NDC { Oslo }

Securing your .NET application software supply-chain

Niels Tanis

Who am I?

- Niels Tanis
  - Security Researcher @ Veracode
  - Background in .NET Development
  - Application Security Consultancy
  - Pen-testing & Ethical Hacking
  - ISC² CSSLP

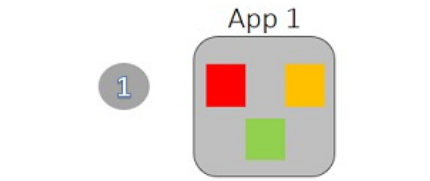
Securing your .NET application software supply-chain

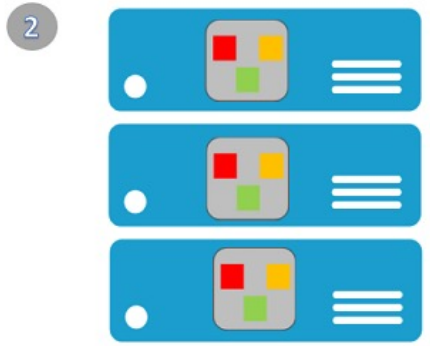
Agenda

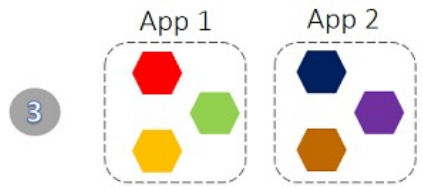
- Definition Software Supply-Chain
- Securing the Software Supply-Chain
  - Developer & Source
  - 3rd Party Libraries
  - Build & Release
- Conclusion and Q&A

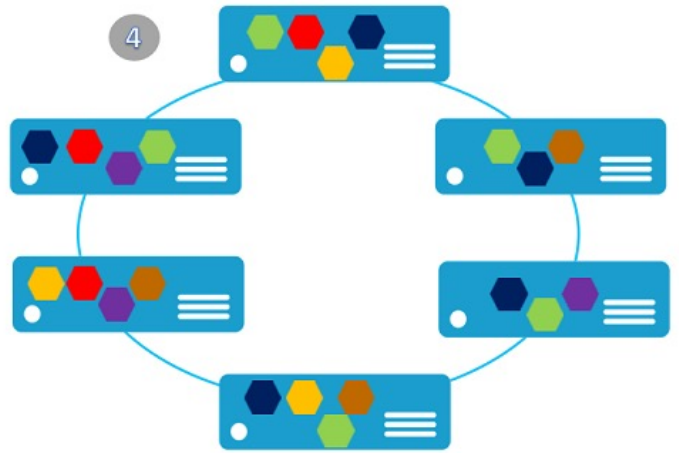
Evolution in Software Architecture

- Monolith
- Microservices
- Serverless
  Cloud-Native



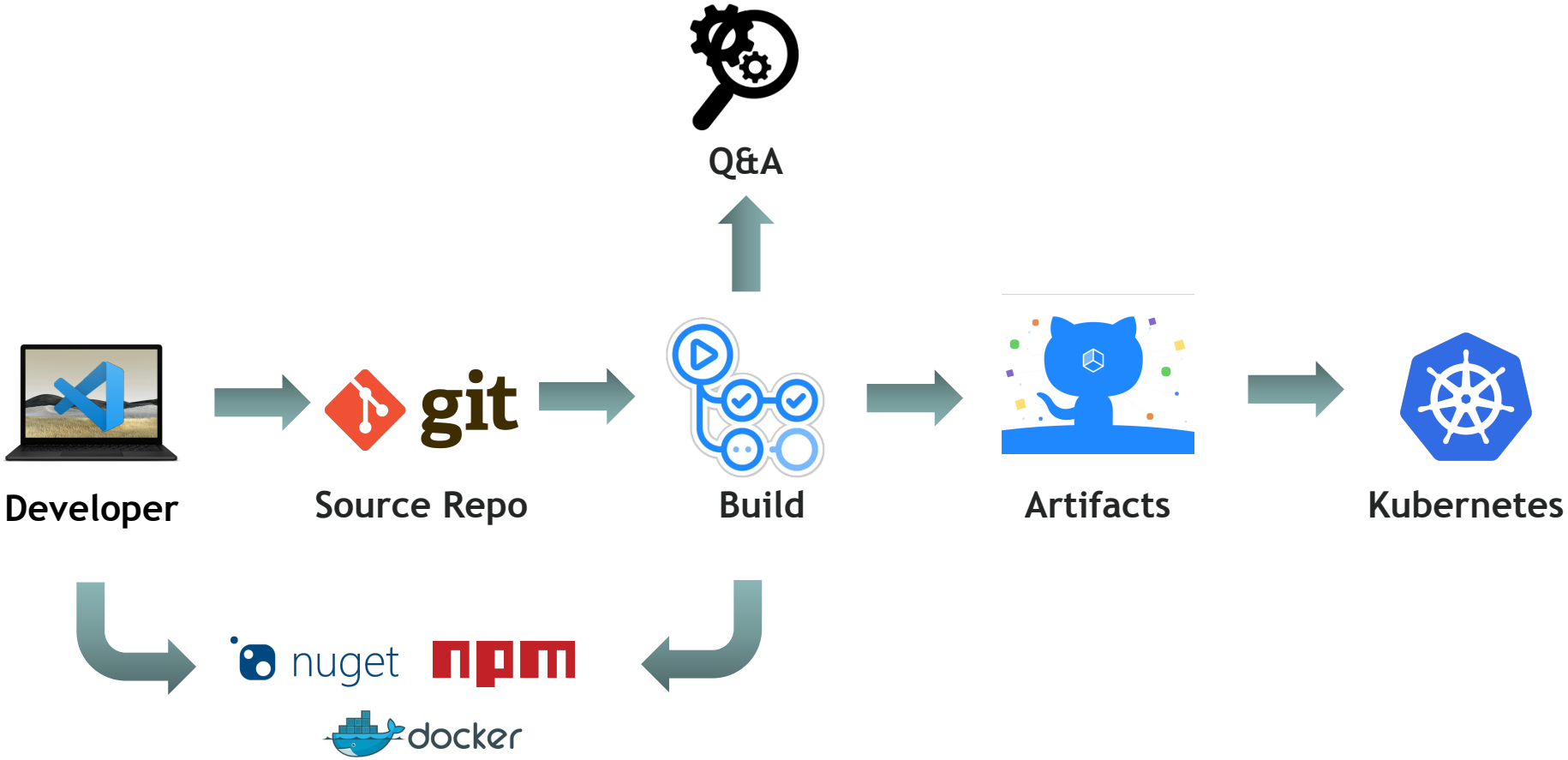




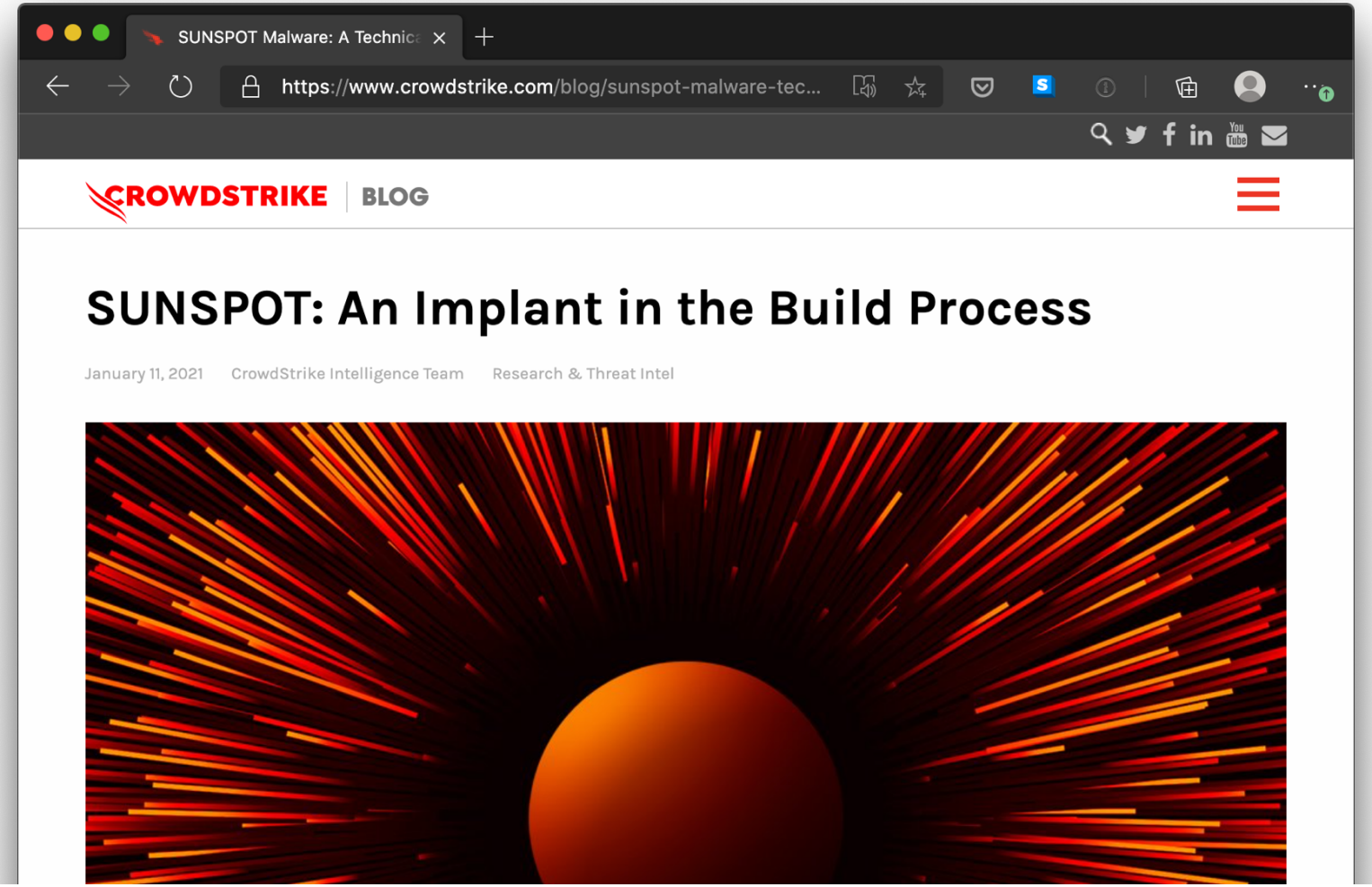


What is a Supply Chain?

Software Supply Chain



SolarWinds SunSpot

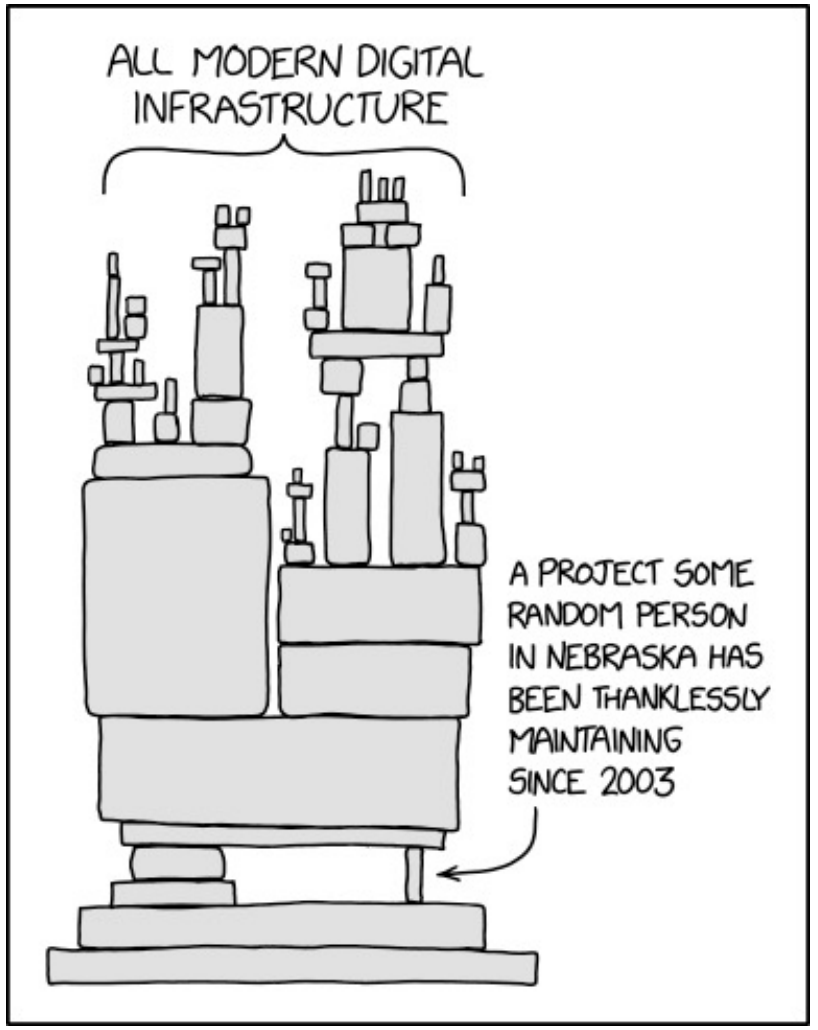


Kaseya

Kaseya Ransomware: a Software Supply Chain Attack or Not?

