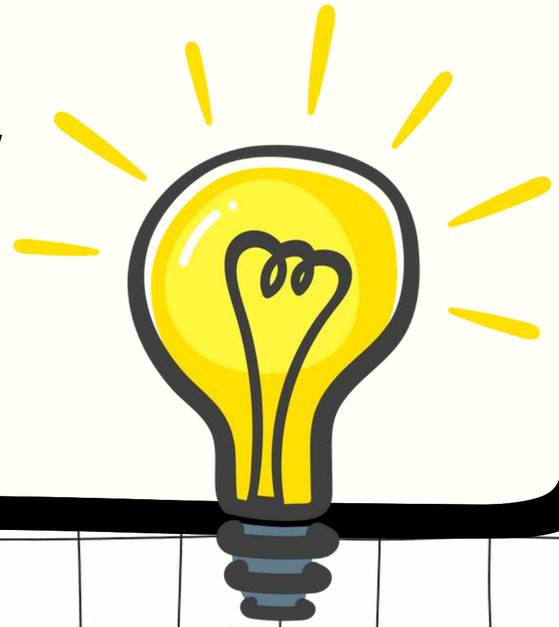
AISA

Enhancing Open Source Security

with Secure Supply Chain Consumption Framework (S2C2F)

Enhancing Open Source Security

with Secure Supply Chain Consumption Framework (S2C2F)



Contents

Speakers

Kaif Ahsan

Atlassian

Kumar Soorya

AWS

Disclaimer #1

The content is our personal research and does not represent the employers.

Disclaimer #2

Operating under tight time conditions so certain elements will be high level.

They serve as references so attendees can dive deeper later.

Let’s understand the problem.

Rise of Supply Chain Threats

Dependency Confusion: How I Hacked Into Apple, Microsoft and Dozens of Other Companies

The Story of a Novel Supply Chain Attack

Alex Birsan · Follow

11 min read · Feb 10, 2021

19K 52

LILY HAY NEWMAN SECURITY DEC 18, 2021 2:54 PM

‘The Internet Is on Fire’

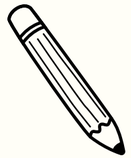
A vulnerability in the Log4j logging framework has security teams scrambling to put in a fix.

Why?

Increased Dependency

More and more software are depending on each other.
Ever increasing complexity.

Low Visibility Into The Supply Chain



It is hard for organisations to get a good visibility of their supply chain.

Project Longetivity

OSS contributors often have little or no incentives to maintain their projects.

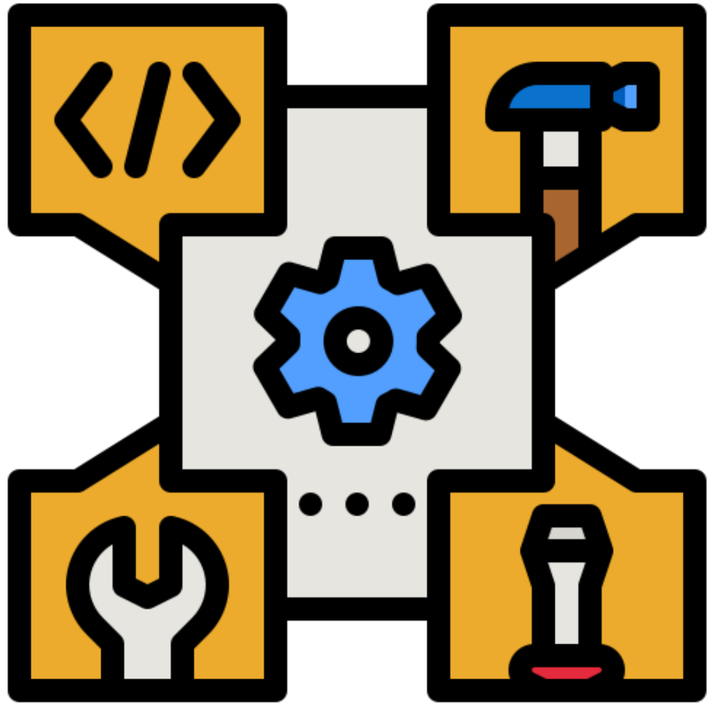
OSS can be a threat but essential for speed and productivity.

Systematic Approach To Securely Consume OSS

Secure Supply Chain Consumption Framework (S2C2F)

What Is S2C2F?

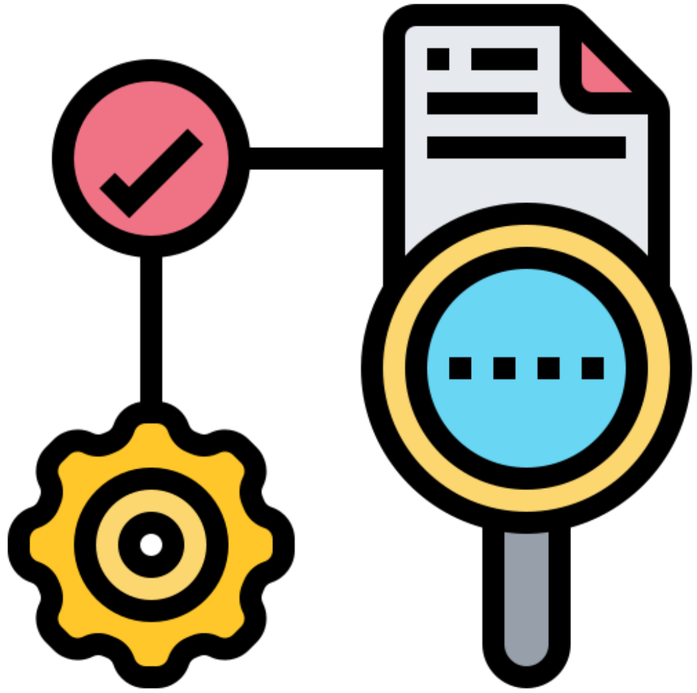
- A framework built by Microsoft and adopted by OpenSSF.
- Guides how to consume OSS dependencies securely.
- Is a collection of practices and controls.
- Provides a roadmap for gradual maturity.



Why S2C2F?

Why S2C2F?

- End-to-end process.
- Technology agnostic.
- Provides teams with a roadmap to follow.
- Lists a wide range of controls with implementation suggestions.



