Dealing with DevSecOps Findings

Open Security Summit 2019

Agenda

Objectives (AM1)

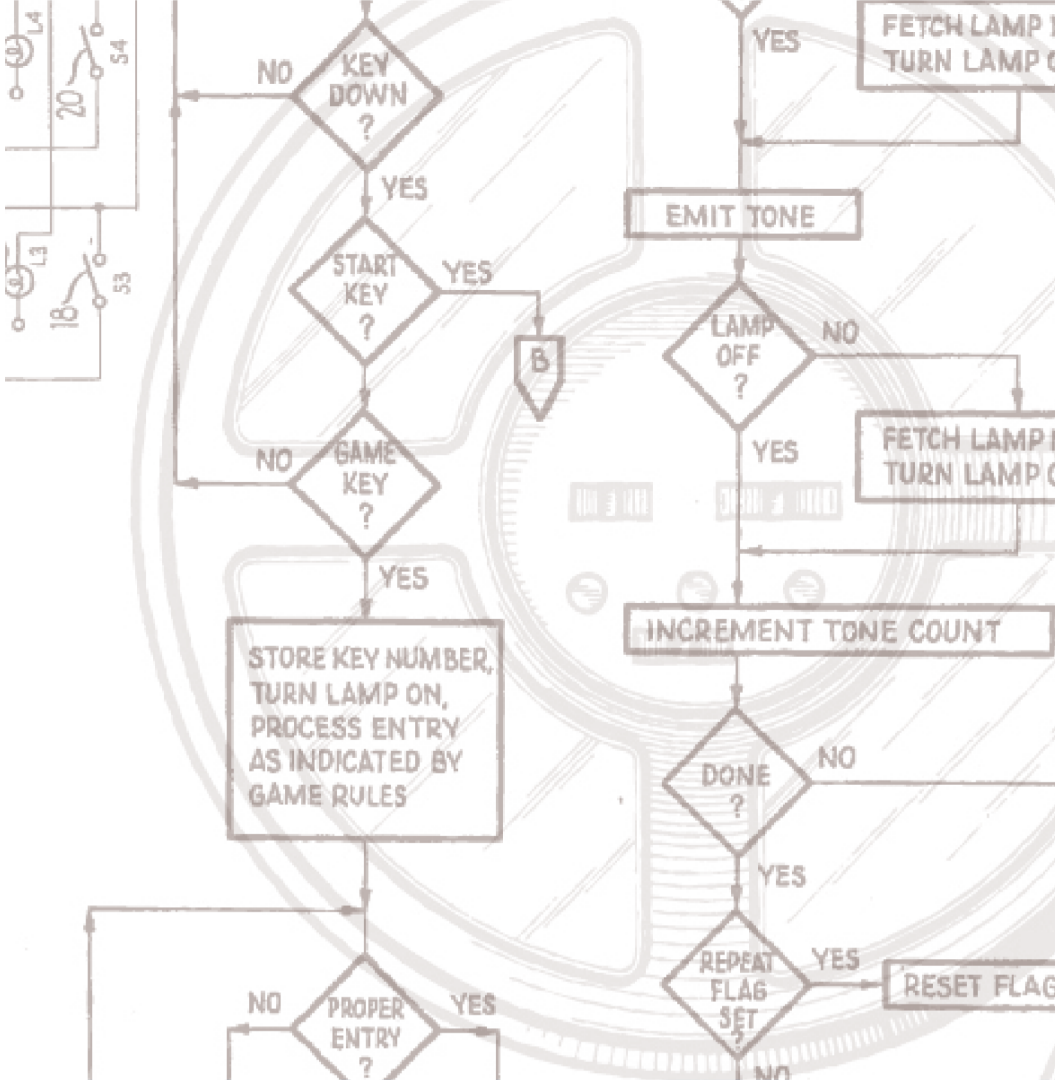
- Project Initiatives & Key Results
- Session Objectives
- The Lab
- The Pipeline

Workflow (PM2)