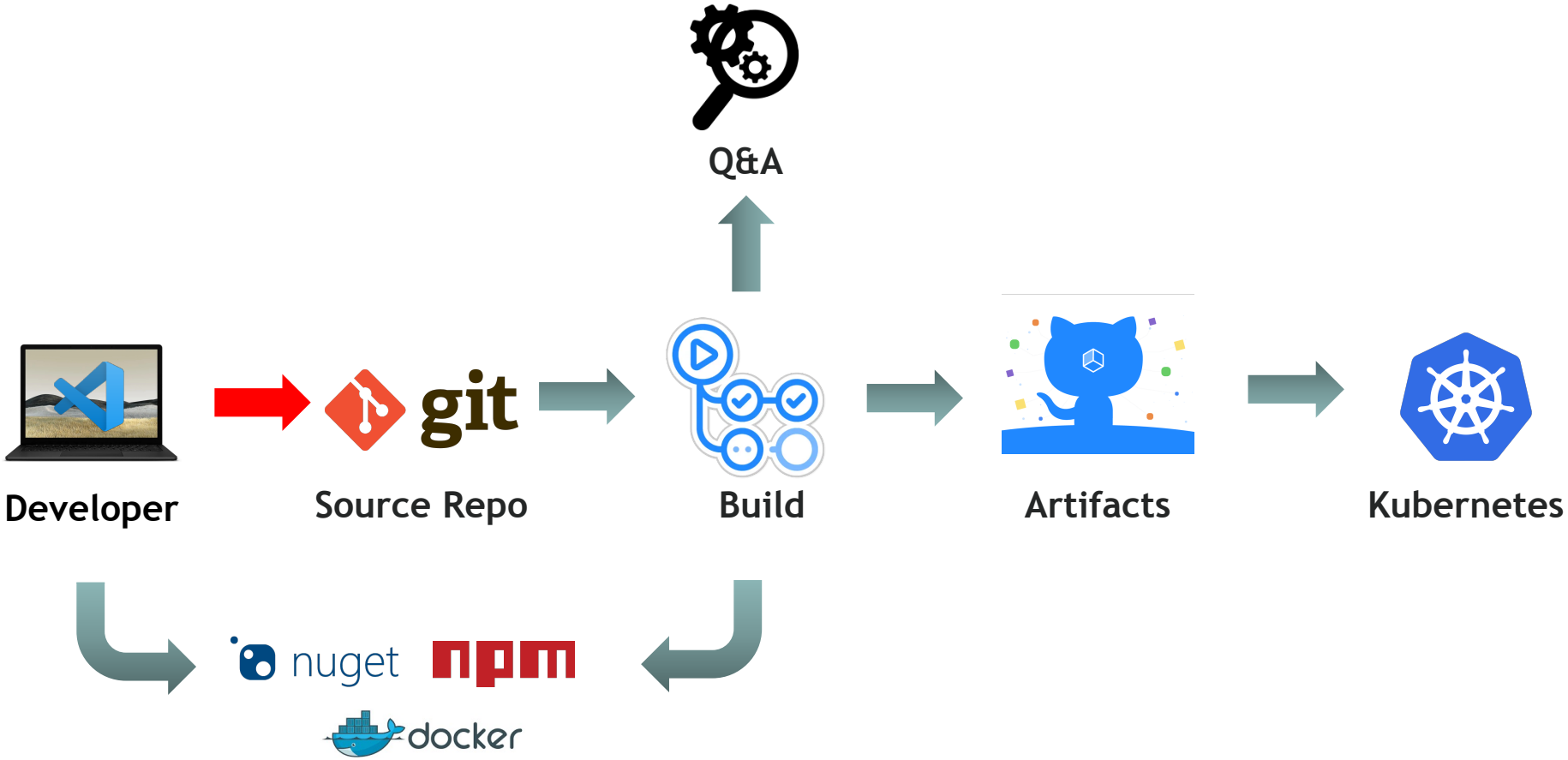
July 06, 2021 By **Matt Howard**

XKDC - Dependency



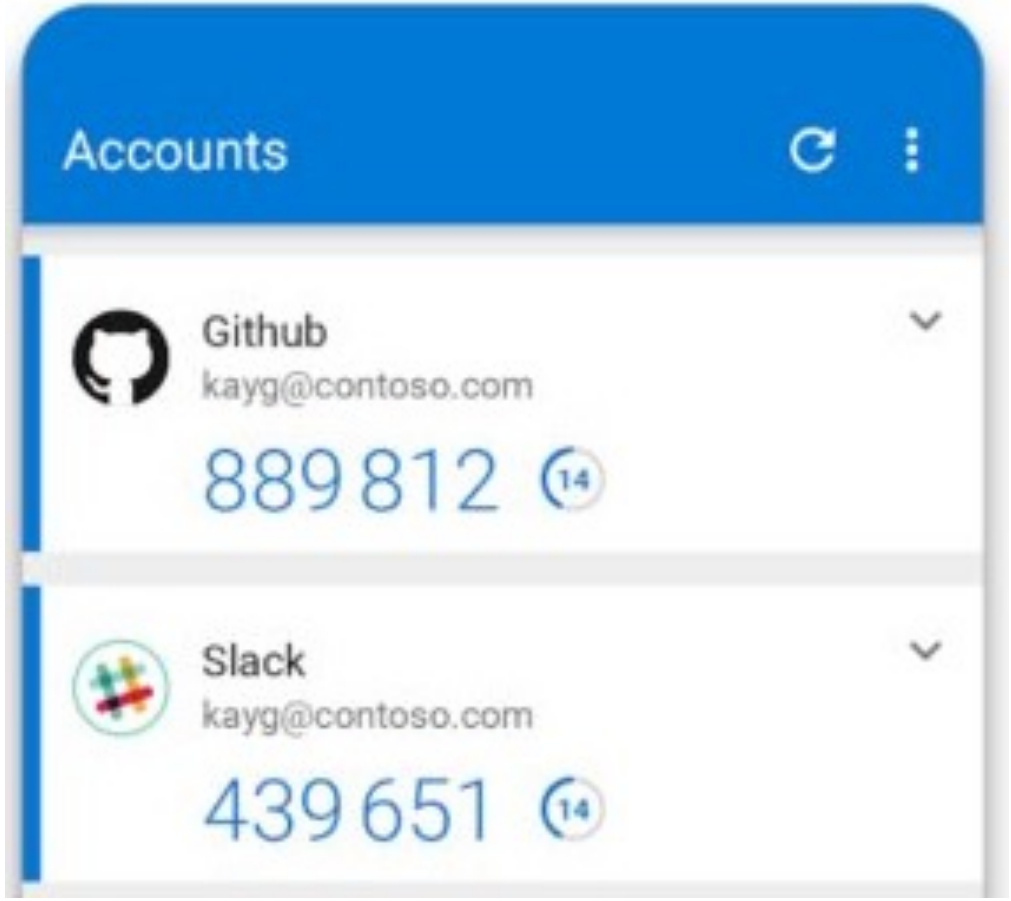
https://xkcd.com/2347/

Software Supply Chain

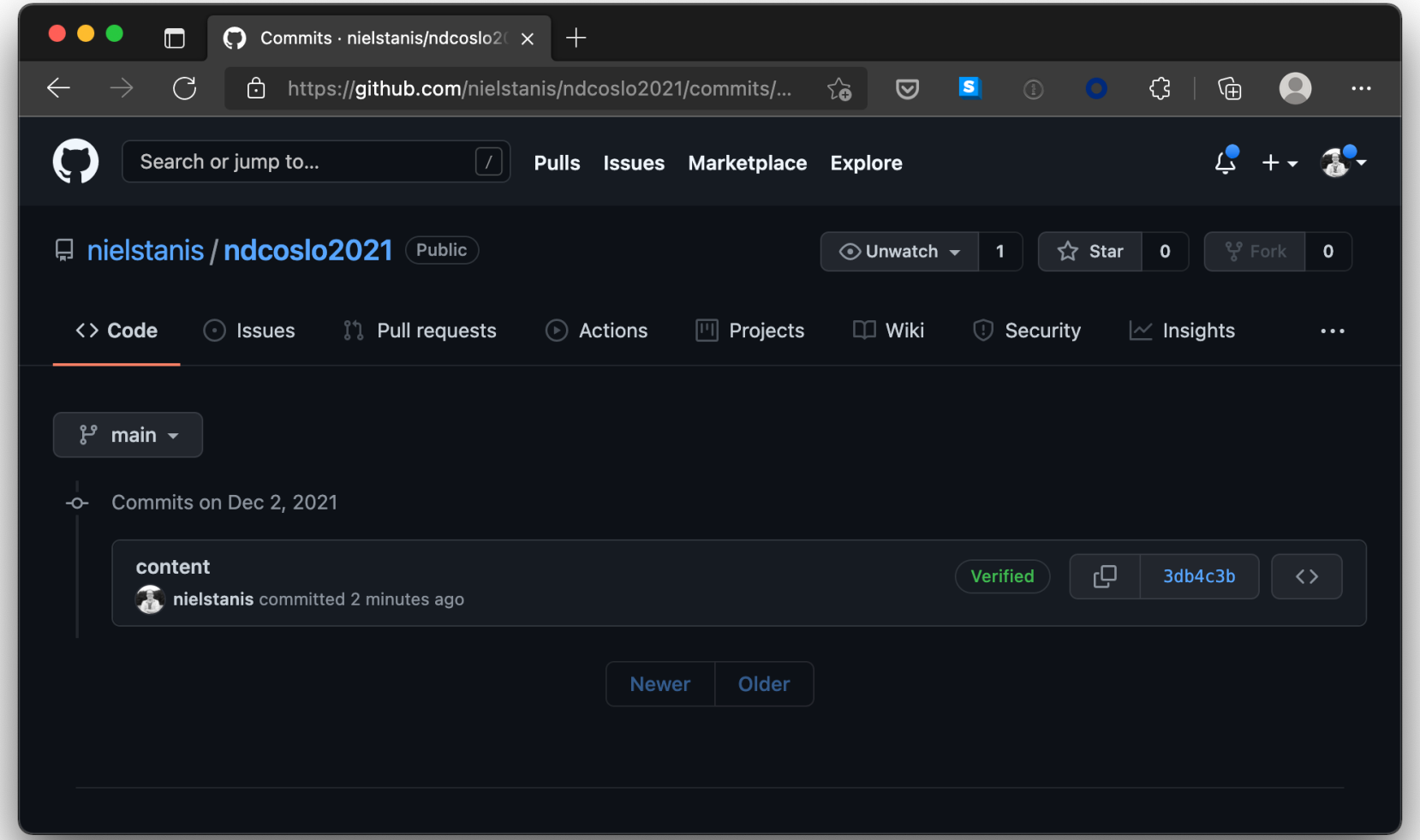


GitHub account

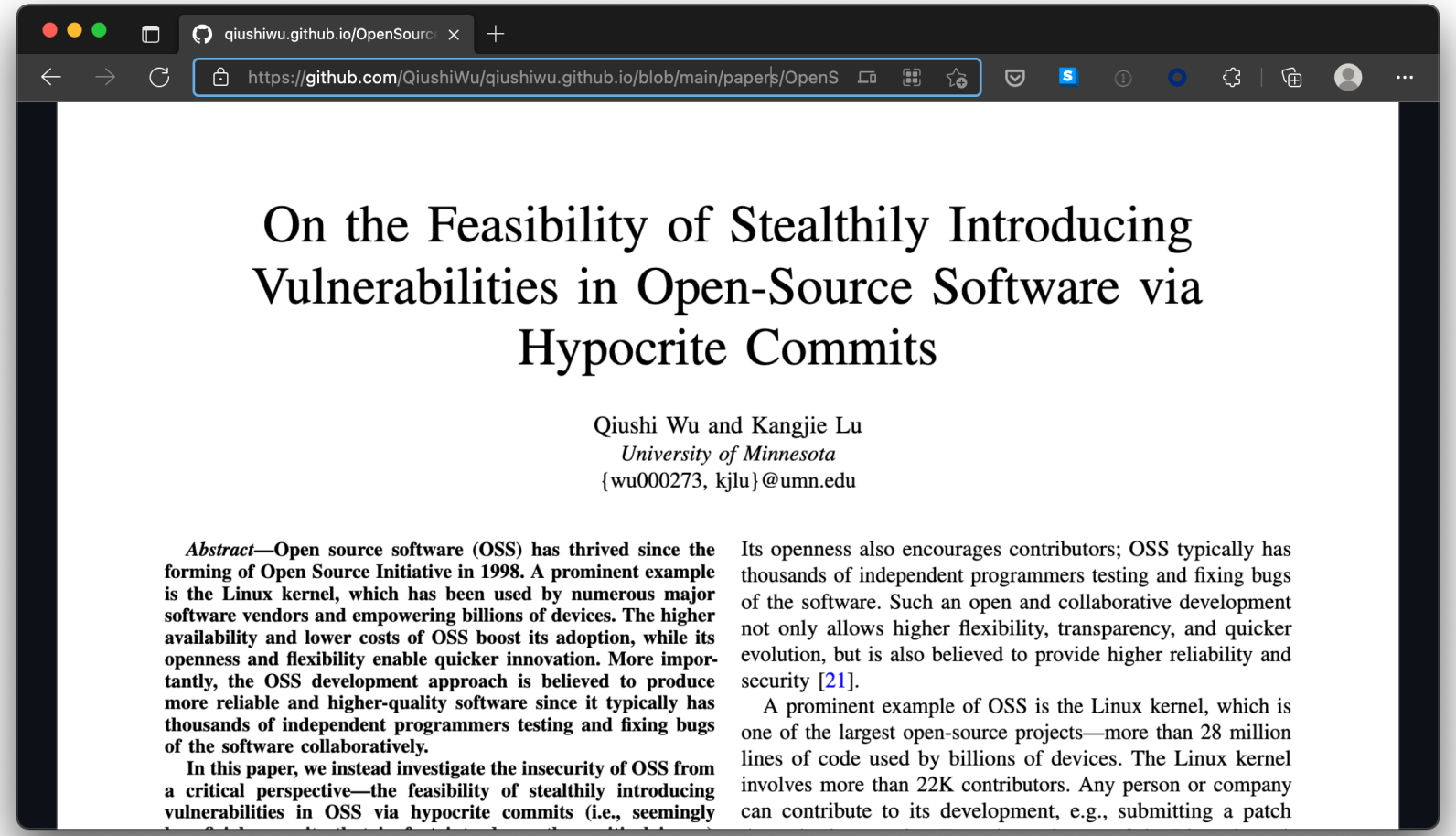
Use MFA on source-repository



GIT Commit Signing



Hypocrite Commits



On the Feasibility of Stealthily Introducing Vulnerabilities in Open-Source Software via Hypocrite Commits

Qiushi Wu and Kangjie Lu
University of Minnesota
{wu000273, kjlu}@umn.edu

***Abstract*—Open source software (OSS) has thrived since the forming of Open Source Initiative in 1998. A prominent example is the Linux kernel, which has been used by numerous major software vendors and empowering billions of devices. The higher availability and lower costs of OSS boost its adoption, while its openness and flexibility enable quicker innovation. More importantly, the OSS development approach is believed to produce more reliable and higher-quality software since it typically has thousands of independent programmers testing and fixing bugs of the software collaboratively.**

In this paper, we instead investigate the insecurity of OSS from a critical perspective—the feasibility of stealthily introducing vulnerabilities in OSS via hypocrite commits (i.e., seemingly

Its openness also encourages contributors; OSS typically has thousands of independent programmers testing and fixing bugs of the software. Such an open and collaborative development not only allows higher flexibility, transparency, and quicker evolution, but is also believed to provide higher reliability and security [21].