8 Key Practices

Ingest

Scan

Inventory

Update

Audit

Enforce

Rebuild

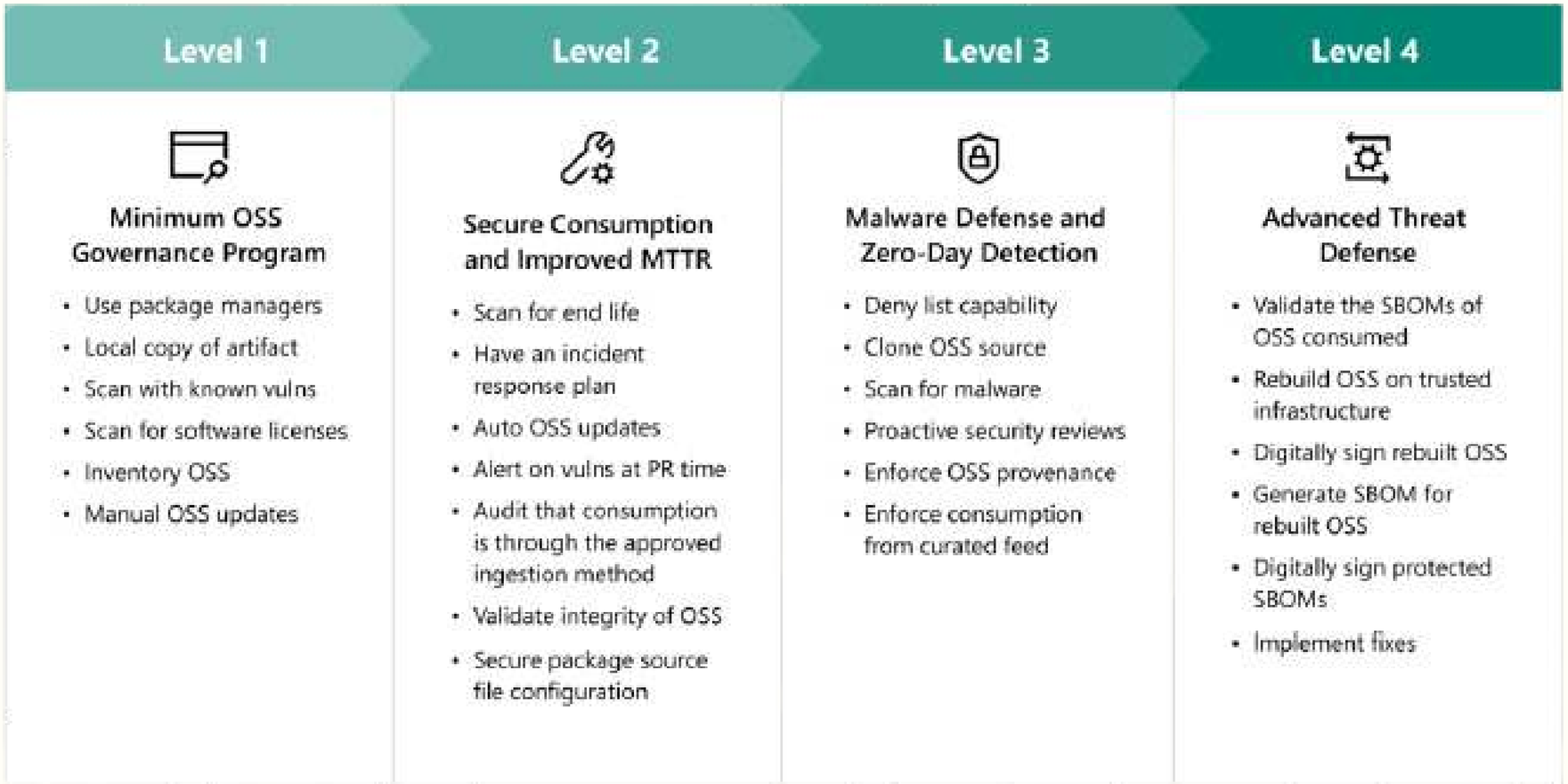
Fix

Comprehensive List of Threats

OSS Supply Chain Threat	Real Example	Mitigation via S2C2F Requirement
Accidental vulnerabilities in OSS code or Containers that we inherit	SaltStack	UPD-2 UPD-3
Intentional vulnerabilities/backdoors added to an OSS code base	colors v1.4.1	SCA-5
A malicious actor compromises a distribution mirror of a package	phpMyAdmin	AUD-3
A malicious actor compromises a known good OSS component and adds malicious code into the repo	ESLint incident	ING-3 ENF-2 SCA-4
A malicious actor creates a malicious package that is similar in name to a popular OSS component to trick developers into downloading it	Typosquatting	AUD-1 ENF-2 SCA-4

Example of list of threats and mitigation.

Maturity Levels



Example of maturity roadmap provided.

Detailed Controls

Secure Supply Chain Consumption Framework Requirements

Below is a table of the requirements mapped to the 8 different practices. Two of the requirements have prerequisites identified that are outside the scope of this document to list as requirements.

Practice	Requirement ID	Maturity Level	Requirement Title	Benefit
Ingest it	ING-1	L1	Use package managers trusted by your organization	Your organization benefits from the inherent security provided by the package manager
	ING-2	L1	Use an OSS binary repository manager solution	Caches a local copy of the OSS artifact and protects against left-pad incidents, enabling developers to continue to build even if upstream resources are unavailable
	ING-3	L3	Have a Deny List capability to block known malicious OSS from being consumed	Prevents ingestion of known malware by blocking ingestion as soon as a critically vulnerable OSS component is identified, such as colors v 1.4.1, or if an OSS component is deemed malicious
	ING-4	L3	Mirror a copy of all OSS source code to an internal	Business Continuity and Disaster Recovery (BCDR)

Example of ingestion controls.

Controls By Levels

Practice name	L1	L2	L3	L4
Ingest it – save a local copy of artifacts and source code	**[ING-1]** Use package managers trusted by your organization **[ING-2]** Saving a local copy of the OSS artifact can be done by adopting an integrated package caching solution into your CI/CD infrastructure. All developers across your organization should standardize their consumption methods (using governed workflows) so that security policy can be enforced. **Free Tools:** VCPKG for C/C++ OSS, Pulp **Paid Tools:** Artifacts, GitHub Packages, Azure		**[ING-3]** Having a Deny List capability to block ingestion of vulnerable and malicious OSS components is a required defensive tool in incident response situations. Having an incident response team that can rapidly respond and update the deny list is also critical. **Paid Tool:** Nexus Firewall **[ING-4]** Saving a local copy of the OSS source code **Free Tool:** Duplicating a repo	

Example of control by level and resources provided.

Shortcomings of S2C2F

- **Early** in the journey. Some rough edges need to be smoothed out.
- **Diminishing returns** of the higher levels.
- **Developer productivity decrease** caused by implementing the controls needs to be carefully considered.

