

Meethack Torino

CI/CD Security Risks & CI/CD Goat



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 their attention to CI/CD**, realizing CI/CD services provide an efficient path to reaching an organization's crown jewels.

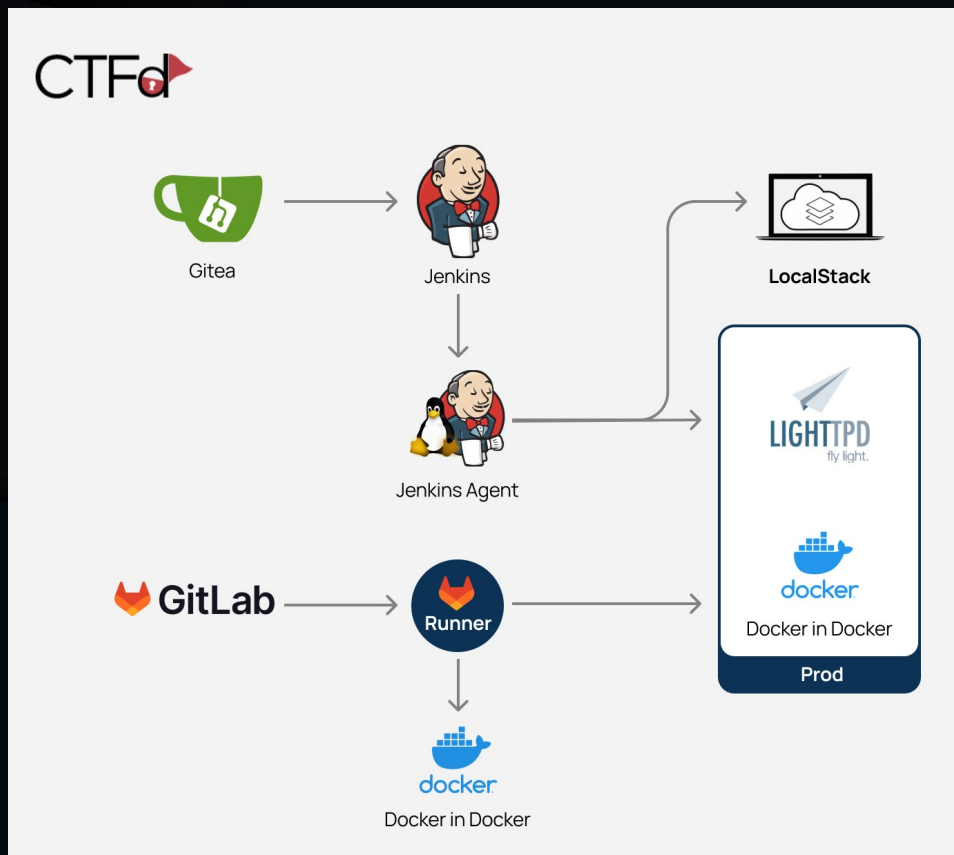
Top 10 CI/CD Security Risks

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

Proposed by **Cider Security** (acquired by Palo Alto Networks).

Still an OWASP “**Lab Project**”.

CI/CD Goat: our playground



- Things to keep in mind:
 - Builds will start on Jenkins, usually after a *Pull Request (PR)*.
 - PRs can be created between *main* branch and a newly created branch on Gitea.
 - *Jenkins will mask plain text secrets* leaked in console output. This will happen in every challenge!
 - *Each challenge stands on its own*. Do not 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*.

Just a couple of prerequisites...

git client 101

- Clone a repository (use Gitea credentials):
 - `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 commit -am "Your message"`
- Push to the remote branch:
 - `git push -u origin <branch_name>`

An example of *Jenkinsfile*

```
1 pipeline {
2   agent any
3   environment {
4     PROJECT = "src/urllib3"
5   }
6
7   stages {
8     stage ('Install_Requirements') {
9       steps {
10        sh ""
11        virtualenv venv
12        pip3 install -r requirements.txt || true
13        ""
14      }
15    }
16
17    stage ('Lint') {
18      steps {
19        sh "pylint ${PROJECT} || true"
20      }
21    }
22
23    stage ('Unit Tests') {
24      steps {
25        sh "pytest"
26      }
27    }
28  }
29  post {
30    always {
31      cleanWs()
32    }
33  }
34 }
35 }
36 }
```