A prominent example of OSS is the Linux kernel, which is one of the largest open-source projects—more than 28 million lines of code used by billions of devices. The Linux kernel involves more than 22K contributors. Any person or company can contribute to its development, e.g., submitting a patch

Octopus Scanner - NetBeans

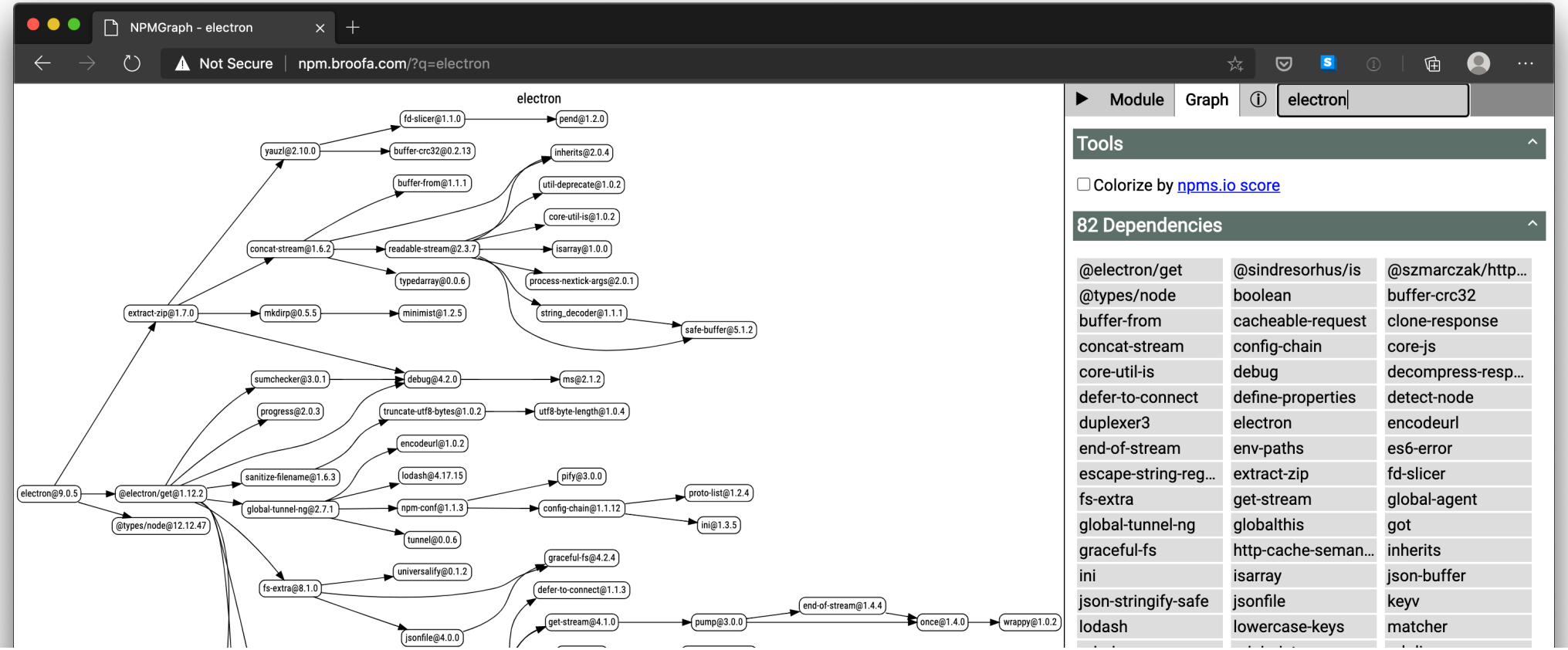
May 28, 2020

The Octopus Scanner Malware: Attacking the open source supply chain

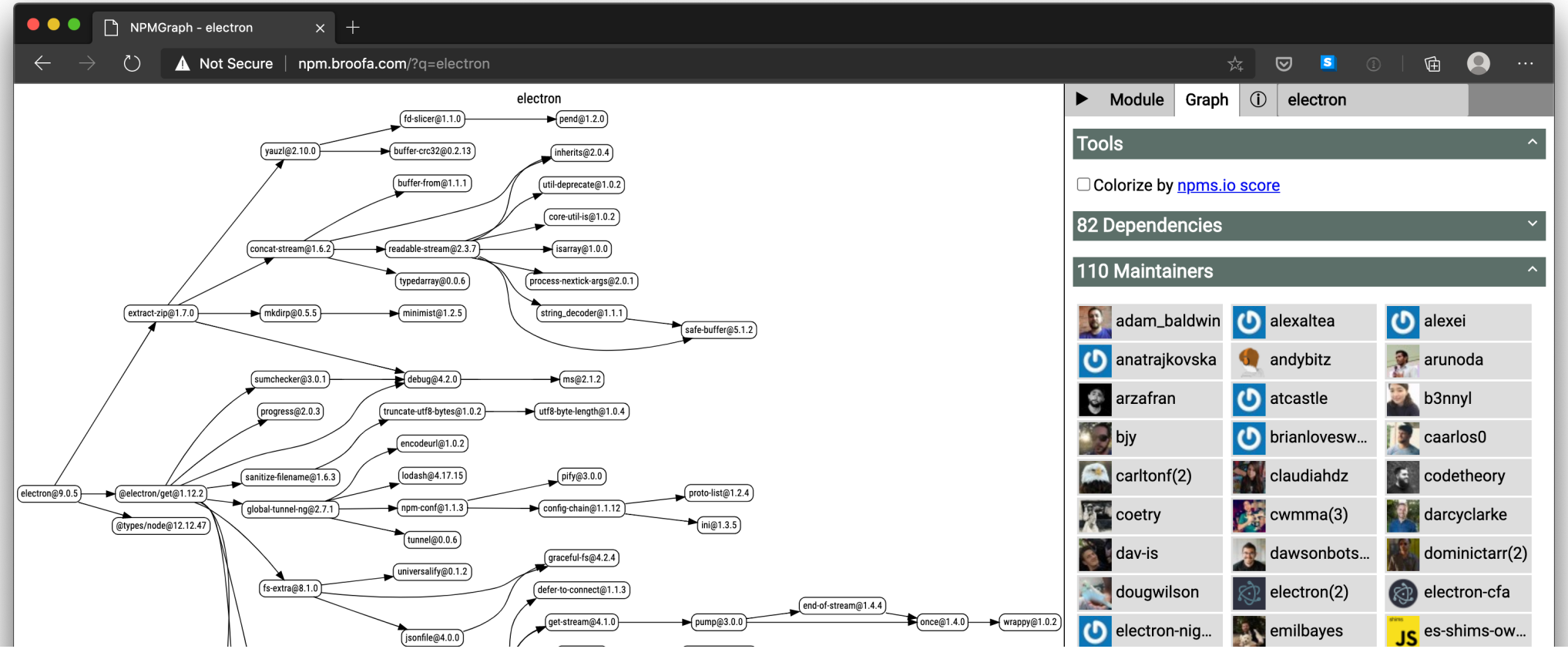
Alvaro Muñoz

Securing the open source supply chain is an enormous task. It goes far beyond a security assessment or just patching for the latest CVEs. Supply chain security is about the integrity of the entire software development and delivery ecosystem. From the code commits themselves, to how they flow through the CI/CD pipeline, to the actual delivery of releases, there's the potential for loss of integrity and security concerns, throughout the entire lifecycle.