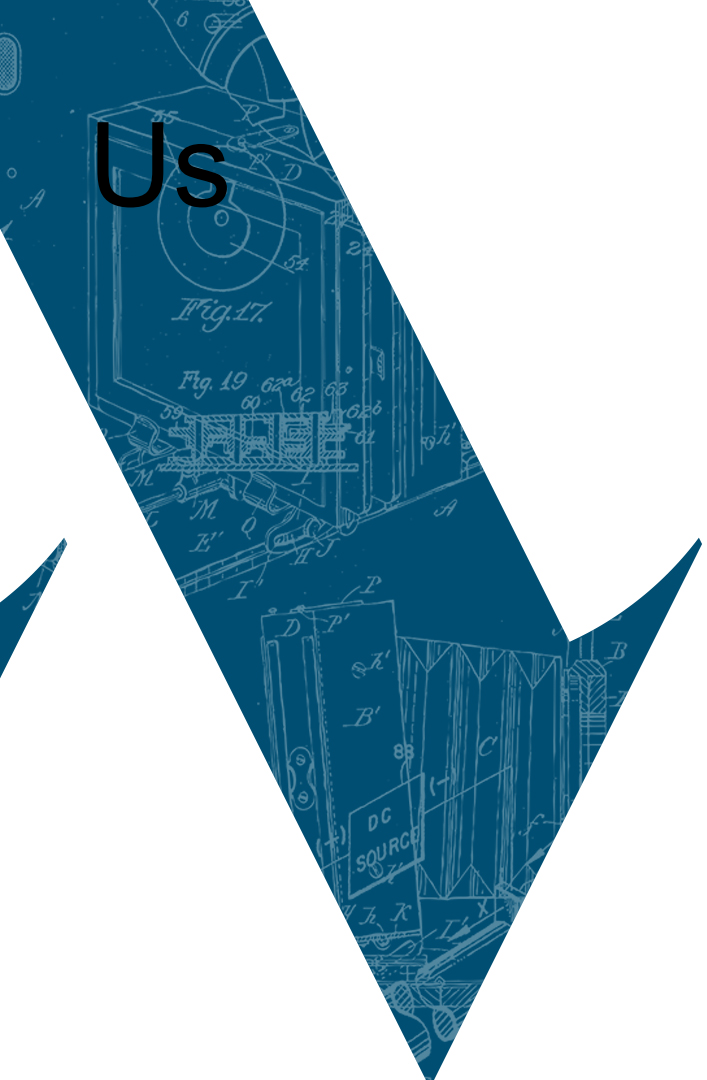
- People
- Process
- Technology
- Q&A

Outcomes (PM3)

- Fix TODOs
- Document process to handle findings



Us



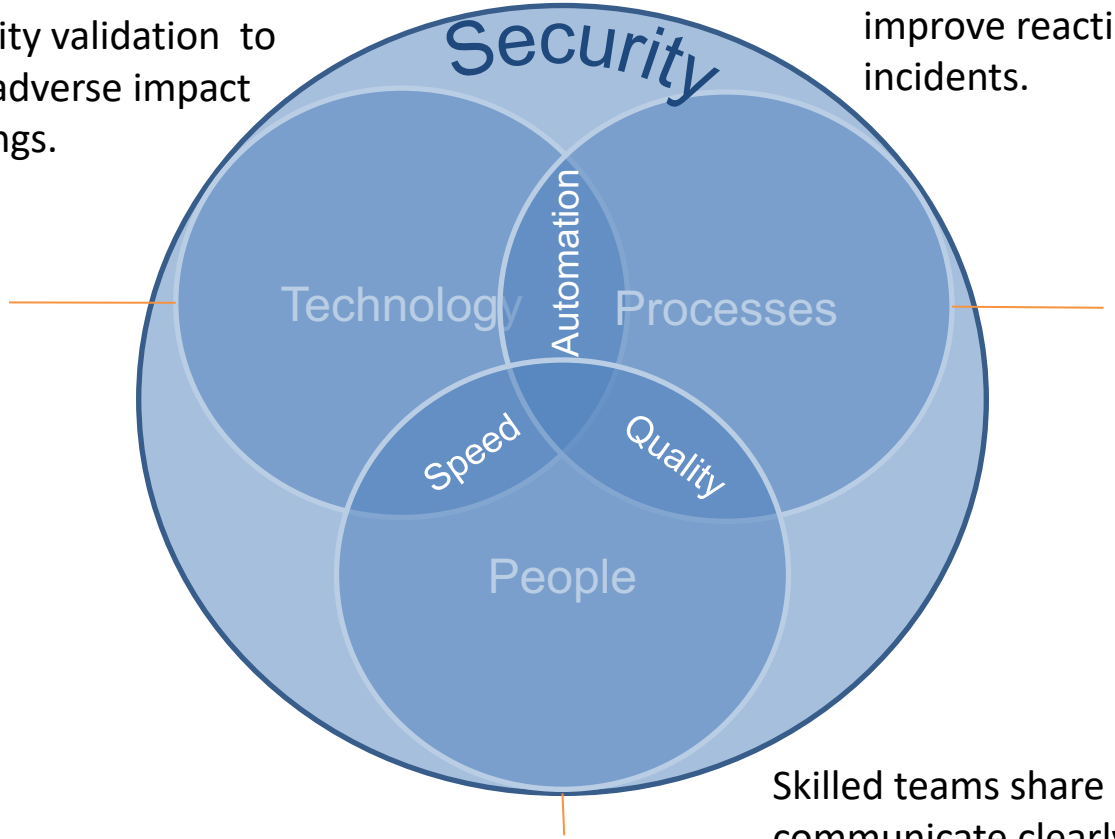
Claudio Camerino - AppSec
Francisco Novo - DevOps
Rafael Jimenez - Delivery Services

