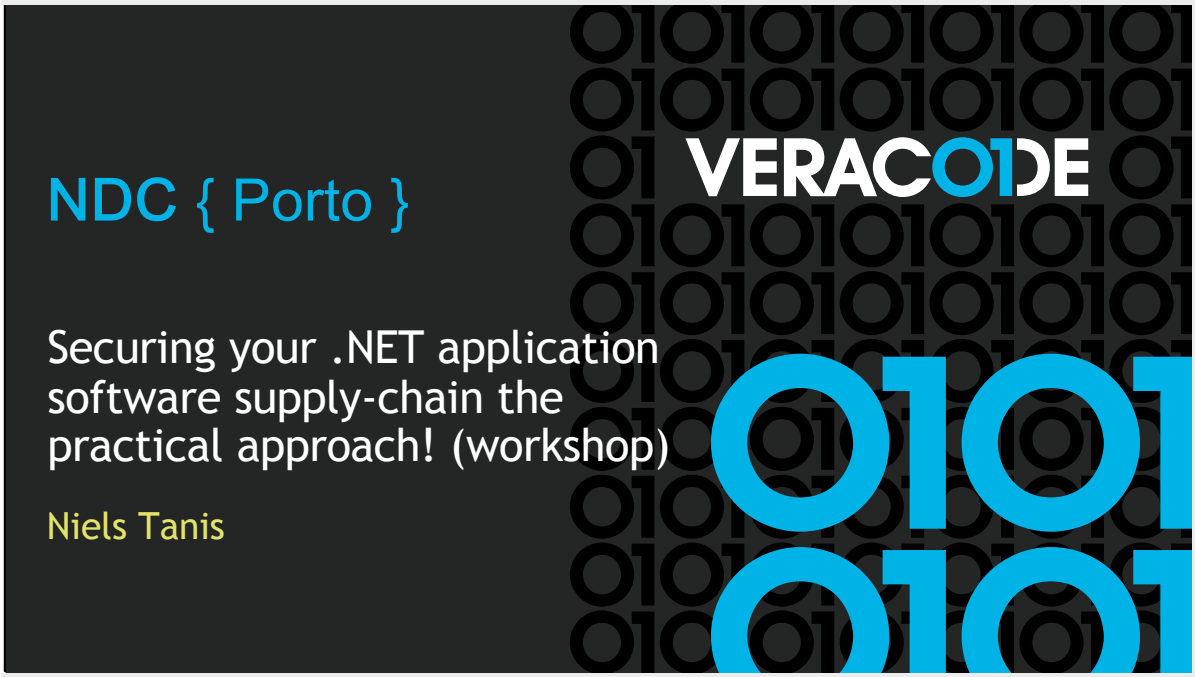
NDC { Porto }

Securing your .NET application software supply-chain the practical approach! (workshop)

Niels Tanis



Who am I?

- Niels Tanis
- Sr. Principal Security Researcher
  - Background .NET Development, Pentesting/ethical hacking, and software security consultancy
  - Research on static analysis for .NET apps
  - Enjoying Rust!
- Microsoft MVP – Developer Technologies

@nielstanis@infosec.exchange

Agenda

- Introduction
- DocGenerator Library
- Signing artifacts
- Reproducible Builds
- Software Bill Of Materials (SBOM)
- Google SLSA
- Docker SBOM
- I got hacked!

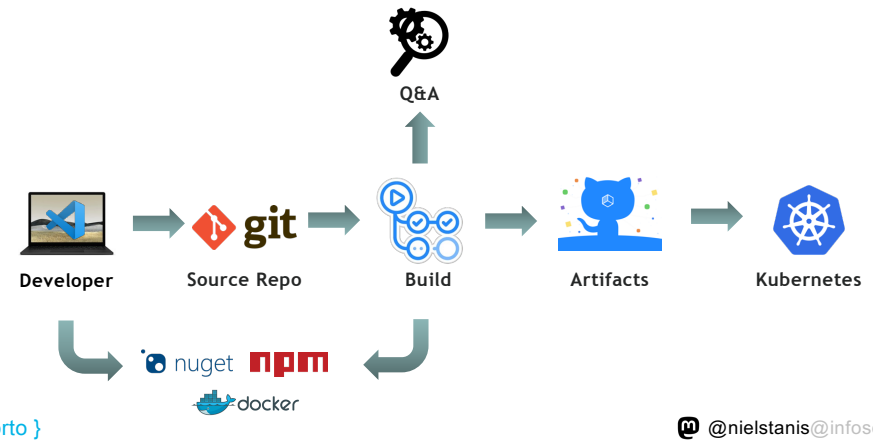
Agenda

- https://github.com/nielstanis/ndcporto2023-supply/
- https://github.com/nielstanis/docgenerator

What is a Supply Chain?