- *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.”*

Challenge: *White Rabbit* - 100

- <http://localhost:8000/challenges#White%20Rabbit-1>
- Scenario:
 - Use your access to the *Wonderland/white-rabbit* repository to steal the *flag1* secret stored in the Jenkins credential store.
 - Secret is stored with the *Global* scope, which makes it accessible to any pipeline on the Jenkins instance.
 - *Jenkinsfile* is not protected.
 - <https://www.jenkins.io/doc/book/pipeline/jenkinsfile/#handling-credentials>
- Solution:
 - *Direct Poisoned Pipeline Execution (D-PPE)*
 - <https://github.com/cider-security-research/cicd-goat/blob/main/solutions/white-rabbit.md>

Challenge: Mad Hatter - 100

- <http://localhost:8000/challenges#Mad%20Hatter-3>
- Scenario:
 - Use your access to the *Wonderland/mad-hatter* repository to steal the *flag3* secret.
 - *Jenkinsfile* is protected: the pipeline is configured in a separate repository from where the application code is stored at. The attacker doesn't have permission to trigger a pipeline with a modified *Jenkinsfile*.
- Solution:
 - *Indirect Poisoned Pipeline Execution (I-PPE)*
 - <https://github.com/cider-security-research/cicd-goat/blob/main/solutions/mad-hatter.md>

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: *-prod* and *-test*.
 - *Think about this repository like an "open source" one... And escalate your privileges to control it.*
- Solution:
 - *Public Poisoned Pipeline Execution (3PE)*
 - <https://github.com/cider-security-research/cicd-goat/blob/main/solutions/caterpillar.md>
 - Considerations about the *when* condition in the *deploy stage* of the pipeline?
 - You can both: remove the condition for the attack and do it via PR or leave it and push directly on *main* branch instead.

Challenge: Cheshire Cat - 200

- <http://localhost:8000/challenges#Cheshire%20Cat-5>
- Scenario:
 - All jobs in your victim's Jenkins instance run on dedicated nodes. You want to execute code on the Jenkins Controller.
 - Use your access to the *Wonderland/cheshire-cat* repository to run code on the Controller and steal *~/flag5.txt* from its file system.
 - <https://www.jenkins.io/doc/book/managing/nodes/#components-of-distributed-builds>
 - <https://www.jenkins.io/doc/book/pipeline/syntax/#agent>
 - <http://localhost:8080/computer/>
- Solution:
 - *Direct Poisoned Pipeline Execution (D-PPE)*
 - *Insufficient PBAC (Pipeline-Based Access Controls)*
 - <https://github.com/cider-security-research/cicd-goat/blob/main/solutions/cheshire-cat.md>
 - Error in the solution: the file name is missing *.txt* extension.

Challenge: *Duchess* - 100

- <http://localhost:8000/challenges#Duchess-4>
- Scenario:
 - You've got access to the *Wonderland/duchess* repository, which heavily uses Python. There must be some *PyPi token* left somewhere. Can you find it?
 - The flag is the token.
- Solution:
 - *Insufficient Credential Hygiene*
 - **You could need external tools for this!**
 - `gitleaks detect -v --enable-rule="pypi-upload-token"`
 - `docker run -v <full_path_to_host_folder_to_scan>:/path zricethezav/gitleaks:latest detect --source="/path" -v --enable-rule="pypi-upload-token"`
 - <https://github.com/cider-security-research/cicd-goat/blob/main/solutions/duchess.md>

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.
 - Trying to add pre or post-install scripts with malicious code should fail, as the *Twiddledum* pipeline runs with the `--ignore-scripts` param.
 - <https://docs.npmjs.com/cli/v9/using-npm/scripts>
- Solution:
 - *Dependency Chain Abuse*
 - <https://github.com/cider-security-research/cicd-goat/blob/main/solutions/twiddledum.md>

Challenge: *Dodo* - 200

- <http://localhost:8000/challenges#Dodo-7>
- Scenario:
 - Your mission is to make the *dodo* S3 bucket public-readable without getting caught.
 - The flag will be printed in the job's console output once you're done.
 - *Checkov* (<https://github.com/bridgecrewio/checkov>) validates that the S3 bucket created by the Terraform code is private, which stops you from making it public.
 - https://registry.terraform.io/providers/hashicorp/aws/latest/docs/resources/s3_bucket_acl
 - <https://www.checkov.io/2.Basics/Hard%20and%20soft%20fail.html>
- Solution:
 - *Insufficient Flow Control Mechanisms*
 - <https://github.com/cider-security-research/cicd-goat/blob/main/solutions/dodo.md>