THE WORKSHOP

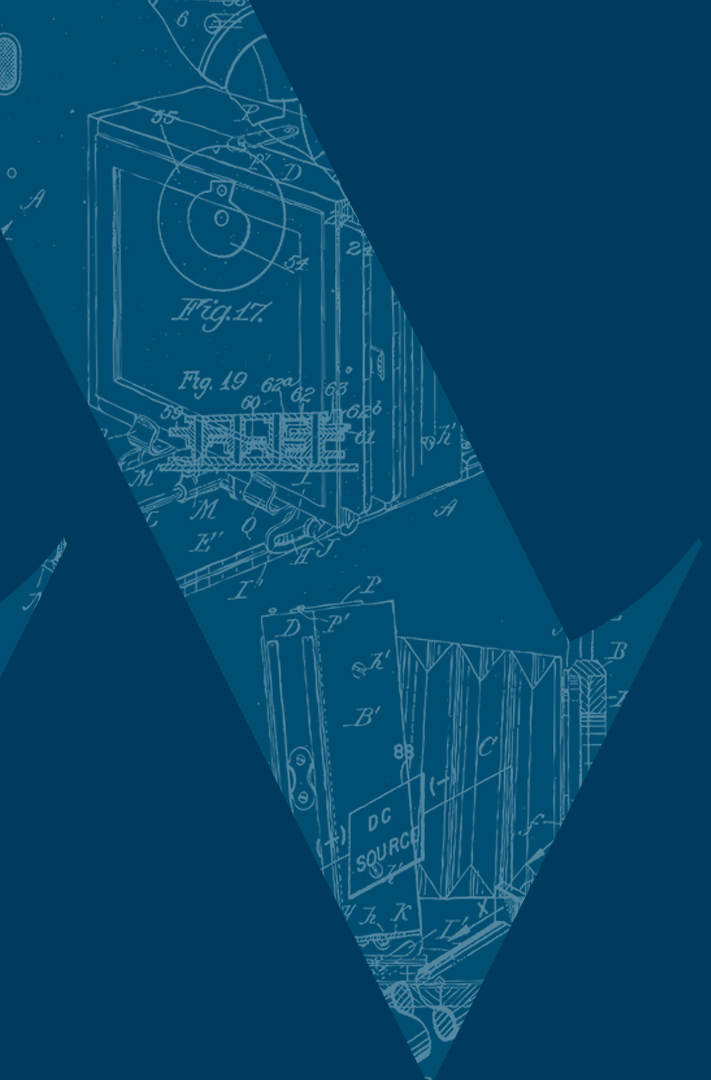
DevSecOps – what is it ?

Tech enabls security validation to scale without an adverse impact on costs and timings.

Processes enable organisations to set measurable goals, drive efficiency and improve reaction time to security incidents.



Skilled teams share a common goal and communicate clearly



Objectives

- What are we doing?
- What are we trying to achieve?

Initiatives

- Scale generation, collection and triaging of security findings through automation of tools and processes
- Introduce appsec tests early in the SDL
- Drive security tests through QA test.
- Inform development teams of security issues with their products
- Triage and action reputable findings
- Dynamically improve testing policies.

