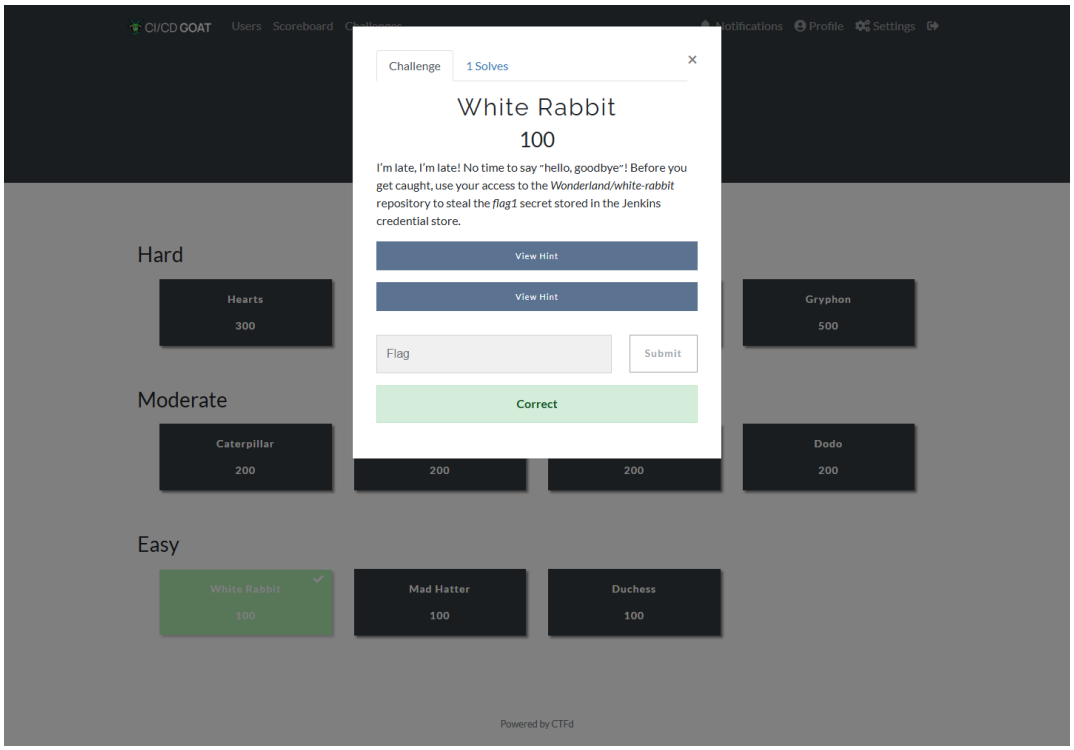

Figure 5.5: Accepted CTFd submission.

Flag: `06165DF2-C047-4402-8CAB-1C8EC526C115`

Remediation

Protect `Jenkinsfile` changes with CODEOWNERS or equivalent. Run untrusted PRs under a *trusted* pipeline definition stored outside the repository. Remove secret-bearing stages from PR jobs entirely.

5.2 Caterpillar — PR Pipeline Token Leakage

▷ Context

Caterpillar began from read-only repository visibility. The question was whether a fork-based PR could reach a Jenkins secret without direct write access.

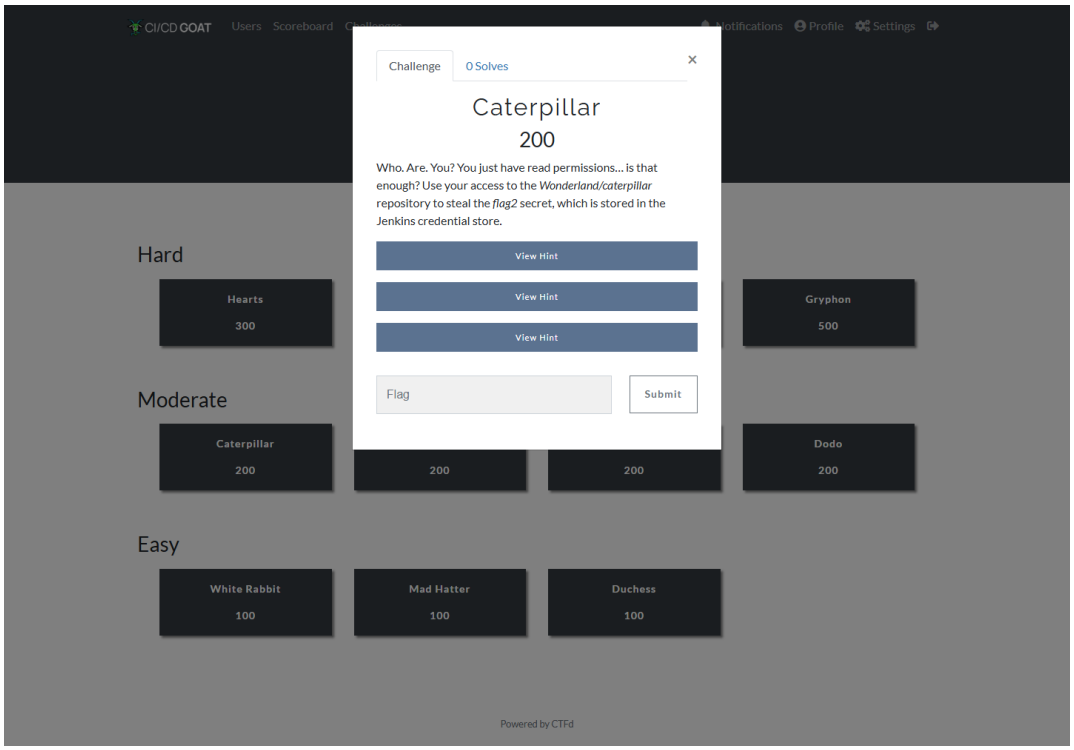

Figure 5.6: Caterpillar prompt.

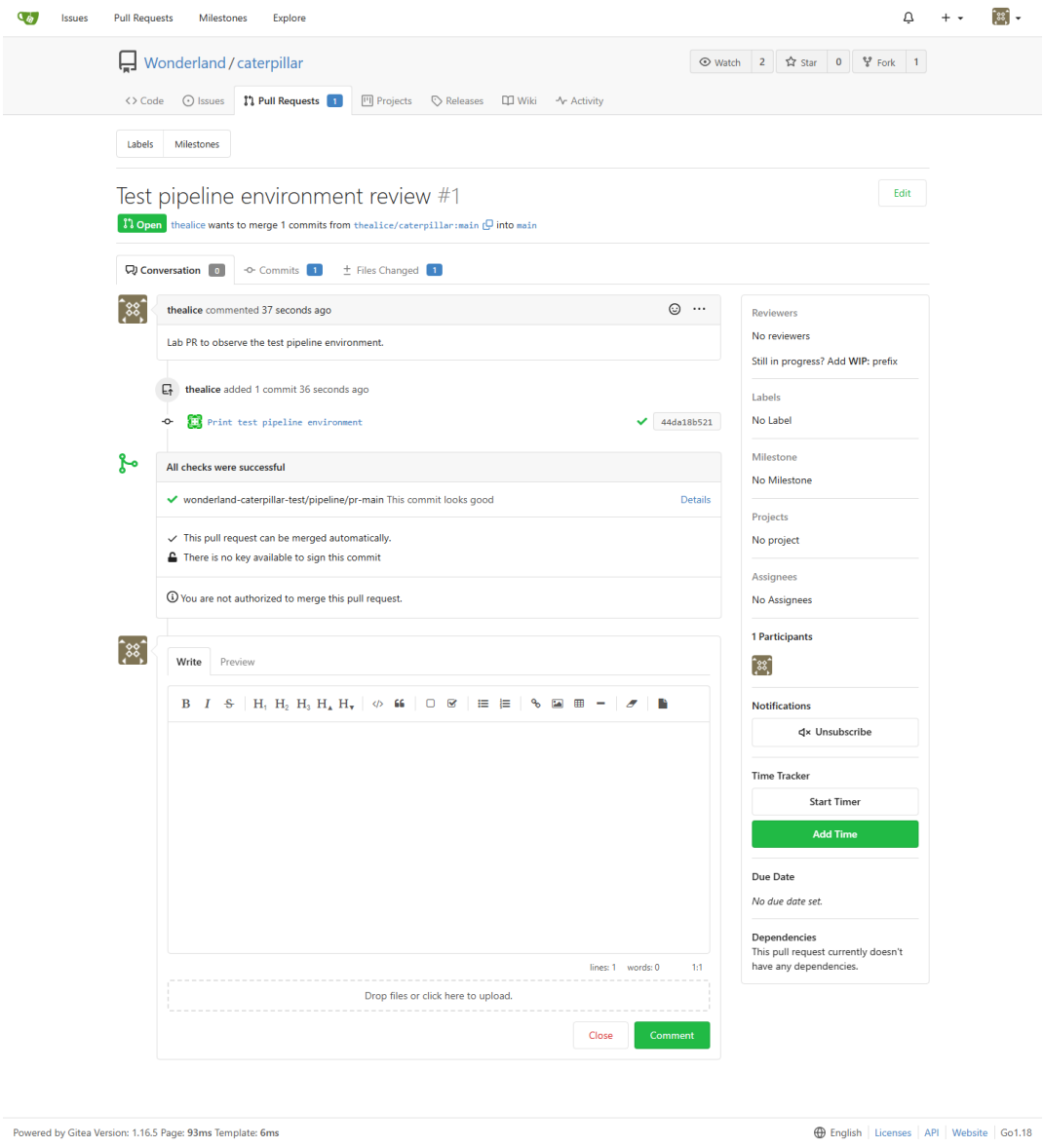

Figure 5.7: Fork-based PR triggering the lower-trust build path.

Figure 5.8: `GITEA_TOKEN` exposed in the fork build output.

Weakness: PBAC Failure / Token Escalation

The fork validation job executed untrusted code in a context that held a write-capable SCM token. Read-only visibility was enough to begin the escalation chain.

Exploitation Path

Fork the repo → modify Jenkinsfile to dump the environment → recover `GITEA_TOKEN` from build output → reuse token to push a malicious Jenkinsfile to `main` → second build discloses `flag2`. Two build cycles, zero write permissions required initially.

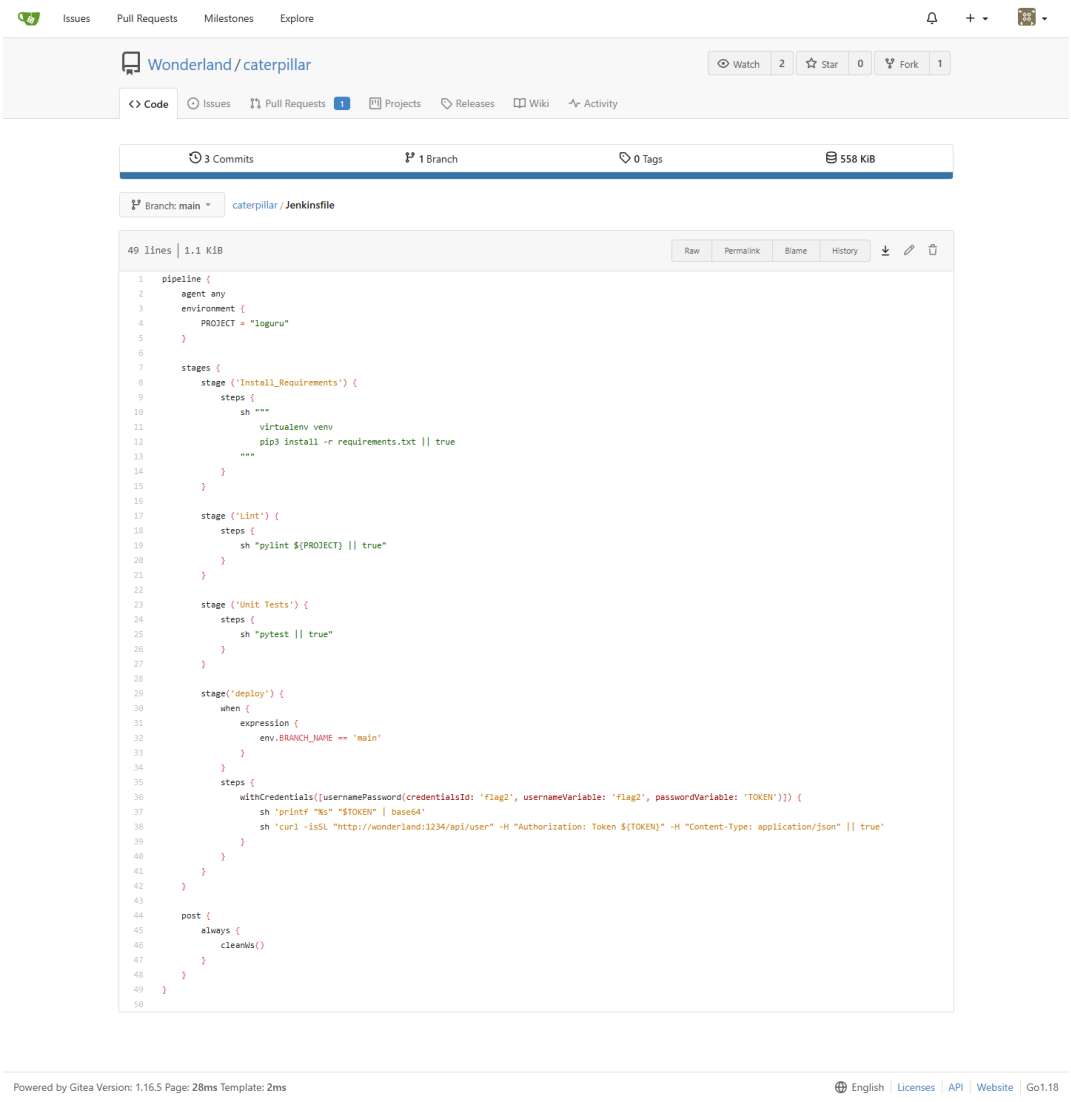

Figure 5.9: Production Jenkinsfile modified using the leaked token.

Figure 5.10: Production build disclosing the flag after trusted-branch modification.

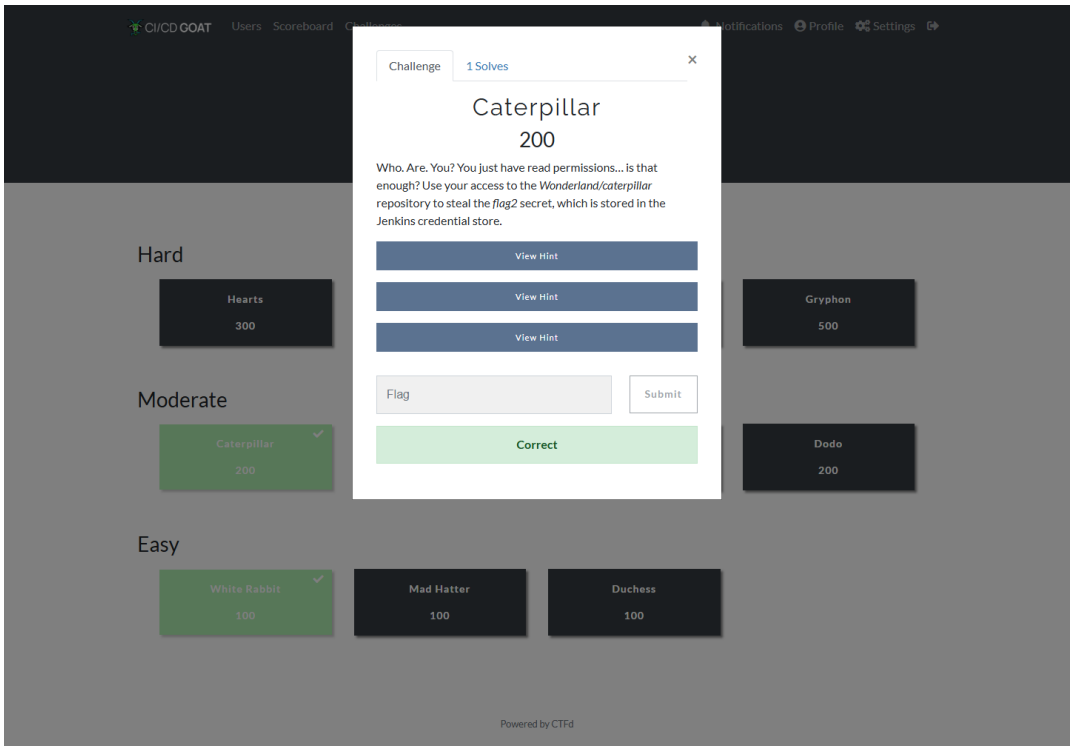

Figure 5.11: Accepted CTFd submission.

Flag: `AEB14966-FFC2-4FB0-BF45-CD903B3535DA`

Remediation

Fork validation jobs must run with read-only or no SCM token. Test-job credentials must not be able to modify protected branches. Separate testing identity from the release identity entirely.

5.3 Mad Hatter — Protected Pipeline Bypass via Build Script Control

▷ Context

Mad Hatter's Jenkinsfile was explicitly protected. The question was whether protection of the top-level file was meaningful when the build delegated execution to a mutable `Makefile`.

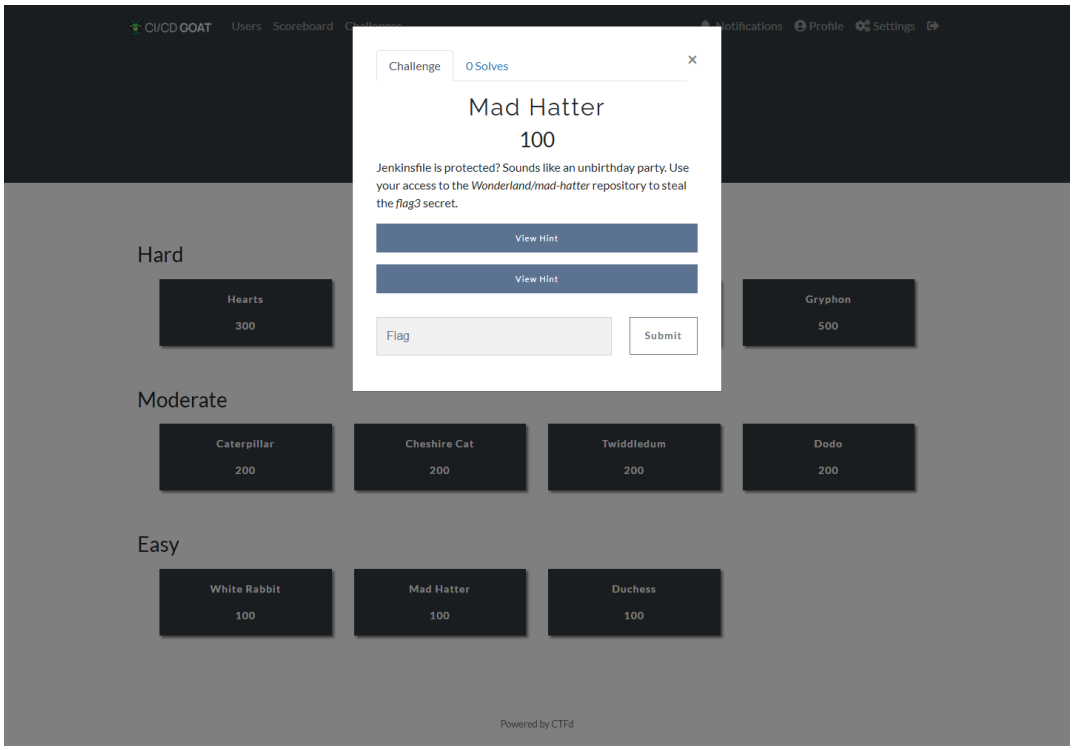

Figure 5.12: Mad Hatter prompt.