8 Key Practices

Ingest

Scan

Inventory

Update

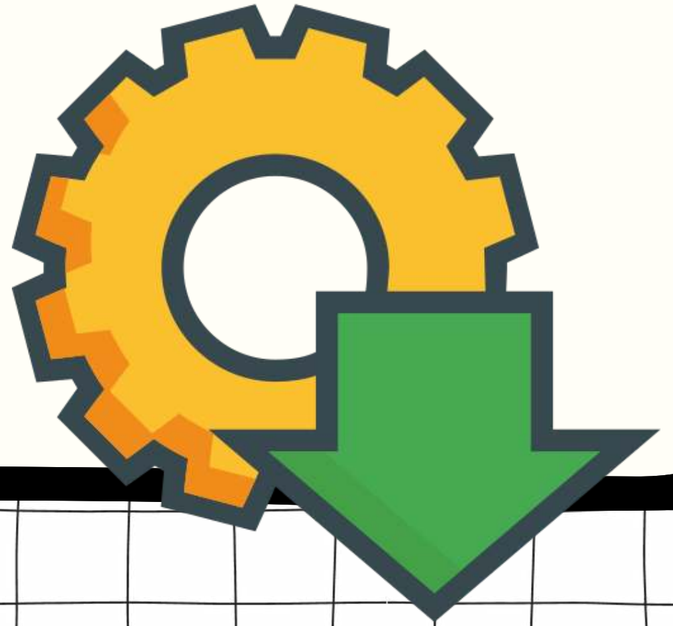
Audit

Enforce

Rebuild

Fix

Ingest



Ingest

Problem

Very little control over the source, resulting in very **hard to prevent** malicious dependencies from entering the supply chain.

Approach

Establish more **control** over **how developers consume** OSS dependencies and prevent malicious packages from being used.

Ingest - Controls

Level 1

- Using package managers.
- Using OSS binary repository managers.

Level 3

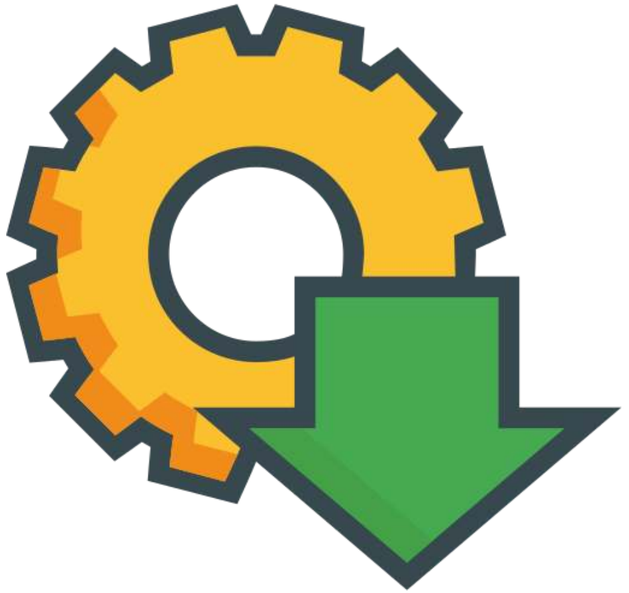
- Having a deny list capability.
- Mirroring a copy of all OSS dependencies internally.

Example

Ingest - Example

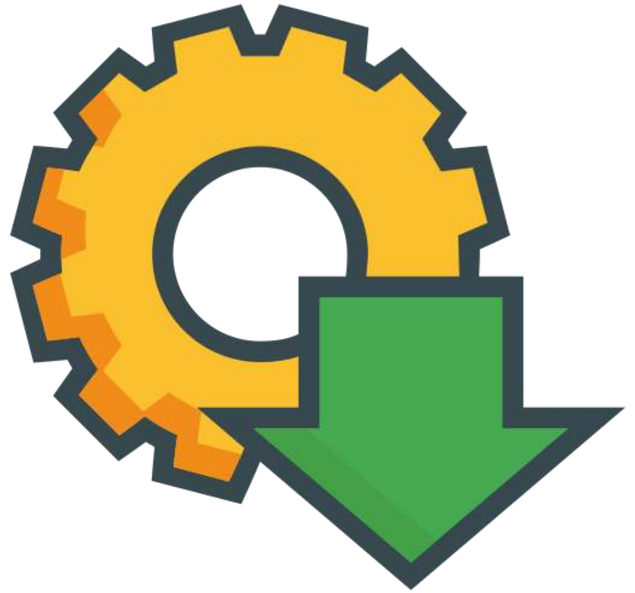
- Preventing the build if malicious dependency exists in the software.

- Leverage existing free-to use Synk GitHub Actions as sample tools.

