Securing the Continuous Deployment Pipeline

Len Bass, Ralph Holz, Paul Rimba, An Binh Tran, Liming Zhu
The software supply chain has a great deal of diversity

Suppliers from all over the world

Operational System

- Open Source
- In House
- Proprietary
- 3rd party
Many opportunities to corrupt delivery

• Rogue versions of 3rd party software
• Replace desired operational system with compromised version
• Leave “back door” in operational system
• Network access
• Credentials
• Software complexity

...
Deployment pipeline is the “last mile” of the supply chain

- The term “Last Mile” comes from telco and logistics
- It refers to the difficulties in getting goods and software to the consumer from a distribution centre
The security requirements and threats

• The security requirement we will discuss in this talk: the image deployed into operation is a valid image
  – This is an integrity requirement
    • The integrity of the specification of the image has not been compromised
      – Example violation: overwrite dockerfile
    • The image built is the image specified
      – Example violation: pulling the “wrong” version of code
    • The image deployed is the image built
      – Example violation: deploy wrong image

• Other security requirements exist but we do not focus on them in this talk
How do we secure a pipeline?

• Analyse a model of the pipeline to detect vulnerabilities (from design perspective)
• Restructure and remodel pipeline to remove vulnerabilities
• Ideally, we are able to remove all of the vulnerabilities. In this case the pipeline is “secure”
• Reality: we are not able to remove all vulnerabilities (at least not now). In this case, the pipeline has been “hardened”
A pipeline is complicated!!
A pipeline is complicated!!
OUR PROCESS
Steps to harden the deployment pipeline

• Identify security requirements to be satisfied
  – Apply principle of least privilege, isolation
    • No components should be able to damage other components
    • Communications between components are well specified and enforced

Diagram:

- A
- B
- C

Arrow from B to C: Write on B's Workspace

NICTA Copyright 2015
Steps to harden the deployment pipeline

• Repeat until all of the requirements have been satisfied OR can no longer decompose the untrustworthy components:
  – Model the interactions between the components
  – Analyse the model to check whether it satisfies our requirements
  – Decompose untrustworthy components causing an unsatisfied requirement into a trustworthy and an untrustworthy portion
    • Reduce the number of untrustworthy portions in the system
    • This is the “hardening” part
Steps to harden the deployment pipeline

• Repeat until all of the requirements have been satisfied OR can no longer decompose the untrustworthy components:
  – Model the interactions between the components
  – Analyse the model to check whether it satisfies our requirements
  – Decompose untrustworthy components causing an unsatisfied requirement into a trustworthy and an untrustworthy portion
    • Reduce the number of untrustworthy portions in the system
    • This is the “hardening” part
Steps to harden the deployment pipeline

- Repeat until all of the requirements have been satisfied OR can no longer decompose the untrustworthy components:
  - Model the interactions between the components
  - Analyse the model to check whether it satisfies our requirements
  - Decompose untrustworthy components causing an unsatisfied requirement into a trustworthy and an untrustworthy portion
    - Reduce the LOC of untrustworthy portions in the system
    - This is the “hardening” part

- Implement new trustworthy components and modify untrustworthy components to utilize the trustworthy components to perform sensitive operations.
Original Build Server

- Build Server is a monolithic component
  - Large code-base
  - All the processes run under the same process space and privileges
Goal: Hardened Pipeline

• Orchestrator + Microservices
  – Many microservices are small enough to be verified
    • We accept that not all can be verified
    • Verified for correctness (i.e. behave as specified)
IN PRACTICE
From theory to practice

- We acknowledge reluctance to change
- Jenkins is the standard go-to build server
  - We use Jenkins as our build server
- Introduce a Jenkins plugin to enable microservices into the build server
  - Take advantage of Microservice architecture through well-defined API that we proposed
  - Microservices will do the actual work
Potential for damage

Pre Steps

Add pre-build step

Build

Root POM
pom.xml

Goals and options
clean package

Post Steps

Run only if build succeeds
Run only if build succeeds or is unstable
Run regardless of build result

Should the post-build steps run only for successful builds, etc.

Execute shell

Command

DOCKER_IMAGE=repo.research.nicta.com.au/${JOB_NAME}:${BUILD_NUMBER}
echo "Build new Docker image ${DOCKER_IMAGE}"
docker build -t ${DOCKER_IMAGE} ${WORKSPACE}
rm -rf ../../Project_B/workspace/*
echo "Push Docker image to remote image repository"
docker push ${DOCKER_IMAGE}
echo "Deploy new image to Chef environment ${JOB_NAME}"
java -jar deployer.jar jobname=${JOB_NAME} dockerimage=${DOCKER_IMAGE}

See the list of available environment variables
Potential for damage

Execute shell

```
DOCKER_IMAGE=repo.research.nicta.com.au/${JOB_NAME}:${BUILD_NUMBER}

echo "Build new Docker image ${DOCKER_IMAGE}"
docker build -t ${DOCKER_IMAGE} ${WORKSPACE}

rm -rf ../../../Project_B/workspace/*

echo "Push Docker image to remote image repository"
docker push ${DOCKER_IMAGE}

echo "Deploy new image to Chef environment ${JOB_NAME}"

java -jar deployer.jar jobname=${JOB_NAME} dockerimage=${DOCKER_IMAGE}
```

--- Running in 7e3d2d3b657b
--- ffdea9243904
Removing intermediate container 7e3d2d3b657b
Successfully built ffdea9243904
+ rm -rf ../../../Project_B/workspace/Dockerfile ../../../Project_B/workspace/README.md
  ../../../Project_B/workspace/pom.xml ../../../Project_B/workspace/src ../../../Project_B/workspace/target
+ echo Push Docker image to remote image repository
Push Docker image to remote image repository

prod.research.nicta.com.au Running handlers:
prod.research.nicta.com.au Running handlers complete
prod.research.nicta.com.au Chef Client finished, 5/9 resources updated in 16.10195661 seconds
Finished: SUCCESS
One working solution: Sandbox shell

- User only interact via API
- API functionalities
  - Spawn Docker container with specified VM settings (Image, CPU/RAM limit, etc.)
  - Map In dir (read-only) & Out dir (r/w access) to folders in container
  - Put shell commands into container
  - Security mechanisms enforcement
  - Reduce attack surface on filesystem of Host to just the specified Out dir
Sandbox shell as Jenkins plugin

**Virtualized Shell execution**

**VM Settings**
- Virtualization Type: Docker
- VM Image Name: maven-oracle-java-8
- Enable Networking: checked

**Execution Request**
- Input Directory: /home/user/code_workspace
- Output Directory: /home/user/target_workspace
- Shell command: mvn package

---

**Virtualized Shell execution**

**VM Settings**
- Virtualization Type: Docker
- VM Image Name: docker-1.6
- Enable Networking: checked

**Execution Request**
- Input Directory: /home/user/target_workspace
- Output Directory: /home/user/image_workspace
- Shell command:
  ```
  DOCKER_IMAGE = repo.research.nicta.com.au/${JOB_NAME}:${BUILD_NUMBER}
docker build -t ${DOCKER_IMAGE} .
  ```
Hardening the pipeline

• When we can fix some vulnerabilities but not all we say we have “hardened” the pipeline

• Our recommendations involve controlling access to resources (network, I/O, CPU, RAM)

• Ongoing: implementing micro components that communicate with Jenkins

• Ongoing: formal verification on the micro components
Summary

• Our contributions are
  – The creation of an engineering process to evaluate/modify the design of a deployment pipeline
  – Architectural recommendations for the tools in the pipeline
  – Presented one practical example of hardening a pipeline
    • A plugin that enables microservice architecture
    • Sandbox shell

• Our process is based on
  – Identifying trustworthy components,
  – Patching vulnerabilities by creating small trustworthy components,
  – Refining until no vulnerabilities remain.

• The specifics of what we have done depends on the technologies we use but the process will work for any collection of technologies.