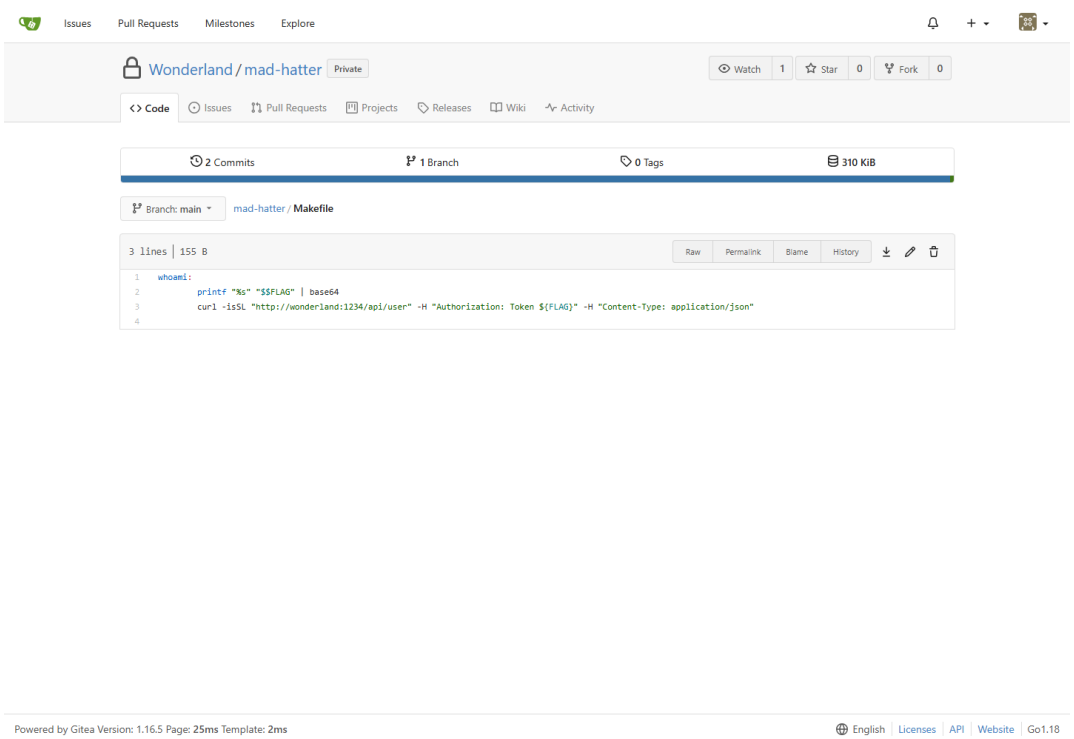

Figure 5.13: Makefile used as the real execution entry point.

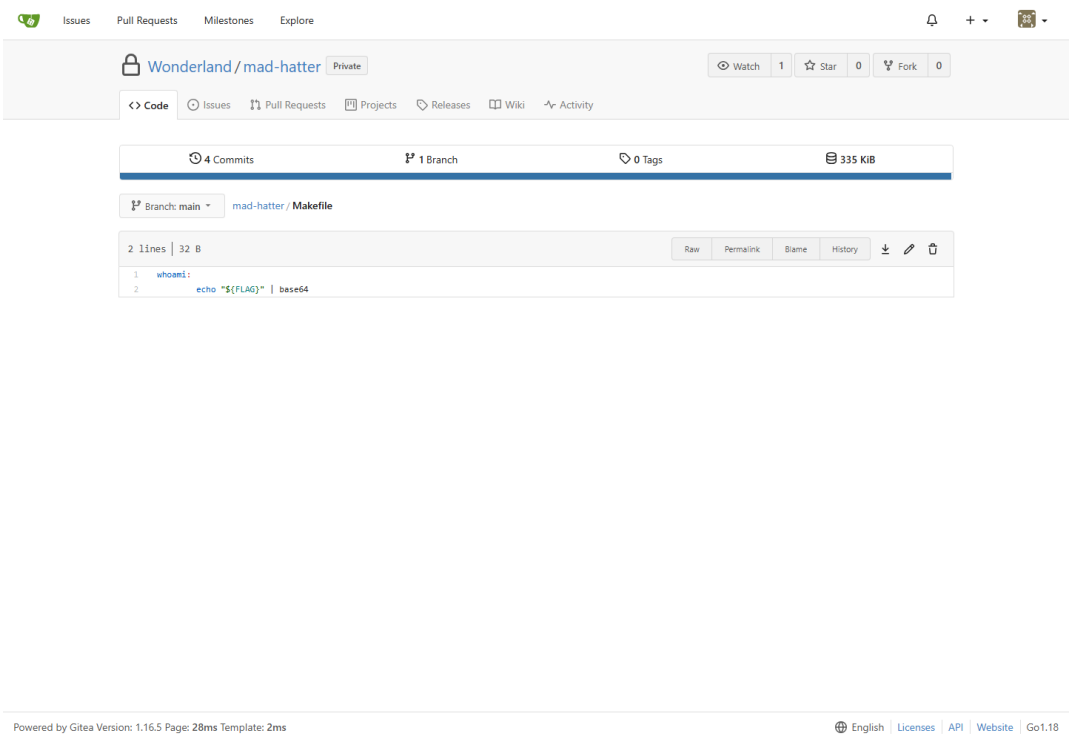

Figure 5.14: Modified Makefile payload reaching the credential-bearing step indirectly.

Weakness: Indirect PPE — Delegated Build Script

The protected pipeline delegated to a repository-controlled Makefile before secrets were bound. Protecting the Jenkinsfile alone created a false sense of control over the full build path.

Exploitation Path

Modify the Makefile target invoked by the protected pipeline so the build discloses the bound flag variable during normal execution. No Jenkinsfile change required.

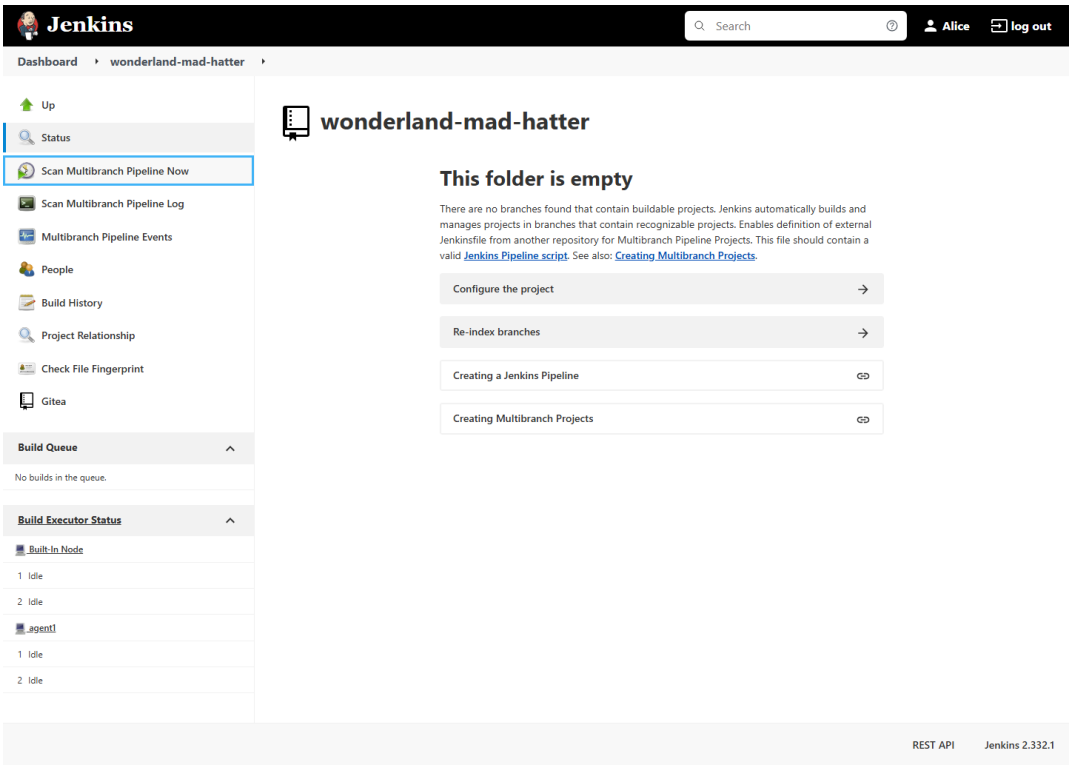

Figure 5.15: Jenkins scan behavior after the Makefile payload.

Remediation

Treat Makefiles, shell helpers, and any file invoked by the pipeline as part of the trusted surface. Apply the same review policy as for the Jenkinsfile. Do not bind secrets before the build input is pinned to a reviewed revision.

5.4 Duchess — Secret Exposure in Git History

▷ Context

Duchess was a source hygiene task. The objective was to recover a PyPI credential removed from the current tree but still present in repository history.

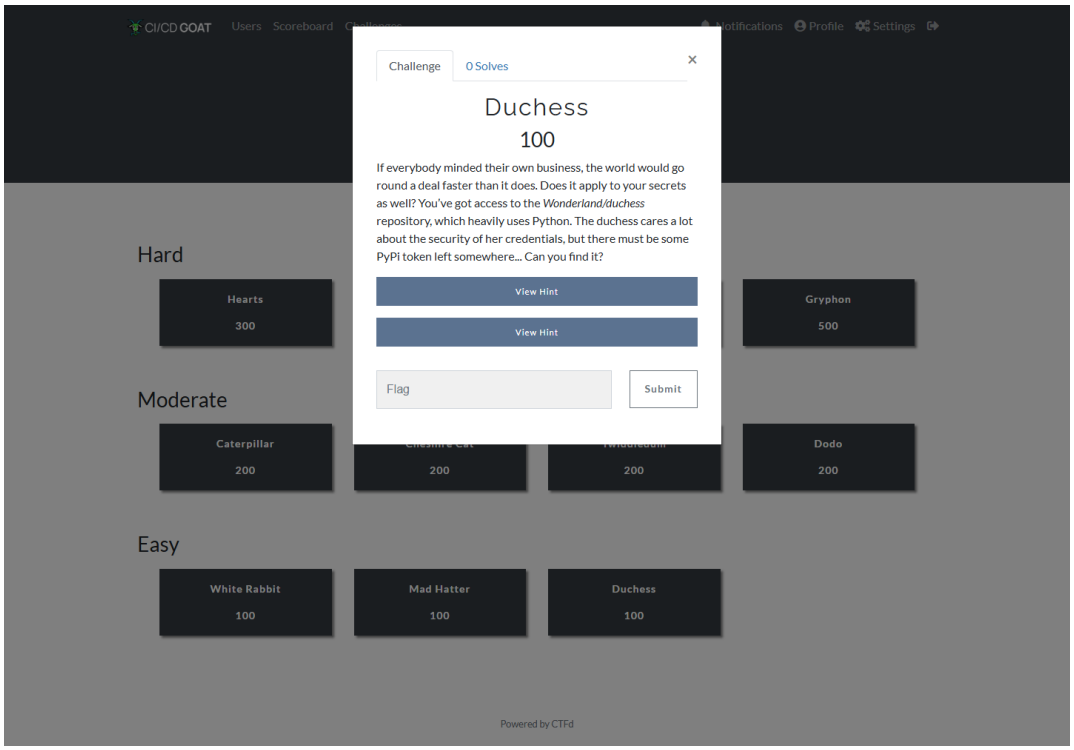

Figure 5.16: Duchess prompt.

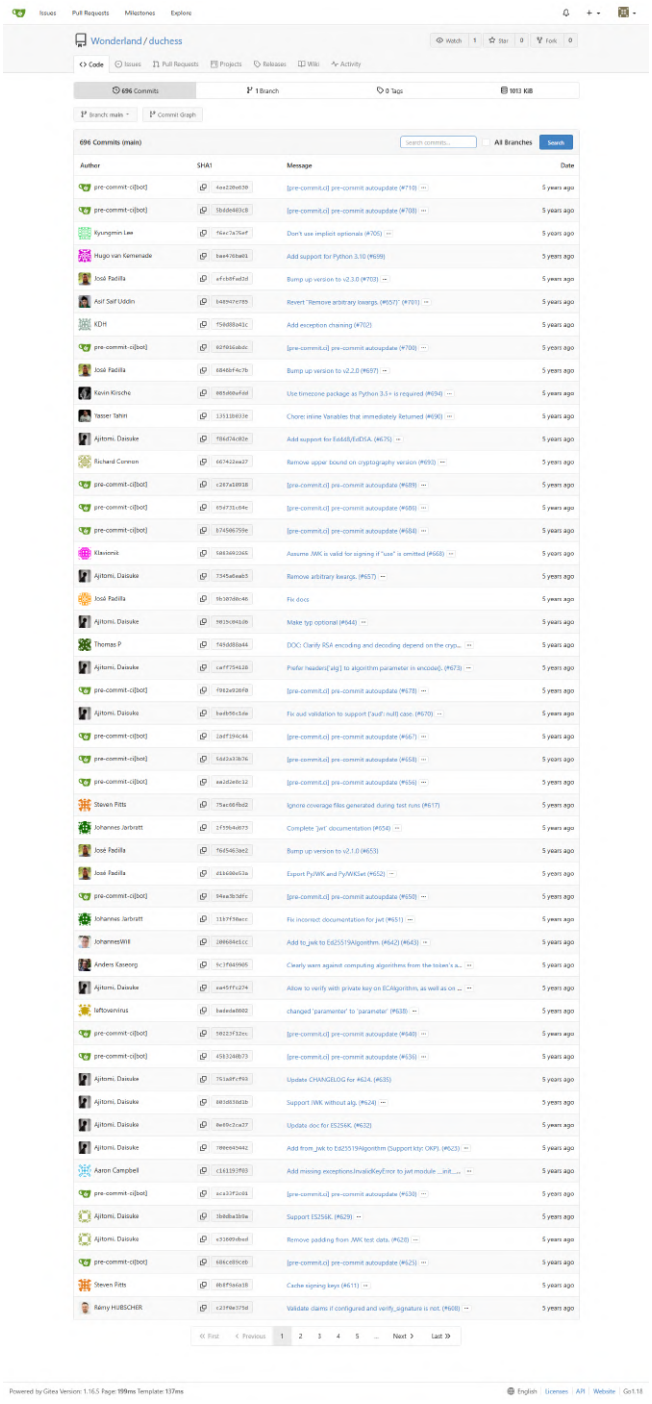

Figure 5.17: Commit history revealing a token-removal event.

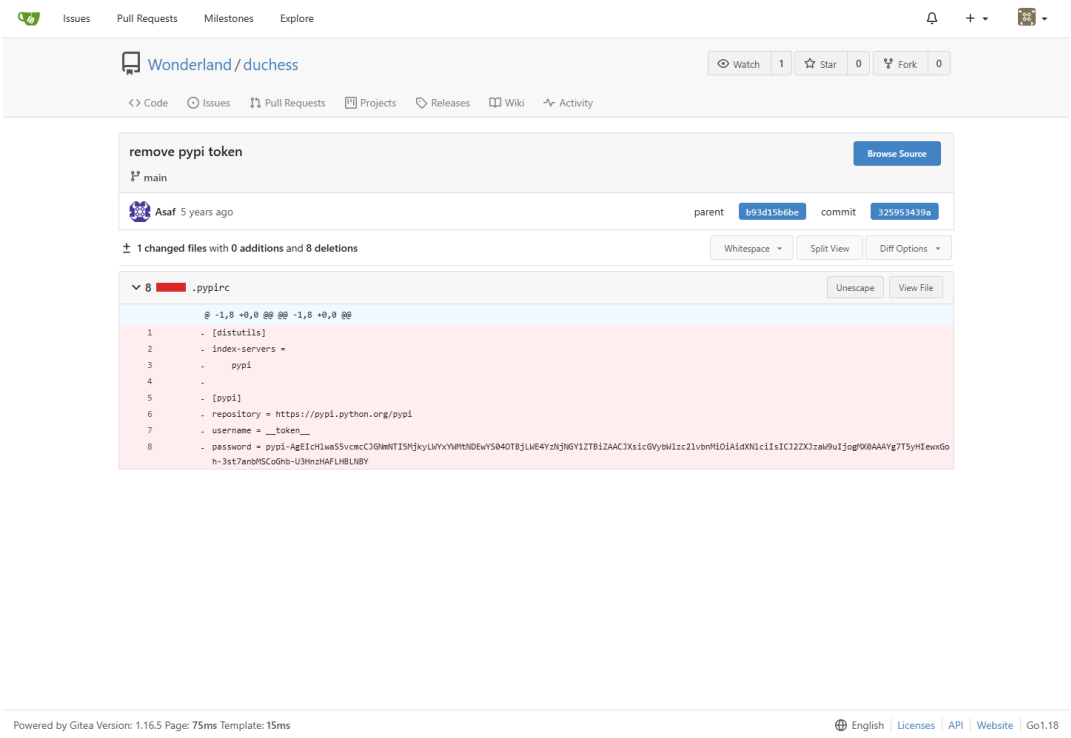

Figure 5.18: Deleted `.pypirc` recovered from Git history.

Weakness: Credential Committed to History

The repository owner treated current-tree state as the full exposure surface. Historical objects are permanently accessible to anyone with clone rights.

Exploitation Path

`gitlog--all--diff-filter=D` → identify the removal commit → `gitshowCOMMIT:~{}/.pypirc` → extract PyPI token. No pipeline interaction needed.

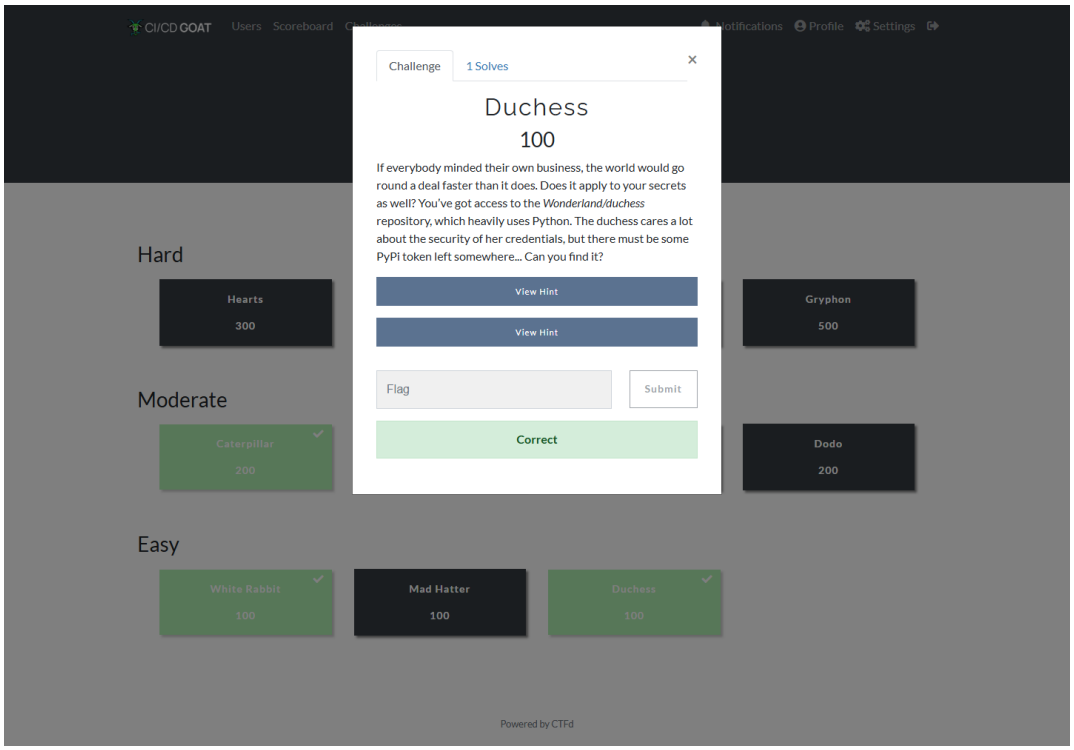

Figure 5.19: Accepted CTFd submission.

Remediation

Pre-commit and pre-push secret scanning on all repositories. Rotate any token found in history immediately. Audit repository history before broadening access or making a project public.

5.5 Cheshire Cat — Jenkins Controller Execution via Agent Label Abuse

▷ Context

Cheshire Cat targeted execution placement. The goal was to move repository-controlled pipeline logic from a build agent onto the Jenkins controller itself.