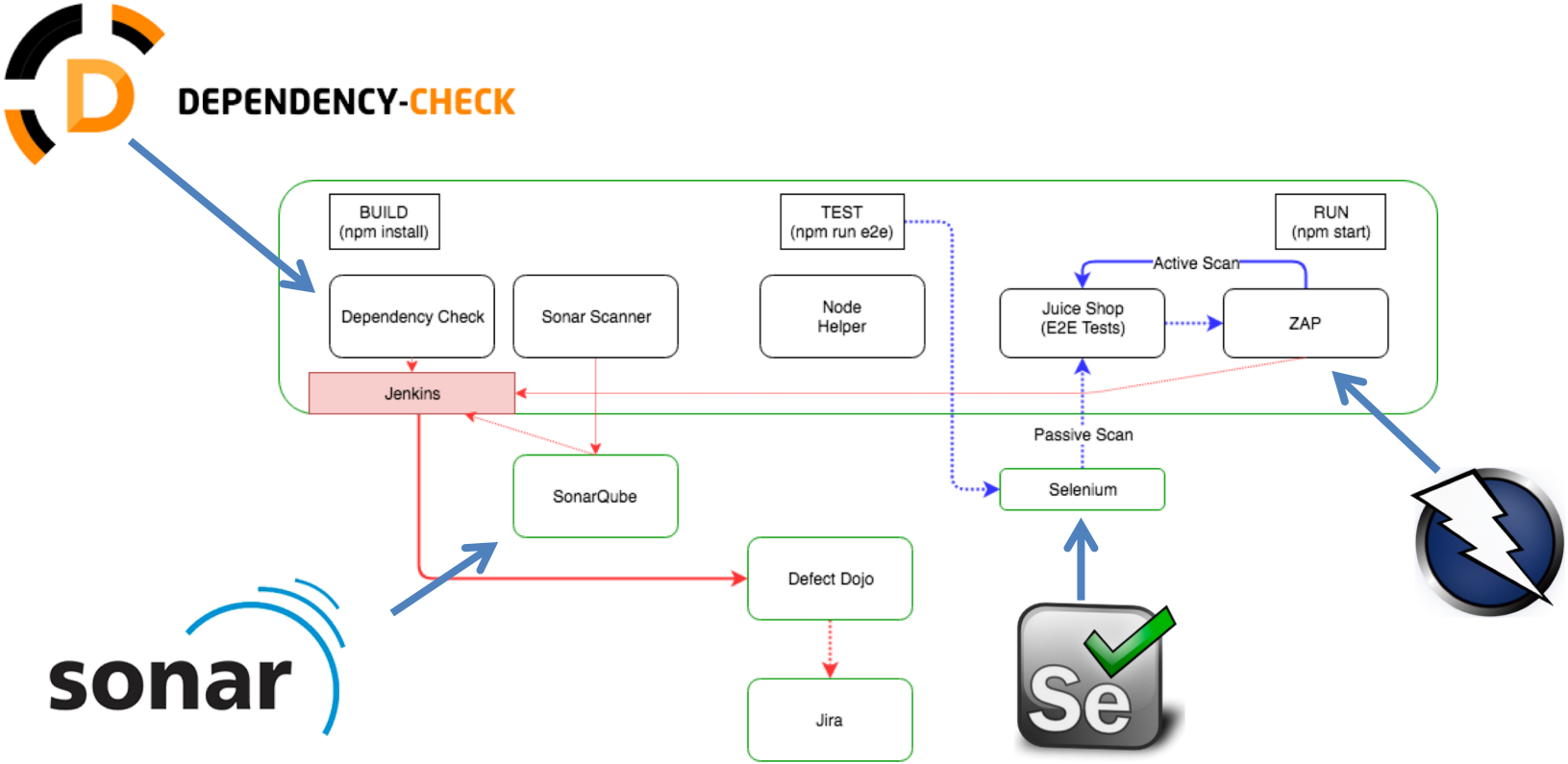
Key Results

- Implemented security checks for CD pipeline, on-board 25% of target products
- Reduced false and true positive for security findings by 30%
- Scale, automate and simplify the collection, triage and mitigation of security defects

What we've done

- Build a working DevSecOps POC lab
- Hacked OSS tools like ZAP and Defect Dojo
- Create CD scripts to generate, collect and route findings
- Jenkins groovy scripts to tie it all together ..

Lab Architecture



Lab - Admin Tools

User(s): dwdsof
Password: oss2019

SSID: OSS19
PWD: dwdsof2019

Jenkins

http://jenkins.oss.local:8081

DEFECT DOJO

http://defect-dojo.oss.local:8082

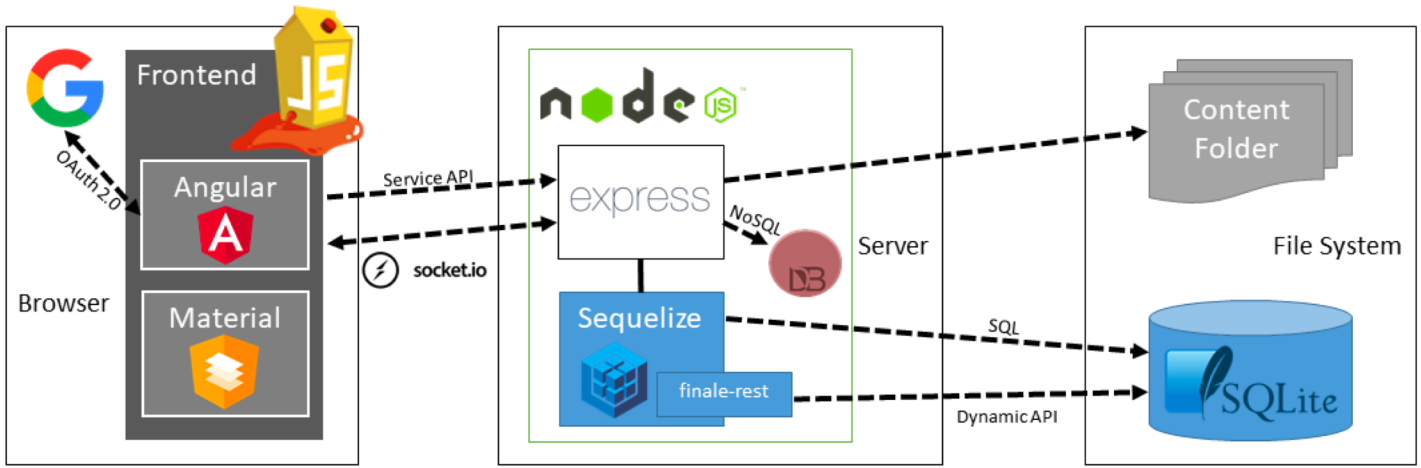
Jira

http://jira.oss.local:8083

http://juice-shop.oss.local:3000

Why Juice Shop ?