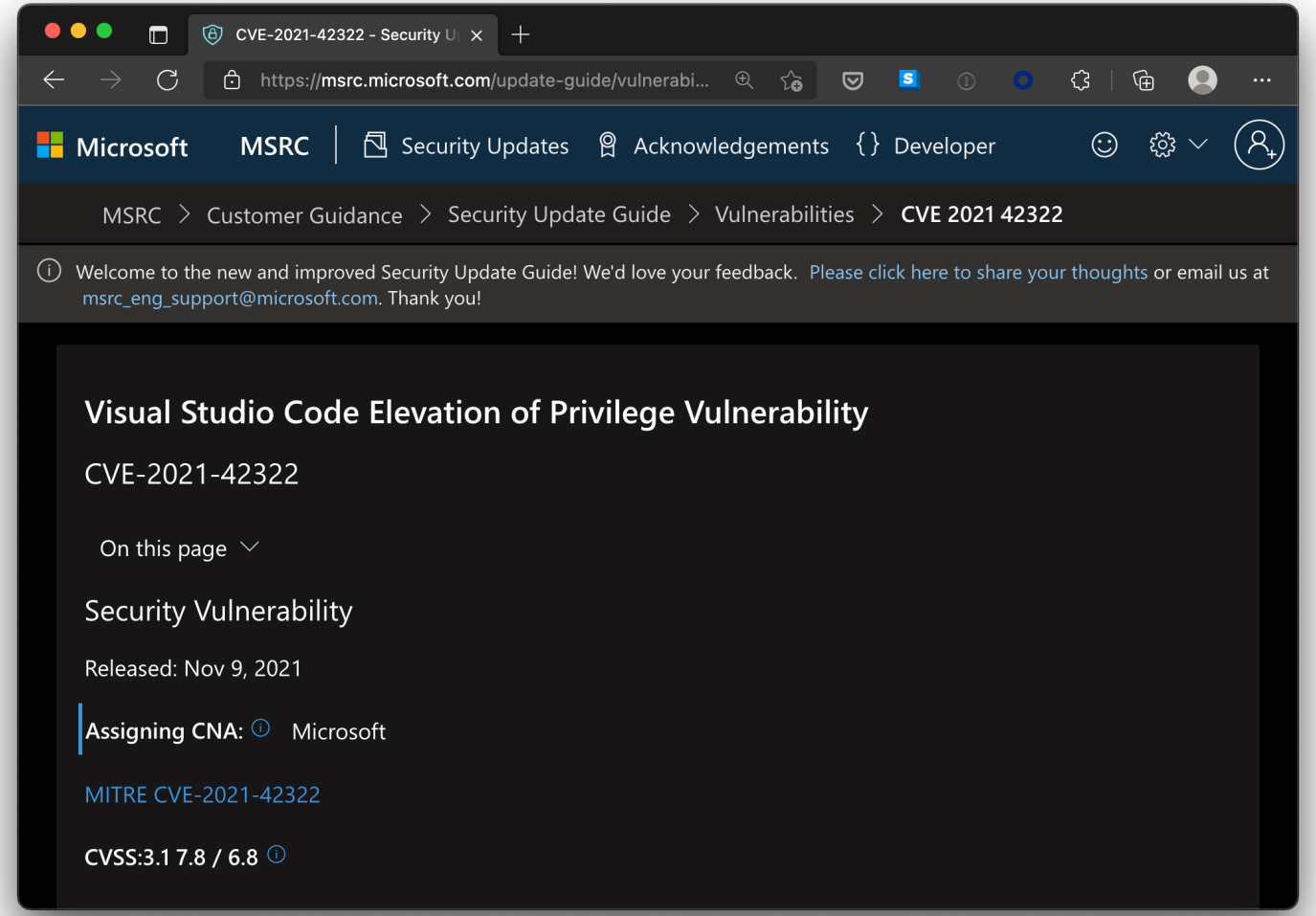
Visual Studio Code



Visual Studio Code

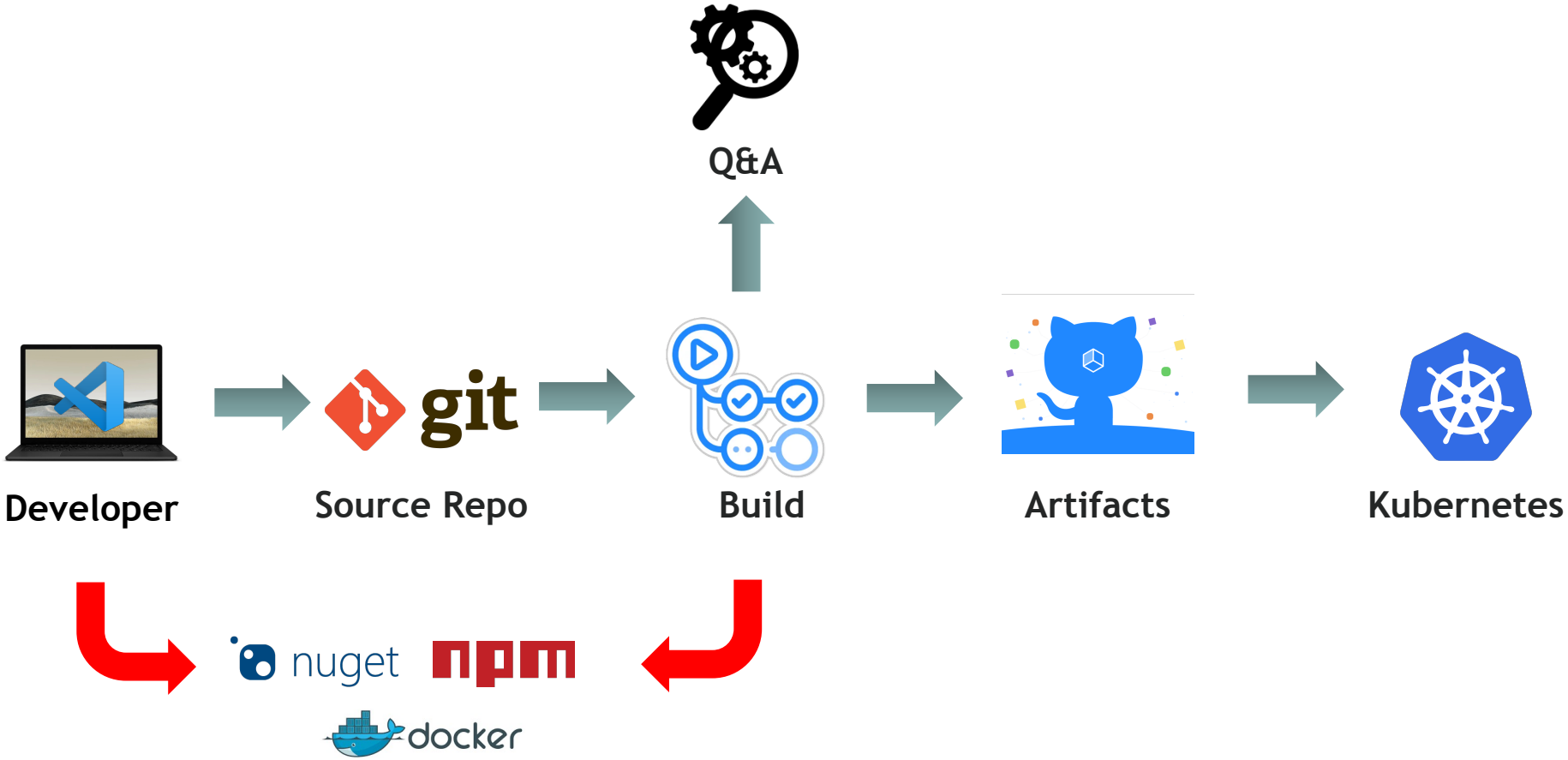


Visual Studio Code



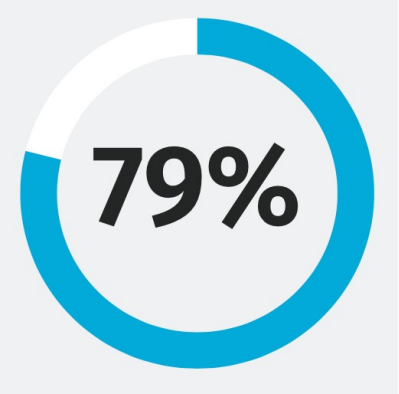
Visual Studio Code

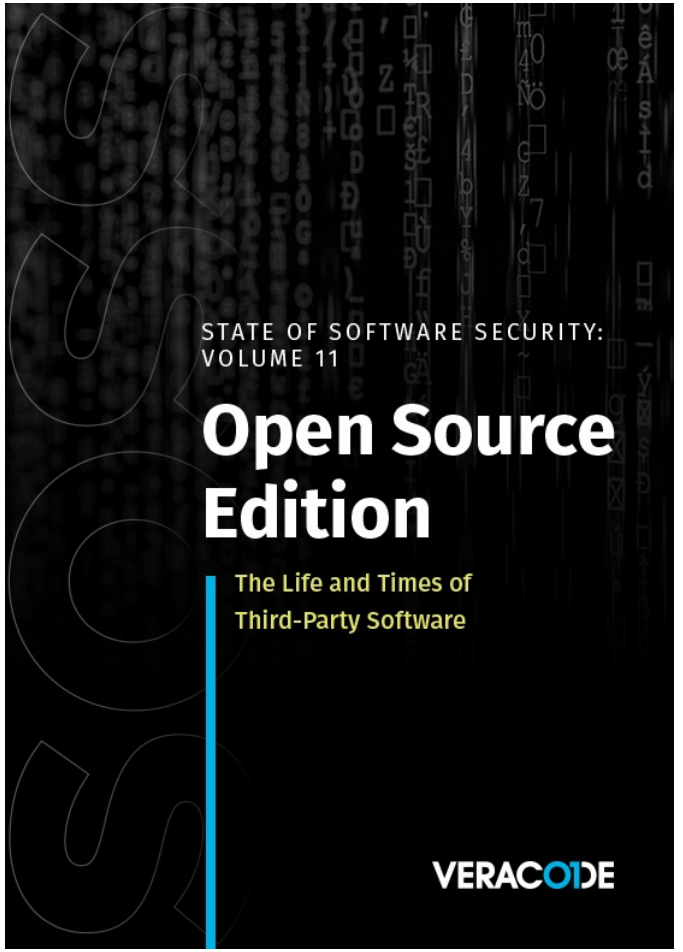
3rd Party Libraries



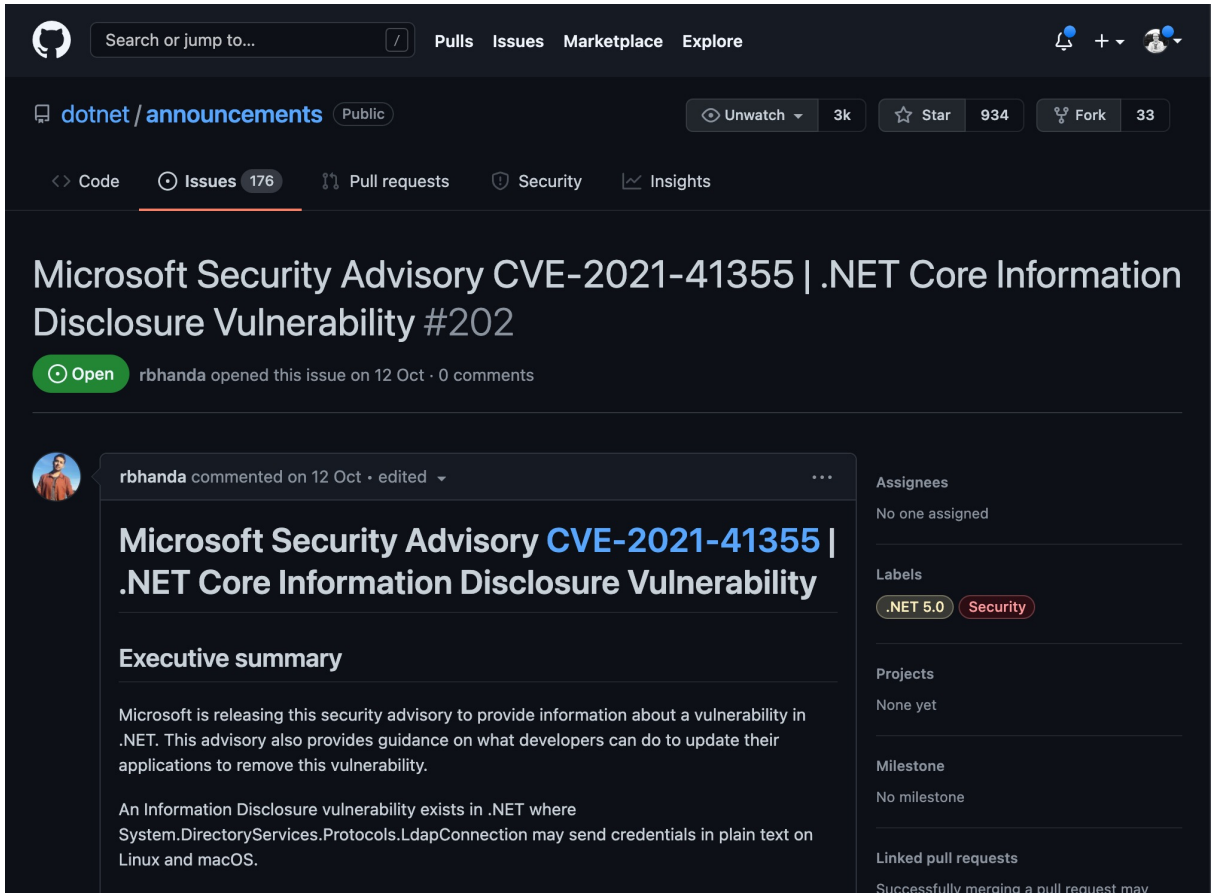
State Of Software Security v11 2021

”Despite this dynamic landscape, 79 percent of the time, developers never update third-party libraries after including them in a codebase.”





Vulnerabilities in libraries



Vulnerabilities in libraries

CYBERSECURITY & INFRASTRUCTURE SECURITY AGENCY

Alerts and Tips | Resources | Industrial Control Systems

National Cyber Awareness System > Current Activity > Malware Discovered in Popular NPM Package, ua-parser-js

Malware Discovered in Popular NPM Package, ua-parser-js

Original release date: October 22, 2021

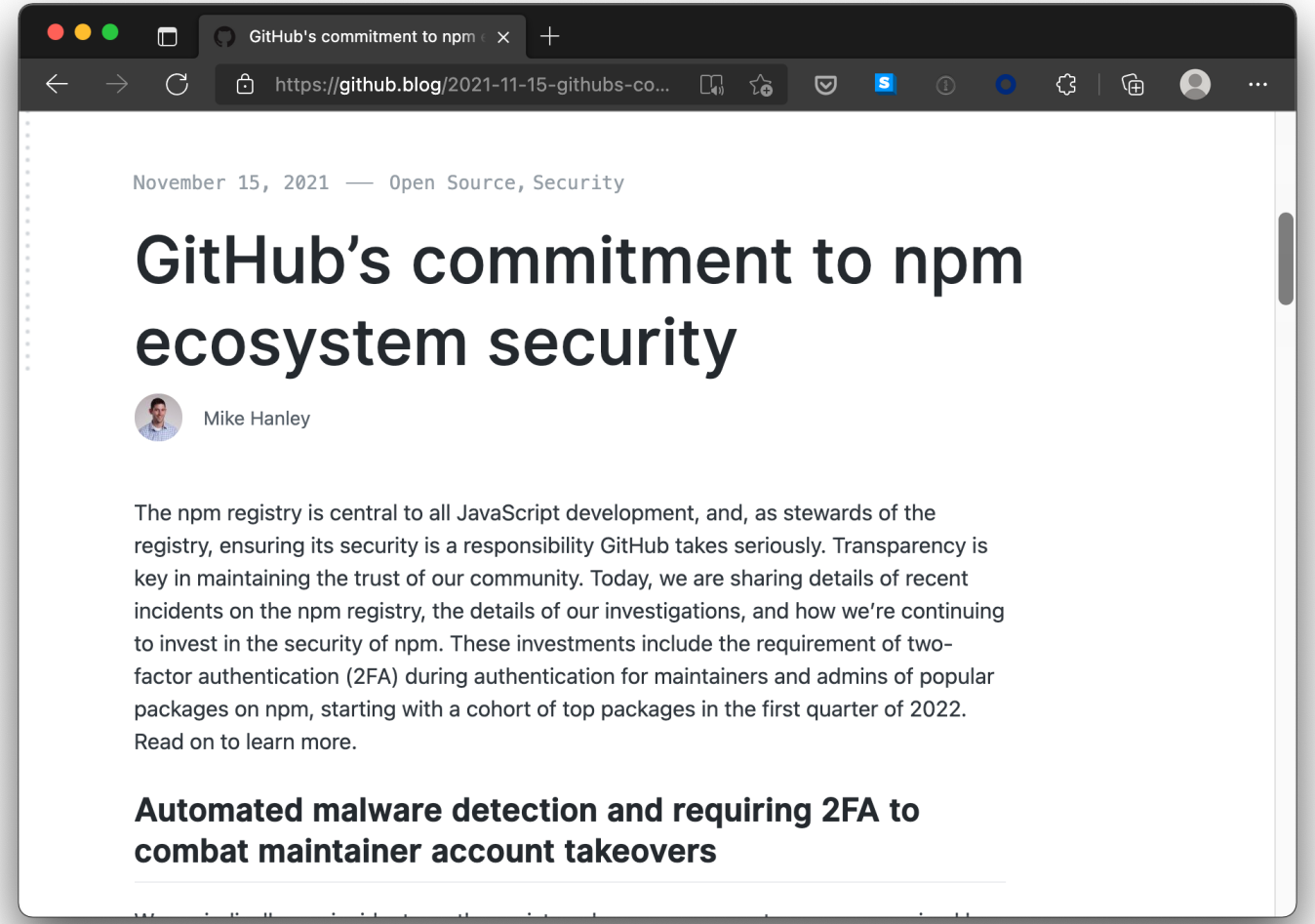
Print | Tweet | Send | Share

Versions of a popular NPM package named `ua-parser-js` was found to contain malicious code. `ua-parser-js` is used in apps and websites to discover the type of device or browser a person is using from User-Agent data. A computer or device with the affected software installed or running could allow a remote attacker to obtain sensitive information or take control of the system.

CISA urges users and administers using compromised ua-parser-js versions 0.7.29, 0.8.0, and 1.0.0 to update to the respective patched versions: 0.7.30, 0.8.1, 1.0.1

For more information, see Embedded malware in ua-parser-js.

Vulnerabilities in libraries

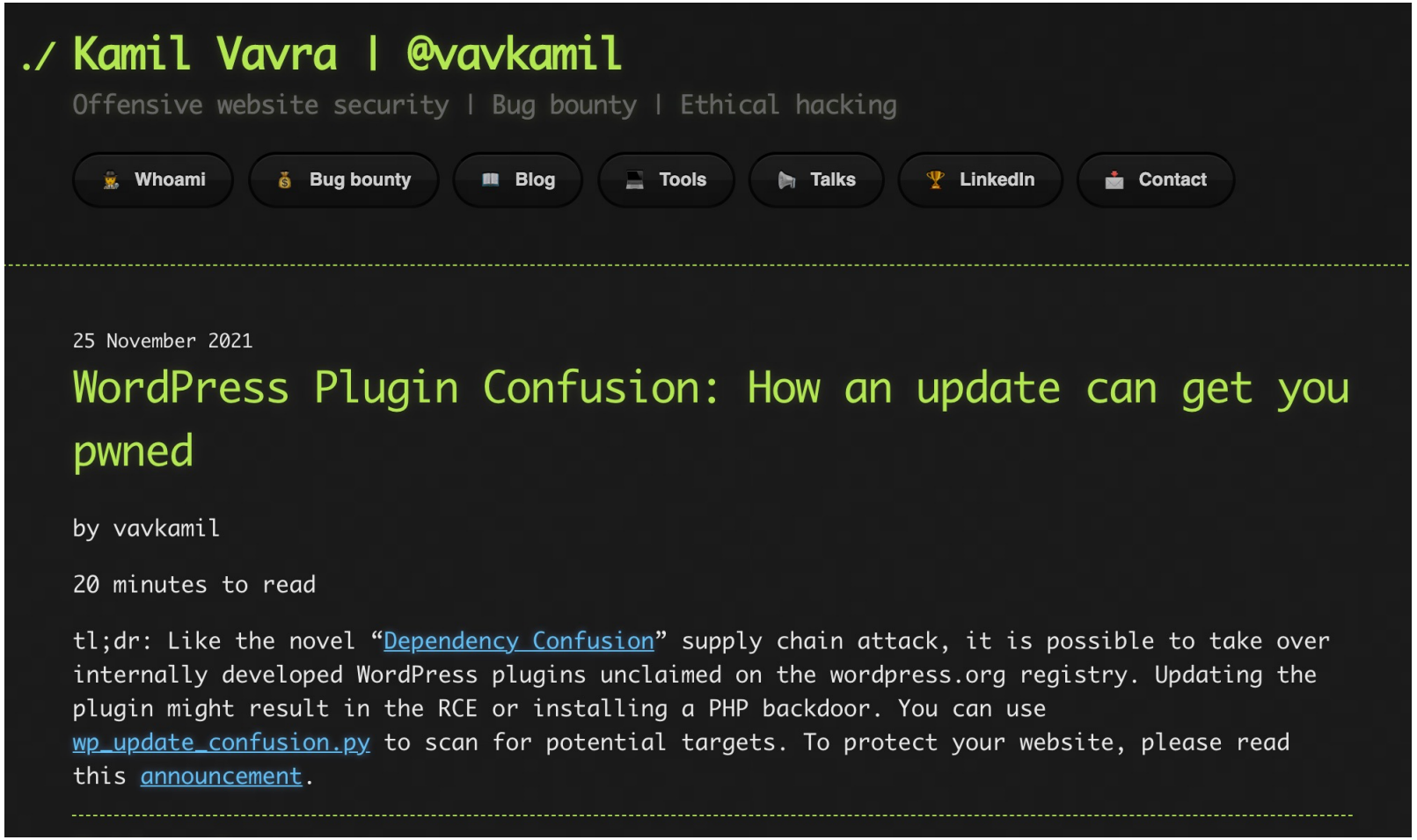


Dependency Confusion

Dependency Confusion - NuGet

- Use single private repository
- Azure Artifacts can help manage and control upstream
- If your company has public packages consider registering prefix
- Lock packages with config