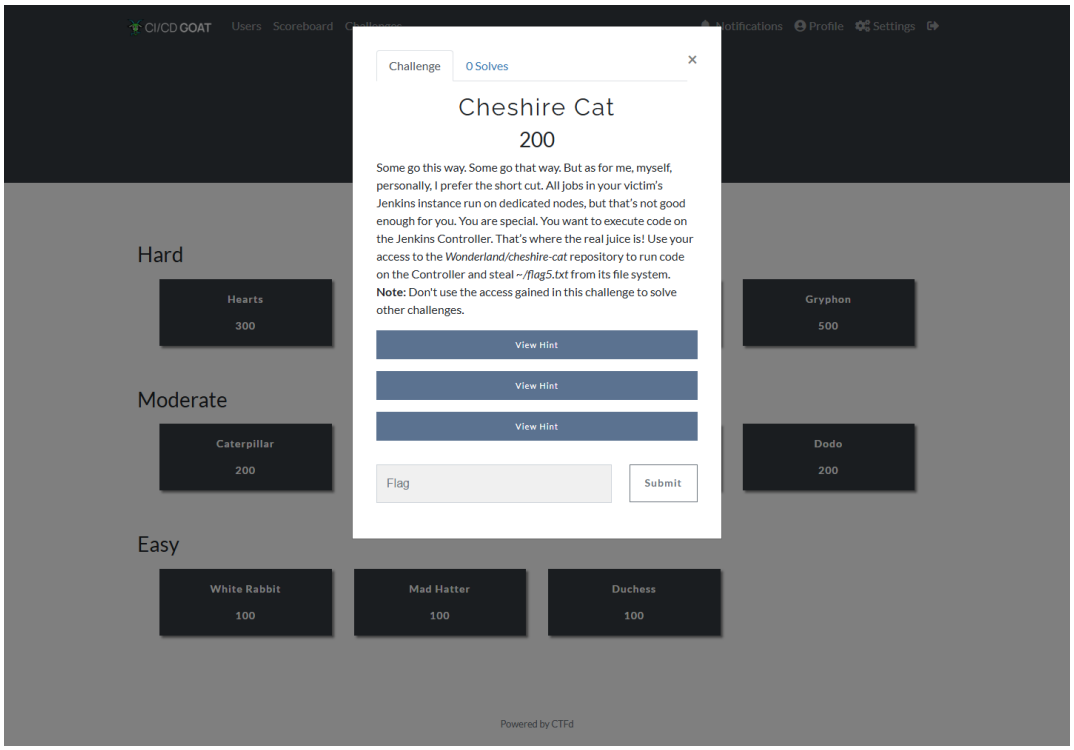

Figure 5.20: Cheshire Cat prompt.

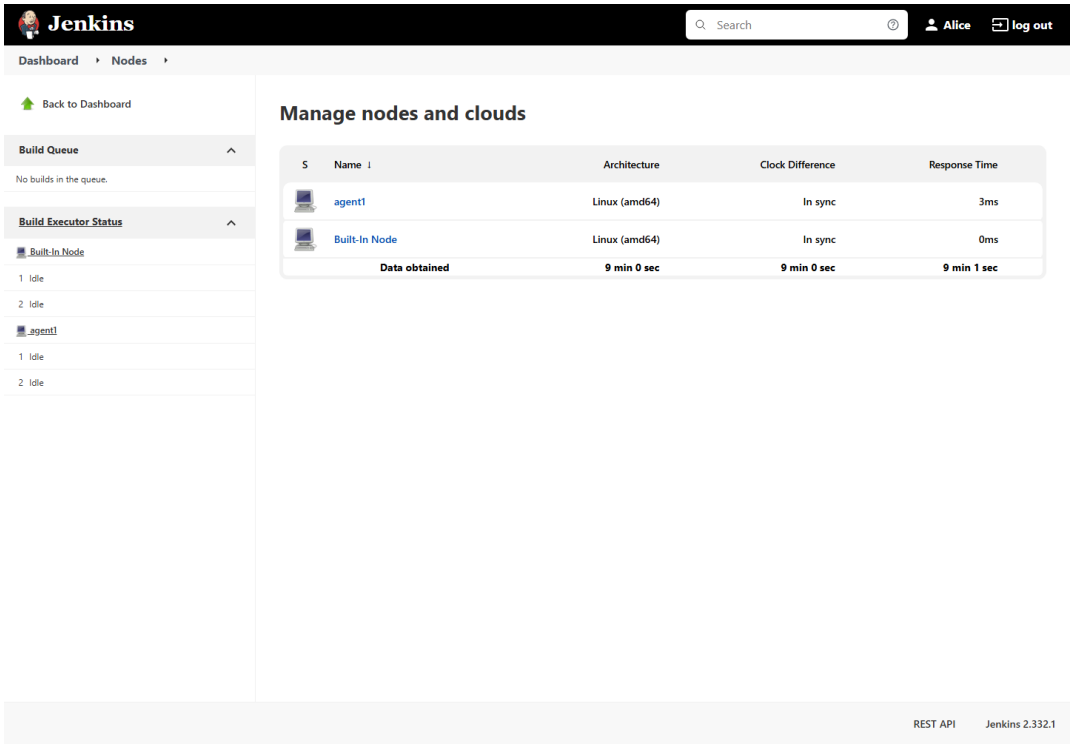

Figure 5.21: Jenkins nodes view exposing the `built-in` controller label.

Weakness: Runner Isolation Failure

Repository-controlled pipeline code could select any execution target, including the privileged controller. The controller label was not restricted to trusted jobs.

Exploitation Path

Change pipeline agent to `label'built-in'` → PR triggers controller execution → read `~{}/flag5.txt` on the controller host → base64 encode and print in build log.

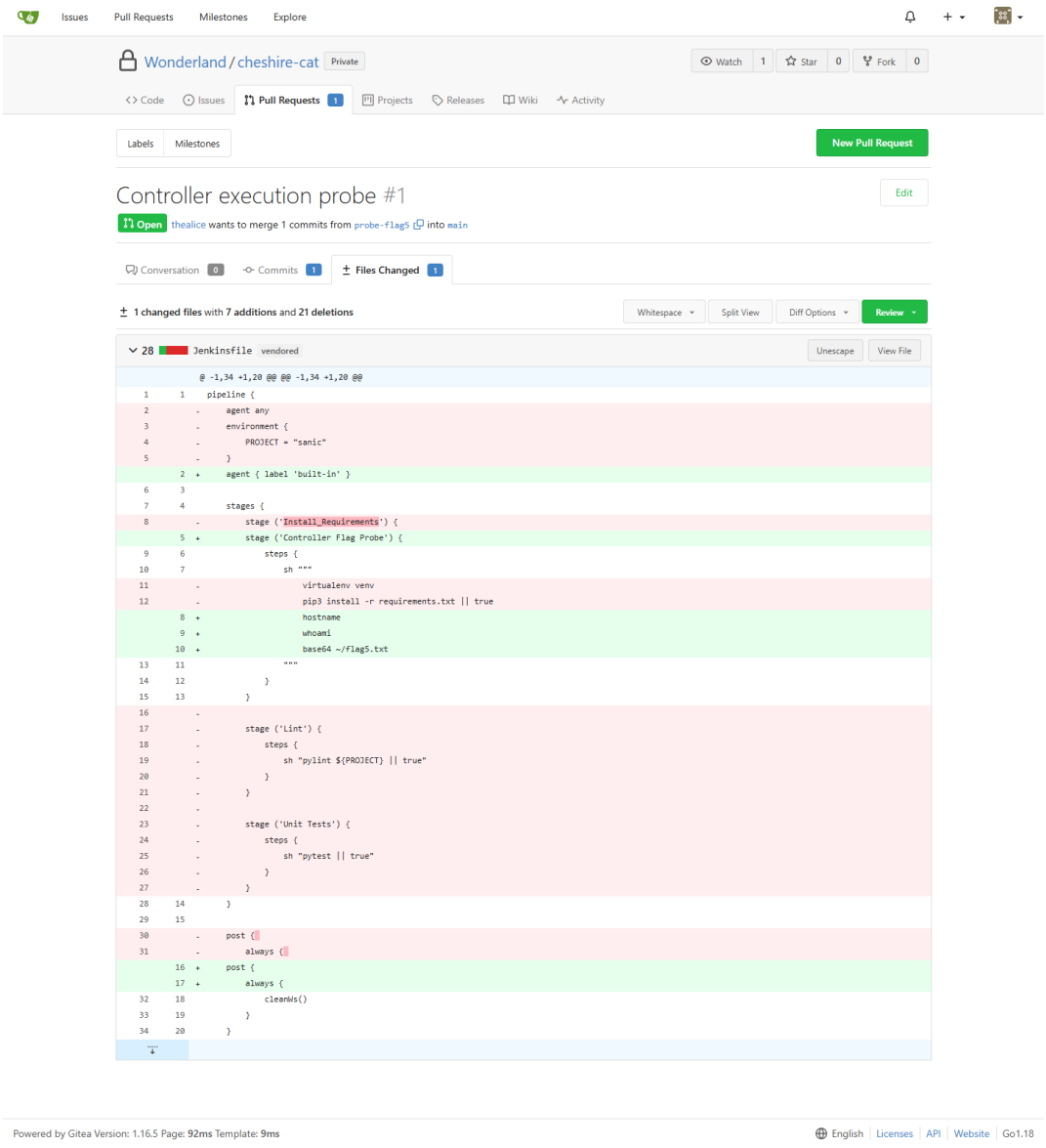

Figure 5.22: PR payload forcing controller node selection.

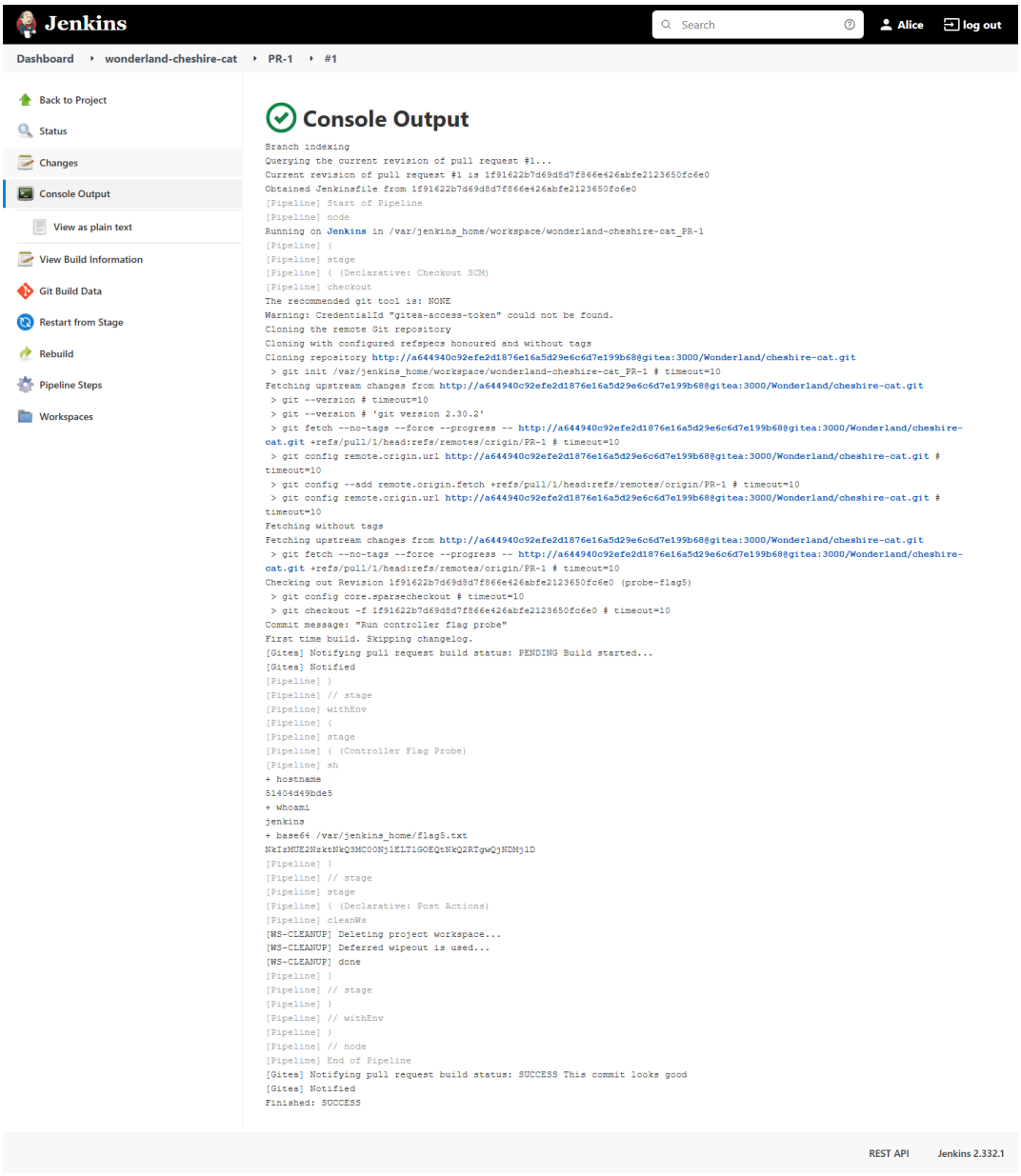

Figure 5.23: Controller-backed build disclosing the encoded flag.

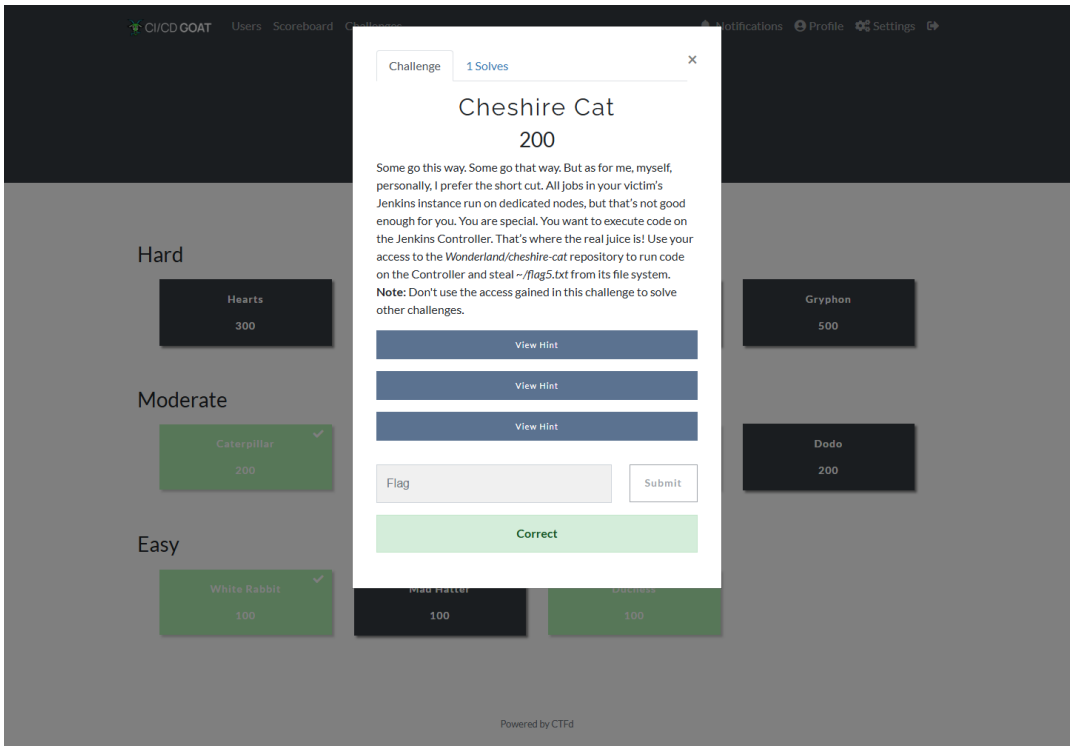

Figure 5.24: Accepted CTFd submission.

Flag: `6B31A679-6D70-469D-9F8D-6D6E80B3C29C`

Remediation

Disable controller executors. Restrict agent label selection in pipeline syntax for untrusted jobs. Route PR builds to dedicated ephemeral agents with no path to the controller or its credential store.

5.6 Twiddledum — Dependency Chain Abuse via Trusted Package Poisoning

▷ Context

Twiddledum revolved around an internal package dependency. The downstream repo was protected; the upstream package was not.

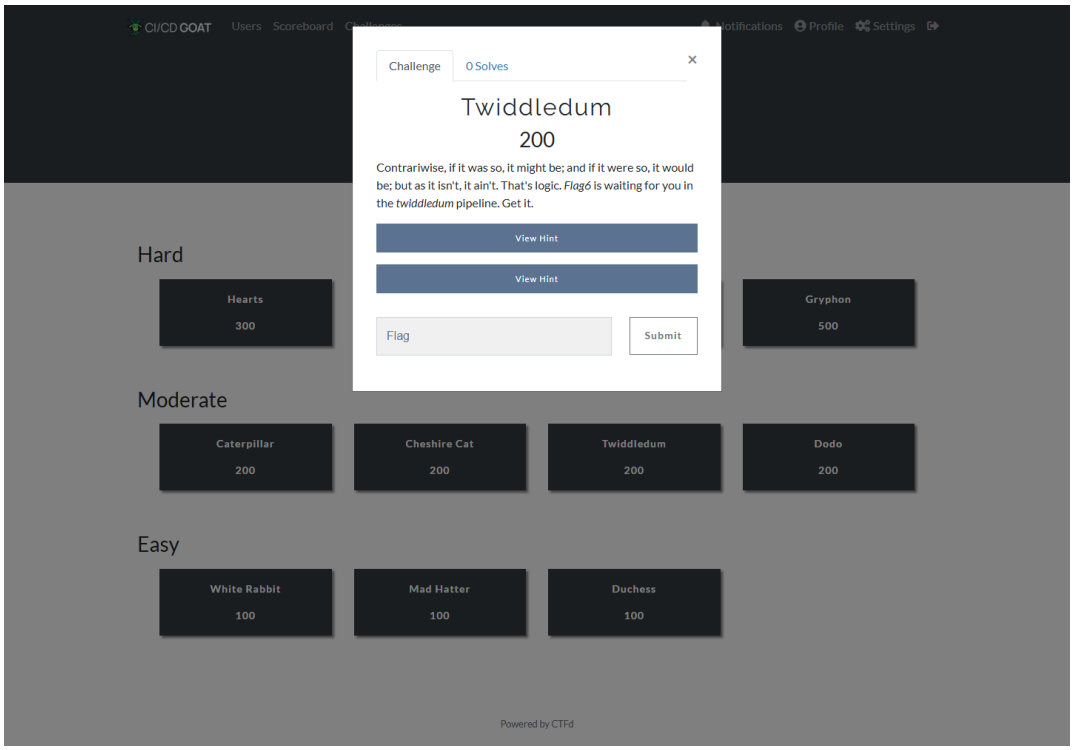

Figure 5.25: Twiddledum prompt.

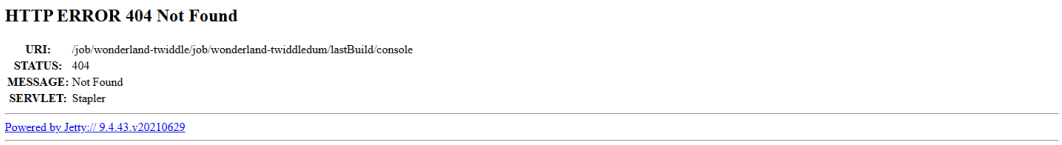

Figure 5.26: Build console showing dependency installation followed by application execution.

Weakness: Mutable Internal Dependency

The pipeline assumed the referenced internal package version was safe. A writable upstream package repo allowed injection of code that ran in a `FLAG6`-bearing context.

Exploitation Path

Publish a modified version of the upstream package that prints `process.env.FLAG6` during import → next downstream build installs it and discloses the flag.