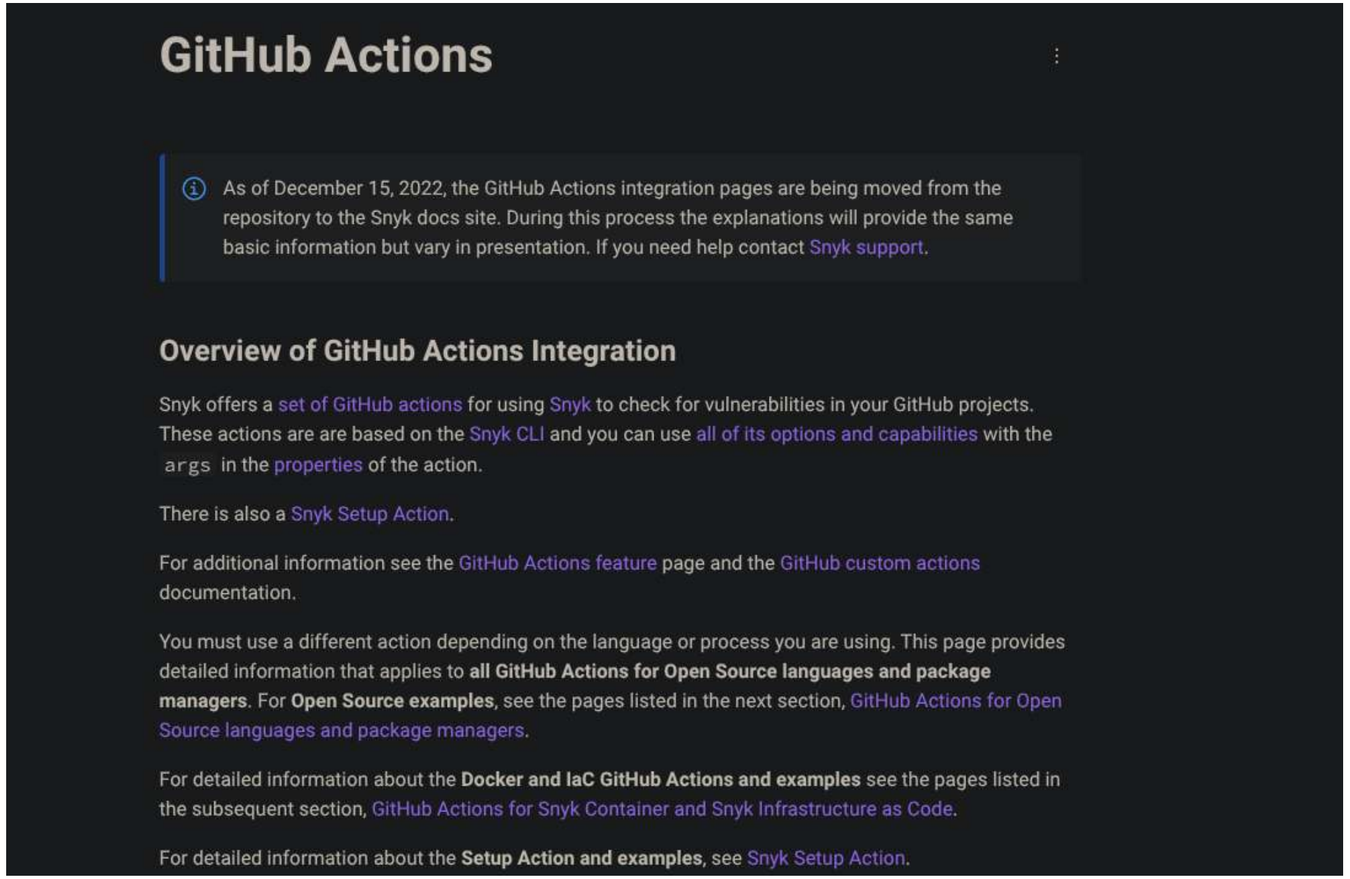


Ingest - Example

- Set up **GitHub Action** to run Snyk scan in every commit push.
- Fail build if critical or high-severity vulnerabilities exist in any of the packages.
- Notify of failure.



Leveraging Pre-built GitHub Actions from Snyk



GitHub Actions

As of December 15, 2022, the GitHub Actions integration pages are being moved from the repository to the Snyk docs site. During this process the explanations will provide the same basic information but vary in presentation. If you need help contact Snyk support.

Overview of GitHub Actions Integration

Snyk offers a set of GitHub actions for using Snyk to check for vulnerabilities in your GitHub projects. These actions are are based on the Snyk CLI and you can use all of its options and capabilities with the `args` in the properties of the action.

There is also a Snyk Setup Action.

For additional information see the GitHub Actions feature page and the GitHub custom actions documentation.

You must use a different action depending on the language or process you are using. This page provides detailed information that applies to **all GitHub Actions for Open Source languages and package managers**. For **Open Source examples**, see the pages listed in the next section, GitHub Actions for Open Source languages and package managers.

For detailed information about the **Docker and IaC GitHub Actions and examples** see the pages listed in the subsequent section, GitHub Actions for Snyk Container and Snyk Infrastructure as Code.

For detailed information about the **Setup Action and examples**, see Snyk Setup Action.

Example Vulnerable Package

```
<!-- Dependency with a known critical vulnerability -->
<dependency>
    <groupId>commons-beanutils</groupId>
    <artifactId>commons-beanutils</artifactId>
    <version>1.9.2</version>
</dependency>

<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
</dependency>
```

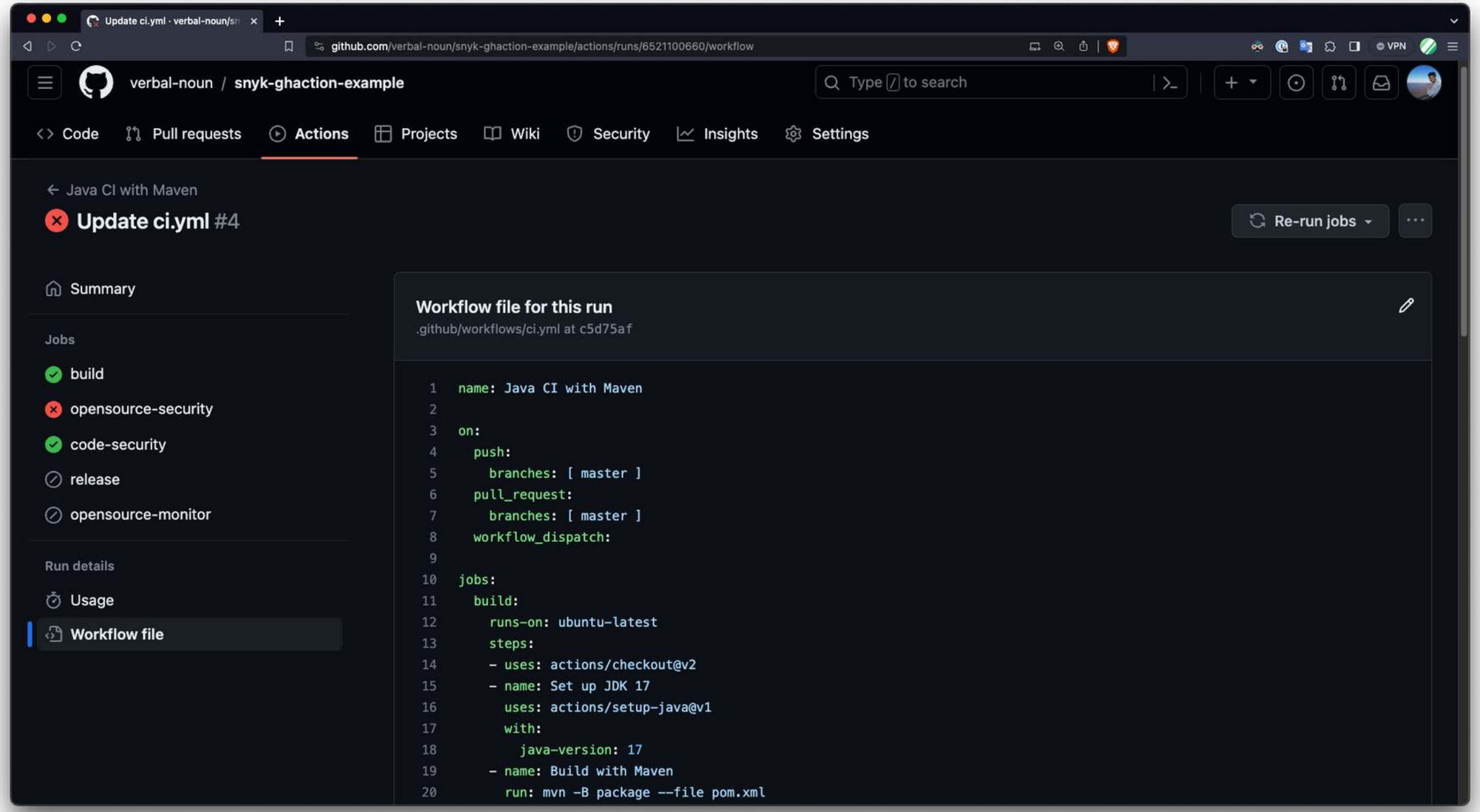
Example vulnerable package in pom.xml file