Juice Shop



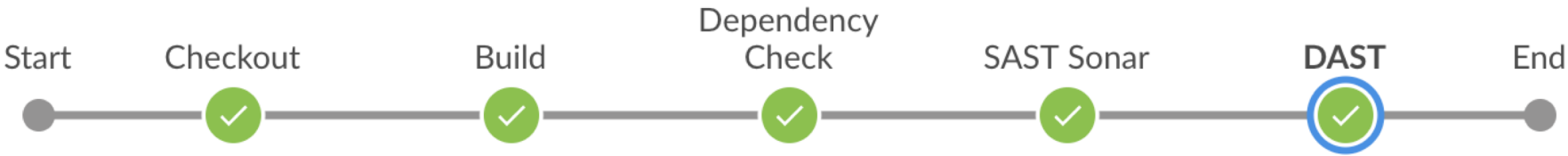
A considerable number of vulnerable web applications already existed before the Juice Shop was created. The OWASP Vulnerable Web Applications Directory (VWAD) maintains a list of these applications. When Juice Shop came to life there were only *server-side rendered* applications in the VWAD. But *Rich Internet Application (RIA)* or *Single Page Application (SPA)* style applications were already a commodity at that time. Juice Shop was meant to fill that gap.

Goals for today

- Review pipeline stages and tools in detail
- Critique our approach, get feedback
- See how findings are generated
- Define ruleset for removal of noise
- Tune security tools policies, contexts and scripts
- Handle security tickets through Jira and Defect Dojo

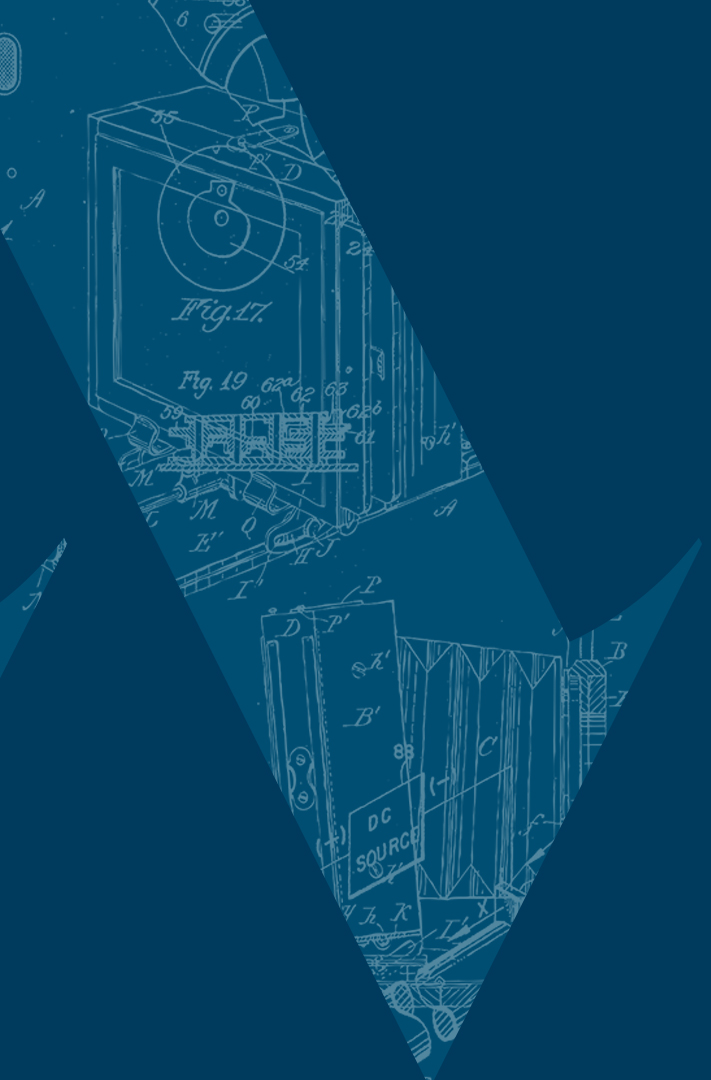
The Pipeline

Juice Shop



Q&A

- What does automation mean to you?
- What are your objectives?
- What problems are you trying to solve?
- What do you expect out of this session?
- What outcomes would you like to see?



Workflow

- What is the lifecycle of a security finding?
- What are the technologies involved?