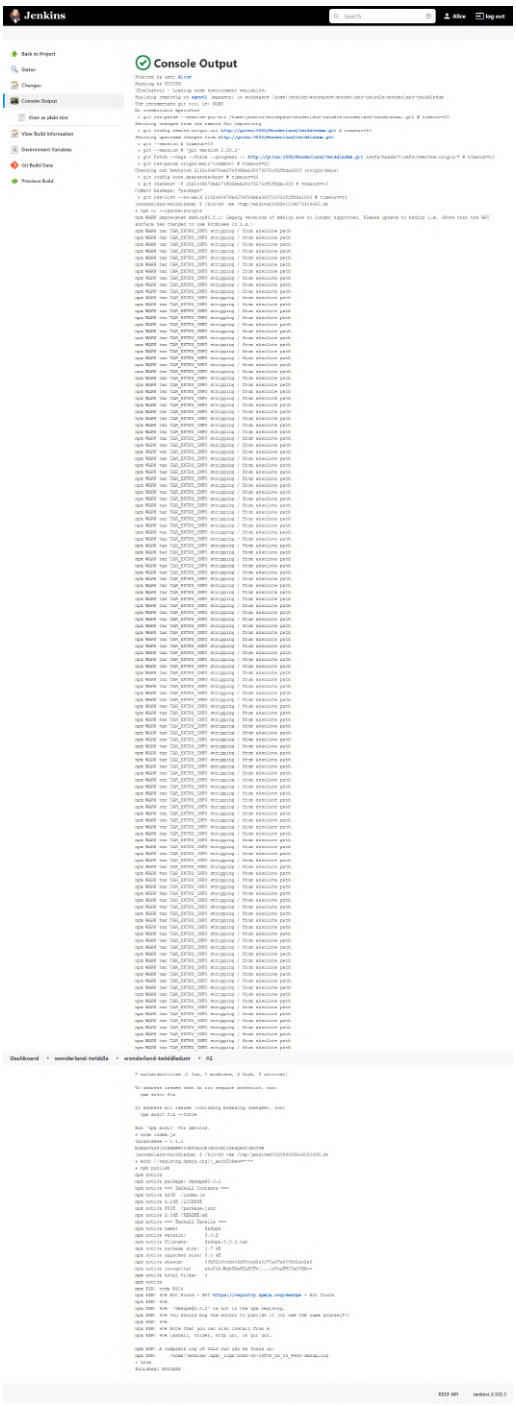

Figure 5.27: Downstream build disclosing the flag after dependency poisoning.

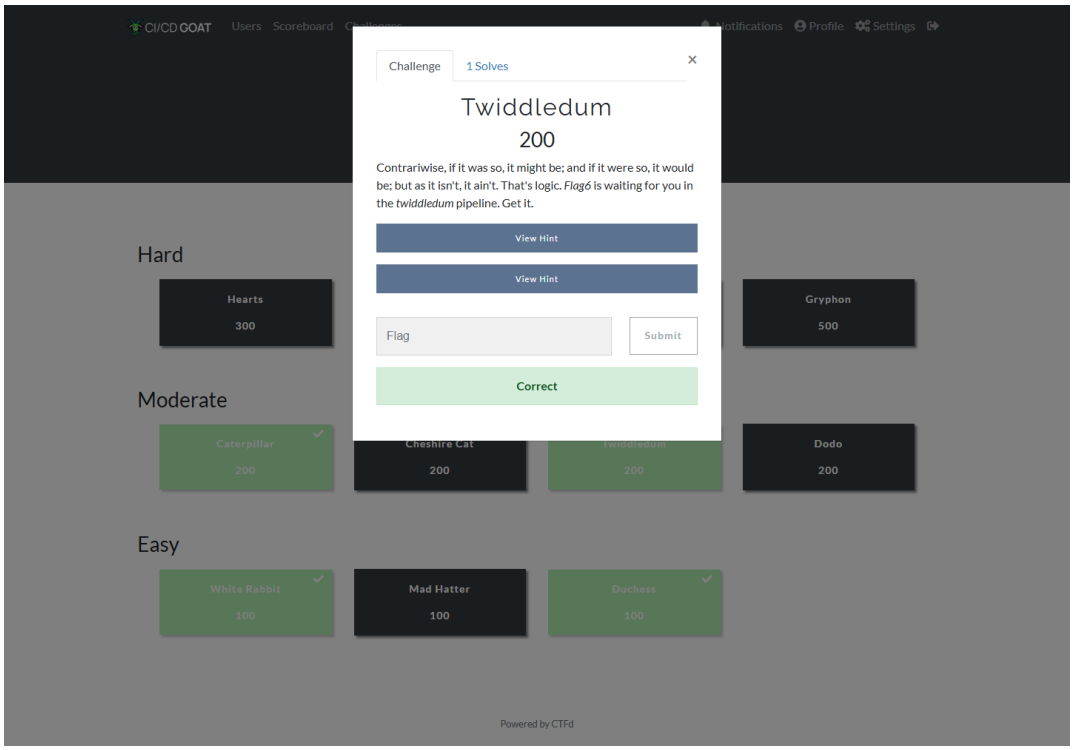

Figure 5.28: Accepted CTFd submission.

Flag: `710866F2-2CED-4E60-A4EB-223FD892D95A`

Remediation

Protect internal package publishing paths. Require reviewed releases and pin exact dependency versions from a controlled registry. Do not expose high-value secrets to jobs that execute unverified dependency code.

5.7 Dodo — IaC Scanner Bypass via Apply-Time Mutation

▷ Context

Dodo examined the gap between static Checkov scanning and actual Terraform apply behavior.

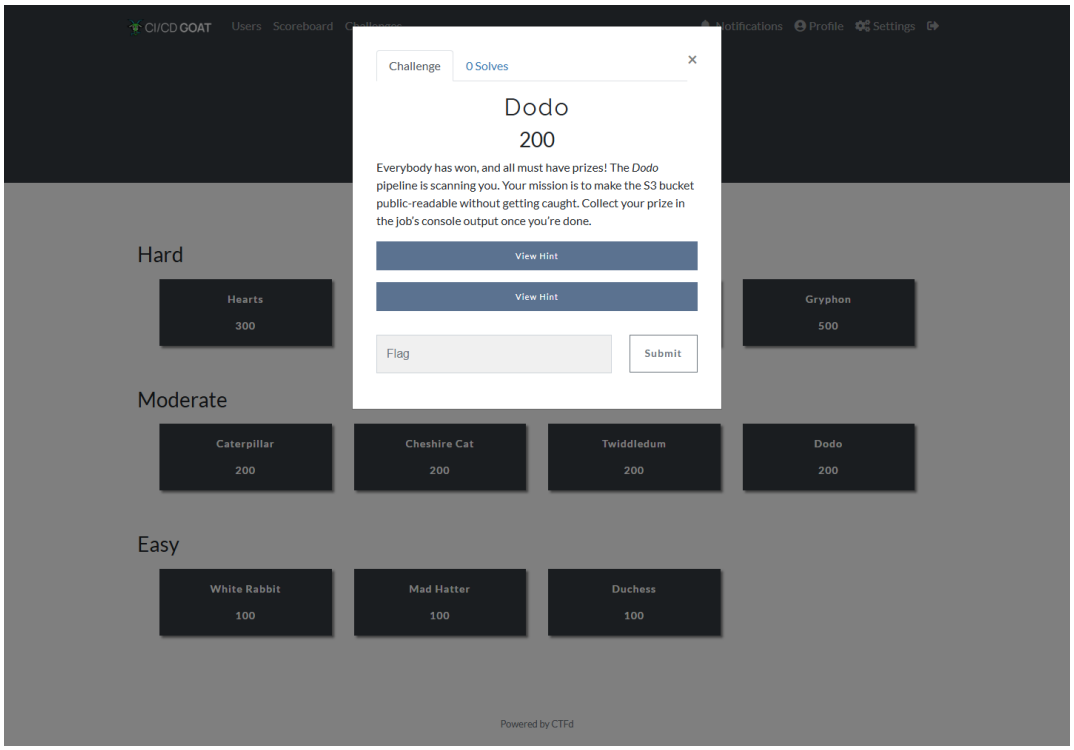

Figure 5.29: Dodo prompt.

Figure 5.30: Initial Terraform configuration before the payload.

Figure 5.31: Payload with `local-exec` provisioner modifying S3 bucket ACL post-scan.

Weakness: Static Scan Does Not Cover Runtime Effects

Checkov approved the static configuration. A `local-exec` provisioner ran `awslocal` after `terraformapply` began, mutating live bucket ACL to `public-read` outside the scanner's view.

Exploitation Path

Keep visible Terraform clean for Checkov → embed `local-exec` provisioner calling `awslocals3apiput-bucket-acl--aclpublic-read` → post-apply verification detects public bucket and produces the flag.

Figure 5.32: Jenkins output: Checkov passes, then bucket ACL mutated at apply time.

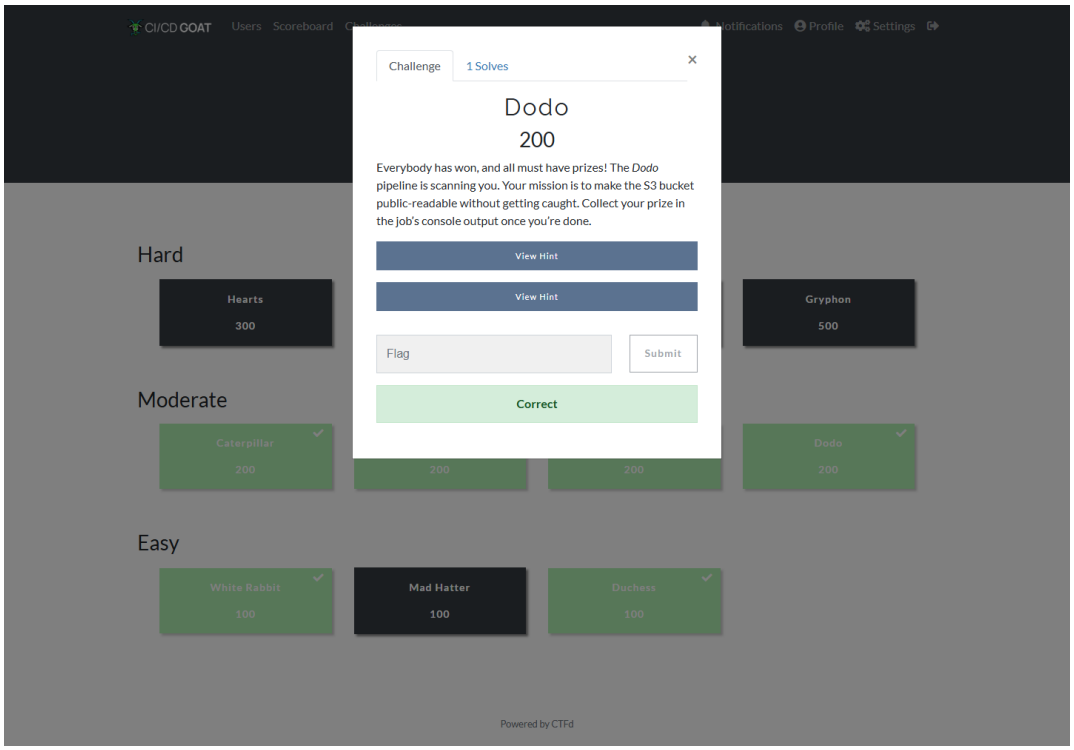

Figure 5.33: Accepted CTFd submission.

Flag: `A62F0E52-7D67-410E-8279-32447ADAD916`

Remediation

Keep policy enforcement outside the application repository. Restrict or ban `local-exec` in delivery pipelines. Validate resulting cloud state, not just static configuration.

5.8 Hearts — Jenkins Agent Credential Replay

▷ Context

Hearts targeted Jenkins administrative features. The abuse path was agent launcher configuration rather than pipeline code.

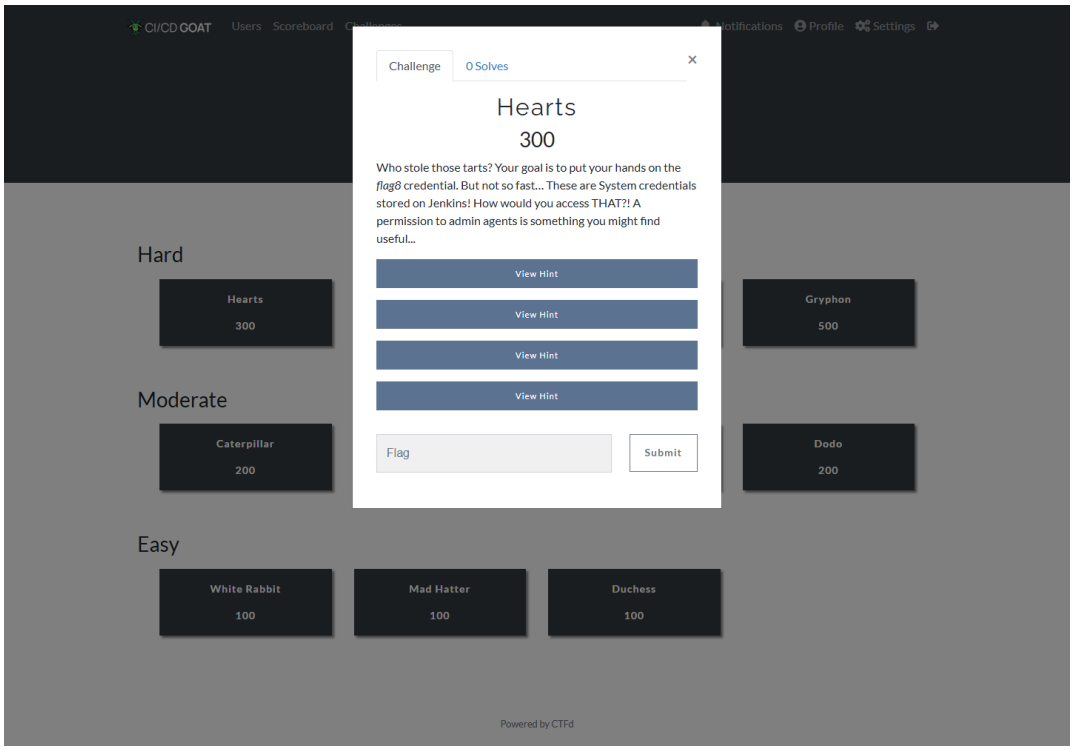

Figure 5.34: Hearts prompt.

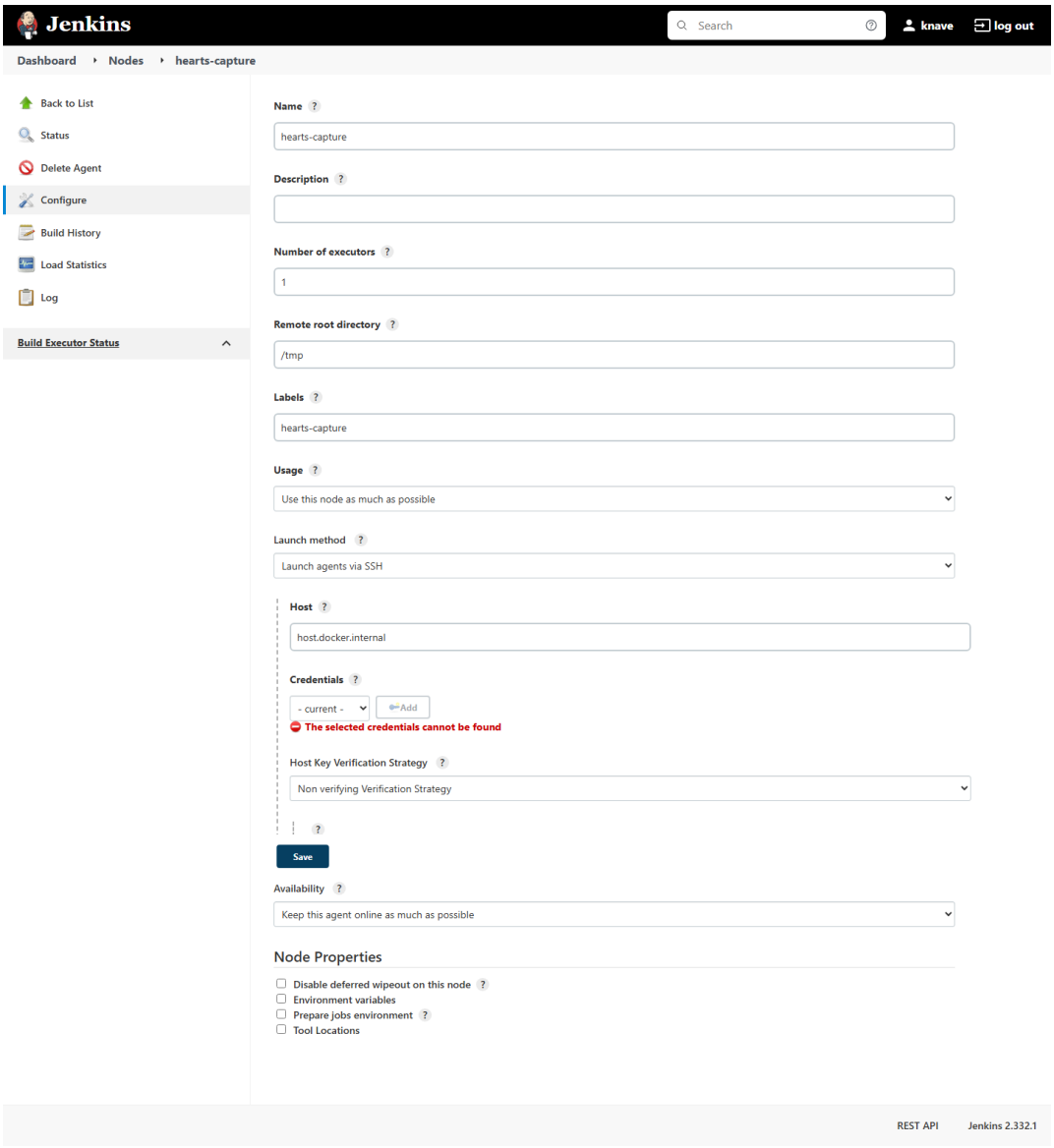

Figure 5.35: Jenkins node configuration pointing the SSH launcher at a controlled host.

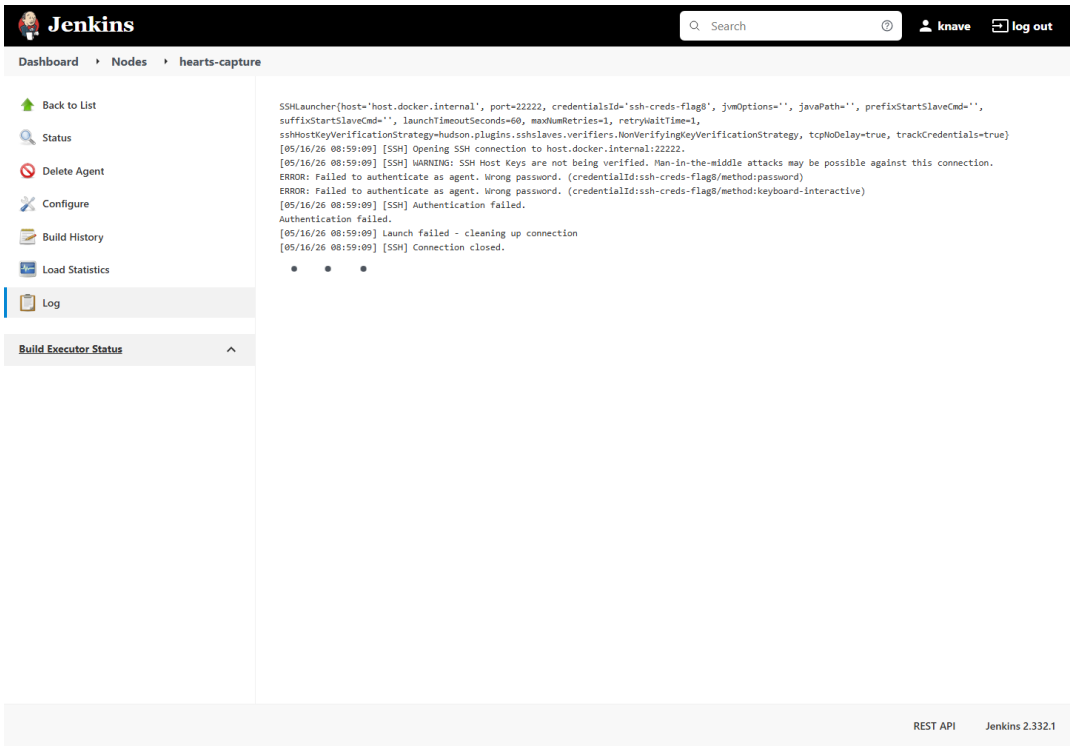

Figure 5.36: Launcher log showing credential replay to the attacker-controlled endpoint.

Weakness: Agent Configuration as Credential Exfiltration Vector

Jenkins replayed a stored system credential to any SSH target defined in node configuration. An attacker with node-management rights could configure a malicious target and harvest the material.