Setting up Actions Workflow

```
22    opensource-security:
23      runs-on: ubuntu-latest
24      steps:
25        - uses: actions/checkout@master
26        - name: Run Snyk to check for vulnerabilities
27          uses: snyk/actions/maven@master
28          env:
29            SNYK_TOKEN: ${{ secrets.SNYK_TOKEN }}
30          with:
31            args: --severity-threshold=high
```

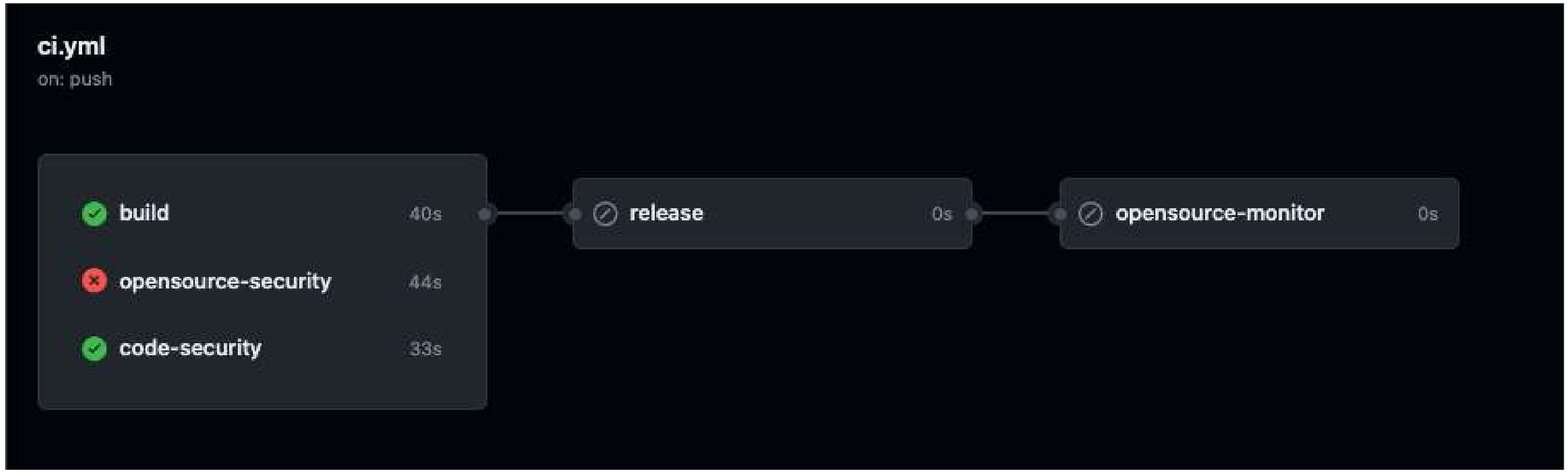
Example step created in GitHub Actions

Zooming out - Entire Workflow



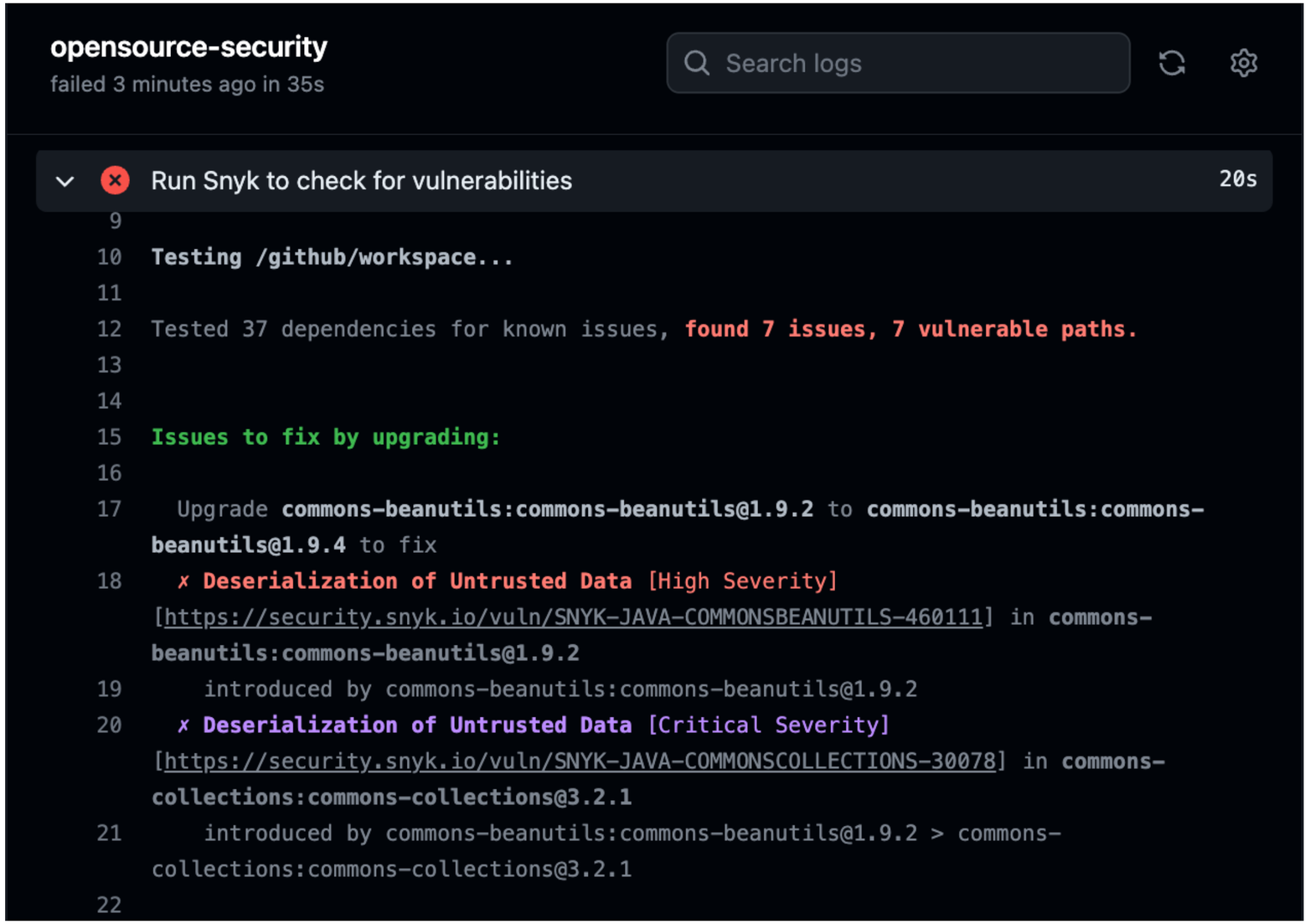
Visual purposes only - no need to read

Let’s see it in action



Visual flow of the Action running

Capturing issues in packages



Scan

Scan

Problem

New vulnerabilities and issues are being **discovered continuously** in the OSS. It is hard to identify new risks without proactive scans.

Approach

Establish scanning capability for a wide variety of issues depending on the company's priorities.

Scan - Controls

Level 1

- Scanning OSS for known vulnerabilities.
- Scanning OSS for licences.

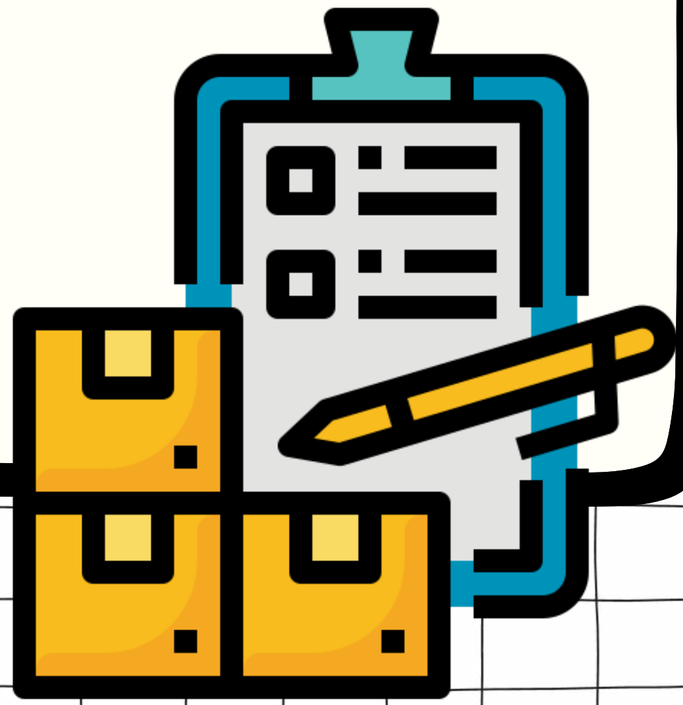
Level 2

- Scanning for End-of-Life

Level 3

- Scanning OSS for malware.
- Security review of OSS dependencies.

Inventory



Inventory

Problem

It is hard to protect against threats if we do **not know** which dependencies we **use** directly and indirectly.

Approach

Producing an **inventory** of dependencies and also **being prepared** to tackle issues when they arise based on inventory.