WordPress Plugin Confusion

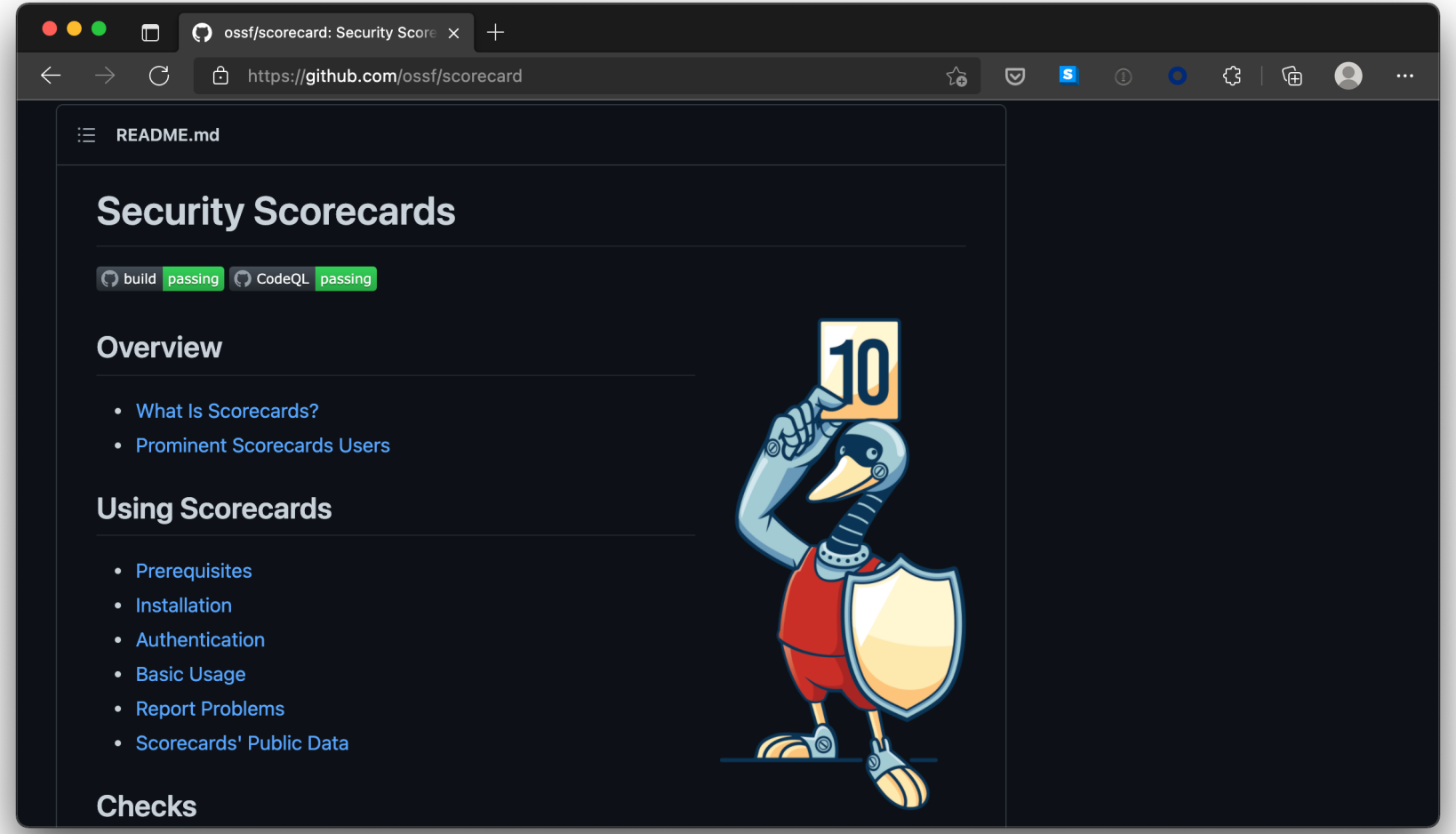


3rd Party Libraries

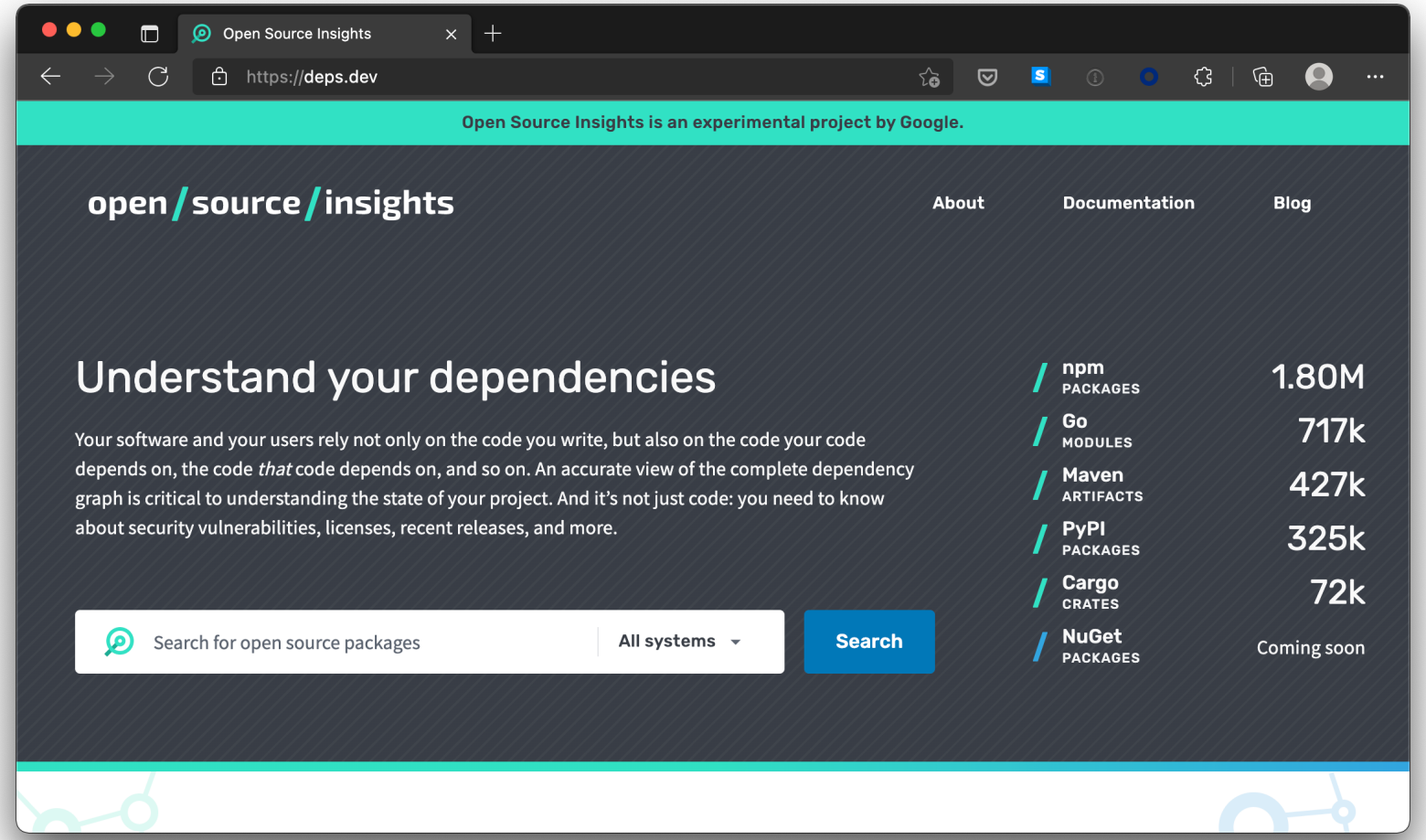
- Intent of library, know what's inside!
- Keep in mind that's a transitive list of dependencies
- Look out for talk on 'Sandboxing .NET Assemblies' I gave in November.

- Open Source Security Foundation - OpenSSF
- Security Scorecards - Security health metrics for Open Source

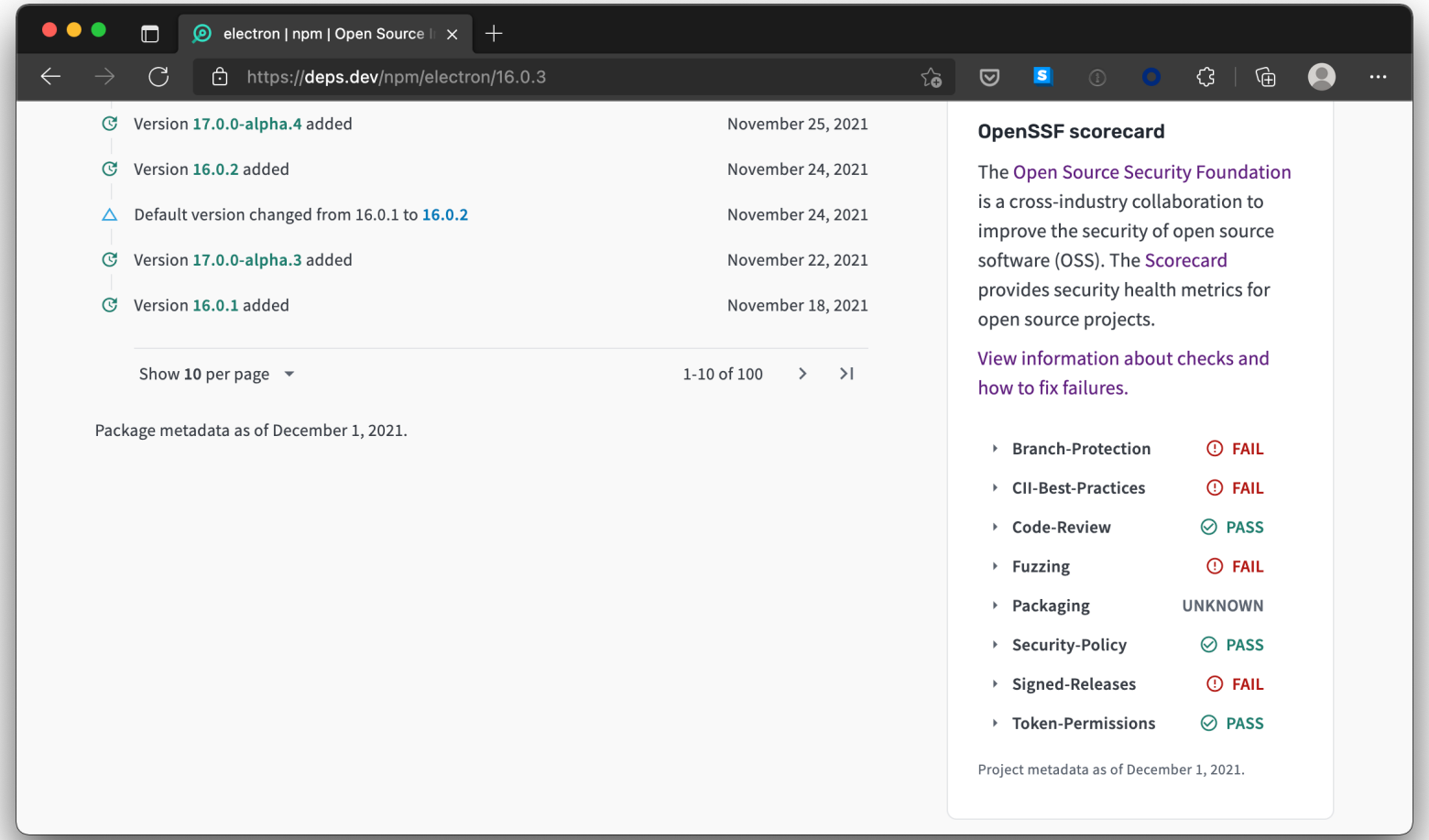
Security Scorecards - OpenSSF



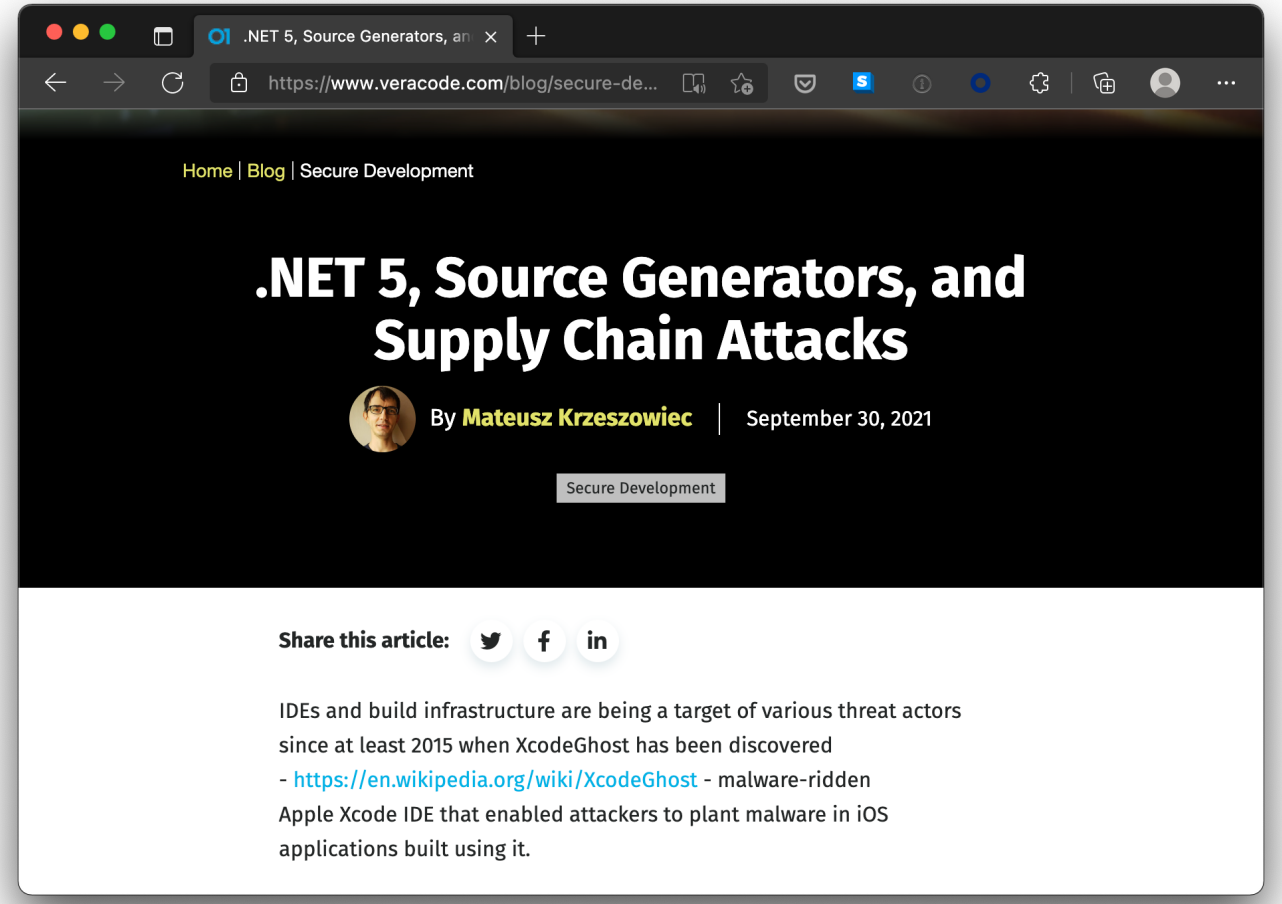
Deps.Dev by Google



Deps.Dev by Google



Source Generators

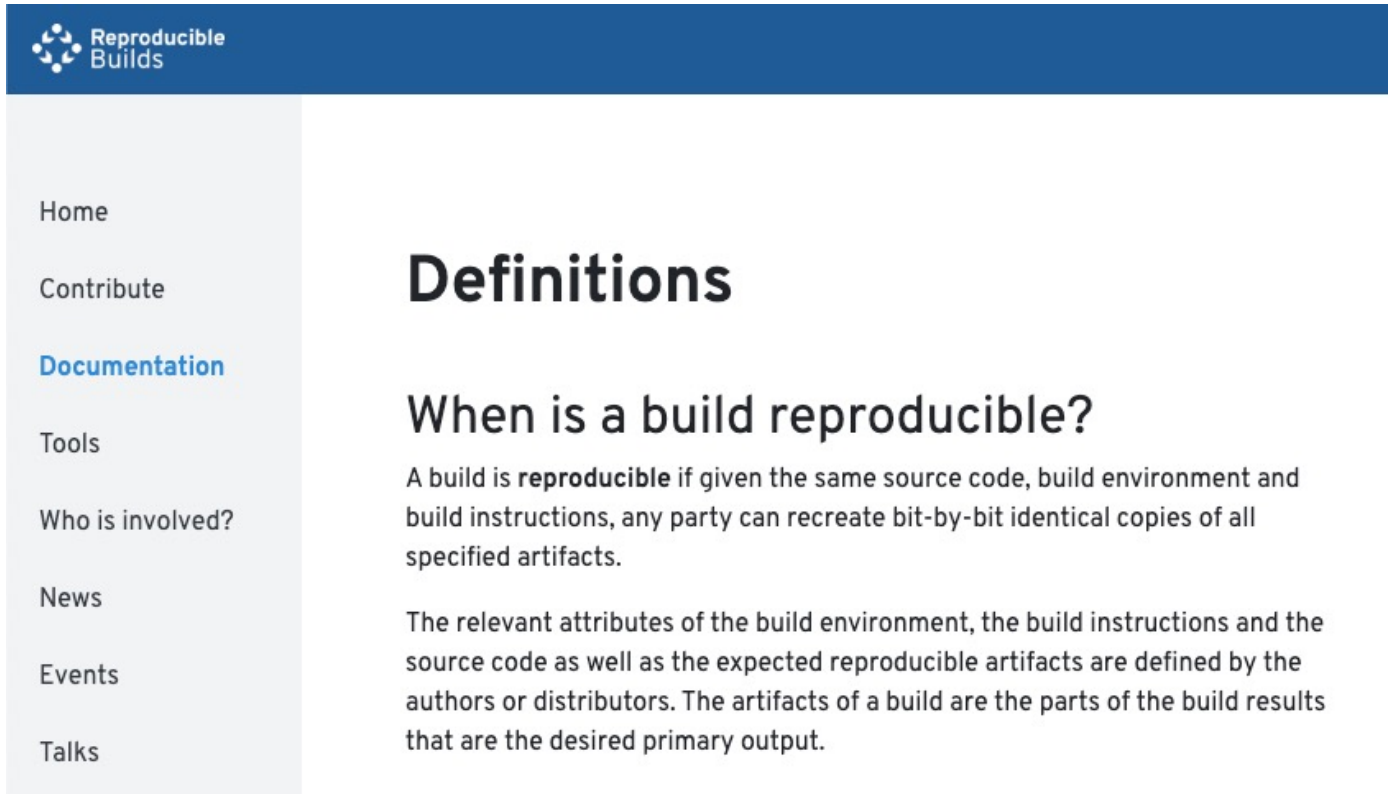


Source Generators

- Any 3rd party library can contain Source Generator!
- Consider disabling on project-level:

```
<Target Name="DisableAnalyzers"
        BeforeTargets="CoreCompile">
  <ItemGroup>
    <Analyzer Remove="@(Analyzer)" />
  </ItemGroup>
</Target>
```