Exploitation Path

Define a new Jenkins node pointing at a local capture listener → select the target credential as the launcher identity → Jenkins initiates SSH connection → capture listener receives and logs the credential.

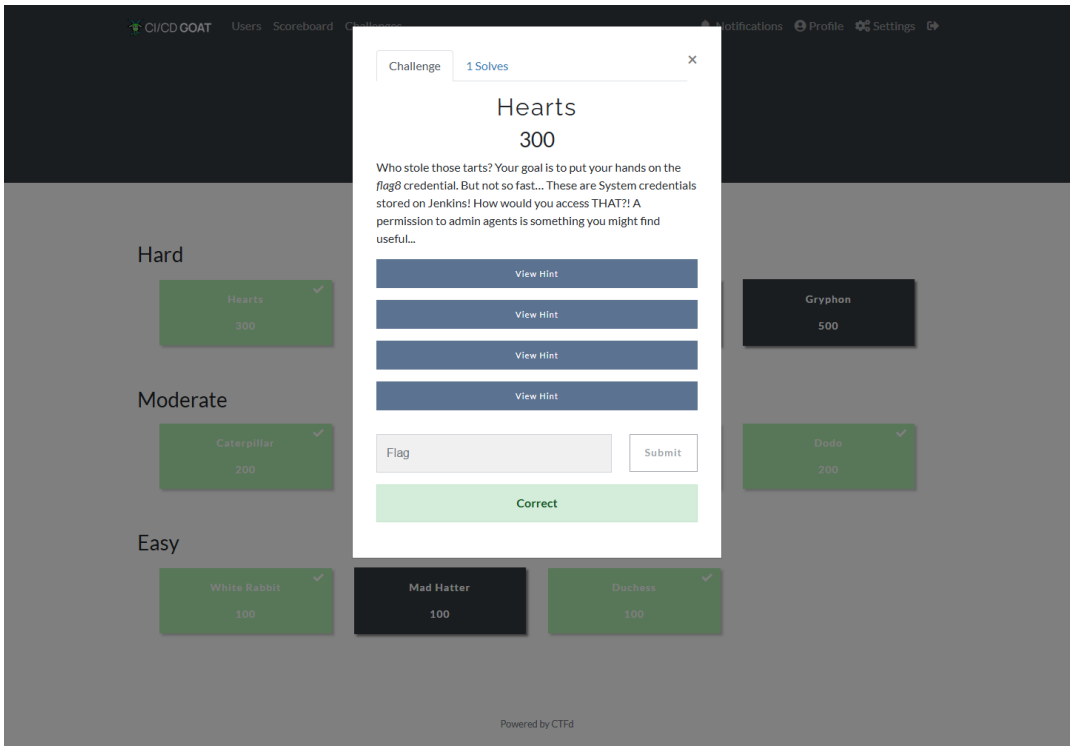

Figure 5.37: Accepted CTFd submission.

Flag: `B1A648E1-FD8B-4D66-8CAF-78114F55D396`

Remediation

Limit node creation and launcher changes to a tightly controlled admin role. Separate launcher credentials from sensitive service credentials. Alert on new SSH nodes or changes to launcher destinations.

5.9 Dormouse — External Script Poisoning via PR Metadata Injection

▷ Context

Dormouse's build downloaded and executed a remote helper script. The Reportcov pipeline that deployed that script interpolated PR metadata into shell execution without sanitization.

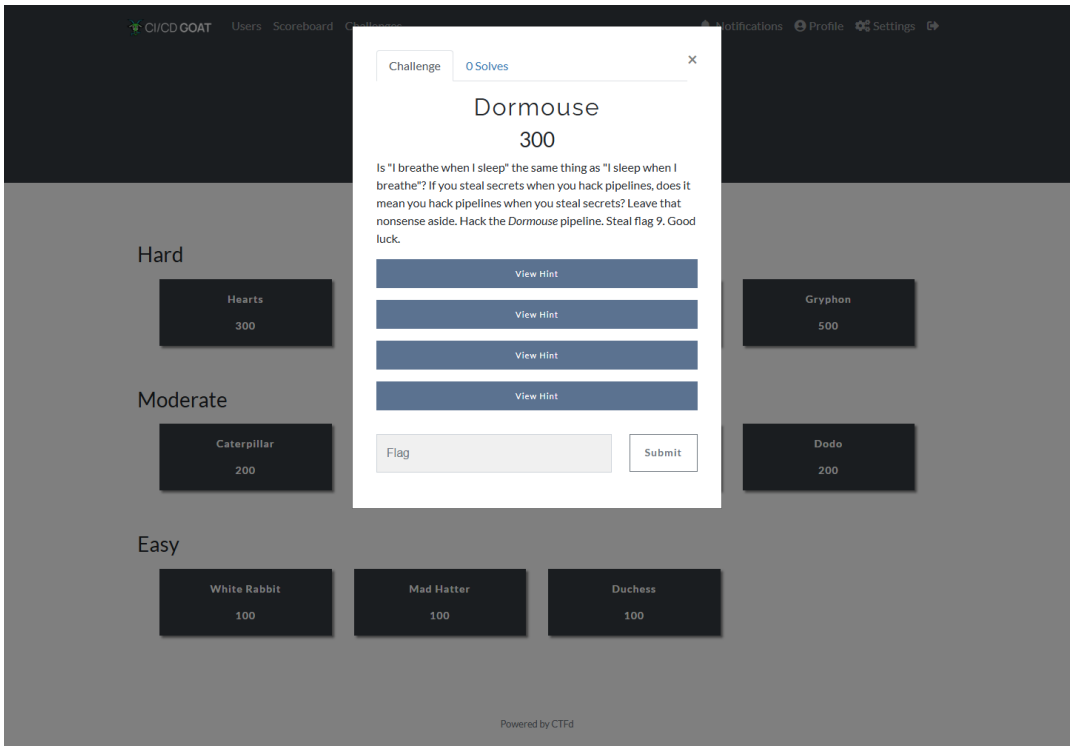

Figure 5.38: Dormouse prompt.

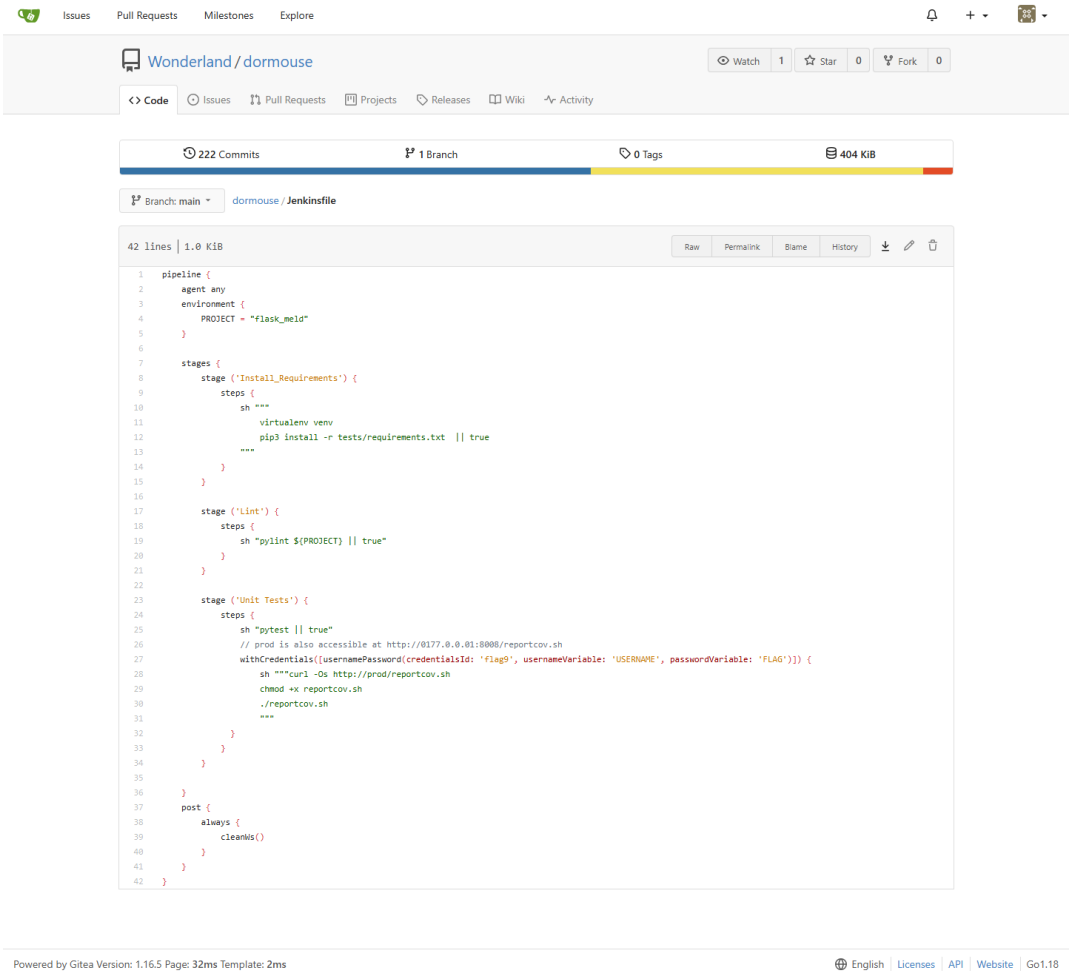

Figure 5.39: Dormouse Jenkinsfile showing dependency on the external helper.

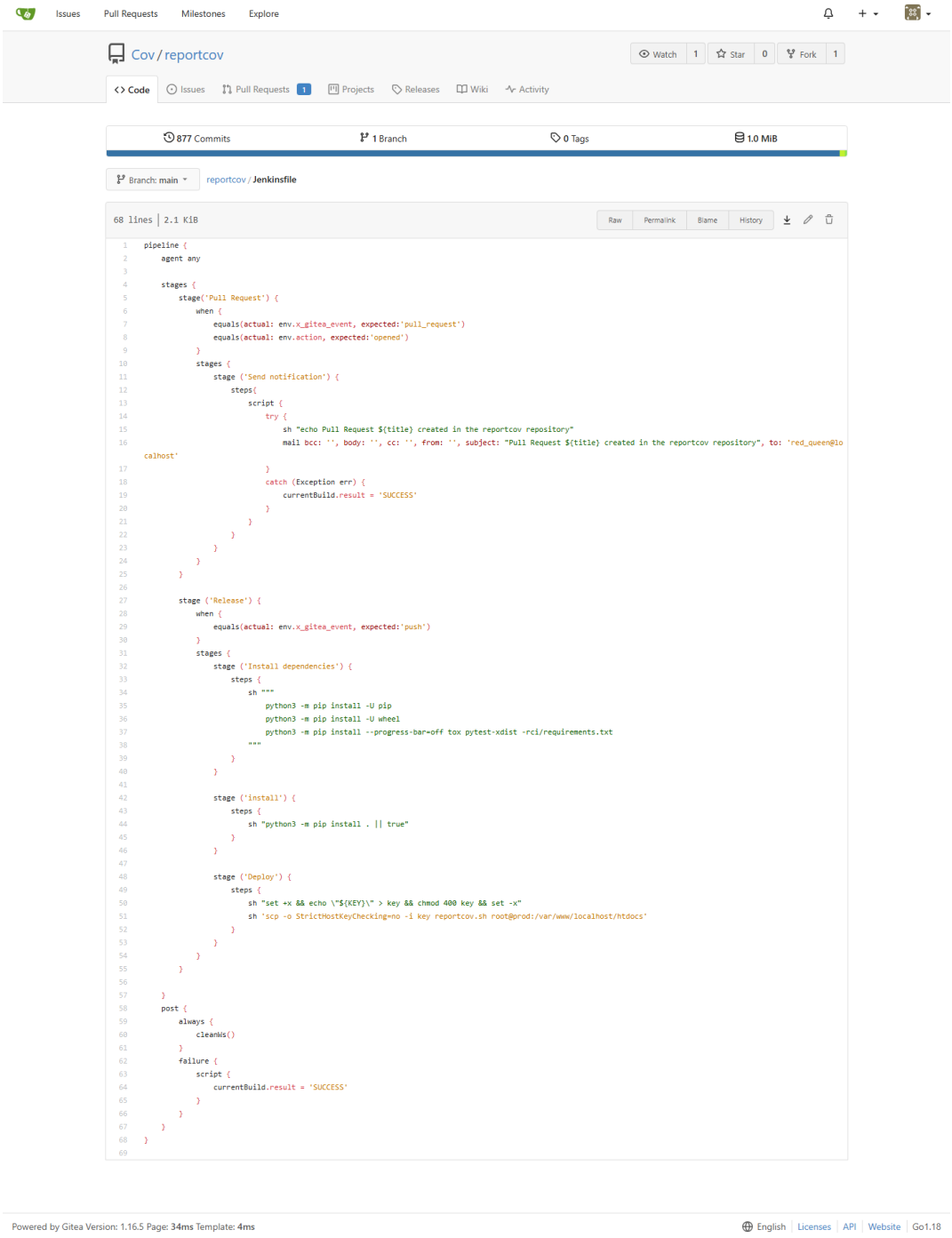

Figure 5.40: Reportcov Jenkinsfile: PR metadata flows into shell execution.

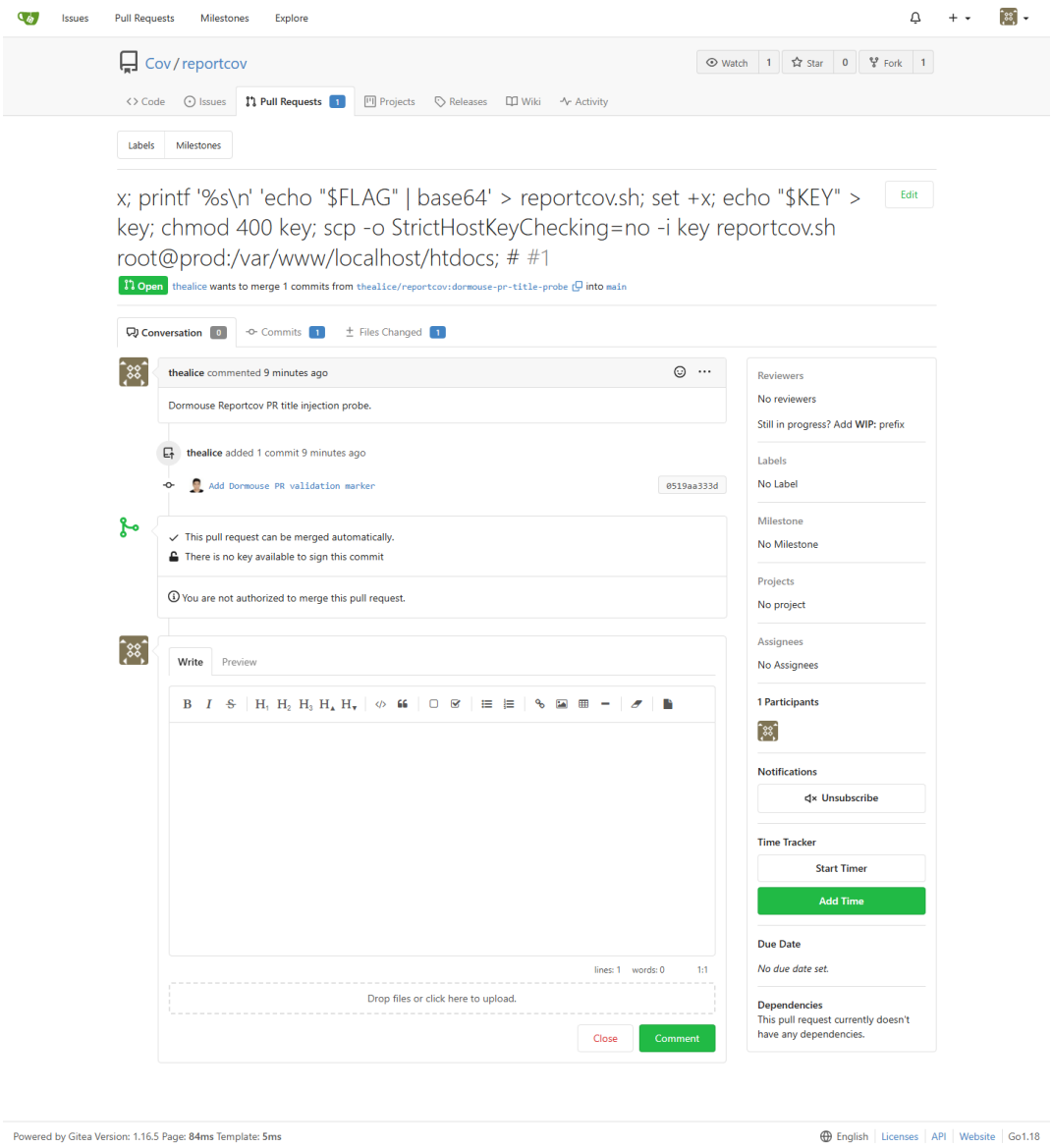

Figure 5.41: PR title crafted as a shell injection payload.

Weakness: Two Chained Trust Failures

(1) Reportcov interpolated PR title text directly into a shell command, allowing helper script overwrite on `prod`. (2) Dormouse treated the remote helper as trusted input inside a credential-bearing job.

Exploitation Path

Craft PR title as shell payload → Reportcov pipeline overwrites `reportcov.sh` on `prod` → Dormouse rebuild fetches and executes the poisoned helper → `flag9` disclosed.

```
echo "$FLAG" | base64
```

Figure 5.42: Poisoned helper script after Reportcov deployment.

Figure 5.43: Dormouse build disclosing the flag.

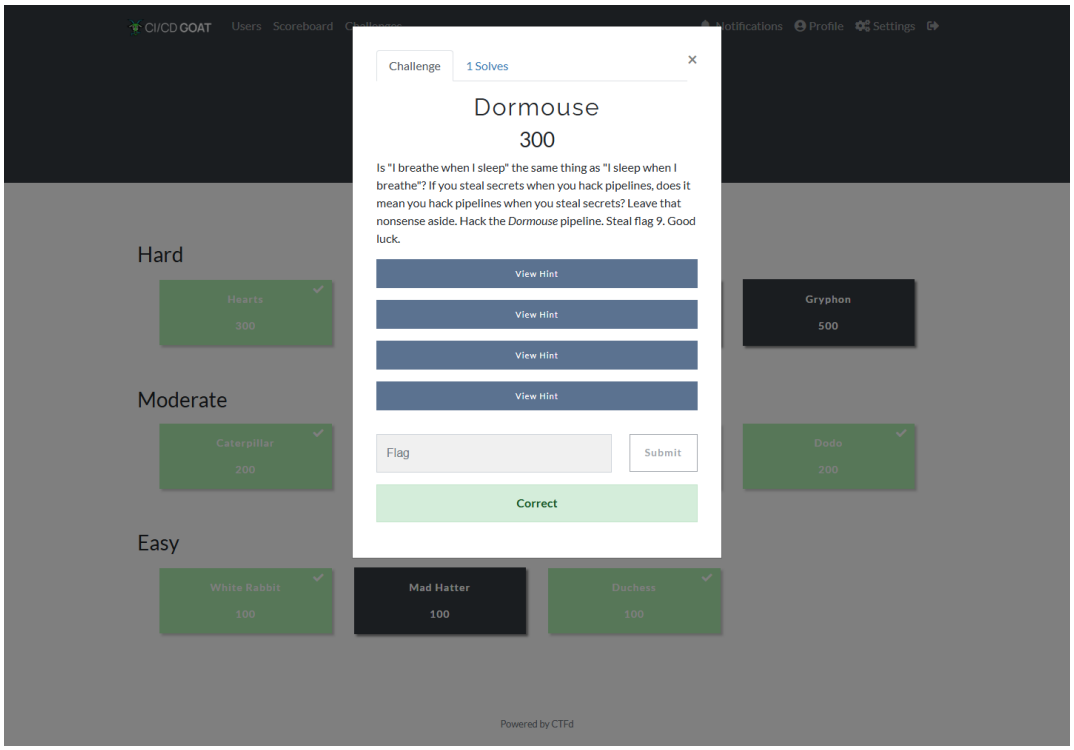

Figure 5.44: Accepted CTFd submission.