Image source:
https://www.wardsauto.com/sites/wardsauto.com/files/styles/article_featured_retina/public/Renault%20Kadjar%20assembly%20line%20-%20Palencia%20Spain-5_8.jpg?

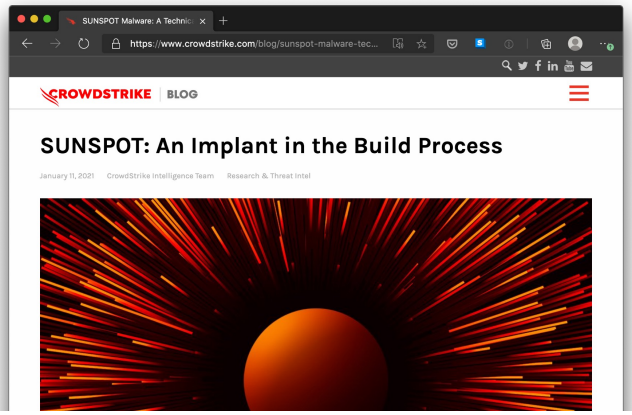
Software Supply Chain



NDC { Porto }

@nielstanis@infosec.exchange

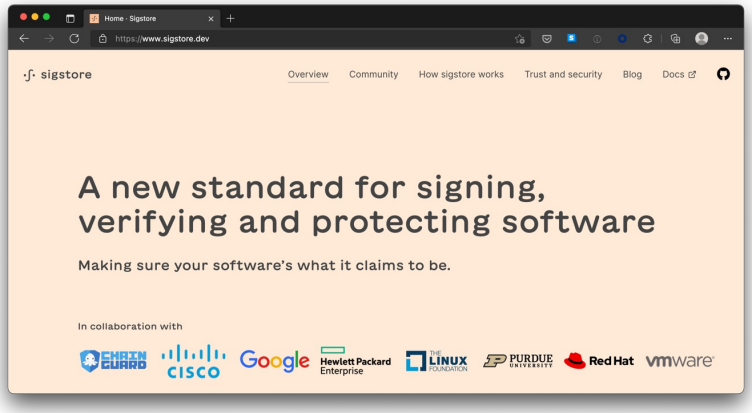
SolarWinds SunSpot



@nielstanis@infosec.exchange

https://www.crowdstrike.com/blog/sunspot-malware-technical-analysis/
https://www.microsoft.com/security/blog/2020/12/18/analyzing-solorigate-the-compromised-dll-file-that-started-a-sophisticated-cyberattack-and-how-microsoft-defender-helps-protect/