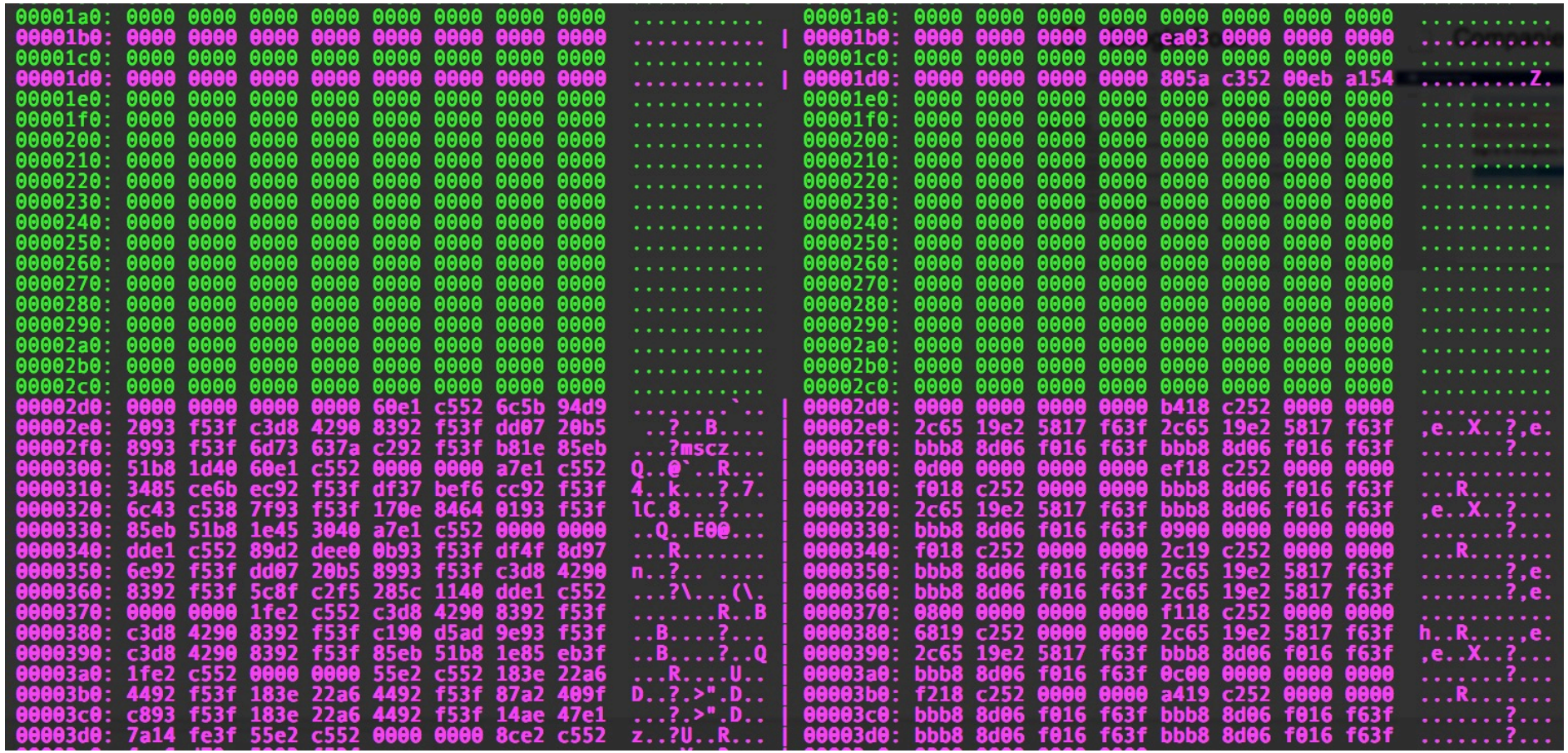
Reproducible/Deterministic Builds



Reproducible/Deterministic Builds

- Roslyn v1.1 started supporting some kind of determinism on how items are emitted
- Given same inputs, the compiled output will always be deterministic
- Inputs can be found in Roslyn compiler docs ‘Deterministic Inputs’

Reproducible Build .NET6

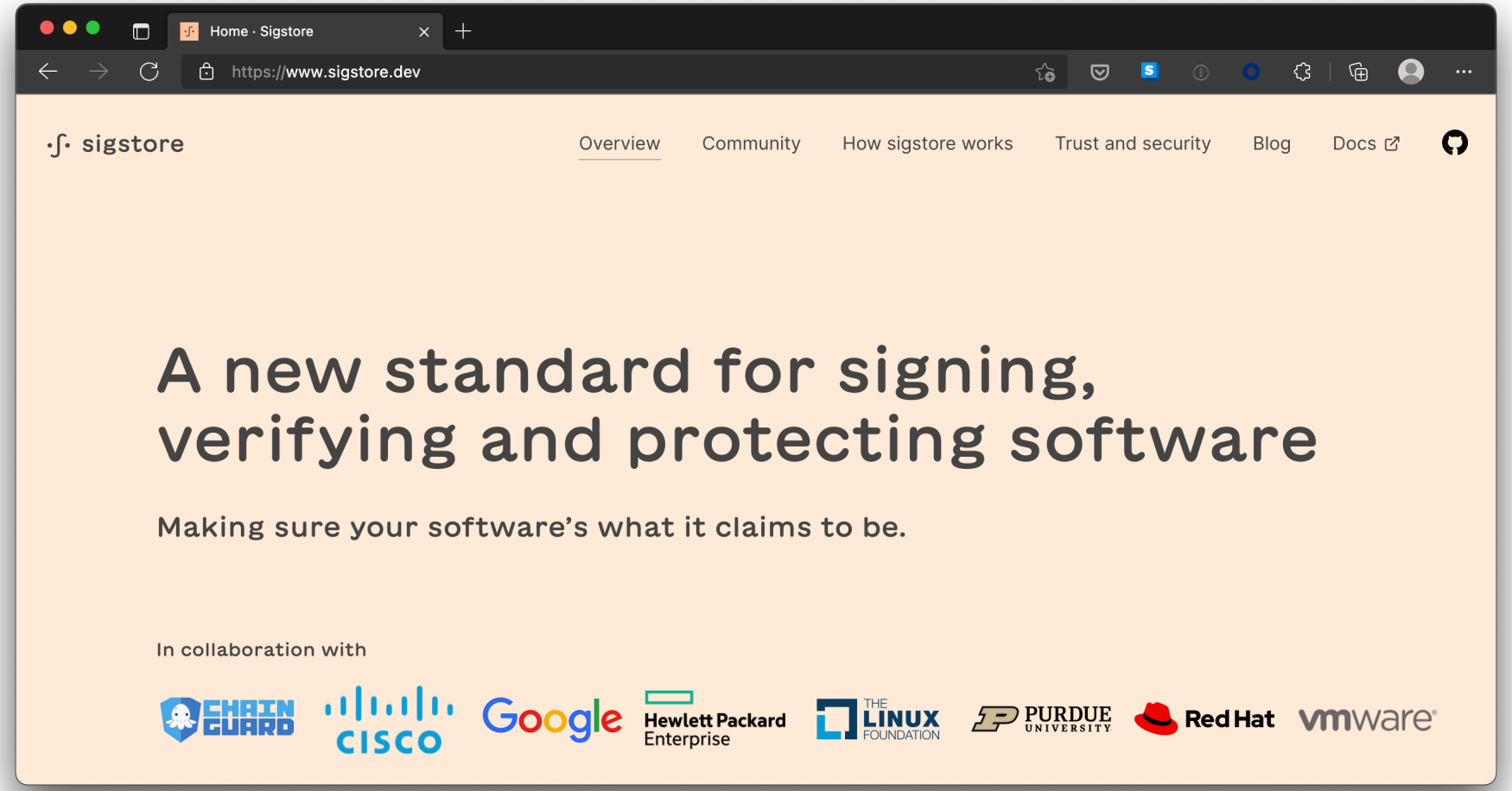


Reproducible/Deterministic Builds

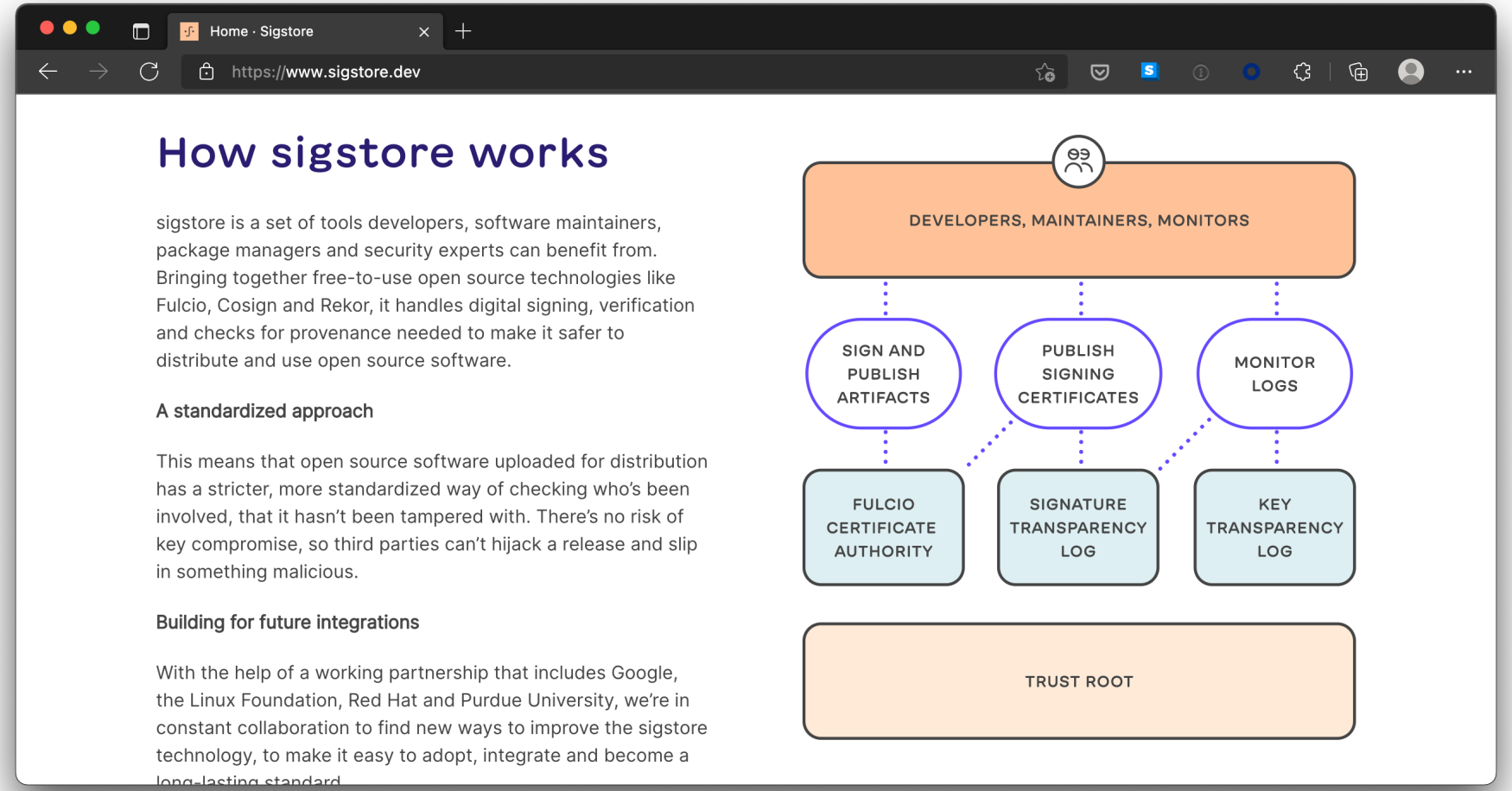
- DotNet.Reproducible Package
  - MSBuild *ContinuousIntegrationBuild*
  - SourceLink

- Dotnet.Reproducible.Isolated
  - Hermetic builds

Signing artifacts



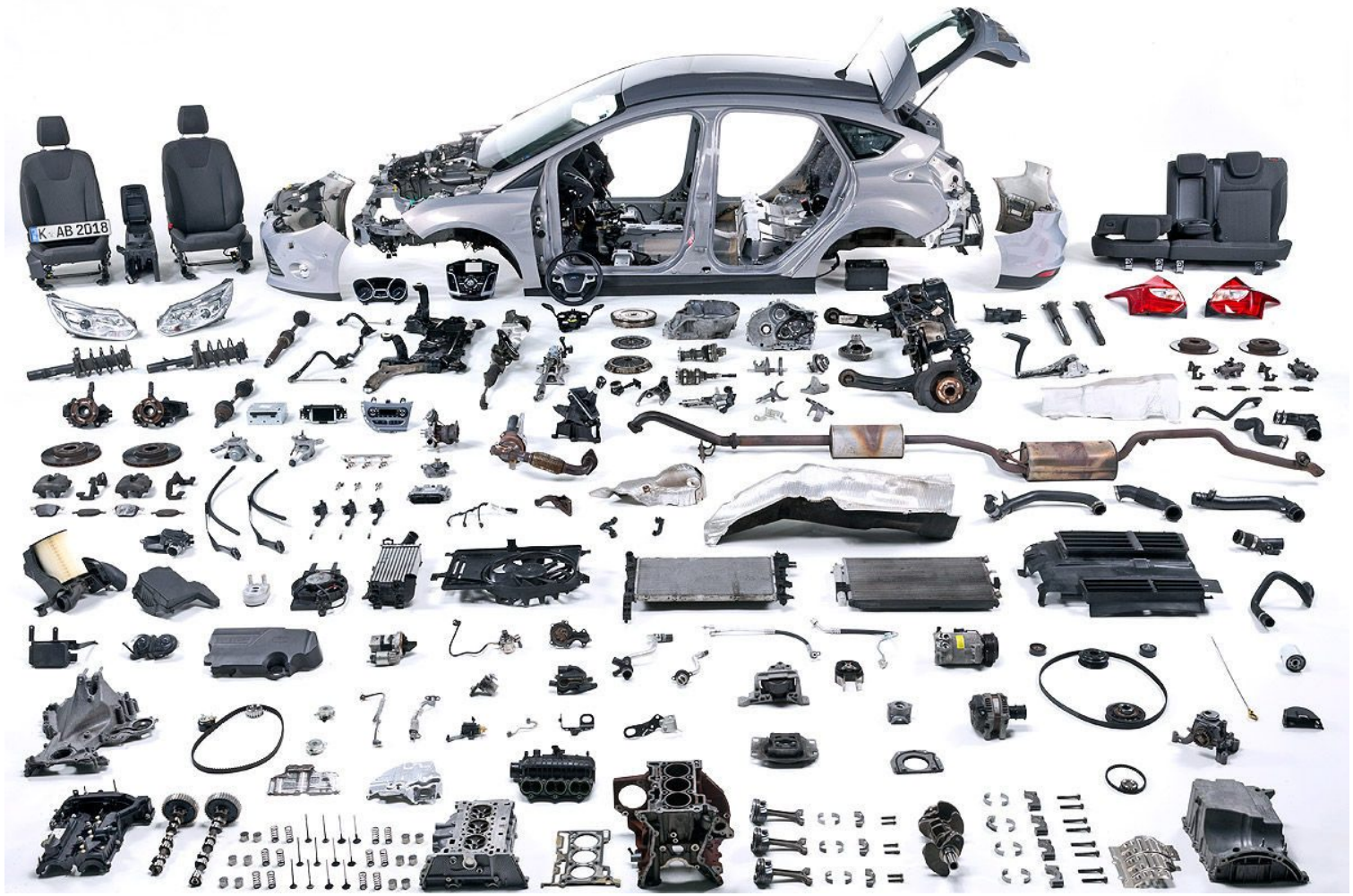
Signing artifacts



Signing artifacts

- Cosign can do keyless signing based on OpenID Connect
- GitHub Actions have released OpenID Connect support
- Demo time!