Flag: `31350FBC-A959-4B4B-A8BD-DCA7AC9248A6`

Remediation

Treat all PR metadata as hostile input. Never shell-interpolate repository metadata. Replace runtime-downloaded helper scripts with reviewed immutable artifacts or versioned packages.

5.10 Mock Turtle — Auto-Merge Token Leakage and Flow-Control Failure

▷ Context

Mock Turtle targeted the merge gate itself: the automation that decided whether a PR could be merged into the trusted branch.

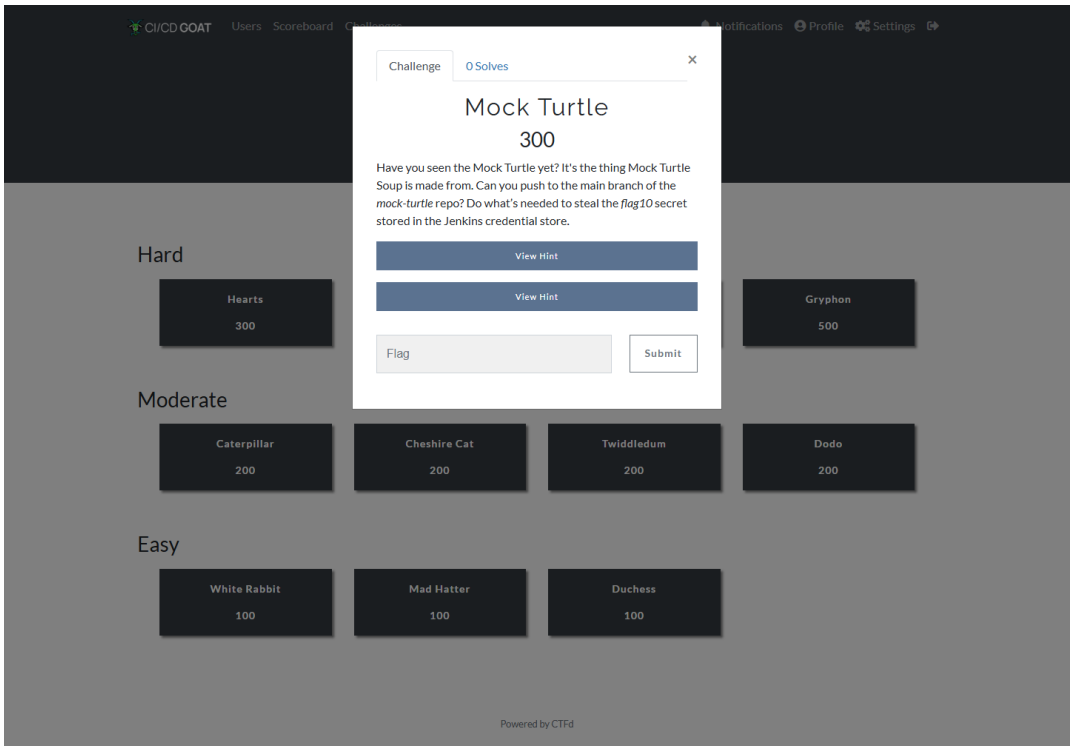

Figure 5.45: Mock Turtle prompt.

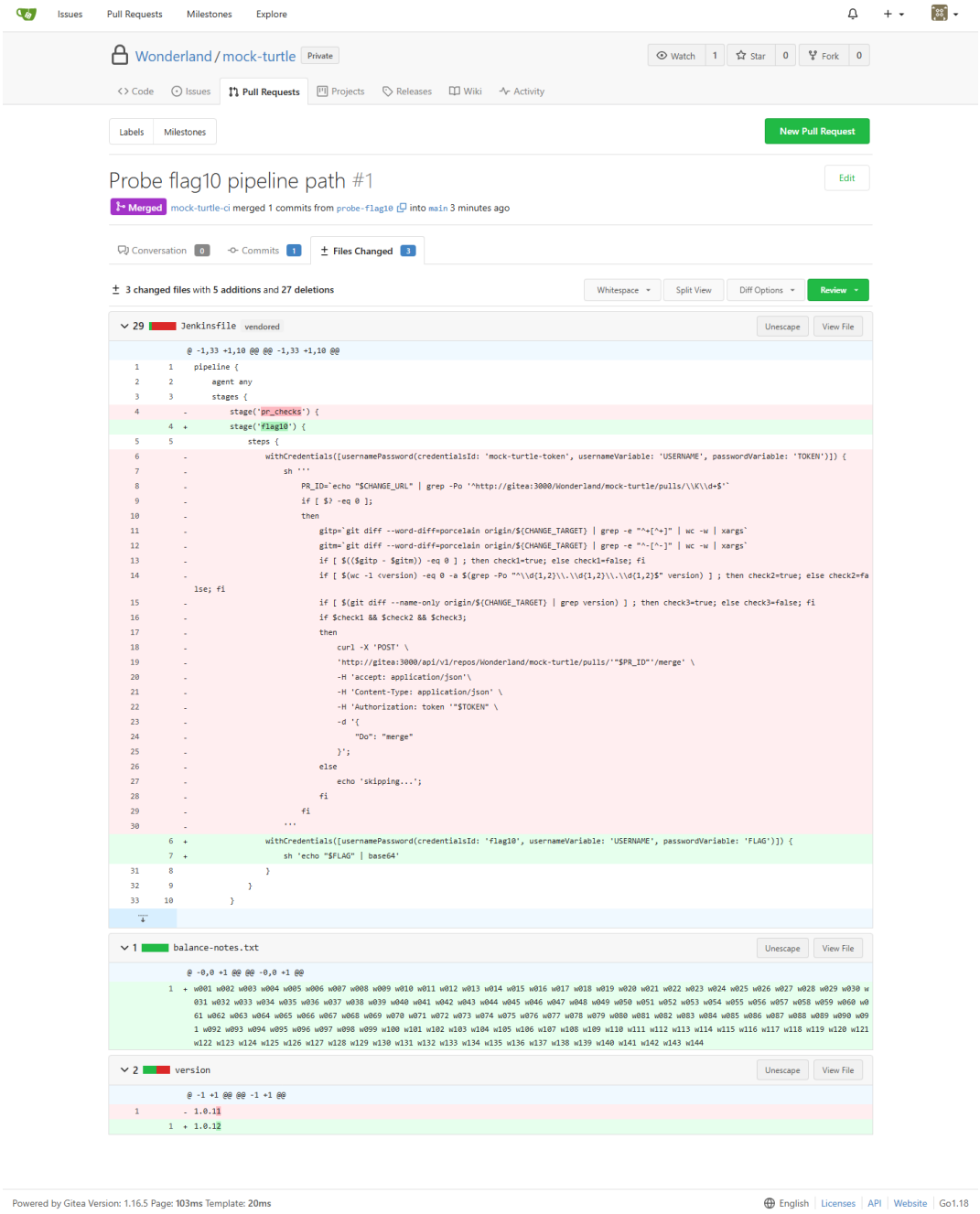

Figure 5.46: PR payload satisfying the merge checks while modifying the trusted pipeline.

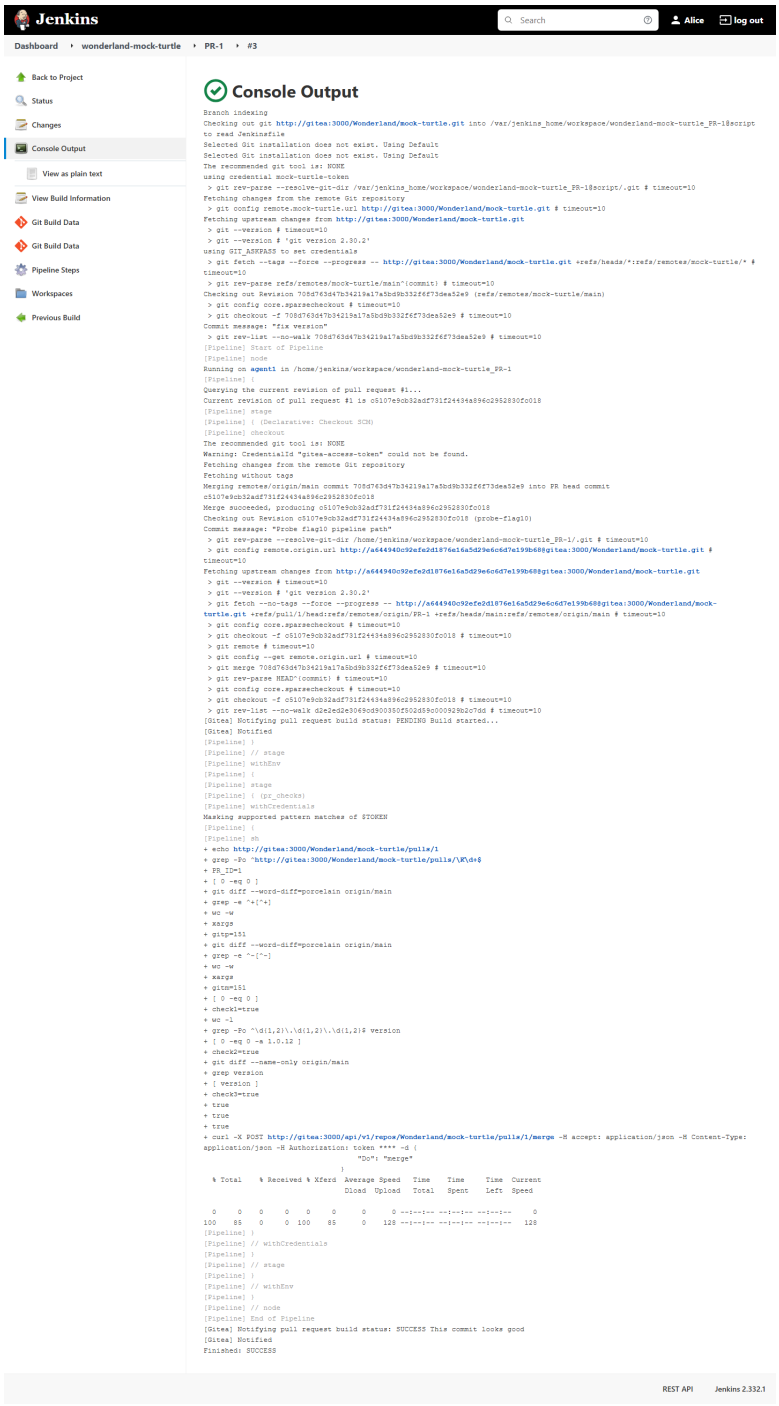

Figure 5.47: Build output: weak diff heuristic passes on a malicious change set.

Weakness: Heuristic Merge Gate

The auto-merge logic relied on shell-based diff arithmetic. Satisfying the heuristic did not prevent the same PR from replacing the Jenkinsfile.

Exploitation Path

Structure the change so all heuristic checks stay green while the PR still replaces the trusted-branch pipeline → Jenkins auto-merges → post-merge build runs attacker-controlled Jenkinsfile → `flag10` disclosed.

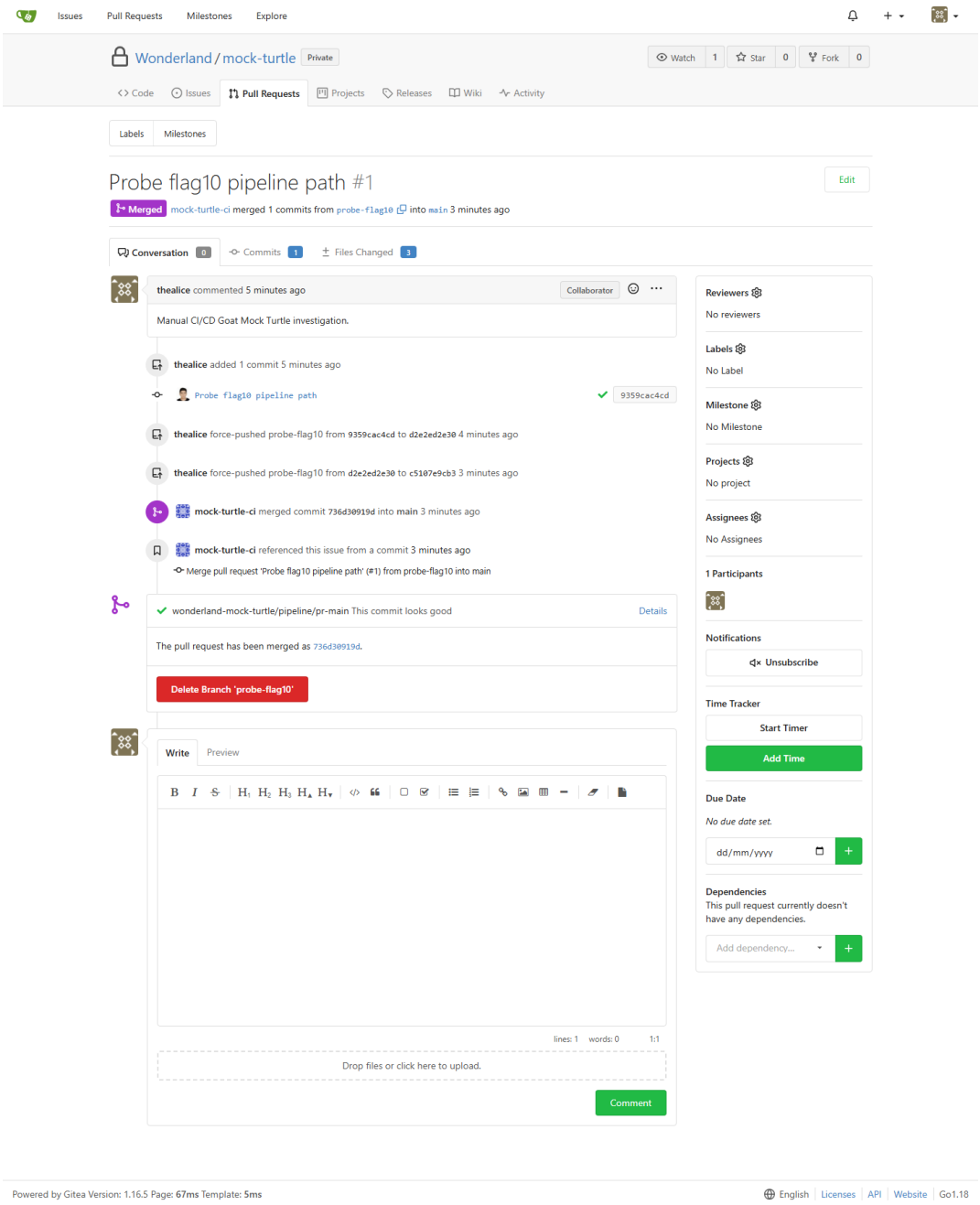

Figure 5.48: PR auto-merged by the automation.

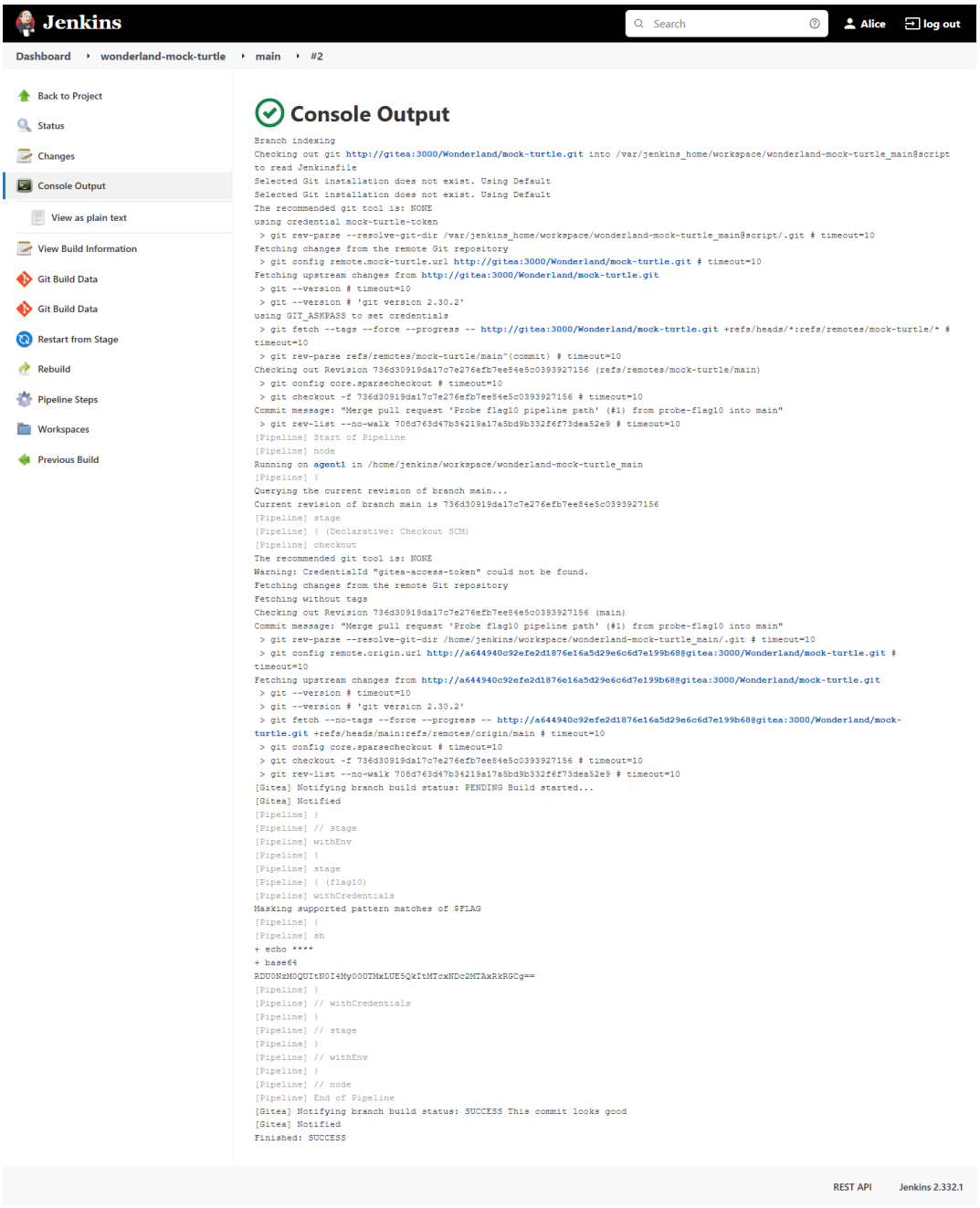

Figure 5.49: Trusted-branch build disclosing the flag.

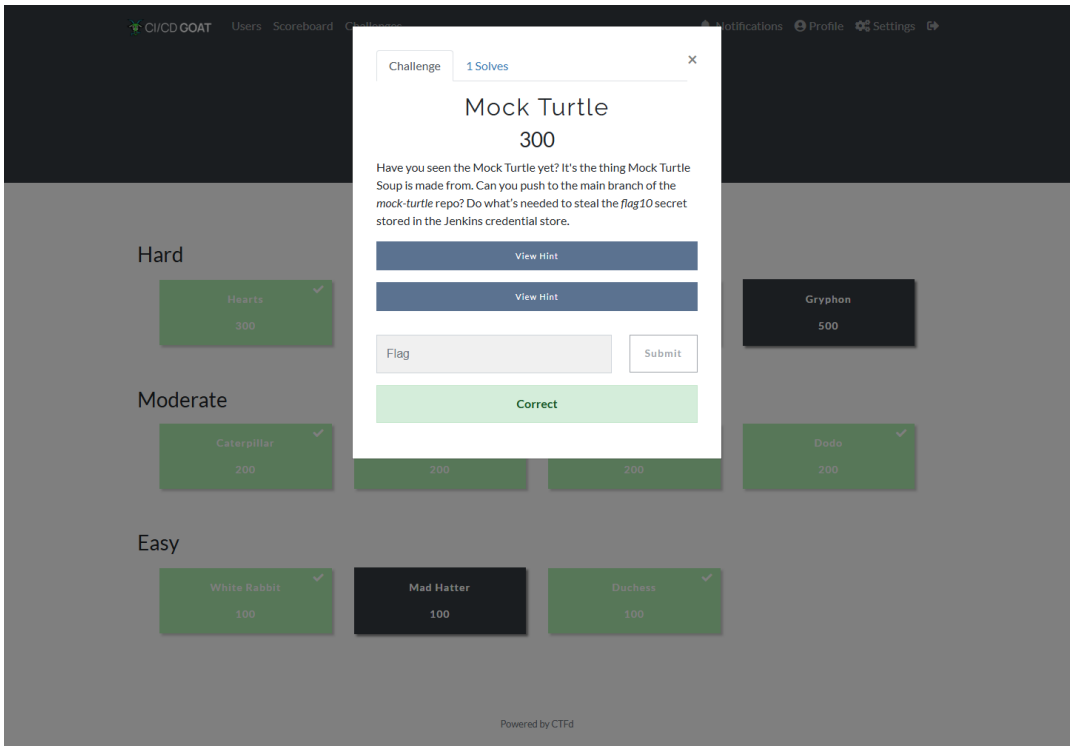

Figure 5.50: Accepted CTFd submission.

