


Scanning Docker Images for Fun and Compliance

Natalia King & David Ochel
BSides Austin, May 05, 2023

Objective & Agenda

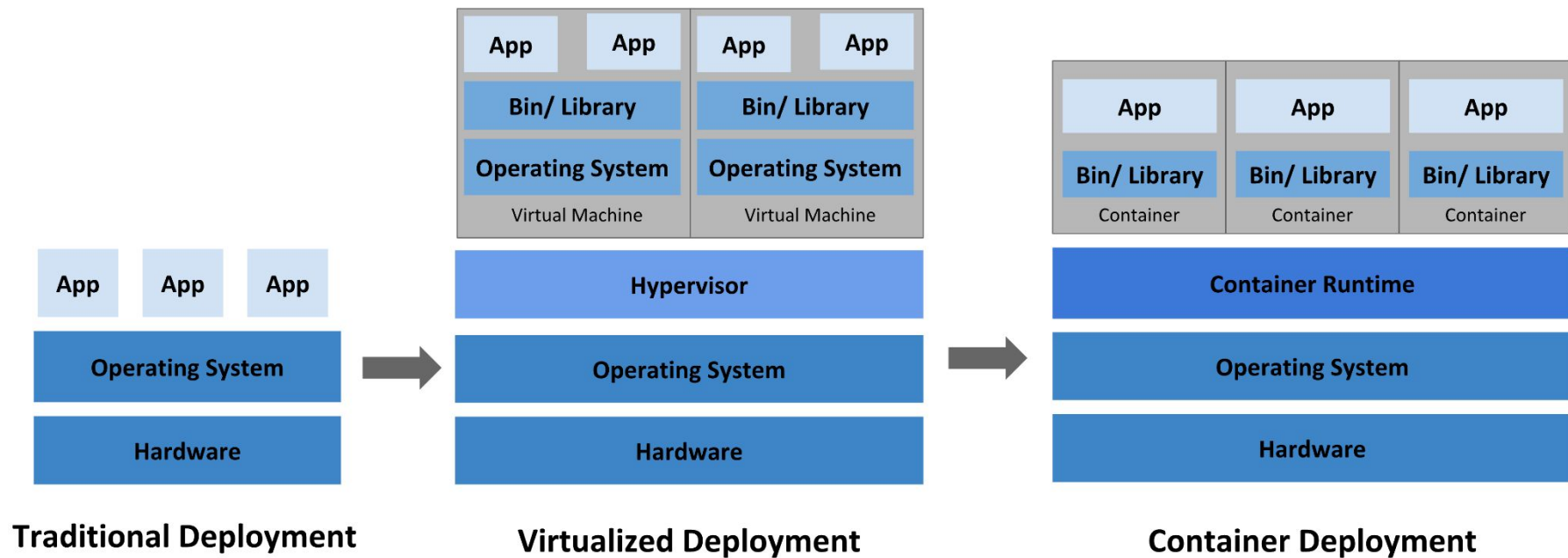
Effectively managing publicly known vulnerabilities in container-based workloads:

1. Concept – docker images and how/why to scan them
 2. Implementation options – when, and where in the pipeline, to scan
 3. Dealing with scan results – applicability, triage, resolution
 4. Compliance reporting – accommodating needs & wants
 5. Recap – common pitfalls and how to avoid them
 6. Questions & Answers
- 



Intro: Docker Images & Containers

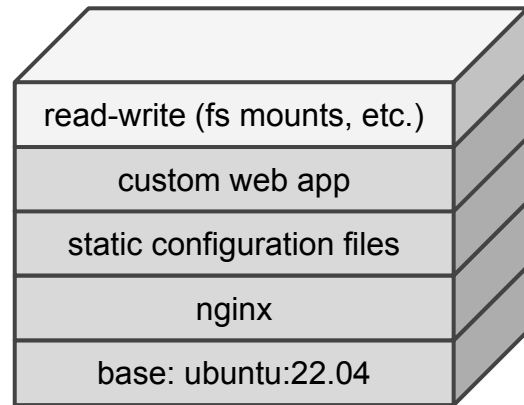
Containerization – Concept



Source: <https://kubernetes.io/docs/concepts/overview/>

Container Images

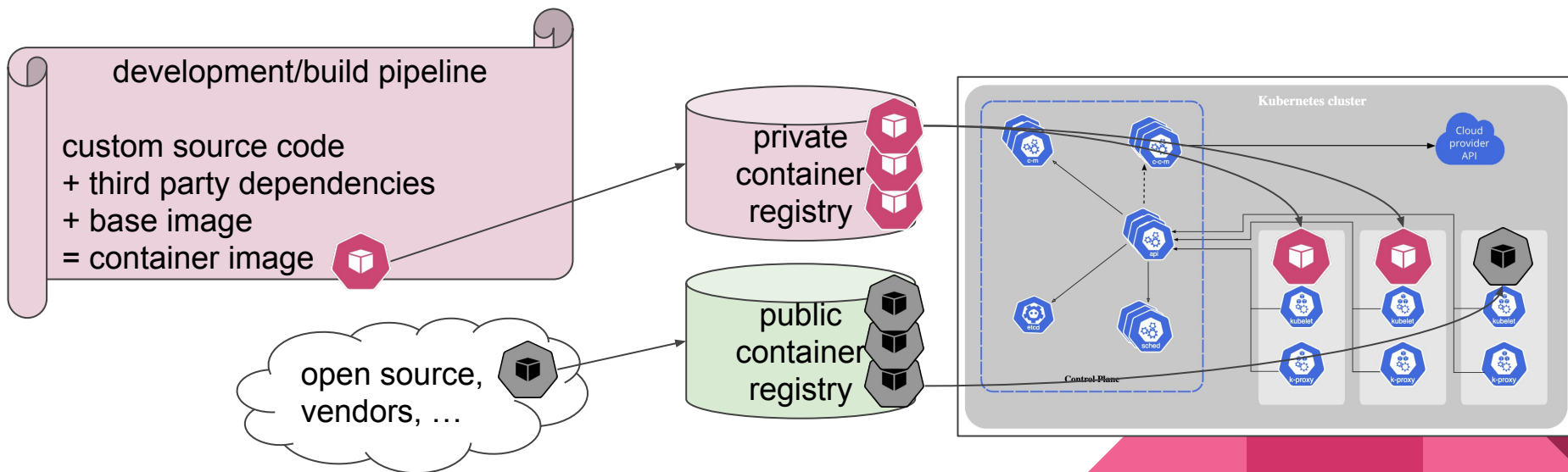
- Container workloads are...
 - ...single purpose (one app per container, think microservices)
 - ...mostly immutable (file systems, secrets, etc. are externally mounted)
 - ...ephemeral (expected to be re-deployed frequently)
- ...instantiated from images (“docker images”)
 - Layered: base (often, a OS flavor) and dependency/app layers
 - Re-built and re-deployed when changes need to be made




Example container layers

Container Orchestration

- Deployment of workloads is typically orchestrated (often using Kubernetes)
 - Declarative approach for describing resources, desired workload versions, etc.
 - Orchestration control plane scales and re-deploys deployments as needed



What to Scan: Image or Container Workloads?