People

Skills

- AppSec - Define and build security policies
- Developers - Tune and maintain product-specific security policies
- Developers - Triage security findings
- Quality Assurance – Integrate security tools, create abuse cases as e2e tests
- AppSec/Leadership - Risk management

Changes

- Re-enforced message that security is everyone’s job
- Security test become an extension of QA test.
- Knowledge sharing and support structure in place for dev teams

People

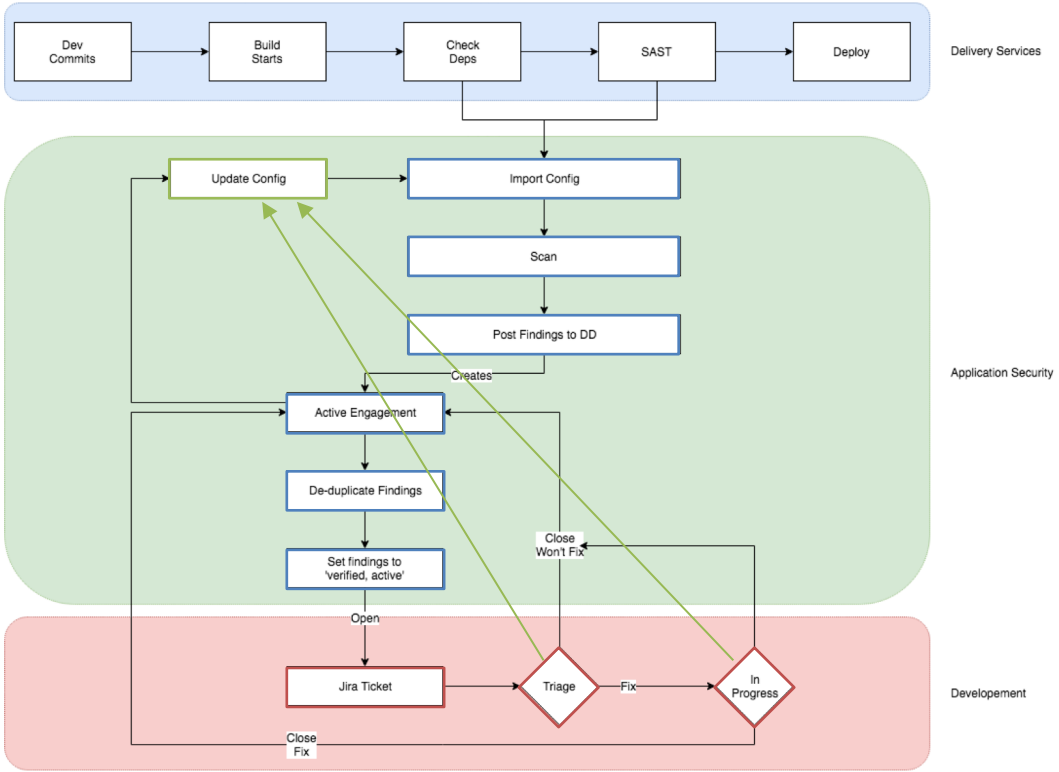
Responsibilities

- Development teams to fix vulnerabilities for own product
- AppSec Team to provide training, support and act as escalation point
- Security Champions to act as the "voice" of security for own channel

Methodology

- Define and formalise the role of Security Champion
- Generate and track metrics for code quality, time fix and rate of improvement
- Triage and fixing of vulnerabilities handled as part of technical debt

Process



Changes

- Centrally managed and dynamically updated policies
- Dev able to review and action own security findings
- Continuous feedback loop drives reduction of defects

Responsibilities

- AppSec team provides governance and support
- Dev teams define Jira workflow
- Delivery Services create and maintain build plans