Flag: `D54734AB-7B83-4931-A9BB-171476101FDF`

Remediation

Use exact file-path allowlists for automation-driven merges. Require human approval when pipeline definitions change. Scope auto-merge tokens so they cannot rewrite the delivery path unchecked.

5.11 Gryphon — Package and Container Supply-Chain Compromise

▷ Context

Gryphon was the most complex chain in the lab: GitLab package publishing, scheduled downstream testing, registry token exposure, and mutable deployment images consumed by a protected project.

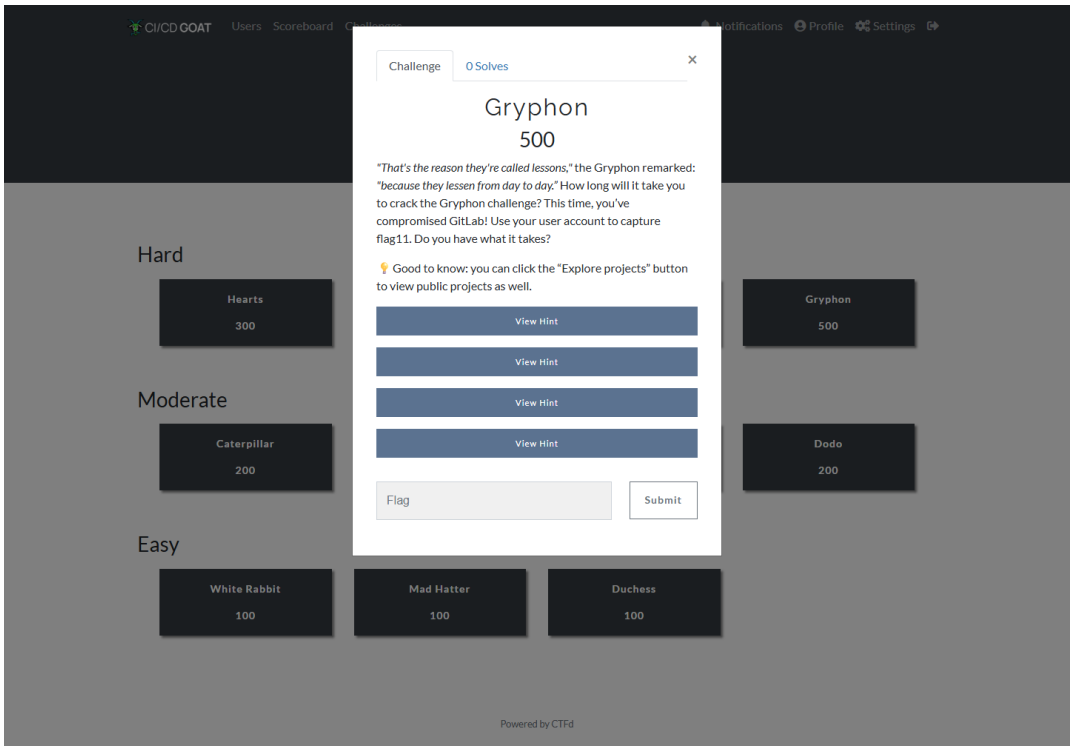

Figure 5.51: Gryphon prompt.

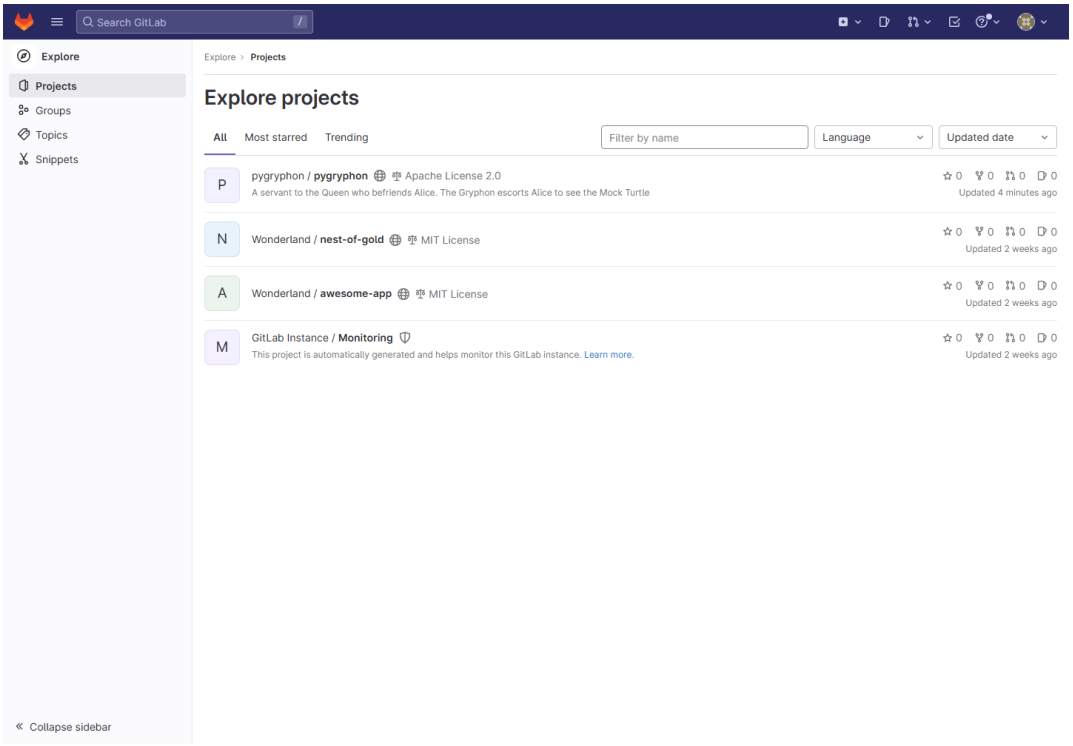

Figure 5.52: GitLab project enumeration showing public project relationships.

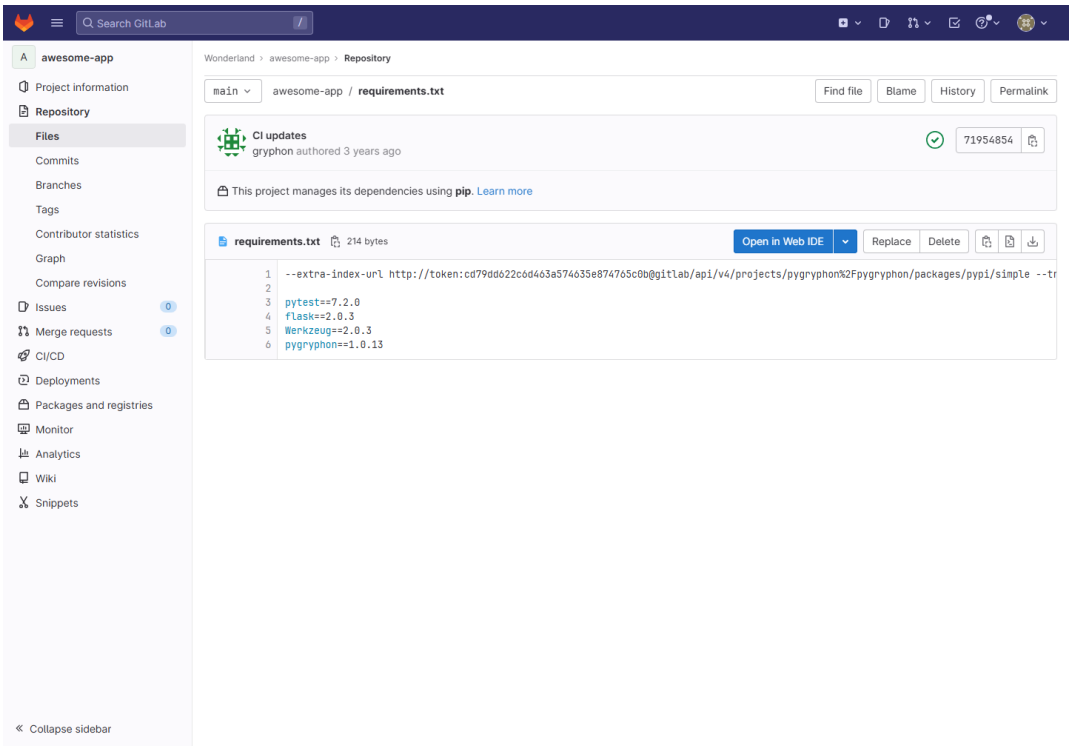

Figure 5.53: `awesome-app` requiring `pygryphon==1.0.13` from the internal registry.

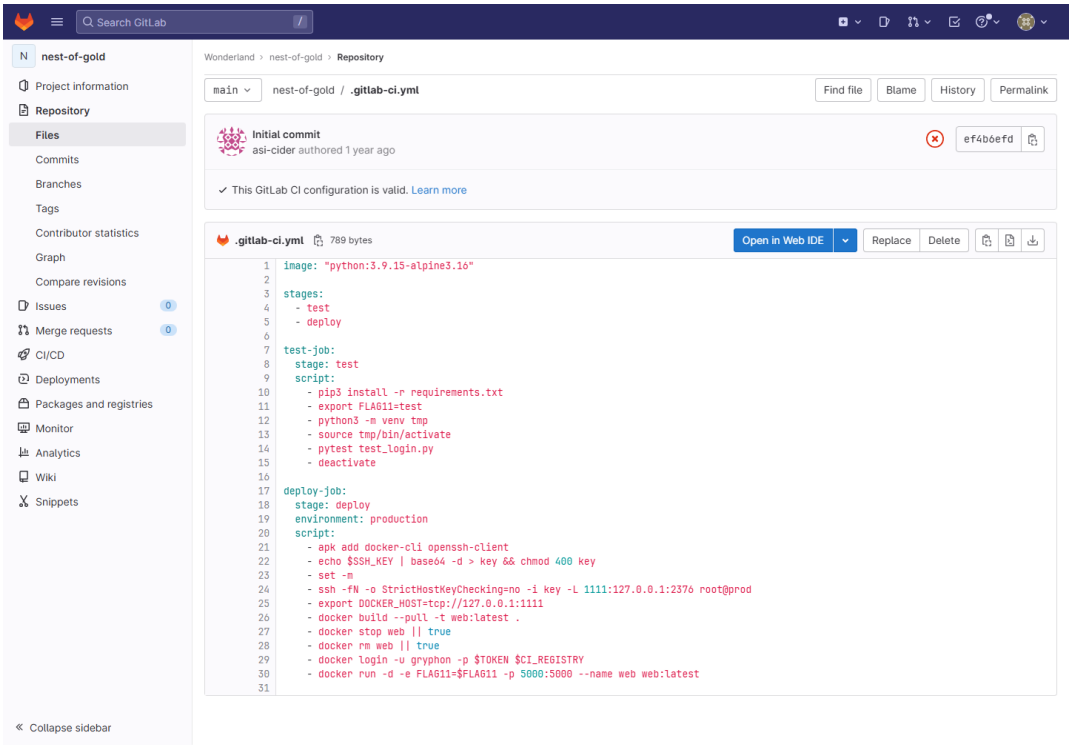

Figure 5.54: `nest-of-gold` consuming a mutable image tag while running with `FLAG11`.

Weakness: Mutable Artifact Trust at Every Stage

Each stage assumed the previous internal artifact (package, registry token, container image) was trustworthy. None were pinned or signed. Compromise propagated up the chain automatically.

Exploitation Path (4 stages)

1. Publish malicious `pygryphon` update → **2.** Scheduled `awesome-app` job imports it, exposes registry token in build log → **3.** Use token to push poisoned image to the namespace used by `nest-of-gold` → **4.** Next scheduled deployment pulls poisoned image, flag becomes accessible at the exposed endpoint.

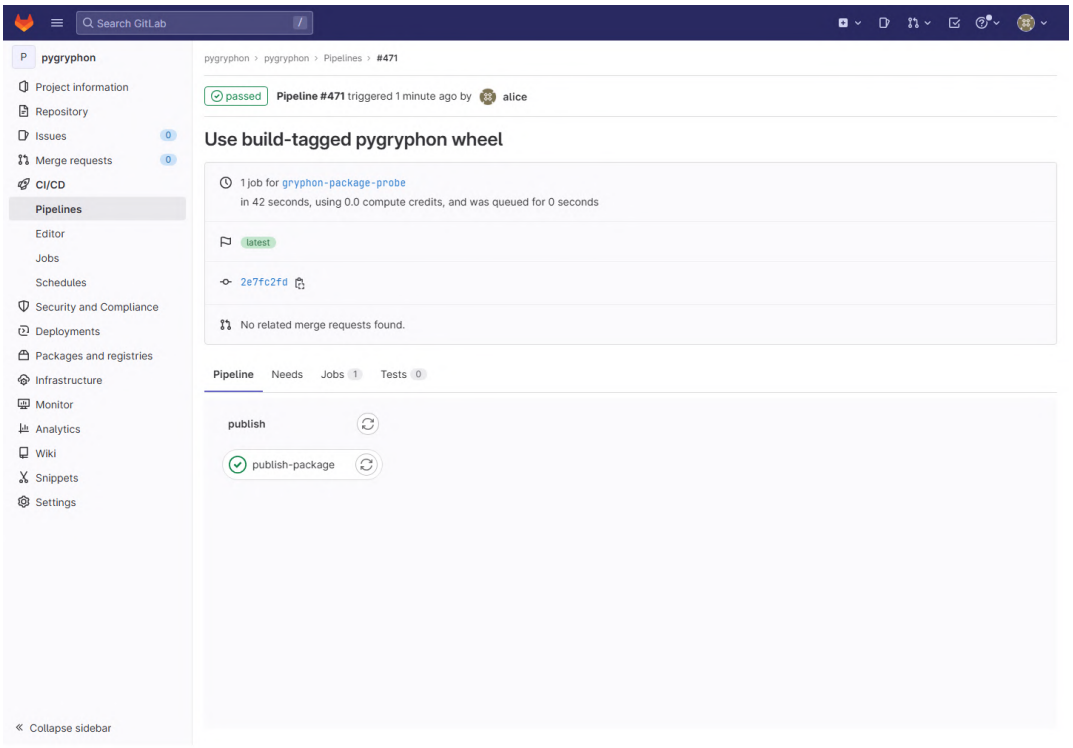

Figure 5.55: Malicious upstream package published in GitLab CI.

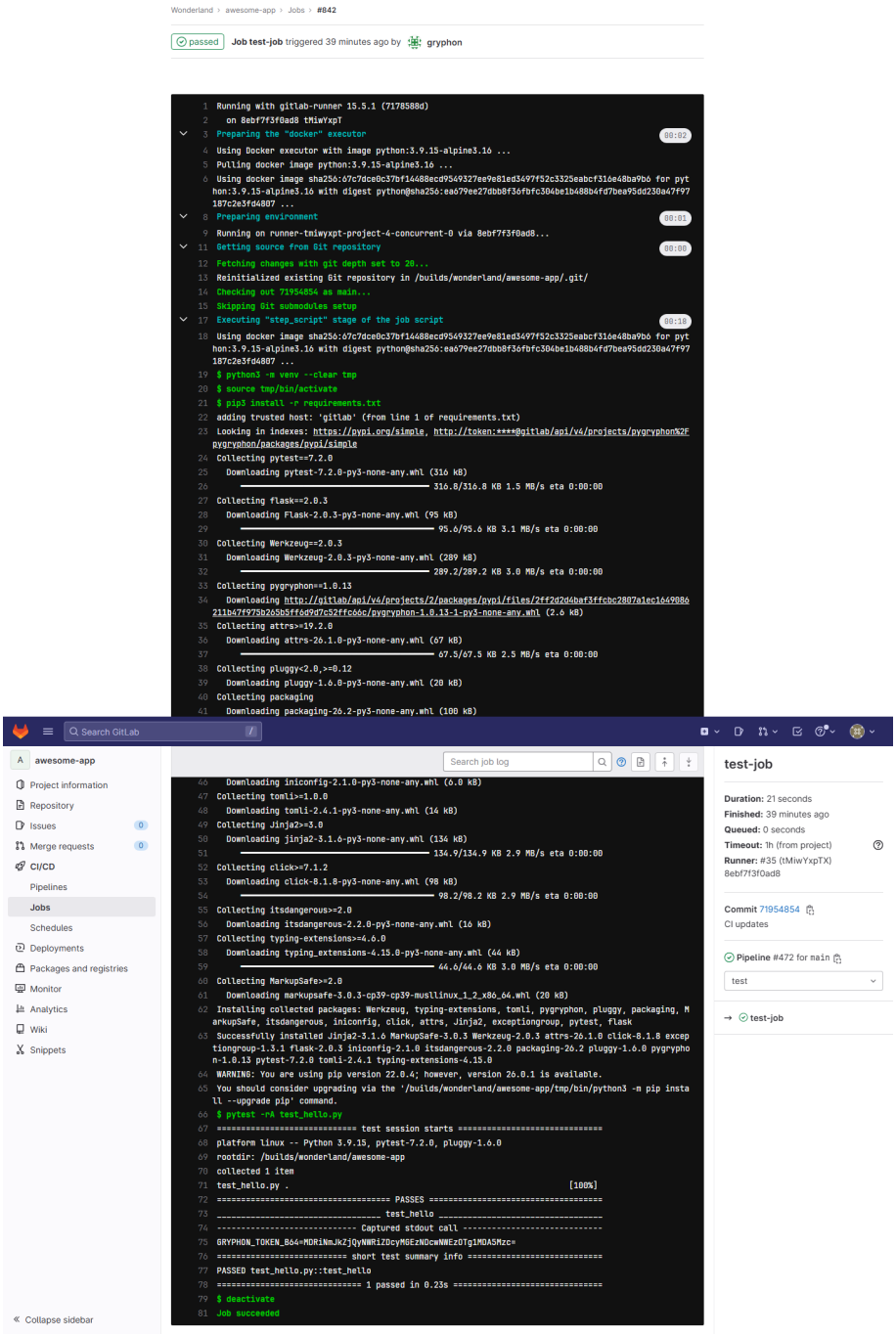

Figure 5.56: Scheduled downstream job exposing the registry token.

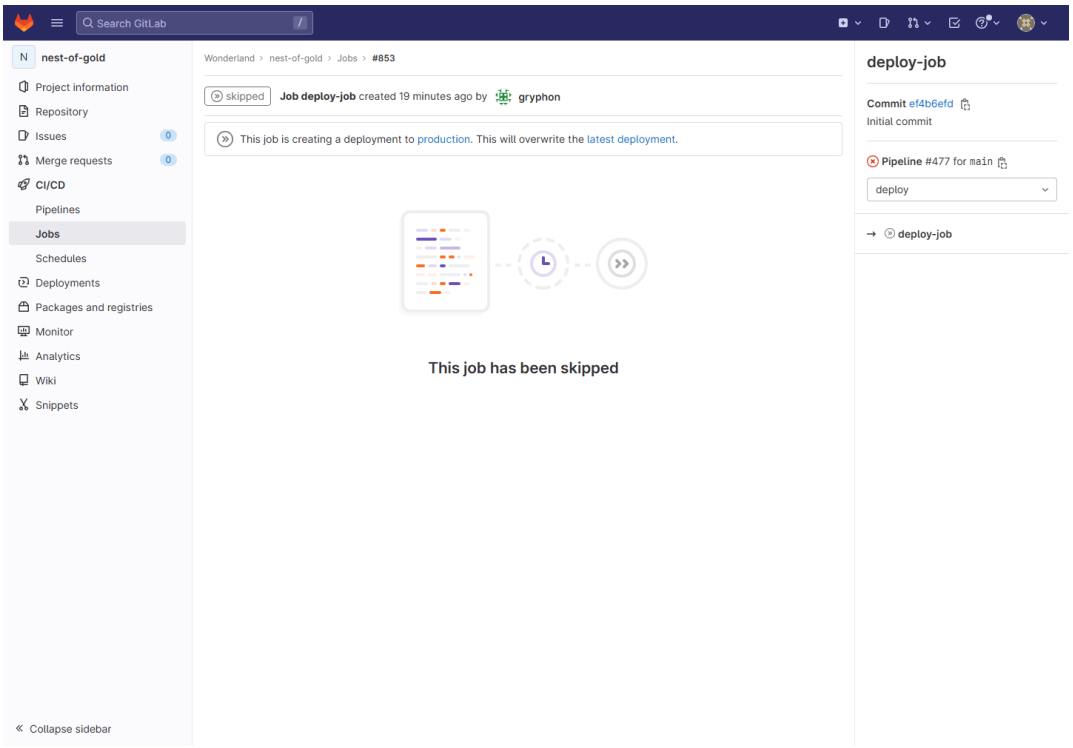

Figure 5.57: Deployment consuming the poisoned image.

```
7ED44218-C9CC-4824-BC85-C9841305A642
```

Figure 5.58: Flag recovered from the poisoned deployment endpoint.

