SpringOne

Building A Secure Software Supply Chain MVP

September 1–2, 2021

springone.io

Safe Harbor Statement

The following is intended to outline the general direction of VMware's offerings. It is intended for information purposes only and may not be incorporated into any contract. Any information regarding pre-release of VMware offerings, future updates or other planned modifications is subject to ongoing evaluation by VMware and is subject to change. This information is provided without warranty or any kind, express or implied, and is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions regarding VMware's offerings. These purchasing decisions should only be based on features currently available. The development, release, and timing of any features or functionality described for VMware's offerings in this presentation remain at the sole discretion of Pivotal. Pivotal has no obligation to update forward looking information in this presentation.

Introductions

Scott Hiland

Solution Lead Manager - Platform Services

VMware Tanzu Labs

Alex Barbato

Solutions Engineer - Federal

VMware Tanzu

Rough Agenda

- What is a Secure Software Supply chain?
- How might I evaluate if I’m doing it “right”?
- Let’s put hands on keyboards!

What is it?

Secure Software Supply Chain is a set of practices that enable organizations to adjust the way they deliver software packages – both first- and third-party – from source code to production at a sustained high speed and quality relative to their accepted risk level.

It provides confidence that the code and its dependencies are:

- trustworthy, compliant, up-to-date and release-ready
- as well as ensuring regular scans are in place to detect, report and eliminate new vulnerabilities.
- With a defined set of policies enforced consistently across all systems in the chain, it prevents unauthorized access and prohibits unsigned packages to run.

tl;dr

What does that really mean?

- Verify your source code
- Verify your dependencies
- Protect your builds
- Protect your artifacts and deployments

...and do it all in a repeatable, reproducible way. (Automate!)

Source: CNCF - Evaluating Your Supply Chain Security

Learning to walk

Let’s MVP a Secure Software Supply Chain together.

Let’s:

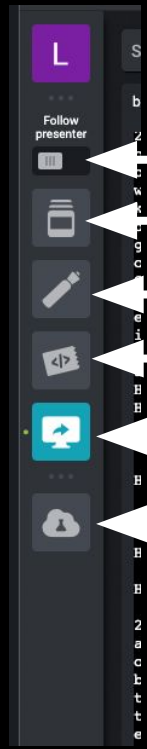
- Validate a commit
  - ...and fail an unverified commit
- Generate a Software Bill of Materials (SBOM)
  - ...to identify and visualize risks in our code and its dependencies
- Build our application
  - ...with a secure base image and components
- Deploy our artifacts
  - ...and check for where our image came from

Before we get started!

There are 7 exercises, following this structure

- Introduction of exercises and context
- Do the work on your own
  - Use the ask for help button and/or unmute if you need help and we will hurry on over.
  - Raise your hand please when you're done working and reading.
- Recap

Strigo Logistics



Panes (Terminal, Dependency Track, etc.)

Quick follow along toggle

Slides

Instructor whiteboard

Code snippets

Instructor lab / screenshare

Your lab!

The items in this green will not be used much or at all during the presentation.

SpringOne

Workshop Configuration: Exercise 1

Let’s dive in!

Let’s get a few minor tasks out of the way to ensure your environment is setup.

Commit validation (Code I Write): Exercise 2

Control(s):

- Do you require signed commits?

What this solves for:

- Knowing that new code comes from a known and verified source.

Generate SBOM (Code I Use): Exercise 3

Control(s):

- Do you create an SBOM of your own artifacts?

What this solves for:

- Knowing that what's in your build and dependencies comes from a known and verified source.

Check for vulnerabilities (Code I Use (cont.)): Exercise 4

Control(s):

- Do you verify that dependencies meet your minimum thresholds for quality and reliability?
- Do you automatically scan dependencies for security issues and license compliance?
- Do you automatically perform Software Composition Analysis on dependencies when they are downloaded/installed?
- Do you monitor dependencies for updates and security issues?

What this solves for:

- Reduces the likelihood of allowing user-impacting security, licensing, and reliability issues to enter your application
- Helps developers know to inspect new dependencies more thoroughly

Secure base image and components (Build the Image): Exercise 5

Control(s):

- Do you use hardened, minimal containers as the foundation for your build workers?
- Do you maintain your build and test pipelines as Infrastructure-as-Code?
- Do you automate every step in your build pipeline outside of code reviews and final sign-offs?
- Do you network isolate your build workers and pipeline as much as possible?

What this solves for:

- Reduces attack surface
- Improves reliability and resilience through repeatability
- Systematically reduces manual toil

Runtime checks (Run the Image): Exercise 6

Control(s):

- Is every artifact your produce (including metadata and intermediate artifacts) signed?
- Can your downstream consumers verify/validate any artifact they ingest from you before they use/deploy it?

What this solves for:

- Automates the security validation process for promoting artifacts into environments

Cleanup: Exercise 7

In the spirit of a secure supply chain, you don't really know us or where these VM's came from... so please clear from GitHub your:

- GPG Keys
- Access or SSH keys generated

This is an MVP!

Securing a Software Supply Chain is a journey, not a destination so:

- Iterate on it as though it were any other software product
- Prioritize the highest impact, lowest effort things first
- Report *then* enforce
- Version control *everything*
- Automate as you move beyond MVP
  - Do and document what you did
  - Script
  - Config Abstraction/Full automation