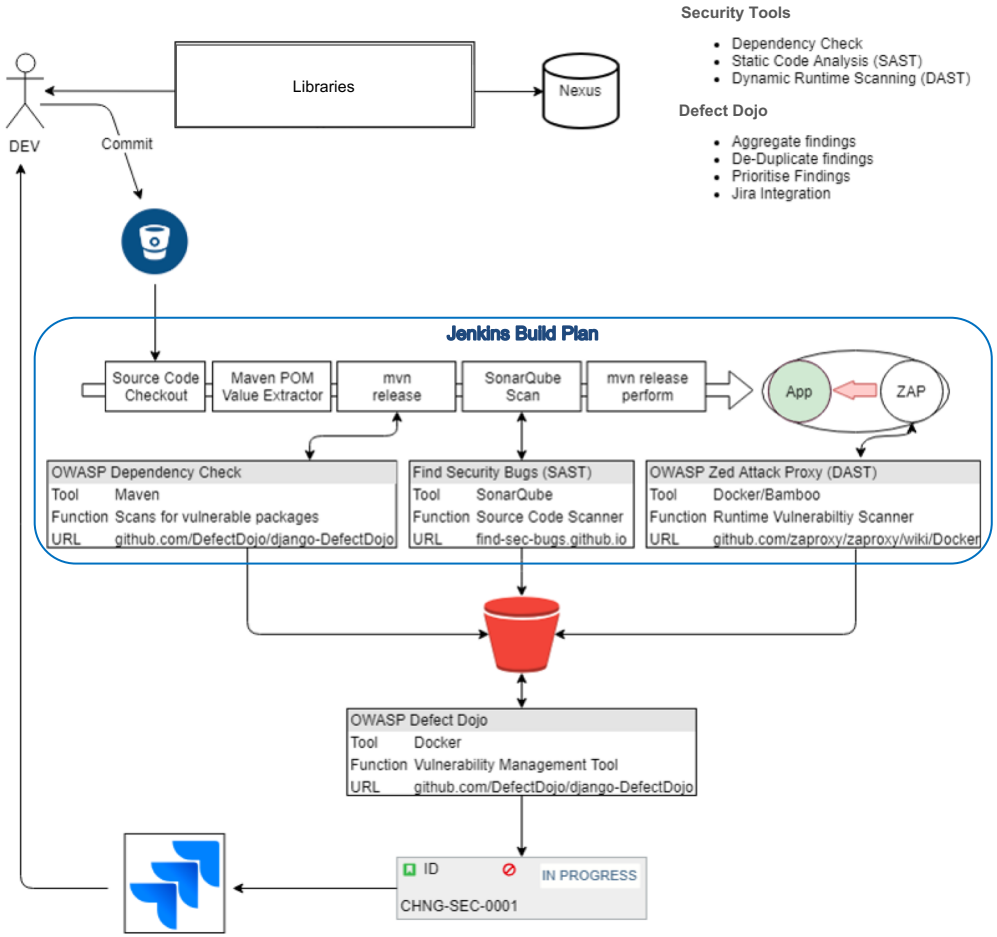
Workflow

Changes

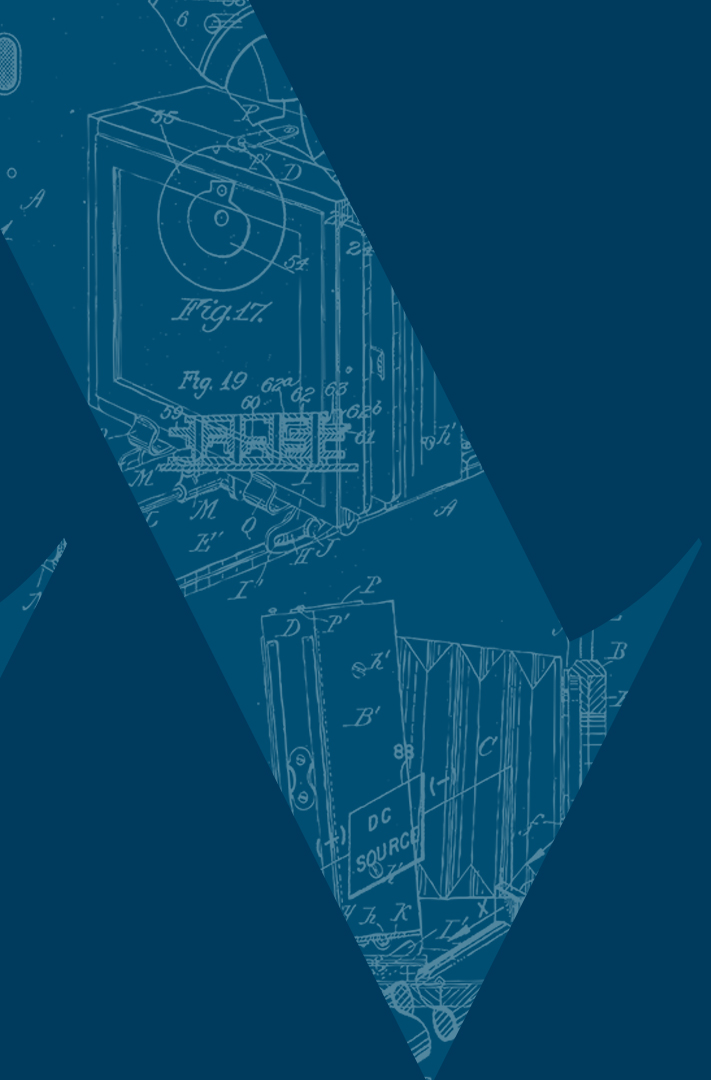
- Check for vulnerable libraries
- Security rules in Sonar
- QA e2e drive security test
- Single SoT for security
- Jira only interface for Dev

Responsibilities

- AppSec team configures tools and policies
- Dev teams tune/maintain policies for own products
- Delivery Services implement bamboo plans




LIVE DEMO



Docker Compose

- ....
- ....

CI/CD Server - Jenkins

```
jenkins.env.local:
  build:
    context: ./jenkins
  image: jenkins:twsoss2019
  ports:
    - '8081:8080'
    - '8443:8443'
  volumes:
    -
'./data/jenkins_data:/var/jenkins_home'
    # Docker in Docker
    -
/var/run/docker.sock:/var/run/docker.sock
```

SonarQube - SAST

```
# SAST
sonar.env.local:
  image: sonarqube
  ports:
    - "9000:9000"
  environment:
    - sonar.jdbc.url=jdbc:postgresql://db:5432/sonar
  volumes:
    - './data/sonarqube_conf:/opt/sonarqube/conf'
    - './data/sonarqube_data:/opt/sonarqube/data'
    - './data/s..extensions:/opt/sonarqube/extensions'
```

```
db:
  image: postgres:9.5-alpine
  volumes:
    - './data/postgresql:/var/lib/postgresql'
    - './data/postgresql_data:/var/lib/postgresql/data'
```