Inventory - Controls

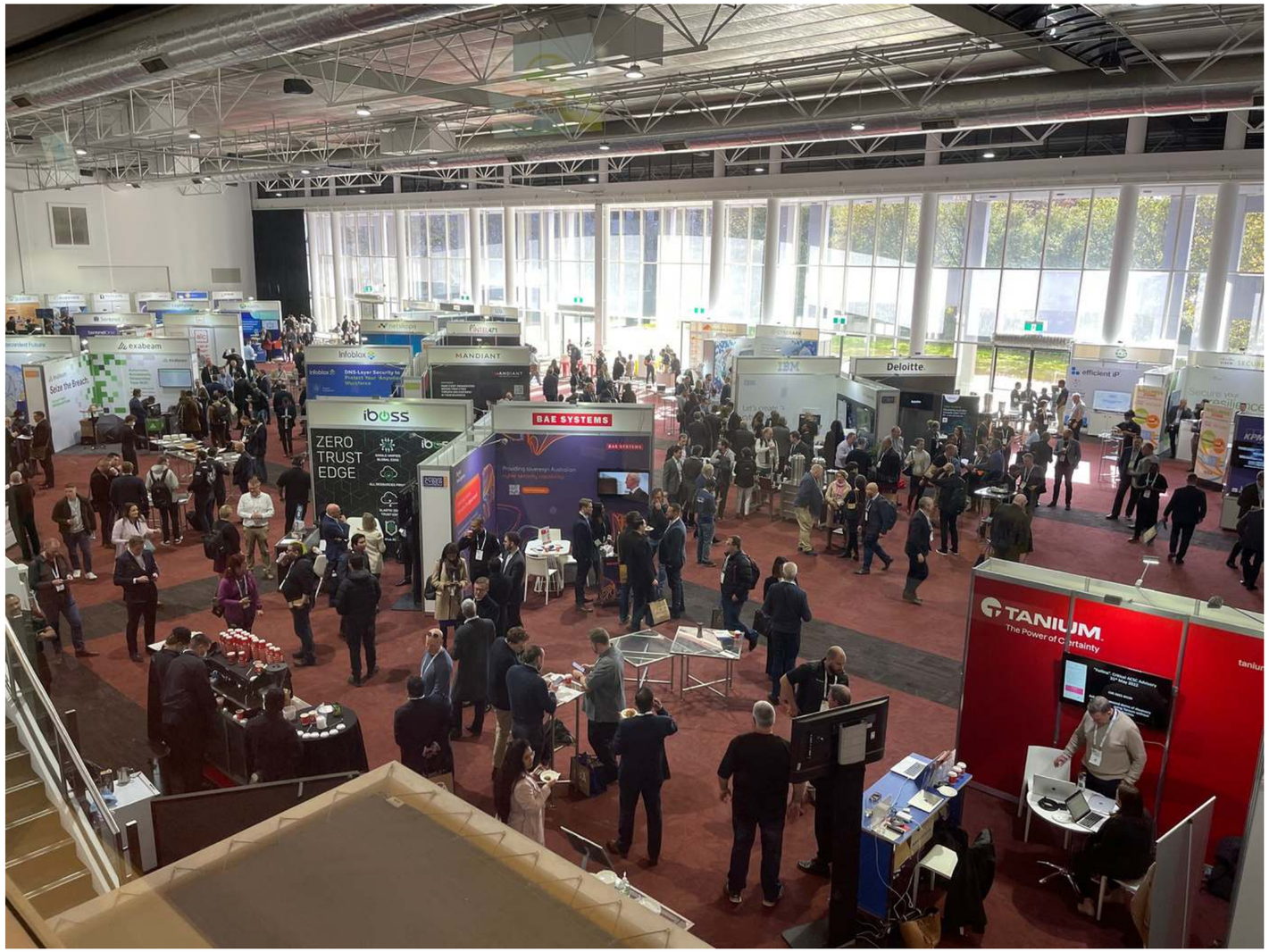
Level 1

- Maintaining an automated inventory of all OSS used / Generating SBOM.

Level 2

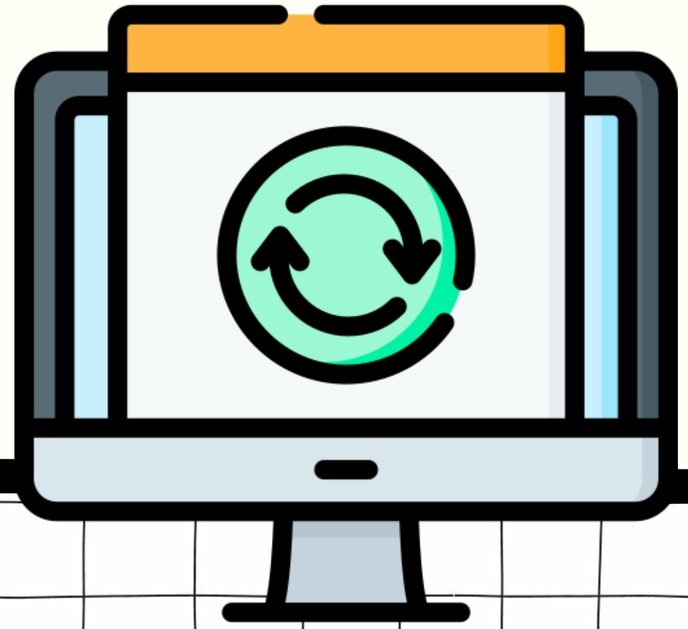
- Having an OSS incident response plan.

Example



Just go to the vendor area, and you will get all the SBOM pitches you want. 😂

Update



Update

Problem

When a new vulnerability is found, **attackers move very quickly**. On the other hand, developers are often **not motivated** to prioritise fixing library issues.

Approach

Reduce the **cognitive load** on developers and **time** to update through **automation** and **tooling**.