Automotive Industry



Car Supply Chain

Tata Steel Factory

- Iron Ore from Sweden
- ISO 6892-1 Tested/Certified
- Batch #1234

Bosch Factory

- Steel Batch #1234 Tata
- ECE-R90 Tested/Certified
- Serie #45678
- Used by Ford, Volkswagen and Renault

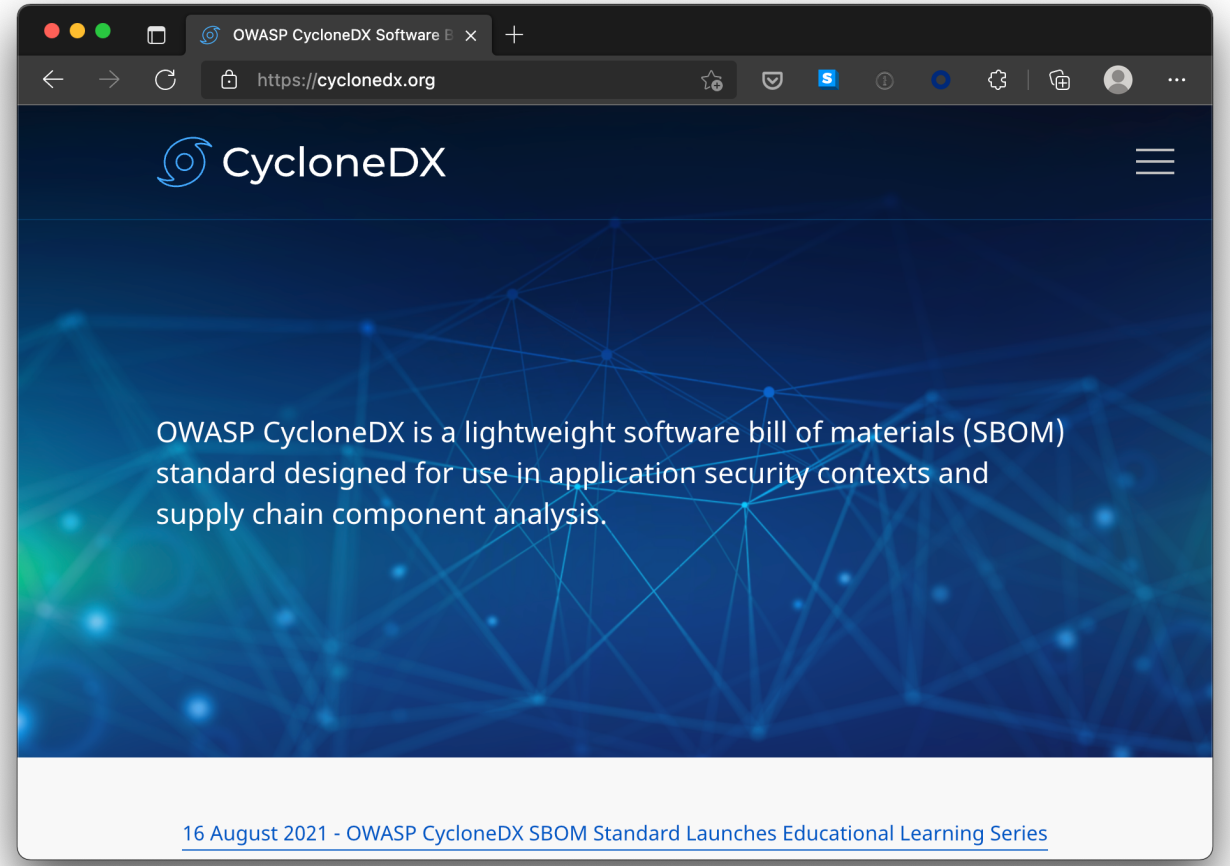
Renault Manufacturing

- Bosch Disk #45678
- Bosal Exhaust #RE9876
- Goodyear Tires #GY8877
- Kadjar VIN 1234567890

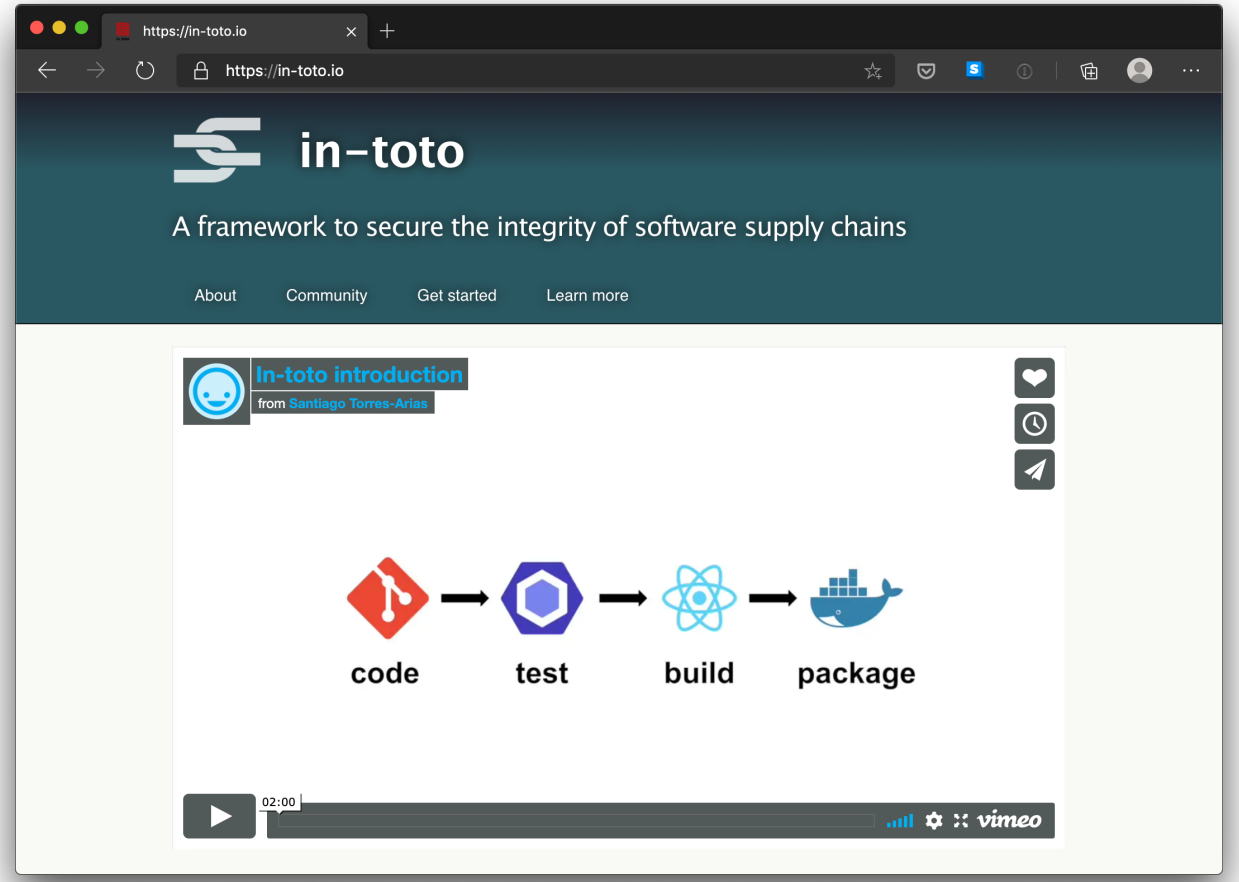
Software Bill of Materials (SBOM)

- Industry standard of describing the software
  - Producer Identity – Who Created it?
  - Product Identity – What's the product?
  - Integrity – Is the project unaltered?
  - Licensing – How can the project be used?
  - Creation – How was the product created? Process meets requirements?
  - Materials - How was the product created? Materials/Source used?

Software Bill of Materials (SBOM)



In-toto


https://in-toto.io
https://in-toto.io
in-toto
A framework to secure the integrity of software supply chains
About
Community
Get started
Learn more
In-toto introduction
from Santiago Torres-Arias
code
test
build
package
02:00
vimeo

In-Toto – Demo - Terminology

- **Functionaries** that are identified by public key our supply chain.
  For example, the Project-Owner, Developer, and Release Manager

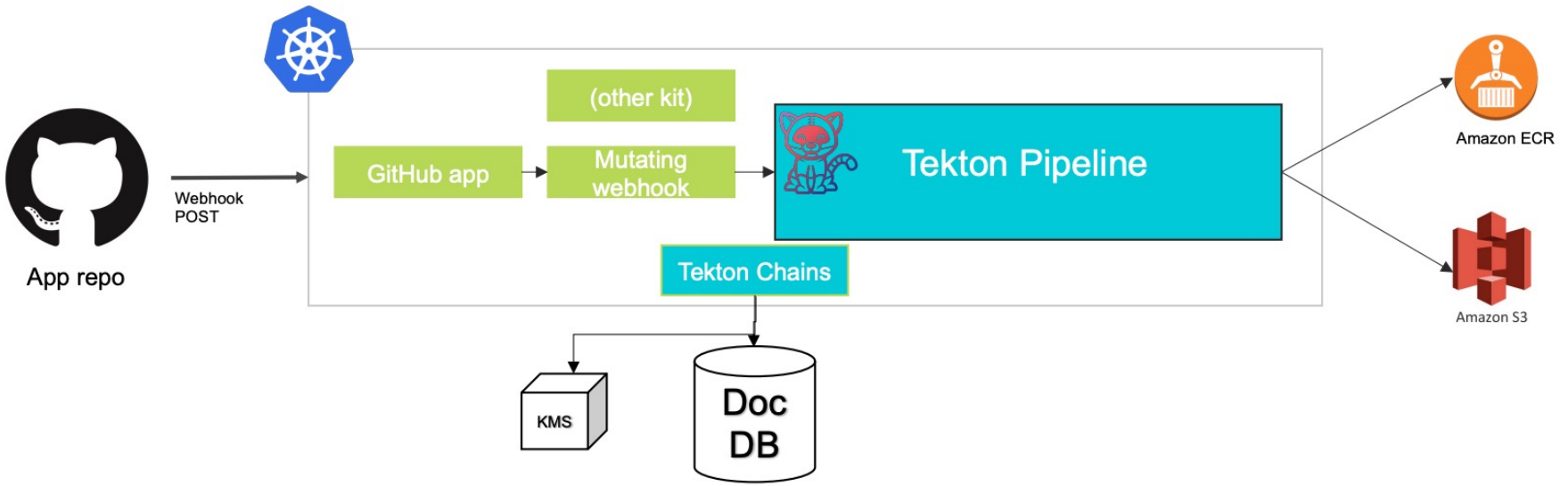
- **Project-Owner** defines a (**Supply Chain) Layout** that describes **what** happens and by **who** and what the produced **Materials** and **Byproducts** are.

- **Link** metadata is output of executed step in the **Layout**
  Materials are input, **Products** are output and can be used as **Materials** in later steps