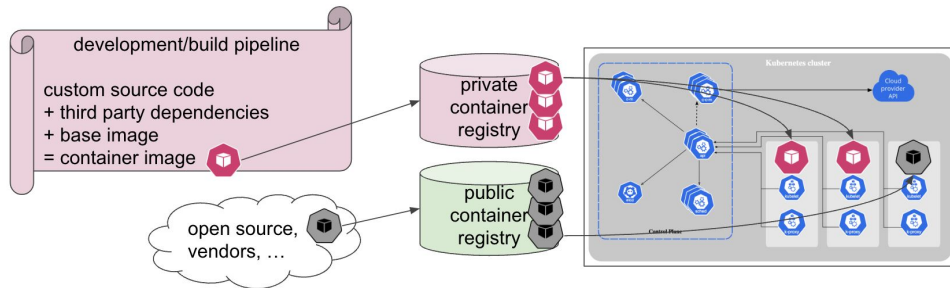
1. No difference in installed dependencies between images and workloads
 - But there might be dozens or 100s of instances of the same image deployed
 2. Workloads are often short-lived
 - By the time you ask a developer to fix a vulnerability, the workload you found it in is likely gone
 3. Containers are designed to not be accessible by humans
 - Can't point a scanner at them via SSH
 4. Containers are intended to be single-purpose
 - Running agents inside a workload would defeat the purpose
- 



Scanning images – where and when?

Implementation options

Which images to scan?



- Pipeline checks vs. registry vs. deployed inventory
 - Time of check vs. time of use
 - Scanning of code base is a good way to prevent vulnerabilities from being merged into main branch (block PRs via Branch Protection Rules)
 - Scanning at build time is helpful as an “early warning”, but doesn’t represent what’s currently deployed as workloads
 - Scanning (everything) that’s in the registry is ineffective, since it doesn’t likely match what’s deployed

✓  **license/snyk** [redacted] — No license issues in [redacted] tests

✓  **security/snyk** [redacted] — No manifest changes detected in [redacted] projects

Working with Images List

```
~ % kubectl get pods --all-namespaces \
-o jsonpath="{range .items[*]}{.metadata.namespace}\
{'\t'}{range .spec.containers[*]}{.image}\
|{.securityContext.privileged}{'\t'}{end}{'\n'}{end}"\
| awk '{for(i=2;i<=NF;i++) printf $i" "$1"\n"};' \
| sort | uniq > list_of_images.txt
```

- Getting a cluster inventory out of k8s
 - Requires far-reaching privileges
 - Is important, because multiple versions of the same image might be deployed at the same time
 - Scanners aren't always good at keeping track of versions in registries
 - Allows to scan *only* images that can put us at risk
- Third party images
 - Built-in registry may have a scanner that is less comprehensive
 - We use same tool for scanning of private and 3rd party images
 - Pull through cache for images can help!

Dealing with scan results

Now that we've scanned our images...

1. Understanding scanner capabilities
2. Triaging reported vulnerabilities
3. Identifying owners for mitigation
4. Asking owners to mitigate vulnerabilities
5. Validating the mitigation of vulnerabilities



Scanner Capabilities – are my scan results complete?

1. Does the scanner speak my language(s)?
 - a. Scanners frequently use binary decomposition to identify the dependencies that went into building a binary (e.g., module information included in Go binaries)
 - b. But if they don't do it well, it might lead to confusion
2. Is the scanner's database up-to-date?
 - a. If CVE databases do not get updated frequently, scanners might miss a critical zero-day vulnerability for too many days

Potential value add: Generation of Software Bill of Material (SBOM)

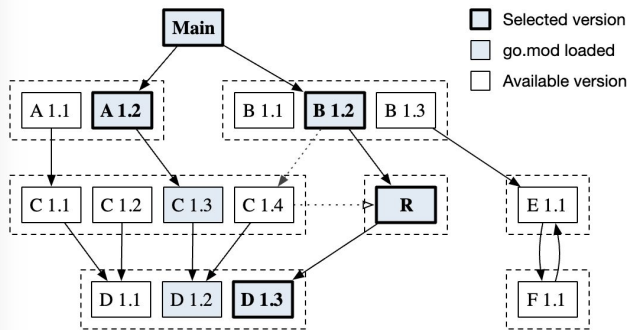
Replacement

<https://go.dev/ref/mod>

The content of a module (including its `go.mod` file) may be replaced using a `replace directive` in a main module's `go.mod` file or a workspace's `go.work` file. A `replace directive` may apply to a specific version of a module or to all versions of a module.

Replacements change the module graph, since a replacement module may have different dependencies than replaced versions.

Consider the example below, where C 1.4 has been replaced with R. R depends on D 1.3 instead of D 1.2, so MVS returns a build list containing A 1.2, B 1.2, C 1.4 (replaced with R), and D 1.3.



Triage – are my scan results applicable?

Exclude:

1. *False positives* – vulnerability actually not present

May want to exclude or reduce criticality (your mileage may vary):

2. *Vulnerability present* – but not in a dependency that's in the code path
3. *Vulnerability present* – but not exploitable
4. *Vulnerability present* – but mitigated in the environment

May want to reduce criticality:

5. *Vulnerability present* – but more difficult to exploit than rated by scanner (CVSS “environmental” scoring)

Examples:

- DoS caused by malformed SQL; but the SQL gets sanitized elsewhere already
- TLS handshake vulnerability can only affect a client; but the library is used to implement a server

Image Composition – who maintains what?

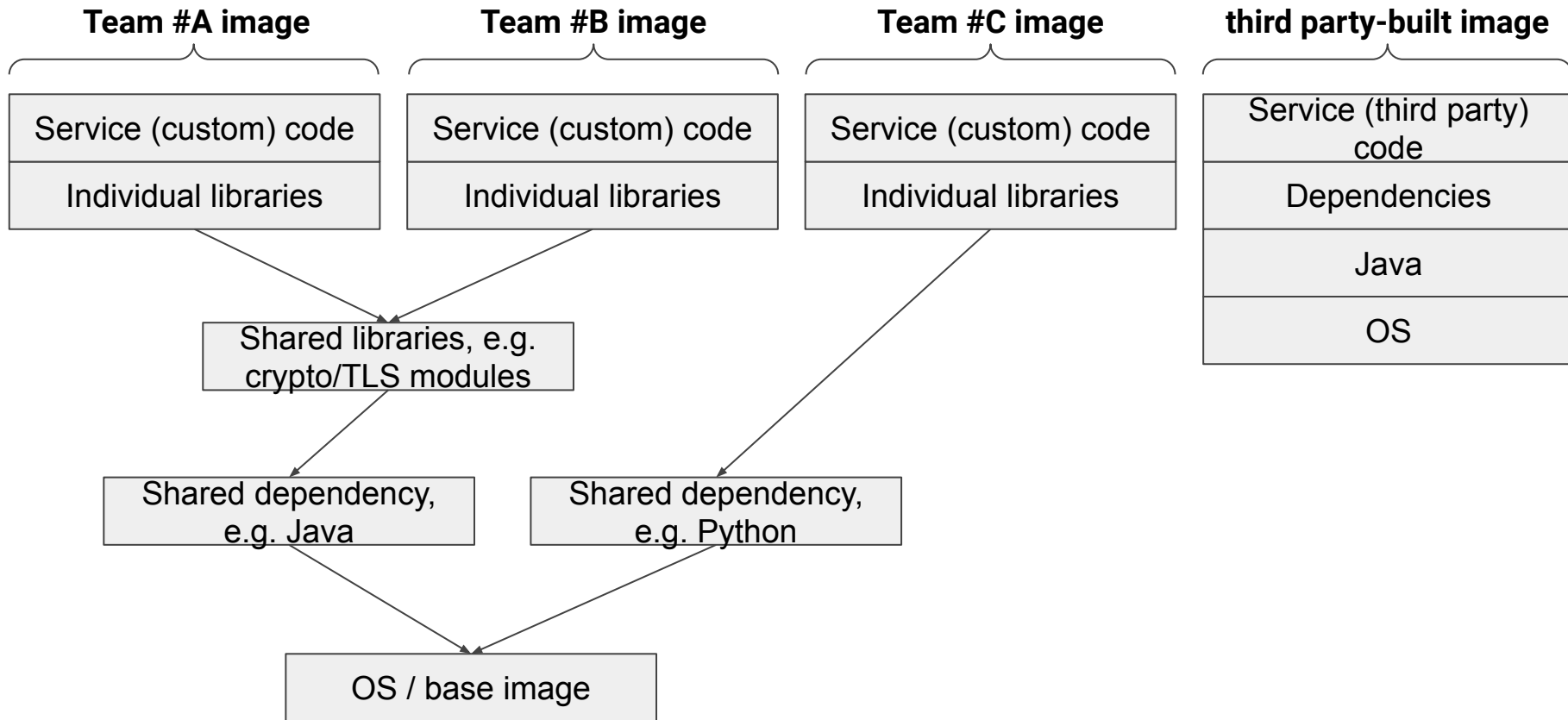


Image Composition – who maintains what?

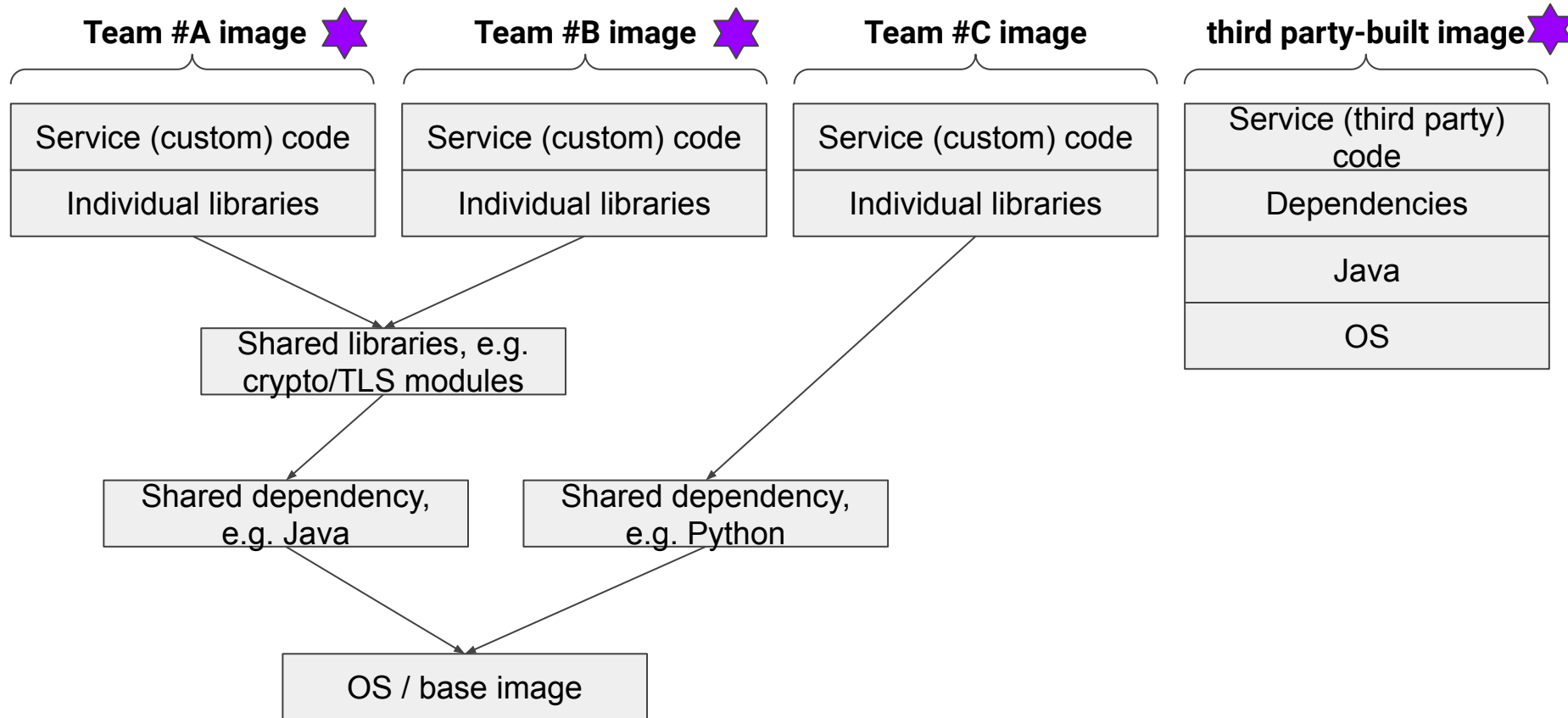
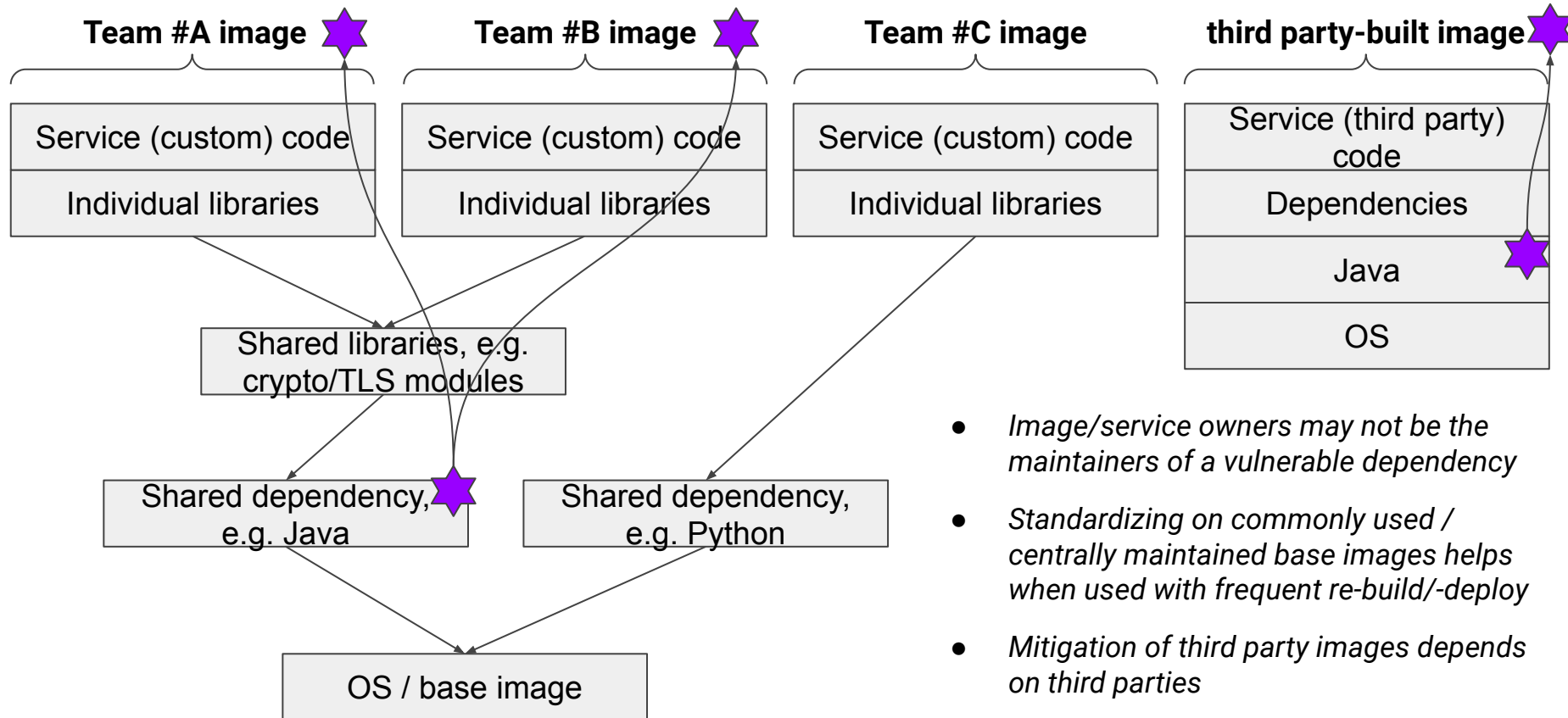


Image Composition – who maintains what?



- *Image/service owners may not be the maintainers of a vulnerable dependency*
- *Standardizing on commonly used / centrally maintained base images helps when used with frequent re-build/-deploy*
- *Mitigation of third party images depends on third parties*

Getting software engineers to mitigate vulnerabilities

1. Make the pipeline the enforcer?

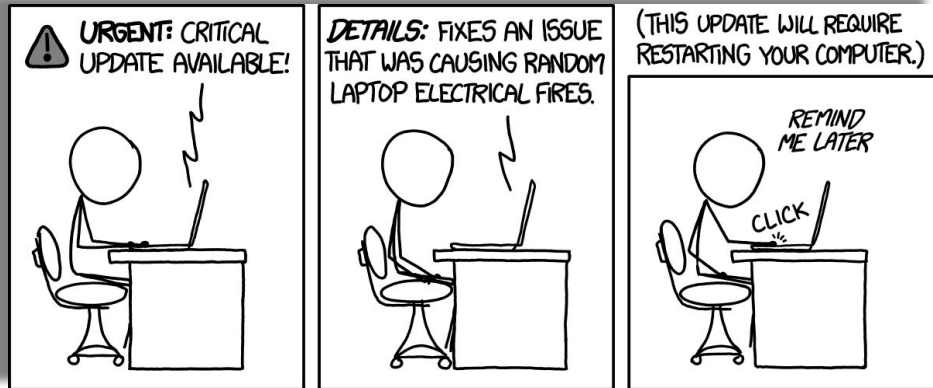
- Build checks – vulnerable images fail the build
 - i. If you don't practice continuous integration & deployment, this might miss recently published vulnerabilities
- Admission checks – vulnerable images will not be deployed
 - i. Risks availability issues, e.g. if workloads cannot be scaled

2. Create tickets... and make them actionable!

- *"Teams A, B, and C need to fix vulnerability X that showed up in all of their images"*

vs.

"Team Y needs to fix vulnerability X in the shared dependency, followed by Teams A, B, and C re-building their images with the updated dependency and re-deploying their images"



Scanner value-add: advisories

Provide upgrade paths as part of findings (some scanners help with this)

- Minimum library version to upgrade to, or “no patch available yet”
- Integration with repository scanners might enable automated PRs for library upgrades

Information Exposure

Affecting `org.graalvm.sdk:graal-sdk` package, versions `[20.3.10]` `[21.0.0,21.3.6]` `[22.0.0,22.3.2]`

INTRODUCED: 18 APR 2023

NEW

CVE-2023-21930 ?

CWE-924 ?

Share ▾

How to fix?

Upgrade `org.graalvm.sdk:graal-sdk` to version 20.3.10, 21.3.6, 22.3.2 or higher.

Validating fixes

How do you know a vulnerability has been mitigated?

1. Self-service option for developers to validate updated images before deployment
2. Back to the inventory of currently deployed image versions – are all vulnerable images gone?

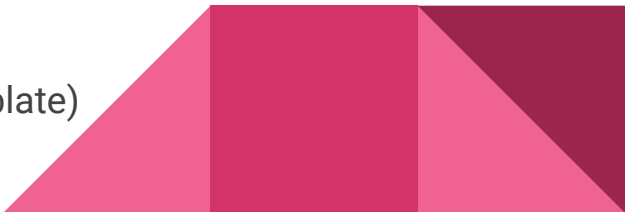


Compliance Reporting

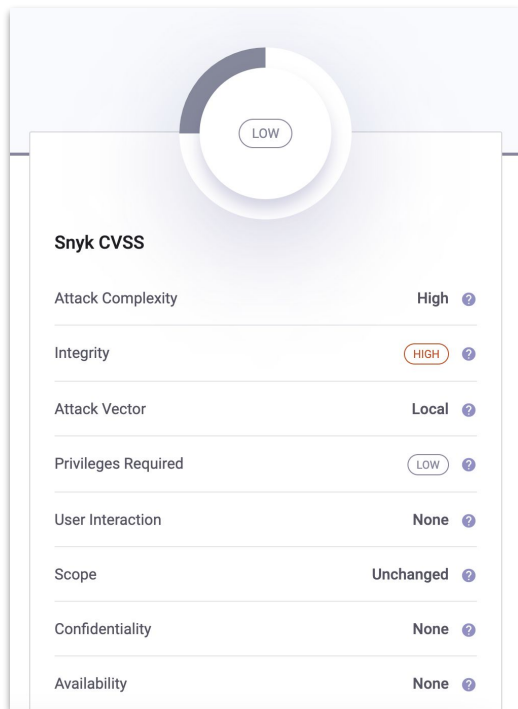
Reporting on vulnerabilities

Reporting on the scan results

- Criticality – your scanner's criticality rating may not match your compliance regime's
 - We use CVSS 3.x to make it universal across the company, and default to the National Vulnerability Database rating when available
 - Scanner may give lower rating due to high complexity, lack of POC/Mature Exploits available
- Reporting formats differ for different compliance regimes
 - Python CSV module or Pandas library to populate reports (e.g., FedRAMP Plan of Action and Milestones (POA&M) Template)



CVSS or custom score?



CVE-2013-4235 Detail

MODIFIED

This vulnerability has been modified since it was last analyzed by the NVD. It is awaiting reanalysis which may result in further changes to the information provided.

Description

shadow: TOCTOU (time-of-check time-of-use) race condition when copying and removing directory trees

Severity

CVSS Version 3.x

CVSS Version 2.0

CVSS 3.x Severity and Metrics:



NIST: NVD

Base Score: **4.7 MEDIUM**

Vector: CVSS:3.1/AV:L/AC:H/PR:L/UI:N/S:U/C:N/I:H/A:N

Some vulnerabilities may be harder to exploit, and scanners can lower score.
Not always useful for compliance!

Reporting on Mitigation Efforts

- Reporting per CVE vs. per container image vs. something else
 - Depends! But most of the time team maintains an image, so per image assignment is a good start
- Eg, FedRAMP Plan of Action and Milestones (POA&M) Template is tracking Jira tickets, which can be organized in many ways
 - Jira ticket per image: easier model for assignment of a ticket. Team that owns a microservice, owns the image
 - SRE/Infrastructure/DevOps team likely owns a base image
 - Occasionally there will be shared dependencies `_(ツ)_/`





Recap – Pitfalls and Opportunities

Pitfalls – Scanner Capabilities & Pipeline Integration

Check for support of your pipelines & infrastructure:

- Binary inspection capabilities for different formats
- Supported image formats (there are others than Docker)
- Public/private container registries
- Workload inventory capabilities
- Scan in the build pipeline? Based on workload inventories?



Pitfalls – Vulnerability management

You may want to be able to:

- Adjust vulnerability severities based on custom analysis
 - Or a specific source (e.g., National Vulnerability Database) rather than scanner-proprietary
 - And/or mark them as false positives
 - For one/multiple/all images
- Automatically identify and report on the resolution of vulnerabilities
- Correlate image scan results with your code base
- Report on Service Level Objectives for mitigation in a “compliant” manner
- Prepare software maintainers for what’s coming at them
 - Consider trial runs, start with a small scope (crawl, walk, run)



How to get started? (Examples, not a complete list!)

- Open Source tools, for example:
 - Gype (<https://github.com/anchore/gype/>)
 - Trivy (<https://github.com/aquasecurity/trivy>)
 - Clair (<https://github.com/quay/clair>)
- Free tiers of commercial solutions, for example:
 - Snyk (100 free scans/month)
 - JFrog (14 day trial)
- Scanners built into container registries, for example:
 - JFrog Xray
 - AWS ECR Basic and Enhanced scanning options
 - GCP On-Demand and Container Scanning APIs
 - Quay Automatic Security Scanning



Questions & Answers