Update - Controls

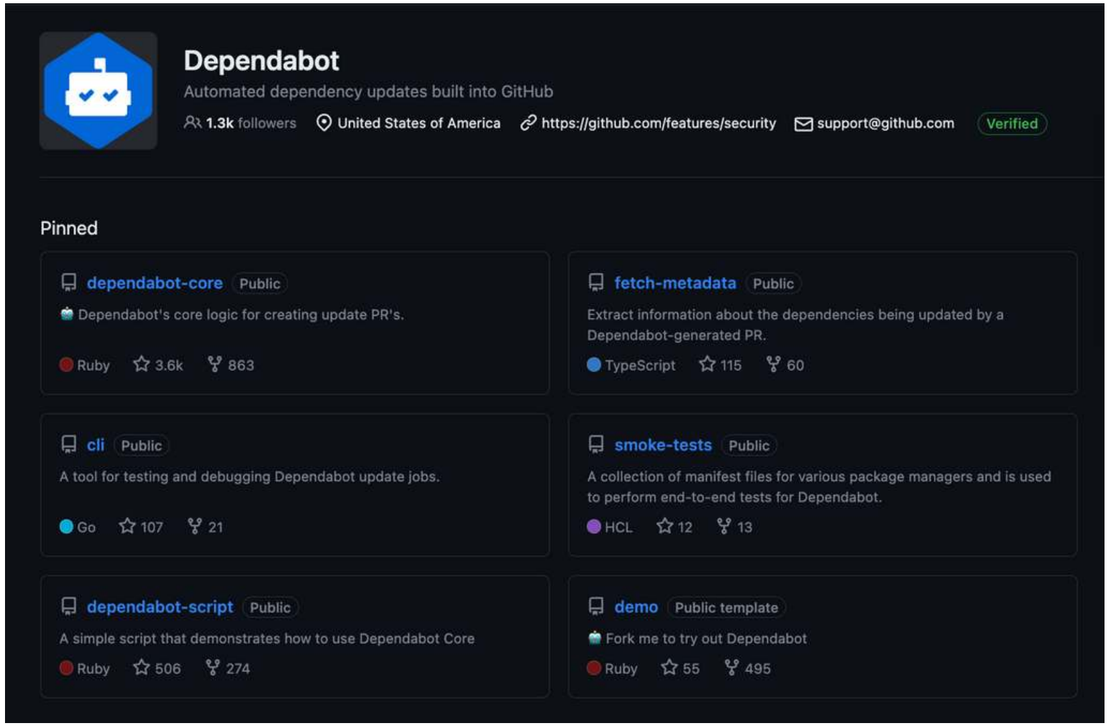
Level 1

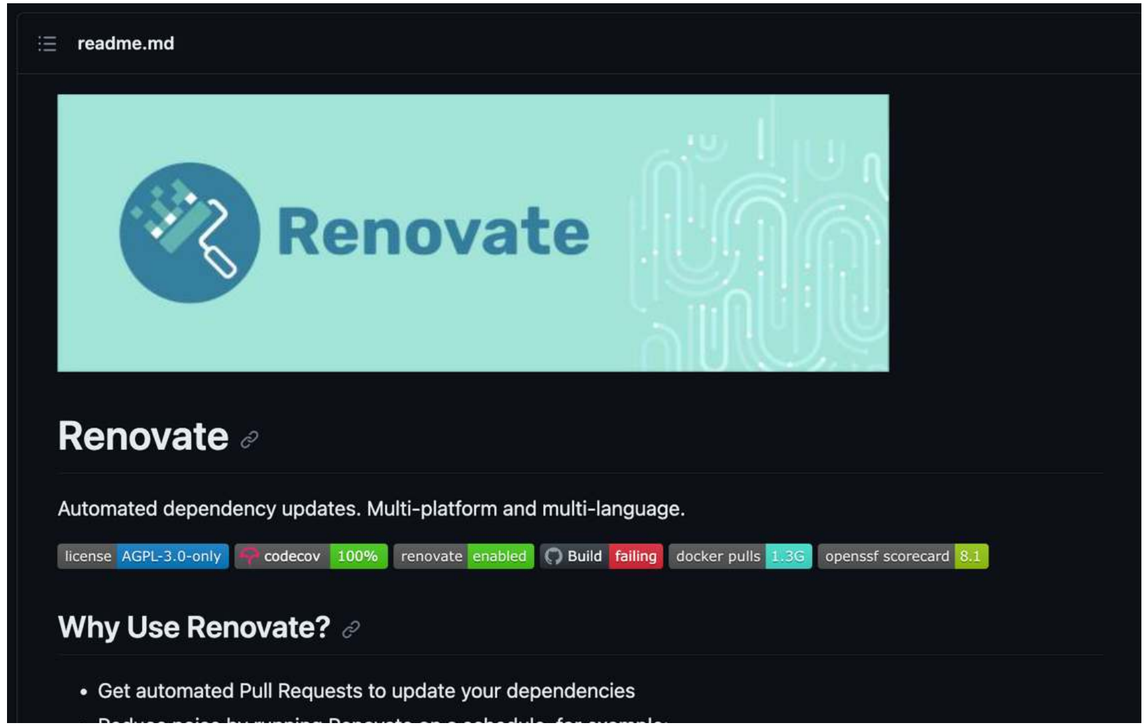
- Update vulnerable OSS manually.

Level 2

- Enable automated OSS updates.
- Display OSS vulnerabilities as comments in Pull Requests (PRs)

Update - Examples





Audit

Audit

Problem

Often, **protective controls are/can be bypassed** by attackers and well-meaning but misguided developers.

Approach

Establishing the **ability to audit OSS consumption** to see if it's coming through the standardised consumption tools or not.

Audit - Controls

Level 2

- Detecting when OSS is consumed using a non-standard medium.
- Validating integrity.

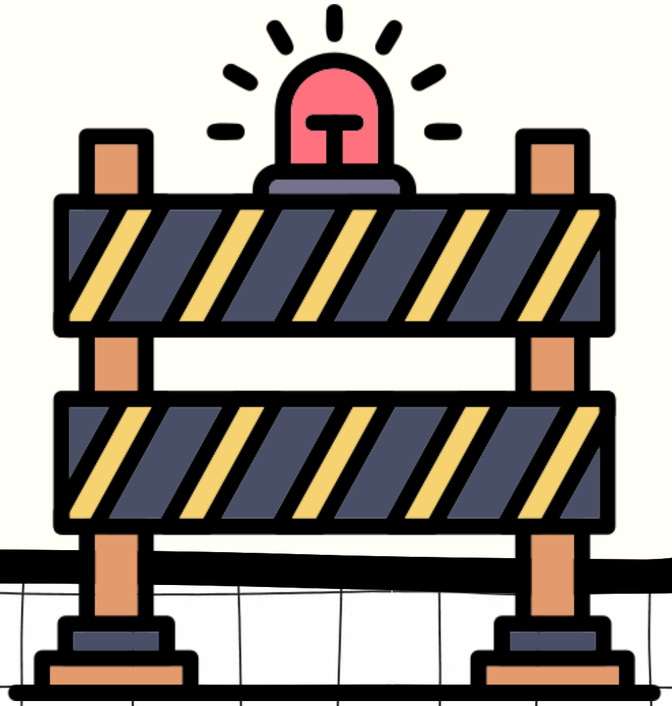
Level 3

- Verifying the provenance of your OSS.

Level 4

- Validating SBOM for provance data and its digital signature for integrity.

Enforce



Enforce

Problem

Often, **protective controls are/can be bypassed** by attackers and well-meaning but misguided developers.

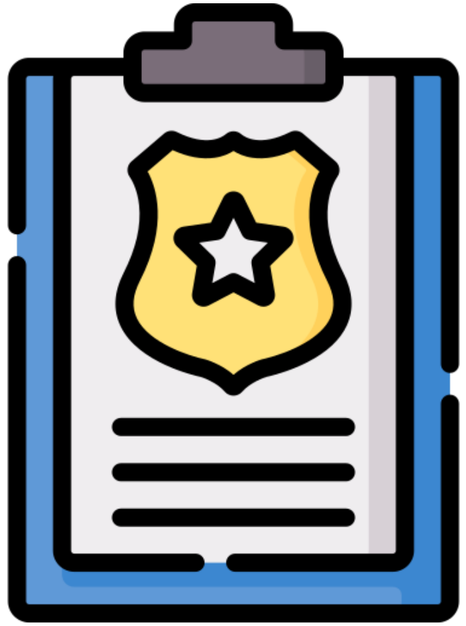
Approach

Enforcement of secure channels to be used only.

Enforce - Controls

Level 2

- Securely configuring package source files.
  - Package source mapping.
  - Single upstream feed.
