Pwn the Pipeline: A Hands-On Tour of CI/CD Security Risks

Meethack

RomHack Camp 2026

2 October 2026

About us: **Meethack**



- Hackerspace / community of cybersecurity enthusiasts.
- Based in Torino @ Toolbox Coworking, since 2019.
- In-person meetings every Tuesday.
- `https://meethack.it`

What we do: vulnerability research, n-day analysis, IoT hacking nights, CTF challenges, *Hack The Box* machines, AWS security, Kubernetes security, etc.

Today's speakers: `m3ssap0`, `IadRabbit`, `0xbro`, `clownfire`.

Introduction

A new attack surface

CI/CD environments, processes and systems are the beating heart of any modern software organization.

They deliver code from an engineer's workstation to production.

However, they have also reshaped the *attack surface* with a multitude of new avenues and opportunities for attackers.

Adversaries ~~are shifting~~ **already shifted** their attention to CI/CD, realizing CI/CD services provide an efficient path to reaching an organization's crown jewels.

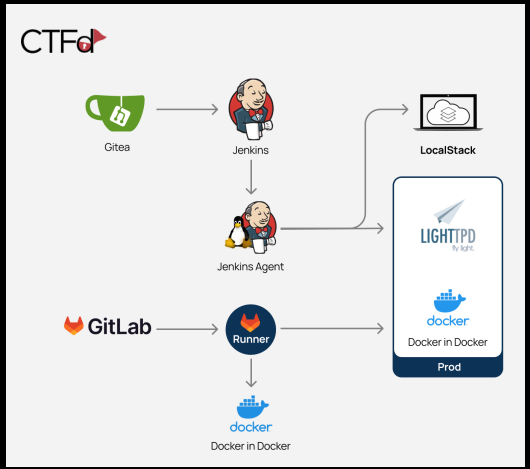
Top 10 CI/CD Security Risks

1. Insufficient Flow Control Mechanisms
2. Inadequate Identity and Access Management
3. Dependency Chain Abuse
4. Poisoned Pipeline Execution (PPE)
   - Direct (D-PPE)
   - Indirect (I-PPE)
   - Public (3PE)
5. Insufficient PBAC (Pipeline-Based Access Controls)
6. Insufficient Credential Hygiene
7. Insecure System Configuration
8. Ungoverned Usage of 3rd Party Services
9. Improper Artifact Integrity Validation
10. Insufficient Logging and Visibility

https://github.com/OWASP/www-project-top-10-ci-cd-security-risks

ROMHACK 2026 CAMP

CI/CD Goat: the playground (credits to Cider Security / Palo Alto Networks)



Things to keep in mind:

- Builds will start on Jenkins, usually after a *Pull Request* (PR).
- PRs can be opened from a newly created branch on Gitea into `main`.
- Jenkins will mask plain text secrets leaked in console output. This will happen in every challenge!
- Each challenge stands on its own. No need to use access gained in one challenge to solve another challenge.
- Don't be afraid to look at hints.
- There is no need to exploit CVEs.

Our fork: https://github.com/meethack-torino/cicd-goat

Git client 101

- Clone a repository (use Gitea credentials `thealice` / `thealice`):
  - `git clone http://localhost:3000/Wonderland/<repository_name>.git`
- Checkout to a new branch:
  - `git checkout -b <branch_name>`
- Add and commit with message:
  - `git add .`
  - `git commit -m "Your message"`
- Push to the remote branch:
  - `git push -u origin <branch_name>`

ROMHACK 2026 CAMP

An example of *Jenkinsfile*

```
pipeline {
    agent any
    environment {
        PROJECT = "src/urllib3"
    }

    stages {
        stage ('Install_Requirements') {
            steps {
                sh """
                    virtualenv venv
                    pip3 install -r requirements.txt || true
                """
            }
        }

        stage ('Lint') {
            steps {
                sh "pylint ${PROJECT} || true"
            }
        }

        stage ('Unit Tests') {
            steps {
                sh "pytest"
            }
        }

    }
    post {
        always {
            cleanWs()
        }
    }
}

```

- *Jenkins Pipeline* is a suite of plugins which supports implementing and integrating continuous delivery pipelines into Jenkins.
- A Continuous Delivery (CD) pipeline is an automated expression of your process for getting software from version control right through to your users and customers.
- The definition of a Jenkins Pipeline is written into a text file (called a `Jenkinsfile`) which in turn can be committed to a project's source control repository.
- This is the foundation of *"Pipeline-as-code"*; treating the CD pipeline as a part of the application to be versioned and reviewed like any other code.

https://www.jenkins.io/doc/book/pipeline

“[...] You take the red pill, you stay in Wonderland, and I show you how deep the rabbit hole goes.”

Real-world case: *Teleport* (2021)

- Teleport's CI/CD ran on *Drone CI* over Kubernetes, serving both public and internal repositories.
- Anyone on the internet could open a PR on a public repository and modify the pipeline configuration file (`.drone.yml`), which the CI would then execute.
- Build containers shared a privileged docker-in-docker sidecar that let a crafted pipeline escape the sandbox and get root on the Kubernetes node, reaching the production AWS credentials of neighbouring pods, cloud backend services, release artifacts and private repositories.
- Exposed from May 2020 to April 2021, found internally, no evidence of exploitation. Fixed by isolating CI from release infrastructure and requiring approval for PRs from forks.

Poisoned Pipeline Execution (PPE)

Refers to the ability of an attacker with access to source control systems — and without access to the build environment, to manipulate the build process by injecting malicious code/commands into the build pipeline configuration, essentially "poisoning" the pipeline and running malicious code as part of the build process. It can be *Direct* or *Indirect*, depending on the attacker's access to the pipeline configuration, and *Public* when anyone on the internet can trigger it.

https://goteleport.com/blog/hack-via-pull-request

Challenge: *Caterpillar - 200*

- http://localhost:8000/challenges#Caterpillar-2
- Scenario:
  - Use your access to the `Wonderland/caterpillar` repository to steal the `flag2` secret, which is stored in the Jenkins credential store.
  - `Jenkinsfile` is protected: the pipeline is configured in the same repository from where the application code is stored at, but the current user can't change it.
  - There are two jobs in Jenkins: `wonderland-caterpillar-prod` and `wonderland-caterpillar-test`.
- Solution:
  - *Public Poisoned Pipeline Execution (3PE)*
  - https://github.com/meethack-torino/cicd-goat/blob/main/solutions/caterpillar.md

Real-world case: *ua-parser-js* (2021)

- `ua-parser-js` is a legitimate npm library used to parse user agent strings, with around 9 million downloads per week.
- In October 2021 the maintainer's npm account was hijacked and three malicious versions were published (`0.7.29`, `0.8.0`, `1.0.0`), one for each major branch in use, so that projects following semantic version ranges would pick them up automatically.
- The malicious code ran automatically at install time, launching cryptominers and stealing credentials.
- Any machine with the package installed had to be considered fully compromised, with every secret on it rotated from a different machine.

Dependency Chain Abuse

Refers to an attacker's ability to abuse flaws relating to how engineering workstations and build environments fetch code dependencies. Performed in different ways: *dependency confusion*, *dependency hijacking*, *typosquatting*, *brandjacking*.

https://github.com/advisories/GHSA-pjwm-rvh2-c87w

Challenge: *Twiddledum - 200*

- `http://localhost:8000/challenges#Twiddledum-6`
- Scenario:
  - The flag is under `process.env.FLAG6`.
  - You can't interact with `Wonderland/Twiddledum` repository.
  - The `Wonderland/Twiddledum` repository is a JS app that uses `Wonderland/Twiddledee` as a dependency. View its `package.json` file.
- Solution:
  - *Dependency Chain Abuse*
  - `https://github.com/meethack-torino/cicd-goat/blob/main/solutions/twiddledum.md`

ROMHACK 2026 CAMP

Real-world case: *Codecov* (2021)

- The *Codecov Bash Uploader* was a script downloaded and executed at run time by thousands of CI pipelines to upload coverage reports.
- A flaw in Codecov's Docker image creation leaked a Google Cloud Storage credential, used by the attackers to modify the hosted script in place so that it exfiltrated environment variables and `git remote -v` to attacker controlled servers.
- Modified on 31 January 2021, detected only on 1 April 2021, by a customer comparing the script's checksum with the published one.

Ungoverned Usage of 3rd Party Services

Refers to the extreme ease with which a 3rd party service can be granted access to resources in CI/CD systems, effectively expanding the attack surface of the organization.

Improper Artifact Integrity Validation

Allows an attacker with access to one of the systems in the CI/CD process to push malicious (although seemingly benign) code or artifacts down the pipeline, due to insufficient mechanisms for ensuring the validation of code and artifacts.

https://about.codecov.io/security-update

Challenge: *Dormouse - 300*

- http://localhost:8000/challenges#Dormouse-9
- Scenario:
  - You can't interact with `Wonderland/dormouse` repository.
  - The `Jenkinsfile` uses a `reportcov.sh` script.
  - `Cov/reportcov` is a public repository of a 3rd party used by other CI pipelines.
    - It has its own Jenkins job (which you can't view) creating an artifact stored remotely, i.e. `reportcov.sh`.
    - All this can be viewed via the `Jenkinsfile`.
    - You can't interact with the repository.
- Solution:
  - *Ungoverned Usage of 3rd Party Services*
  - *Improper Artifact Integrity Validation*
  - https://github.com/meethack-torino/cicd-goat/blob/main/solutions/dormouse.md

That's all, folks. Thank you!

The other challenges will be solved at Meethack's stand tomorrow from 10:00 to 12:00.

These slides: https://meethack.it/asset/meethack-romhack2026-cicd-security.pdf

ROMHACK 2026 CAMP