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*
 - **You could need external tools for this!**
 - <https://github.com/cider-security-research/cicd-goat/blob/main/solutions/dormouse.md>
 - Error in the solution: the command to craft *reportcov.sh* is wrong, here the correct one → `echo 'echo "$FLAG"' | base64' > reportcov.sh`

Challenge: *Mock Turtle* - 300

- <http://localhost:8000/challenges#Mock%20Turtle-10>
- Scenario:
 - Can you push to the *main* branch of the *Wonderland/mock-turtle* repository? Do what's needed to steal the *flag10* secret stored in the Jenkins credential store.
 - The pipeline is used to automatically merge code into the *main* branch if it introduces just a version bump (stored in the *version* file).
- Solution:
 - *Insufficient Flow Control Mechanisms*
 - <https://github.com/cider-security-research/cicd-goat/blob/main/solutions/mock-turtle.md>
 - Pay attention to the `\n` in *version* file! Be sure to remove them!

Challenge: *Hearts* - 300

- <http://localhost:8000/challenges#Hearts-8>
- Scenario:
 - You have to exfiltrate *agent* System credentials stored on Jenkins via a user that has privileged access to manage agents.
 - The users in the Jenkins instance are managed by Jenkins' own user database, which lacks basic security controls against various types of attacks, e.g., credentials bruteforce attempts.
 - <http://localhost:8080/asynchPeople/>
 - <https://www.jenkins.io/blog/2022/12/27/run-jenkins-agent-as-a-service/> (Jenkins version is different)
 - <http://localhost:8080/computer/>
- Solution:
 - *Inadequate Identity and Access Management*
 - Let's skip the bruteforce part... If you need a password, it's "*rockme*".
 - **You could need external tools for this!** To do everything via containers, follow these steps:
 - Launch *SSH-MITM* with:
 - `docker network ls`
 - `docker run --network=<the_goat_network_name_retrieved_before> --name evil-jenkins-agent -it --rm positronsecurity/ssh-mitm`
 - Use *evil-jenkins-agent* as host and *2222* as port.
 - Check for incoming connections entering in the container:
 - `docker exec -it evil-jenkins-agent /bin/bash`
 - `ls /home/ssh-mitm/log/`
 - `cat /home/ssh-mitm/log/sftp_session_*.html`
 - This doesn't make any sense, but it works...
 - <https://github.com/cider-security-research/cicd-goat/blob/main/solutions/hearts.md>

Challenge: Gryphon - 500

- <http://localhost:8000/challenges#Gryphon-11>
- Scenario:
 - You've compromised GitLab. The compromised user is the maintainer of `pygryphon/pygryphon` package.
 - You can click the "Explore projects" button to view public projects.
 - <http://localhost:4000/explore>
 - There are also public projects: `Wonderland/nest-of-gold` and `Wonderland/awesome-app` to which you have read-only access.
 - <https://docs.gitlab.com/ee/ci/yaml/>
 - <https://packaging.python.org/en/latest/tutorials/packaging-projects/>
 - <https://packaging.python.org/en/latest/specifications/pypirc/>
 - https://docs.gitlab.com/ee/user/packages/pypi_repository/#authenticate-with-a-personal-access-token
 - http://localhost:4000/-/profile/personal_access_tokens
- Solution:
 - *Dependency Chain Abuse*
 - *Insufficient Credential Hygiene*
 - <https://github.com/cider-security-research/cicd-goat/blob/main/solutions/gryphon.md>
 - `git clone http://localhost:4000/pygryphon/pygryphon.git`
 - Patch the source code and create `.pypirc` file.
 - `python3 -m build`
 - Remove existing packages under <http://localhost:4000/pygryphon/pygryphon/-/packages>
 - `python3 -m twine upload -r gitlab --config-file .pypirc --verbose dist/*`

Bugged challenge: it fails during the test stage!
T_T

<https://github.com/cider-security-research/cicd-goat/issues/71>

Backup