  - Version pinning and lock files.



Level 3

- Enforce usage of a curated OSS feed.

Audit + Enforce Operating Together For Increased Protection.

Parting Thoughts

- It’s not one size fits all.
- We must identify our organisation’s risks and prioritise controls accordingly.
- Contributing back to OSS and community.

Everything Cyber

Everything Cyber

@Everything_Cyber 244 subscribers 26 videos

Welcome to Everything Cyber. Our channel is dedicated to up-skilling ourse...

linkedin.com/company/everything-cyber

Subscribed

HOME VIDEOS PLAYLISTS COMMUNITY CHANNELS ABOUT

Videos ▶ Play all

Ep. 16 - Cybersecurity In Healthcare w/ Alana Tobgui

32 views • 8 days ago

Ep. 15 - Building Zero Trust Network Infrastructure w/...

80 views • 1 month ago

Ep. 14 - Data-driven security w/ Jeevan Singh

88 views • 1 month ago

Ep. 13 - Leadership In Application Security w/...

330 views • 1 month ago

Ep. 12 - Level-up your Threat Intelligence game with the...

225 views • 4 months ago

Ep. 11 - Leading with Impact: Strategies for Effective...

92 views • 5 months ago

CC

Thank you

Please feel free to reach out.

Our Socials

Kaif Ahsan

Kumar Soorya

Slides & More

AISA

THANK YOU

Slides and contact info: