The State of your (Container’s) Supply Chain

@sublimino & @MayaKaczorowski
I’m:
- Andy
- Dev-like
- Sec-ish
- Ops-y
What is a supply chain?

Anything that we depend upon

- e.g., the military need to know where all their hardware and software comes from and who builds them, to protect against state attacks
- e.g., pharmaceutical companies likewise need to know the provenance of their ingredients
What is a **software** supply chain?

Developer → CI/CD pipeline → Production environment

Any code that ends up running in production
Software supply chains can be exploited

- Vulnerabilities in dependencies, e.g., open-source packages
- Deliberate backdoors
- Compromised downloads, e.g., typosquatting
Software supply chains can be exploited

- Vulnerabilities in dependencies, e.g., open-source packages
- Deliberate backdoors
- Compromised downloads, e.g., typosquatting
What's different about supply chains with containers

- Debug
- Patch
- Update
- Restart
- Manual adjustment

Production environment

VM based

Hard
What's different about supply chains with containers

**VM based**
- Hard
  - Debug
  - Patch
  - Update
  - Restart
  - Manual adjustment

**Container based**
- Easy
  - Build & deploy
  - CI/CD pipeline
  - Manual adjustment
  - Manual adjustment

**Production environment**

- **VM**
- **Monolithic application**
- **VM**
- **VM**

- **Pod**
- **Microservice**
- **Pod**
- **Pod**
- **VM**
- **VM**
- **VM**
Stages of the CDLC (Container Delivery Lifecycle)

- **Base image**: Developer
- **Code**: Build, Test, Scan, Analysis, QA
- **Build**: CI/CD pipeline
- **Application image**: Pod, Microservice, Pod, VM
- **Deploy**: Production environment

Diagram showing the stages of the container delivery lifecycle from base image creation to deployment in a production environment.
“I find your lack of security disturbing.”
Base Image

- **Controlled base images**: official external images, copied into the organisation and promoted through dedicated pipelines
  - e.g. Docker Hub official images
- **Hash based addressing**: image has a verifiable “identity”
  - Hashes help ensure we have immutable images
  - Hashes are static - whereas tags are transitory and a possible risk
Code

- **Static analysis**: of code in-IDE (style, AST-analysis, atoms of confusion)
- **Dependency analysis**: Immediate and transitive (pom.xml, package.json, requirements.txt and pals)
Build

- **Hermetic builds**: Isolated build environment
  - No inter-build data or artefact leakage
- **Reproducible builds**: Repeatable build from source to binary
  - Build dependencies cached within an organisation's estate
  - Pinned versions for deterministic builds
  - Only helps security if you actually do reproduce it - not great for incremental builds
- The future: **rootless builds**: Build without privileged access
  - Tools like umoci, img, buildah, kaniko are moving towards a safer build environment
  - The class of build-time attacks this is mitigating against are aspirational rather than in-the-wild right now
Application Image scans

- **Vulnerability scanning:** CVE scans (operating system components, installed binaries/JARs/tarballs)
  - Patching
  - Removing packages
  - Smaller distribution

- **Configuration scanning:** Make it easy to do the right thing
  - Secrets in code
  - Images running as root
  - Misconfigurations

- **Policy:** filesystem configuration and Discretionary Access Controls, xattrs, SUID/GUID, runtimes and debug tools, etc.
Deploy

- **Admission control:** Gated admission to production based on policy, compliance, and other metadata from previous build stages
- **Runtime configurations:** Adherence to PodSecurityPolicy and Kubesec.io risk based on runtime configuration of the images that comprise a pod
Enforced Governance

Containers are short lived and frequently re-deployed, you can constantly be patching.

Containers are immutable, you can control what is deployed in your environment.
### Ideal, security-hardened container supply chain

<table>
<thead>
<tr>
<th>Base image</th>
<th>Code</th>
<th>Build</th>
<th>Application image</th>
<th>Deploy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Controlled base images</td>
<td>Static analysis</td>
<td>Hermetic</td>
<td>Vulnerability scanning</td>
<td>Admission control</td>
</tr>
<tr>
<td>Hash based addressing</td>
<td>Dependency analysis</td>
<td>Reproducible</td>
<td>Configuration scanning</td>
<td>Runtime configurations</td>
</tr>
</tbody>
</table>
State of the Ecosystem
## Open-source supply chain today

<table>
<thead>
<tr>
<th>Base image</th>
<th>Code</th>
<th>Build</th>
<th>Application image</th>
<th>Deploy</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Images:</strong> Docker Hub</td>
<td><strong>Updates:</strong> TUF, Notary</td>
<td><strong>Pipeline metadata:</strong> Grafeas, in-toto</td>
<td><strong>Vulnerability scanning:</strong> Clair, Micro Scanner, Anchore Open Source Engine</td>
<td><strong>Admission control:</strong> K8s admission controllers, Kritis, Portieris</td>
</tr>
</tbody>
</table>