Vulnerabilities Tracker - Defect Dojo

```
dd.env.local:
    image: defectdojo/defectdojo-nginx:latest
    depends_on:
      - uwsgi
    ports:
      - "8082:8080"
uwsgi:
    image: defectdojo/defectdojo-django:latest
    depends_on:
      - mysql
rabbitmq:
    image: rabbitmq:3.7
```

```
celerybeat:
    image: defectdojo/defectdojo-django:latest
    depends_on:
      - mysql
      - rabbitmq
celeryworker:
    image: defectdojo/defectdojo-django:latest
    depends_on:
      - mysql
      - rabbitmq
mysql:
    image: mysql:5.7
    volumes:
      - './data/mysql_data:/var/lib/mysql'
```

Issues Tracker - Jira

```
jira.env.local:
   depends_on:
     - postgresql
   build:
     context: ./jira
   image: teamatldocker/jira
   ports:
     - '8083:8080'
   volumes:
     -
'./data/jira_data:/var/atlassian/
jira'

postgresql:
   image: postgres:9.5-alpine
   volumes:
     -
'./data/jira_postgresqldata:/var/
lib/postgresql/data'
```

Selenium Chrome Standalone

```
selenium:
  image: selenium/standalone-chrome:3.141.59-oxygen
  ports:
    - "4444:4444"
  volume:
    - /dev/shm:/dev/shm
```

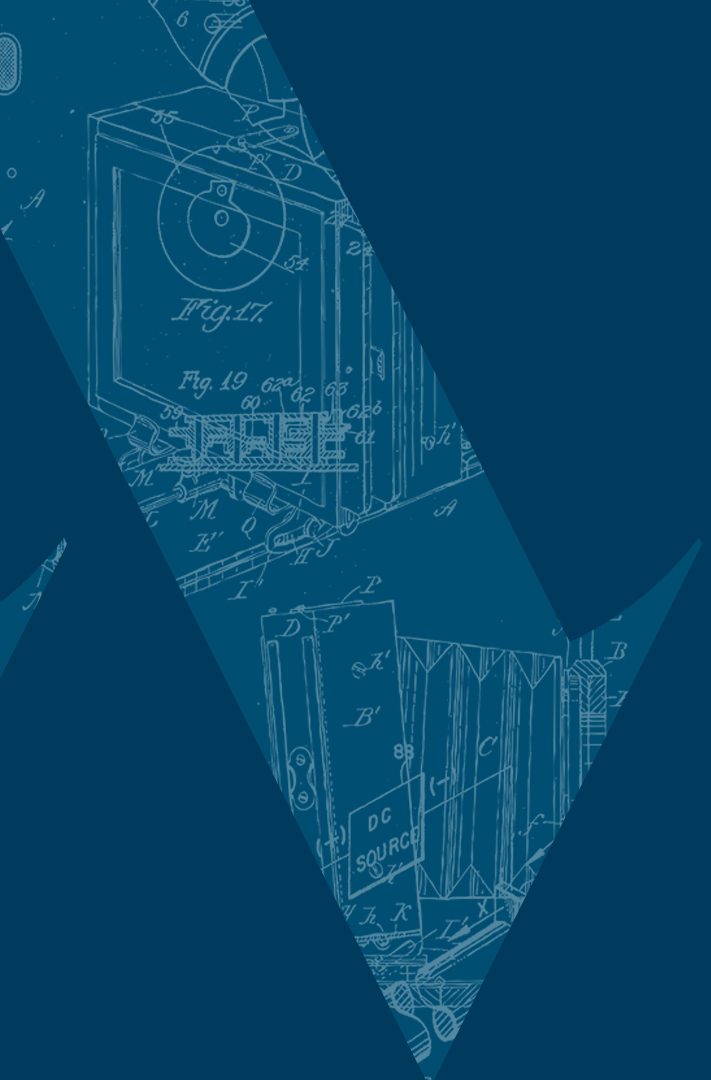
Tools Configuration Deep Dive

Groovy Script

- Review and critique pipeline
- ZAP - Tune policies, contexts and scripts
- SAST Configure - SonarQube quality profiles
- Dependency Check - Configuration

Dojo Code

Defect Dojo/Jira - Integration Scripts



Outcomes

What we want to achieve

Session Outcomes

- Measure Key Results
- Stakeholders Management
- Best Practices
- Top 5 (or better) common automation issues

Best Practices

- Robust product selection mechanism
- Route findings to author or team
- Tune scan policies
- Inform, don’t enforce
- Use right tools for the job
- Try not introduce new tools to Devs

Top 5 Automation Challenges

- Detect business logic flaws
- Generate and maintain rulesets
- Determine how much detail to include
- Create feedback loops
- Policy tuning

Long-term Outcomes

1. Publish Repo
    1. Docker Compose
    2. Integration/automation scripts
    3. Security policies
2. Pull Request for Defect Dojo
    1. Parametised scan types
    2. Set finding status ”in progress”
3. Document process to handle findings