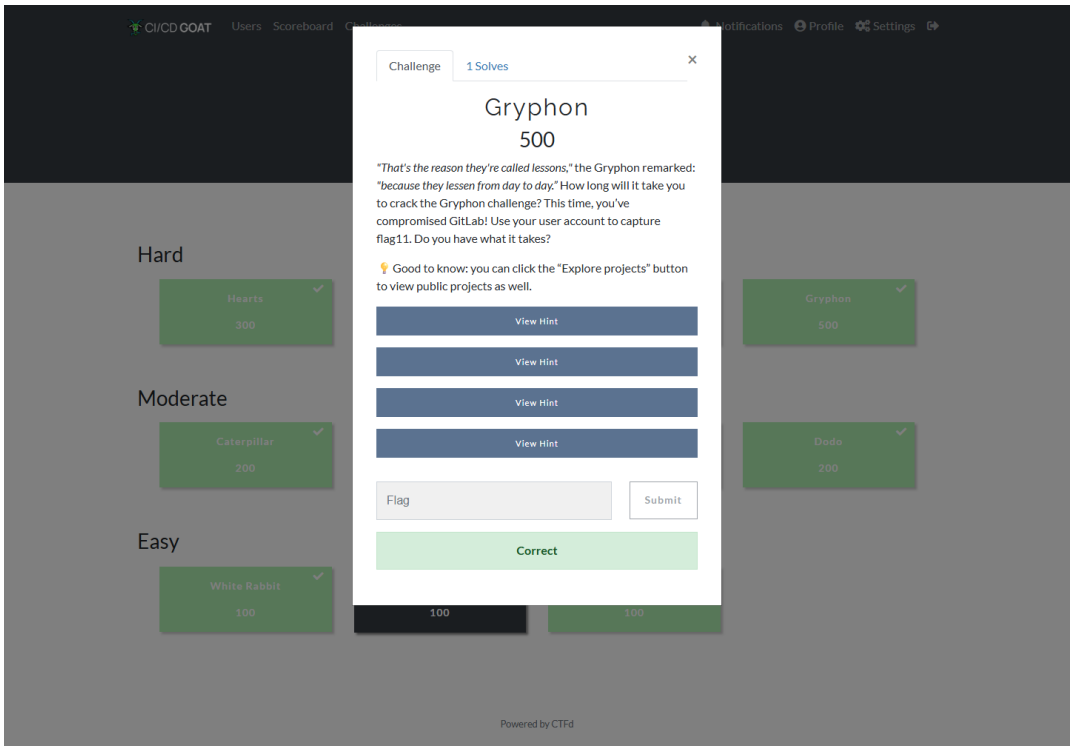

Figure 5.59: Accepted CTFd submission.

Flag: `7ED44218-C9CC-4824-BC85-C9841305A642`

Remediation

Restrict internal package publishing. Pin images by digest, never mutable tags. Require provenance and signing for all internal artifacts. Reduce scope of registry tokens exposed to scheduled jobs.

CHAPTER 6

Cross-Task Analysis

6.1 Recurring Trust Boundary Failures

Across all eleven challenges, the protection was applied at the wrong layer:

- Protecting `main` did not prevent the fork PR job from leaking the SCM token that could overwrite it (Caterpillar).
- Protecting the Jenkinsfile did not constrain the Makefile it delegated to (Mad Hatter).
- Protecting the downstream repo did not protect the upstream package it consumed (Twiddledum).
- Running Checkov did not prevent apply-time infrastructure mutation (Dodo).

The recurring failure was not missing tooling. It was trusting the wrong input boundary.

6.2 Lateral Movement and Privilege Escalation Patterns

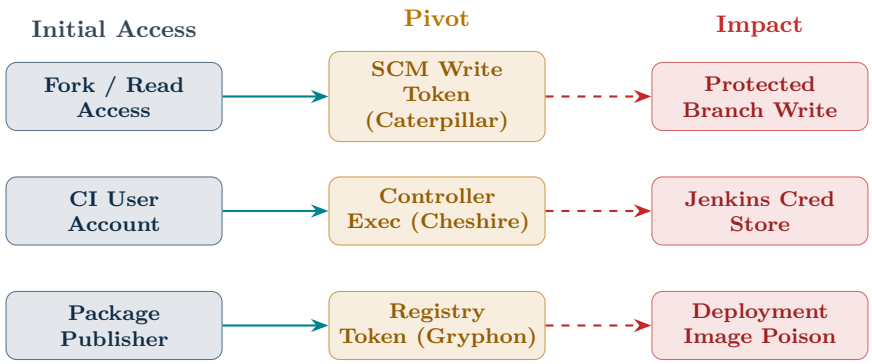

Figure 6.1: Lateral movement and privilege escalation paths observed across the challenge set.

6.3 Secret Exposure Taxonomy

Exposure Class	Mechanism	Challenges
Direct PPE disclosure	Repo-controlled pipeline prints bound credential	White Rabbit, Mad Hatter
Token leakage via CI context	Fork/PR job executes in privileged SCM token scope	Caterpillar, Mock Turtle
Git history recovery	Credential removed from tree but persists in objects	Duchess
Credential store replay	Agent SSH launcher sends stored cred to controlled host	Hearts
Supply-chain execution	Attacker code runs inside secret-bearing downstream job	Twiddledum, Gryphon
Metadata injection chain	PR text becomes shell input, overwrites trusted helper	Dormouse

Figure 6.2: Secret exposure taxonomy across the eleven challenges.

6.4 CVSS 3.1 Scores

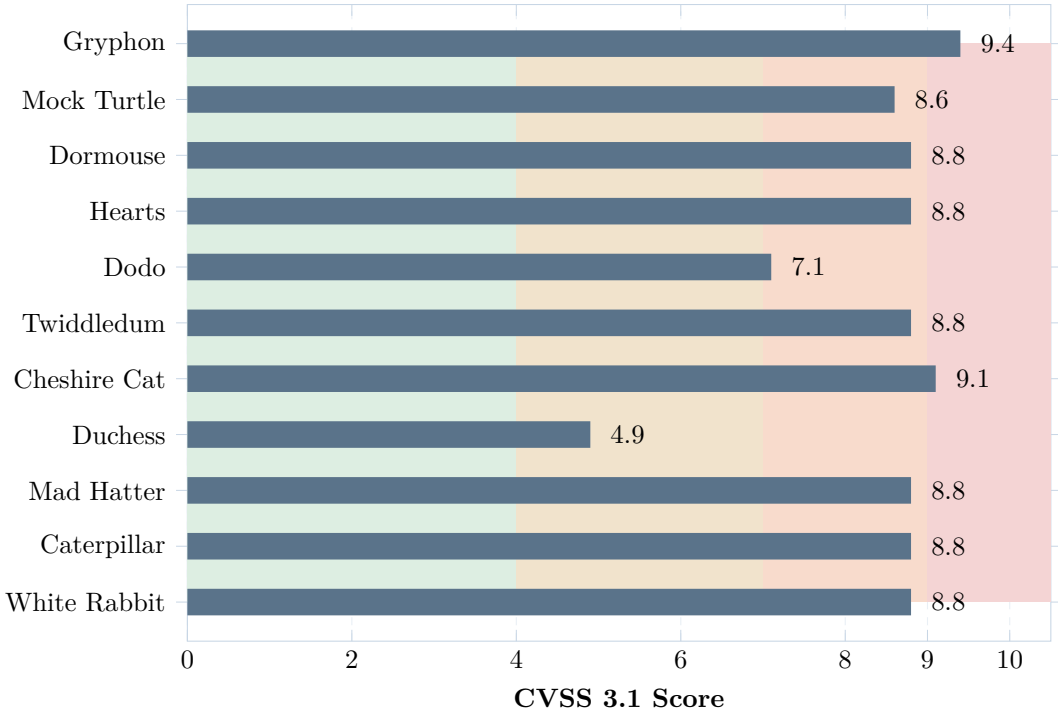

Figure 6.3: CVSS 3.1 scores for all eleven challenges with severity band shading (yellow ≥ 4, orange ≥ 7, red ≥ 9).

6.5 Full CVSS Table

Challenge	Vector	Score	Sev.	Rationale
White Rabbit	`AV:N/AC:L/PR:L/UI:N/S:C/C:H/I:H/`	8.8	HIGH	PR pipeline exposes Jenkins credential.
Caterpillar	`AV:N/AC:L/PR:L/UI:N/S:C/C:H/I:H/A:L`	8.8	HIGH	Read access escalates to protected-branch write.
Mad Hatter	`AV:N/AC:L/PR:L/UI:N/S:C/C:H/I:H/`	8.8	HIGH	Protected pipeline bypassed via mutable Makefile.
Duchess	`AV:N/AC:L/PR:L/UI:N/S:U/C:H/I:N/A:N`	4.9	MEDIUM	Credential in history; no pipeline interaction.
Cheshire Cat	`AV:N/AC:L/PR:L/UI:N/S:C/C:H/I:H/`	9.1	CRITICAL	PR forces controller execution and local file access.

Challenge	Vector	Score	Sev.	Rationale
Twiddledum	AV:N/AC:L/PR:L/ UI:N/S:C/C:H/I:H/A:L	8.8	HIGH	Mutable internal dep executes in secret-bearing job.
Dodo	AV:N/AC:L/PR:L/ UI:N/S:U/C:L/I:H/.	7.1	HIGH	Static policy bypass enables live infra mutation.
Hearts	AV:N/AC:L/PR:L/ UI:N/S:C/C:H/I:H/A:L	8.8	HIGH	Agent config replays system credential to attacker.
Dormouse	AV:N/AC:L/PR:L/ UI:N/S:C/C:H/I:H/	8.8	HIGH	PR metadata chain poisons helper in credential scope.
Mock Turtle	AV:N/AC:L/PR:L/ UI:N/S:C/C:H/I:H/A:L	8.6	HIGH	Weak merge gate allows malicious pipeline on main.
Gryphon	AV:N/AC:L/PR:L/ UI:N/S:C/C:H/I:H/	9.4	CRITICAL	4-stage supply-chain compromise reaches deployment.

Table 6.1: CVSS 3.1 assessment for all lab challenges.

CHAPTER 7

Remediation Strategy

The findings share a single root cause: the execution trust model was wider than the review model. Every remediation axis below narrows one of those gaps.

7.1 Defense-in-Depth for CI/CD

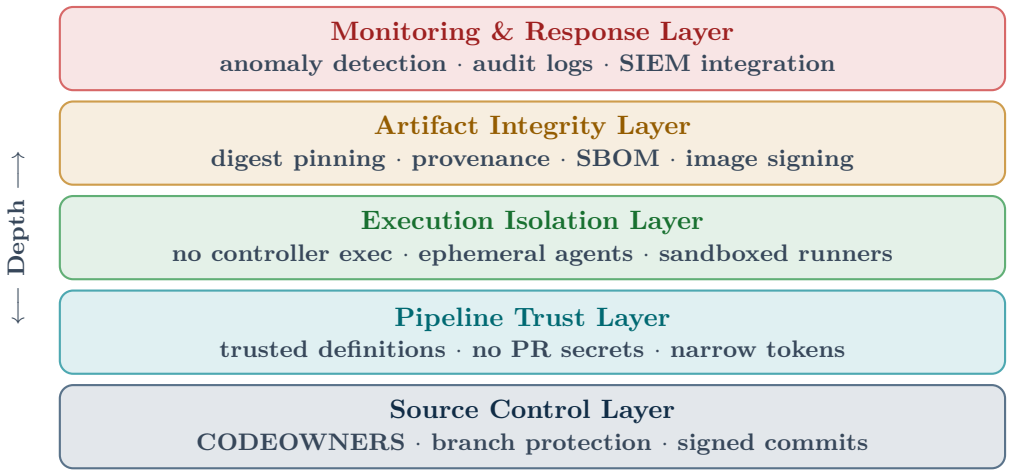

Figure 7.1: Defense-in-depth layers for CI/CD security. Each layer independently reduces the blast radius of a compromise in the layer below it.

7.2 Source Control Hardening

Branch protection governs only what it explicitly tracks. Any file invoked by the pipeline—Makefiles, shell helpers, setup scripts—must be under the same review discipline as the `Jenkinsfile` or `.gitlab-ci.yml` itself. CODEOWNERS-style path-based ownership enforces this automatically.

Controls

- Require human review for all pipeline-relevant file paths, not just the pipeline file.
- Enable pre-receive hooks with secret scanning on every repository.
- Sign all commits that touch release-critical paths.
- Audit repository access grants regularly and remove stale permissions.

7.3 Pipeline and Jenkins Hardening

Controls

- Run untrusted PR content under a separate, trusted pipeline definition stored outside the target repository.
- Disable controller executors; all production builds run on dedicated agents.
- Restrict agent label selection to an allowlist for untrusted job categories.
- Treat node administration as a privileged function with audit trails and change approval.
- Remove high-value credentials from jobs that accept unreviewed content.

7.4 Registry and Artifact Integrity

Controls

- Pin all internal package versions to exact immutable references (digest for images; hash-locked for packages).
- Restrict internal registry publishing to reviewed, signed release pipelines.
- Require SBOM generation and provenance attestation for all published artifacts.
- Validate live artifact integrity at deploy time, not just at build time.

7.5 Secrets Management

Secrets must never enter repository history. The Duchess case demonstrates that "already deleted" is not equivalent to "inaccessible." Token rotation must follow immediately upon discovery.

Controls

- Pre-commit and pre-push scanning on every developer workstation and CI runner.
- Rotate any secret identified in history immediately; treat the old value as fully compromised.
- Use short-lived, scoped tokens instead of long-lived static credentials wherever possible.
- Inject secrets at runtime from a dedicated secrets manager, not from environment baking.

7.6 IaC and Deployment Governance

Static scanning is necessary but not sufficient. Dodo demonstrated that an approved Terraform plan can still produce an insecure live state via apply-time side effects.

Controls

- Disallow `local-exec` provisioners and external script calls in delivery pipelines.
- Run post-apply state validation (e.g., AWS Config, OPA Gatekeeper) as a hard gate.
- Enforce policy as code outside the application repository so it cannot be overridden by a PR.
- Separate `plan` approval from `apply` execution with distinct credential scopes.

7.7 Monitoring and Detection Priorities

CRITICAL: New Jenkins node definitions or changed SSH launcher targets

CRITICAL: Automated PR merge by a CI service account

HIGH: Build log content matching base64-encoded credential patterns

HIGH: Unexpected internal package version installs in production pipelines

HIGH: Writes to deployment image namespaces from non-release jobs

MEDIUM: Helper script changes on `prod` deployment hosts

MEDIUM: Scheduled job runtime significantly deviating from baseline

Figure 7.2: Prioritized CI/CD monitoring detection points derived from lab findings.

CHAPTER 8

Conclusion

Eleven CI/CD Goat challenges, eleven validated flags, one consistent root cause: the execution trust model was always wider than the protection model.

Protecting the Jenkinsfile without protecting the Makefile it calls. Blocking direct pushes to `main` without restricting what a fork PR job can print. Running Checkov without validating what `terraformapply` actually deploys. Each of these reflects the same reasoning error: applying control at the visible layer while leaving the actual execution path open.

The practical defense is not more scanners or more rules. It is disciplined trust boundaries: reviewed pipeline definitions for any unreviewed input, narrowly scoped tokens that expire, ephemeral isolated runners, immutable artifact references, and continuous validation of live state. That architecture makes the execution trust model match the review model, and removes the gaps that every challenge in this lab exploited.

CHAPTER 9

References

Bibliography

[1] Cider Security Research, *CI/CD Goat: A deliberately vulnerable CI/CD environment*, GitHub repository. https://github.com/cider-security-research/cicd-goat

[2] National Institute of Standards and Technology, *SP 800-218: Secure Software Development Framework (SSDF) Version 1.1*, 2022. https://csrc.nist.gov/pubs/sp/800/218/final

[3] CISA & NSA, *Defending Continuous Integration/Continuous Delivery (CI/CD) Environments*, 2023. https://www.cisa.gov/resources-tools/resources/defending-continuous-integrationcontinuous-delivery-cicd-environments

[4] OWASP, *Top 10 CI/CD Security Risks*, 2022. https://owasp.org/www-project-top-10-ci-cd-security-risks/

[5] Bridgecrew, *Checkov — Open-source static analysis for IaC*. https://www.checkov.io/

APPENDIX A

Challenge-to-Vulnerability Mapping

Challenge	Platform	Weakness	Failed Boundary	Observed Outcome
White Rabbit	Jenkins + Gitea	PR pipeline poisoning	PR code vs. credential-bearing exec	Flag via PR Jenkinsfile abuse
Caterpillar	Jenkins + Gitea	CI token leakage	Fork job vs. SCM token scope	Flag after trusted-branch modification
Mad Hatter	Jenkins + Gitea	Protected-pipeline bypass	Protected Jenkinsfile vs. mutable Makefile	Makefile execution path completed
Duchess	Gitea	Secret in history	Current tree vs. history objects	Token from deleted `.pypirc`
Cheshire Cat	Jenkins + Gitea	Controller execution	Untrusted PR vs. privileged node	Controller local file access
Twiddledum	Jenkins + Gitea	Dependency poisoning	Protected consumer vs. mutable dep	Flag via upstream package
Dodo	Jenkins + Terraform	Scanner bypass	Static scan vs. runtime mutation	Flag after public ACL change
Hearts	Jenkins	Credential replay	Agent config vs. system credential	Flag via SSH launcher capture
Dormouse	Jenkins + Gitea	Metadata injection	PR metadata vs. shell; helper vs. cred scope	Flag via poisoned remote script
Mock Turtle	Jenkins + Gitea	Weak merge gate	Diff heuristic vs. branch integrity	Flag after auto-merge abuse
Gryphon	GitLab	Supply-chain abuse	Mutable artifacts vs. protected deploy	Flag via 4-stage chain

APPENDIX B

Challenge Outcomes

Challenge	Outcome	Validation
White Rabbit	Jenkins credential recovered through direct poisoned PR pipeline execution.	Jenkins log + CTFd
Caterpillar	Read-only access escalated to trusted-branch control via PR token leakage.	Token + prod log + CTFd
Mad Hatter	Protected Jenkinsfile bypassed via mutable Makefile execution path.	Payload + scan behavior
Duchess	PyPI credential recovered from Git history after deletion.	History view + CTFd
Cheshire Cat	Controller execution achieved by controlling agent label selection.	Node view + log + CTFd
Twiddledum	Internal dependency poisoning exposed build-time secret.	Downstream log + CTFd
Dodo	Terraform apply-time mutation bypassed static Checkov review.	Terraform + log + CTFd
Hearts	Agent launcher coerced into replaying stored credential.	Node config + log + CTFd
Dormouse	Reportcov metadata injection poisoned external script consumed by Dormouse.	Helper chain + log + CTFd
Mock Turtle	Weak auto-merge checks allowed trusted-branch pipeline replacement.	PR checks + merge + CTFd
Gryphon	4-stage package and image chain compromise reached protected deployment.	GitLab jobs + endpoint + CTFd