- **Deploy**
  - Admission control: K8s admission controllers, Kritis, Portieris

- **Base image**
  - Images: Docker Hub

- **Code**
  - Updates: TUF, Notary

- **Build**
  - Pipeline metadata: Grafeas, in-toto

- **Application image**
  - Vulnerability scanning: Clair, Micro Scanner, Anchore Open Source Engine

- **Deploy**
  - Admission control: K8s admission controllers, Kritis, Portieris
Images
Docker Hub

- Offers hundreds of ‘official’ images, including base images
  - Alpine
  - Debian
  - Ubuntu

- Best practices
  - Pull latest
  - Don’t trust blindly: check when last patched, scan for vulnerabilities
Updates
TUF vs Notary

The Update Framework (TUF) is a secure distribution mechanism, for signing software package updates.

Notary is an implementation of TUF for container images specifically.

Both CNCF projects.
The Update Framework (TUF)

- Software package signing
- Secure key distribution mechanism
  - Update keys delegated by root key
  - Offline rotation
  - Temporal expiration
  - Resistant to replay attacks
Notary

- Implementation of TUF for image distribution
  - Server + database
  - Signer + database
- Signs and validates images
  - Signed collections
  - Key delegation
- Best practices
  - Store the master root key offline
  - Key rotation
Pipeline metadata
Why track pipeline metadata?

- Pipeline metadata is rich and varied
  - Initiating user(s) and/or events
  - Installed dependencies and their versions
  - Veracity test data, e.g., unit/integration/acceptance/&c tests
  - Security test data
  - Compliance and policy
- Data can be used for recording (audit) and reporting/enforcing (policy)
Grafeas

- Structured artifact metadata repository
  - Meant to be used as part of a container registry
- Spec includes multiple kinds of metadata
  - Package, Vulnerabilities, Discovery, Builds, Image
    basis, Deployment history, Attestation
- Can use multiple metadata providers
  - Providers include other scanning companies, e.g., JFrog, Red Hat, IBM, Black Duck, Twistlock, and Aqua
- You can use this metadata for enforcing restrictions on which containers get deployed
  - E.g., use “Admission” metadata with an admission controller to ensure compliance with your policies before deploying
Grafeas: concepts

- **Notes** are the definition of something that can be found or detected through analysis
- **Occurrences** are instances of a Note
- **Providers** are sources of metadata
- **Projects** are namespaces for metadata
- **Attestations** are cryptographic signatures
  - They aren't a separate object - but rather a metadata type part of Notes and Occurrences
in-toto

- Framework to provide whole software supply chain security
- Provides tooling and a metadata format to ensure all steps:
  - Are performed by the right party
  - Follow the expected policy
  - Use the right artefacts
  - Report the artefacts that were produced
- Just been submitted to the CNCF sandbox
in-toto: layouts

"_type": "layout",
"expires": "2018-11-30T12:44:15Z",
"keys": {
  "0c6c50": {...}
},
"signatures": {...}
"steps": [{
  "_type": "step",
  "name": "checkout-code",
  "expected_command": ["git", "clone", "..."]
  "expected_materials": [ ],
  "expected_products": [ ["CREATE", "demo-project/foo.py"], ... ]
  "pubkeys": ["0c6c50"],
  "threshold": 1
}, ... ]
"inspections": [...]