In-Toto - Demo

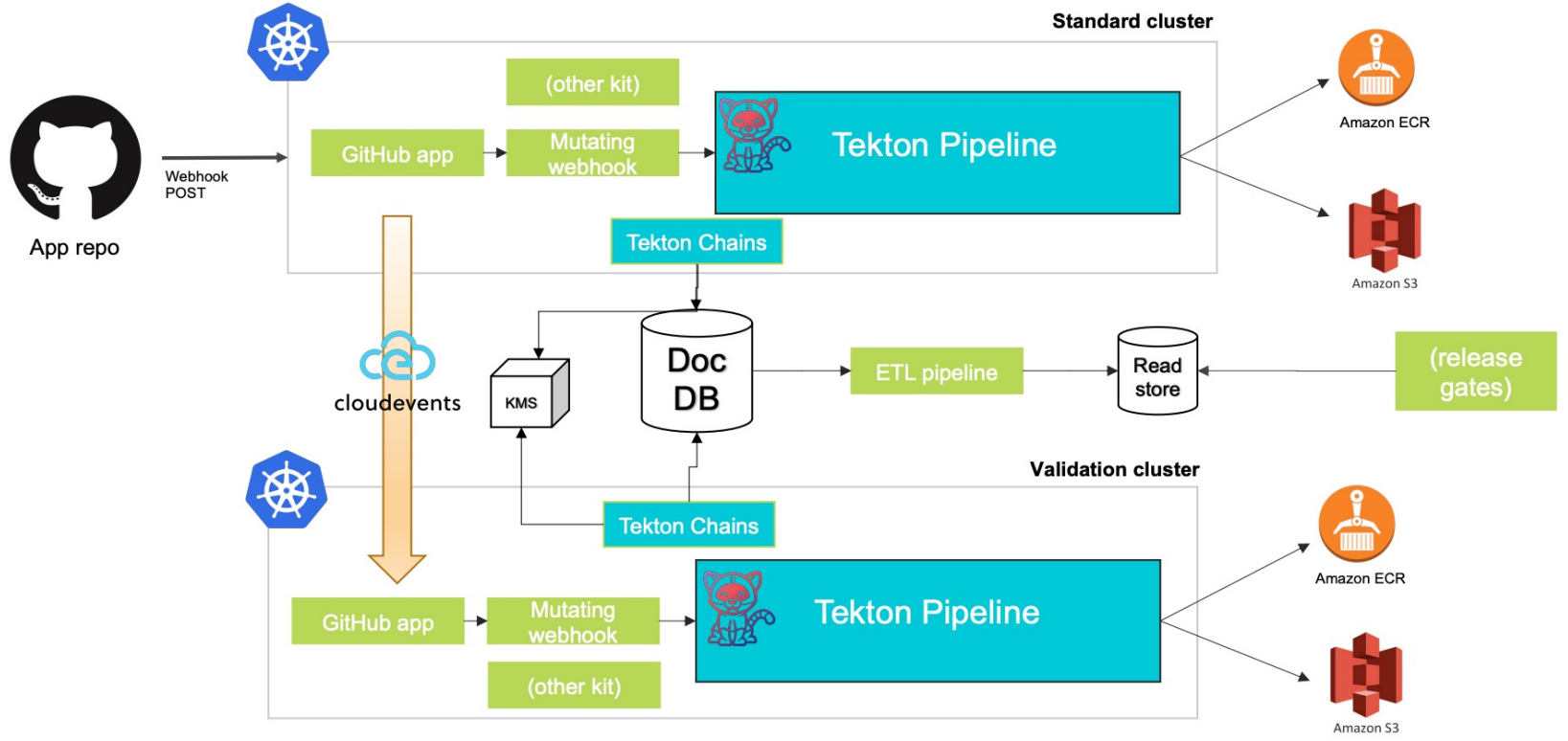
SolarWinds Project Trebuchet

Pipeline With Attestations

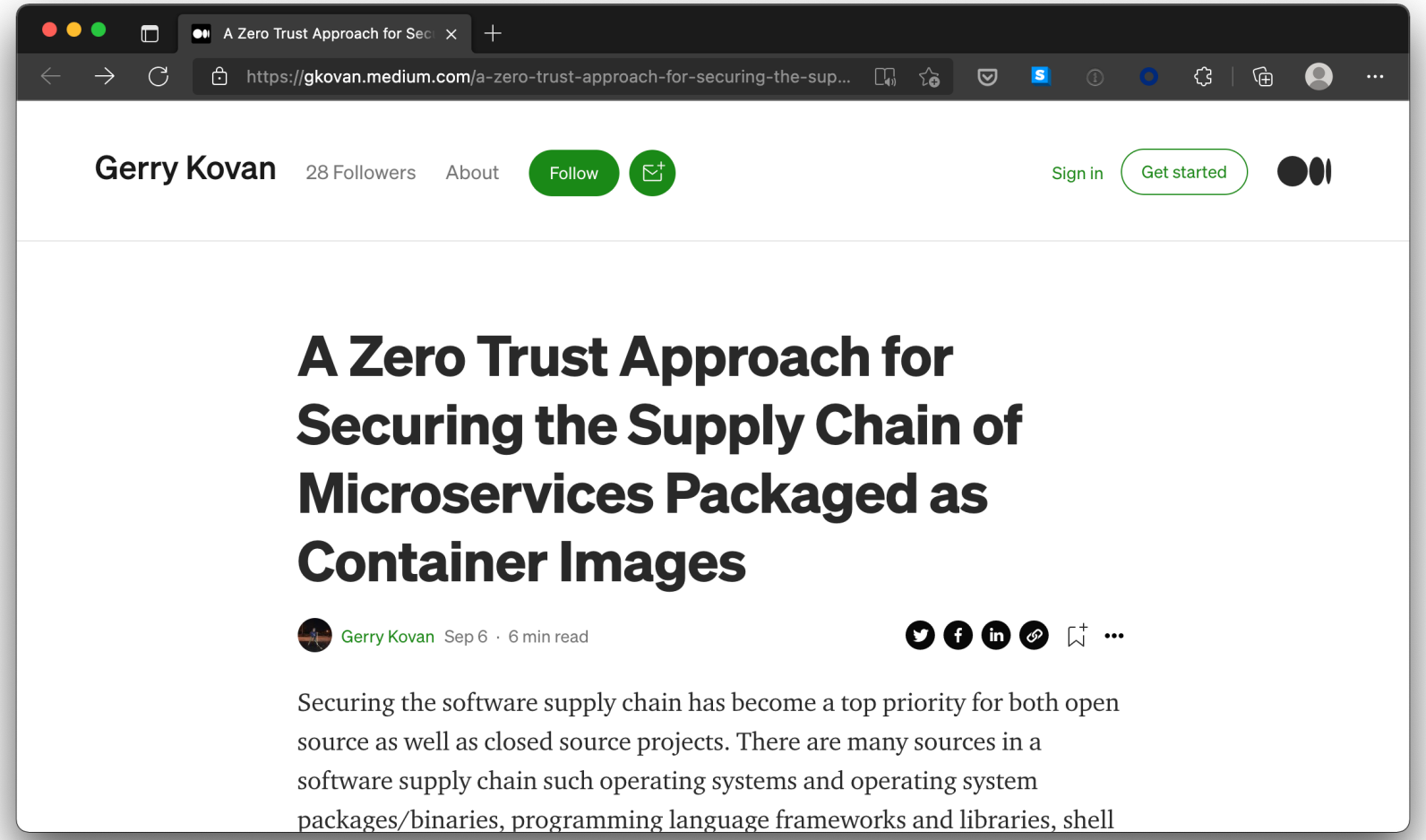


SolarWinds Project Trebuchet

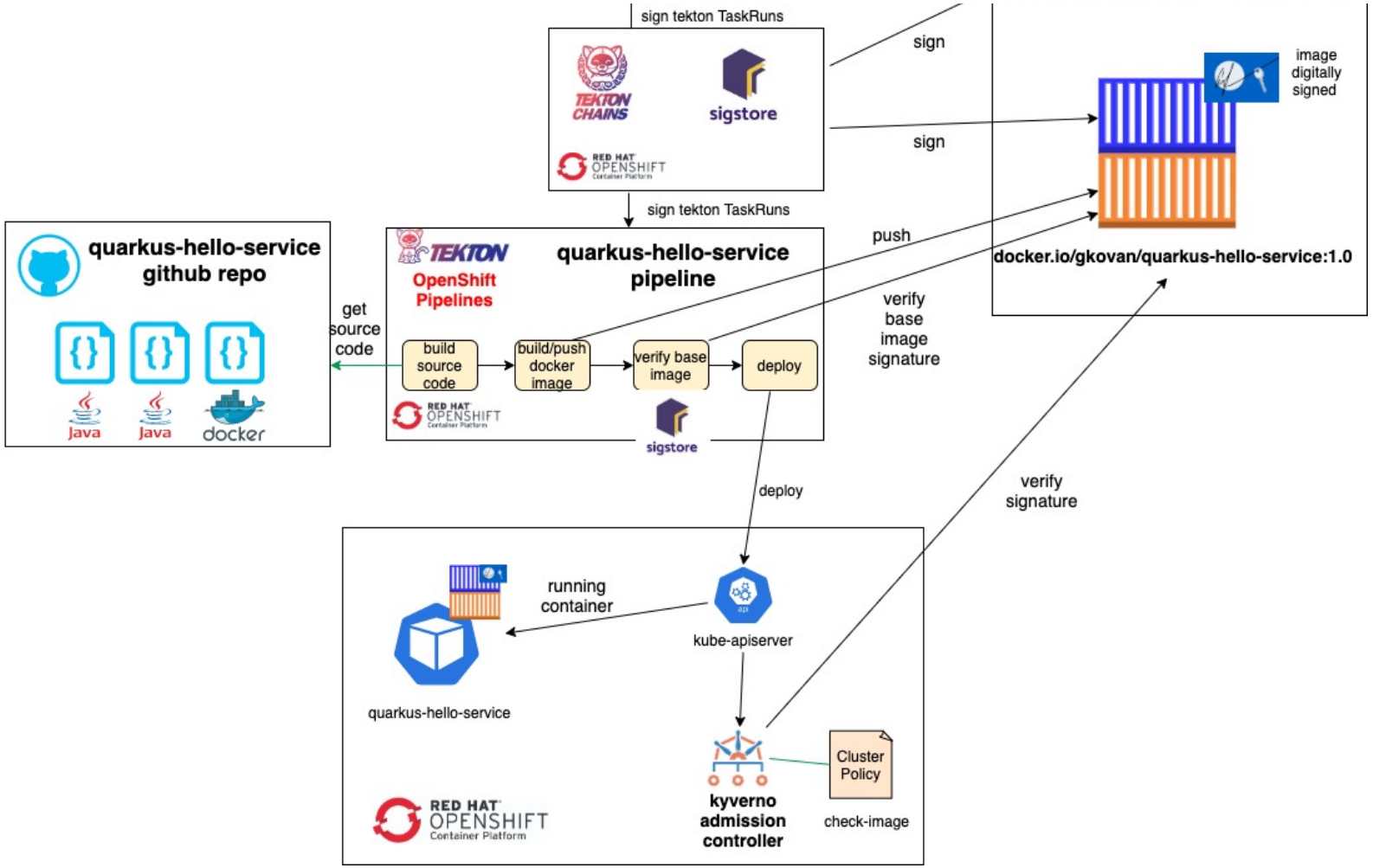
Reading Results



IBM OpenShift

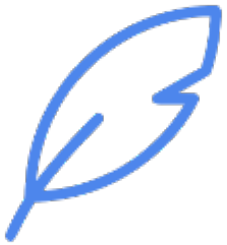


IBM OpenShift

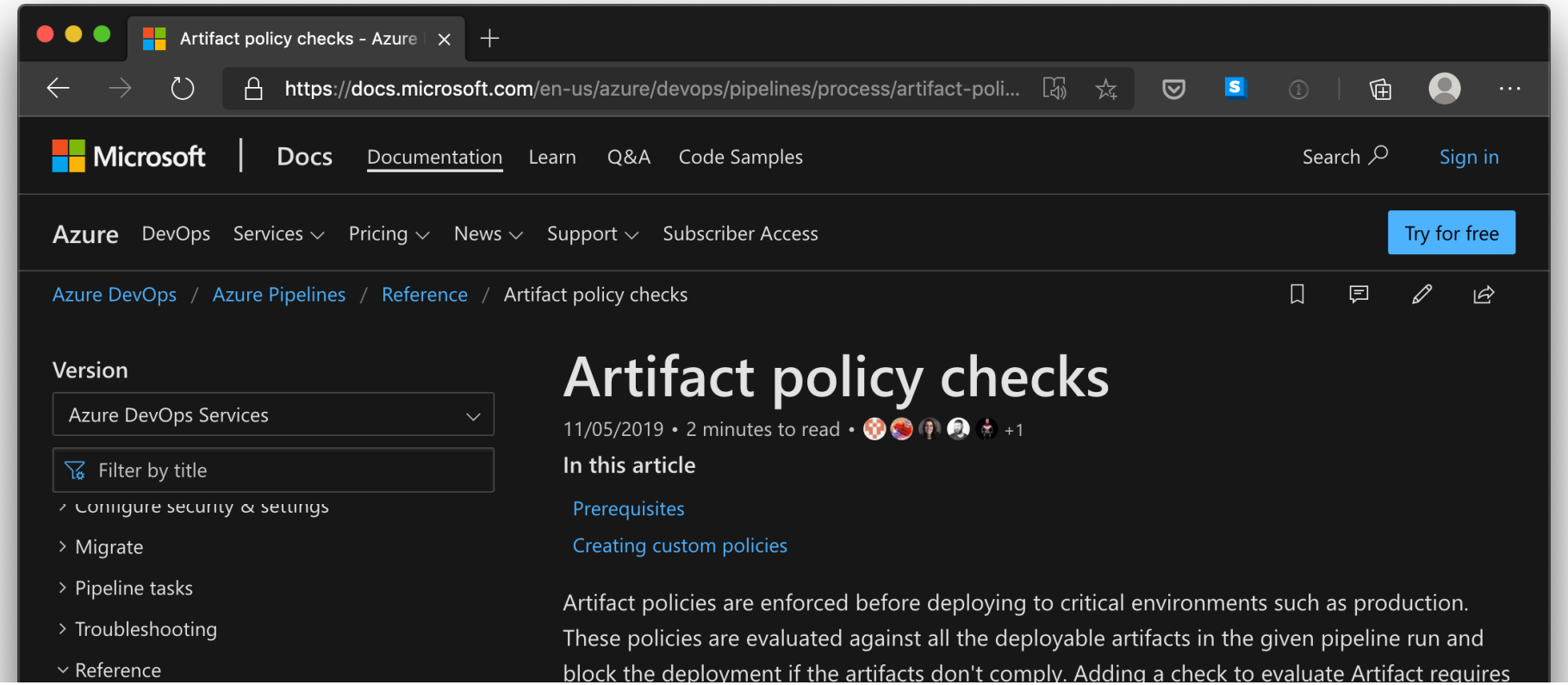


Grafeas and Kritis by Google

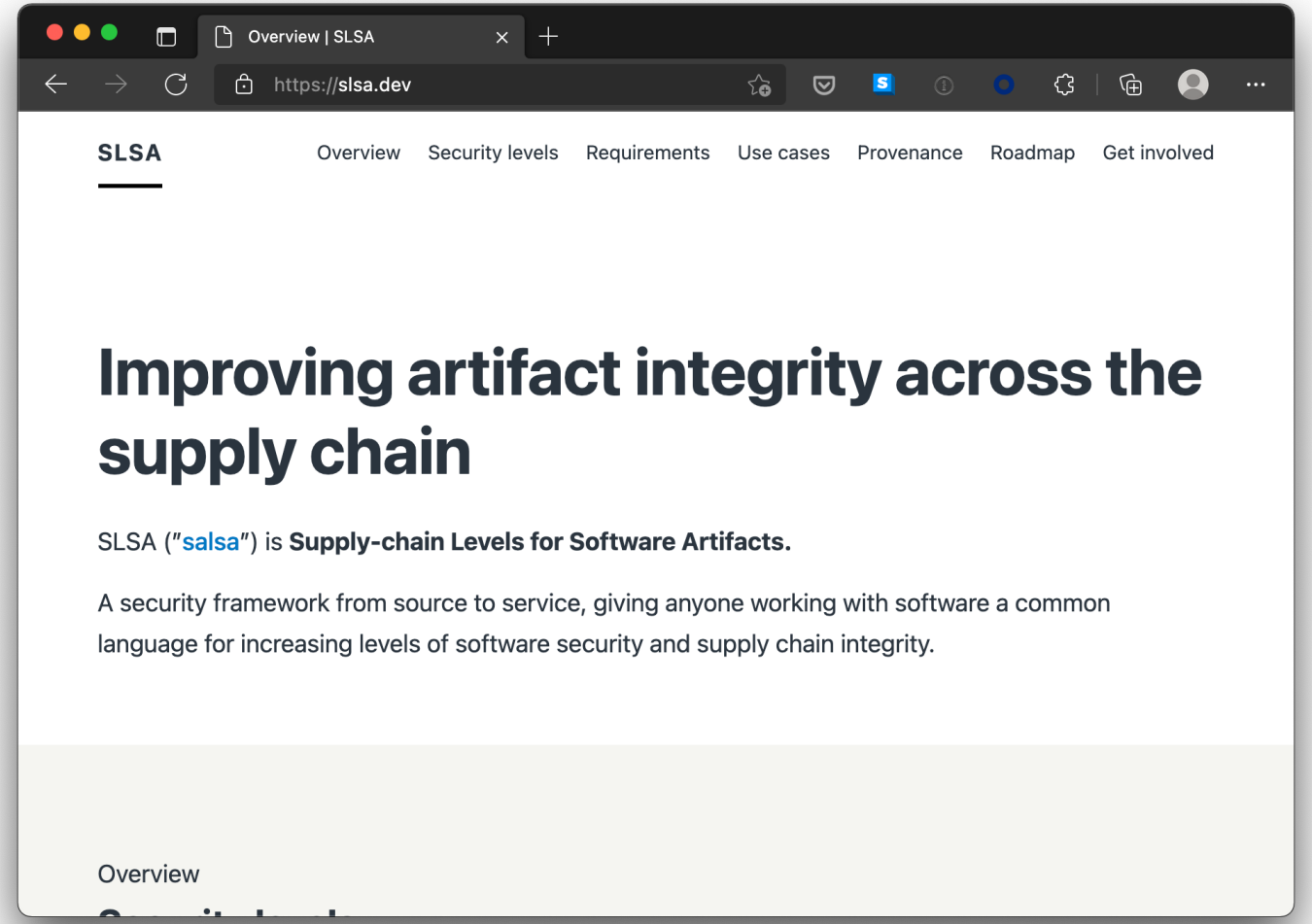
- Grafeas – Component Metadata API
  - Container Analysis API on Google Cloud Platform
- Kritis – Deployment Authorization for Kubernetes Apps
  - Binary Authorization on Google Cloud Platform



Azure Pipelines Artifact Policy



Google SLSA


Overview | SLSA
https://slsa.dev
SLSA
Overview
Security levels
Requirements
Use cases
Provenance
Roadmap
Get involved
Improving artifact integrity across the supply chain
SLSA ("salsa") is Supply-chain Levels for Software Artifacts.
A security framework from source to service, giving anyone working with software a common language for increasing levels of software security and supply chain integrity.
Overview

Conclusion

- It’s not how it’s more a matter of when!
- Be aware of your used software supply chain(s).
- Know what you’re using and pulling into projects.
- Integrate security into your software lifecycle.
- Start working on creating SBOM’s and see how SLSA can fit into your process.

VERACODE

Thanks! Questions?

https://github.com/nielstanis/ndcoslo2021
https://github.com/nielstanis/ndcoslo2021-webapp
ntanis at veracode.com
@nielstanis on Twitter