in-toto: execution parties and links

- Three types of parties
  - **Project owner**: defines a policy
  - **Functionary**: carries out a step and produces a statement as link metadata
  - **Verifier**: ensures all the link metadata matches the layout policy
- Links are cryptographically signed by the functionary

```json
"_type": "link",
"name": "build",
"byproducts": {"stderr": "", "stdout": ""},
"command": [...],
"materials": {...},
"products": {
  "foo": {"sha256": "..."}
},
"return_value": 0,
"signatures": [...]"
in-toto: verification

- Checks for compliance using Link metadata and the Layout metadata
- Verification can be done in many steps:
  - Continuously (e.g. polling the Docker API endpoint)
  - Upon installation (e.g. hooking the package manager)
  - Before deployment (e.g. a Kubernetes admission controller)
- in-toto doesn’t care what you’re verifying
  - It’s just verifying a chain of signatures
  - With a little change-management tooling integration, it could help automate bureaucratic releases processes
in-toto: call for contributions!

- [https://github.com/in-toto:](https://github.com/in-toto:)
  - in-toto: Python reference implementation
  - docs: specification
  - ITE: governance documentation
  - in-toto-java: Java implementation
  - in-toto-webhook: admission controller
  - kubectl-in-toto: kubectl plugin
  - in-toto-golang: Go implementation

- [https://github.com/jenkinsci](https://github.com/jenkinsci):
  - in-toto-jenkins-plugin: jenkins plugin for artifact provenance
Grafeas vs in-toto

Grafeas
- Strict opinionated API schema - "on rails"
- Supported by Google
- Limited documentation

in-toto
- Adaptable to your environment, supports unstructured data
- Can chain together attestations to assert the integrity of a whole supply chain
- Can use different storage backends

Integration between Grafeas & in-toto proposed
Vulnerability scanning
Image vulnerability scanning approaches

- Components to scan: package-level vs. code-level
  - OS packages
  - App library packages
  - JARs, WARs, TARs, etc.
  - Malware
  - Misconfigurations, e.g., secrets

- Scan type
  - Layer-by-layer
  - UnionFS top layer only

https://sysdig.com/blog/container-security-docker-image-scanning/
Clair vs. MicroScanner vs. Anchore

<table>
<thead>
<tr>
<th>Scanning depth</th>
<th>OS covered</th>
<th>Maintainer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Packages</td>
<td>Alpine, CentOS, Debian, Oracle Linux, RHEL, Ubuntu</td>
<td>Anchore</td>
</tr>
<tr>
<td>Packages</td>
<td>Aqua Security</td>
<td>CoreOS</td>
</tr>
<tr>
<td>Packages, files, software artifacts</td>
<td>Anchore</td>
<td>Anchore</td>
</tr>
</tbody>
</table>
Admission control
Kubernetes admission controllers

- Admission controllers are a concept built into Kubernetes
  - **Mutating**: can modify objects
  - **Validating**: can’t modify objects
- Can customize for whatever you want to check
Kritis

- Signing and deploy enforcement tool for Kubernetes
  - Implemented as a Kubernetes admission controller
  - Integrates with Grafeas attestation metadata APIs
- Generate attestations based on your requirements
  - Build provenance
  - Vulnerability findings
apiVersion: kritis.grafeas.io/v1beta1
kind: ImageSecurityPolicy
metadata:
    name: my-isp
spec:
    imageWhitelist:
        - gcr.io/kritis-int-test/nginx-digest-whitelist:latest
        - gcr.io/kritis-int-test/nginx-digest-whitelist\@sha256:56e0af16f4a9d2401d3f55bc8d214d519f070b5317512c87568603f315a8be72
    packageVulnerabilityRequirements:
        maximumSeverity: HIGH  # BLOCKALL|LOW|MEDIUM|HIGH|CRITICAL
        whitelistCVEs:
            - providers/goog-vulnz/notes/CVE-2017-1000082
            - providers/goog-vulnz/notes/CVE-2017-1000081
Portieris

- Notary Admission Controller
- Portieris enforces Content Trust
  - Different levels of trust for different images
- A mutating admission webhook ensures Kubernetes pulls the signed version
- Enforces trust pinning, and blocks the creation of resources that use untrusted images
- **Supports** IBM Cloud Container Registry, Quay.io, Docker Hub
## Ideal, security-hardened container supply chain

<table>
<thead>
<tr>
<th>Base image</th>
<th>Code</th>
<th>Build</th>
<th>Application image</th>
<th>Deploy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Controlled base images</td>
<td>Static analysis</td>
<td>Hermetic</td>
<td>Vulnerability scanning</td>
<td>Admission control</td>
</tr>
<tr>
<td>Hash based addressing</td>
<td>Dependency analysis</td>
<td>Reproducible</td>
<td>Configuration scanning</td>
<td>Runtime configurations</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Rootless</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Open-source supply chain today

### Base image
**Images:** Docker Hub

### Code
**Updates:** TUF, Notary

### Build
**Pipeline metadata:** Grafeas, in-toto

### Application image
**Vulnerability scanning:** Clair, Micro Scanner, Anchore Open Source Engine

### Deploy
**Admission control:** K8s admission controllers, Kritis, Portieris
Summary

- TUF protects automatic software updates from attack and makes them more resilient to compromise.
- Notary uses TUF to secure Docker image distribution from registries, guaranteeing correctness and freshness.
- Grafeas holds information about your systems' compliance and risk state.
- in-toto secures build steps with cryptographic metadata, enhancing accountability of individuals involved in the chain, and the ordering of their actions.
- Clair, Micro Scanner, Anchore, and friends are essential to prevent known CVEs getting pushed to production.
- Protect your clusters with admission control: Kritis, in-toto, and Kubescr
With thanks to

- @MayaKaczorowski
- @torresariass
- https://control-plane.io