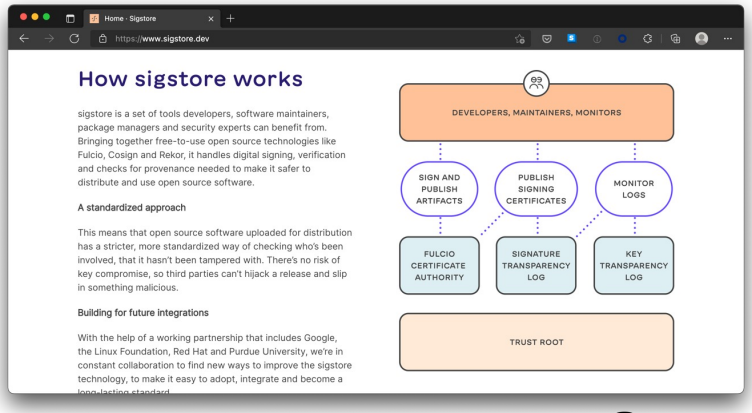
Signing artifacts



@nielstanis@infosec.exchange

https://sigstore.dev

Signing artifacts



@nielstanis@infosec.exchange

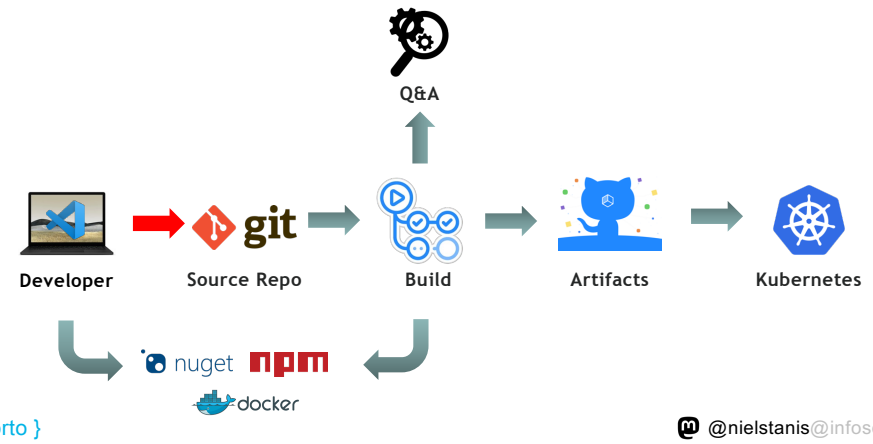
https://sigstore.dev

Lab 1

Signing artfifacts with Sigstore

Divider slide optio
please left justify t
text. Delete Sub titl
not needed. Delet
presenter if not need

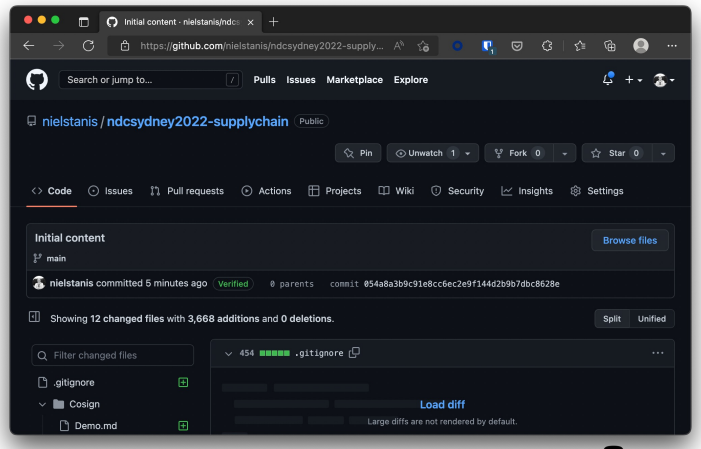
Software Supply Chain



NDC { Porto }

@nielstanis@infosec.exchange

GIT Commit Signing



NDC { Porto }

@nielstanis@infosec.exchange

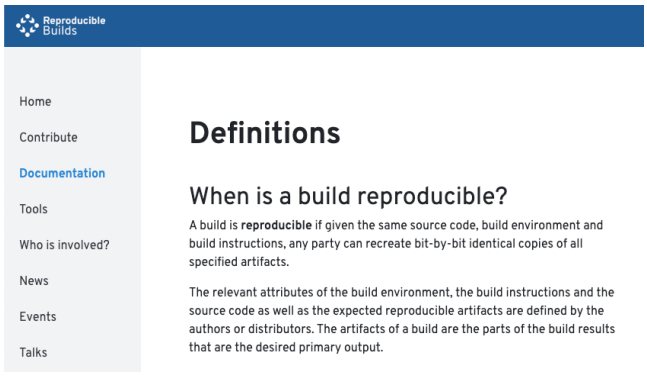
https://www.hanselman.com/blog/HowToSetupSignedGitCommitsWithAYubiKeyNEOAndGPGAndKeybaseOnWindows.aspx

Lab 2

Setup repository, build and git signing with GitSign

Divider slide optio
please left justify t
text. Delete Sub titl
not needed. Delet
presenter if not need

Reproducible/Deterministic Builds



https://reproducible-builds.org/docs/definition/

Lab 3

.NET reproducibility

Divider slide optio
please left justify t
text. Delete Sub titl
not needed. Delet
presenter if not need

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’

https://blog.paranoidcoding.com/2016/04/05/deterministic-builds-in-roslyn.html
https://github.com/dotnet/roslyn/blob/master/docs/compilers/Deterministic%20Inputs.md
https://github.com/clairernovotny/DeterministicBuilds

Reproducible/Deterministic Builds

- DotNet.ReproducibleBuilds NuGet Package
  - MSBuild *ContinuousIntegrationBuild*
  - SourceLink
- Dotnet.ReproducibleBuilds.Isolated NuGet Package
  - Hermetic builds

https://blog.paranoidcoding.com/2016/04/05/deterministic-builds-in-roslyn.html
https://github.com/dotnet/roslyn/blob/master/docs/compilers/Deterministic%20Inputs.md
https://devblogs.microsoft.com/dotnet/producing-packages-with-source-link/
https://github.com/dotnet/reproducible-builds

Lab 4

DotNet.Reproducible

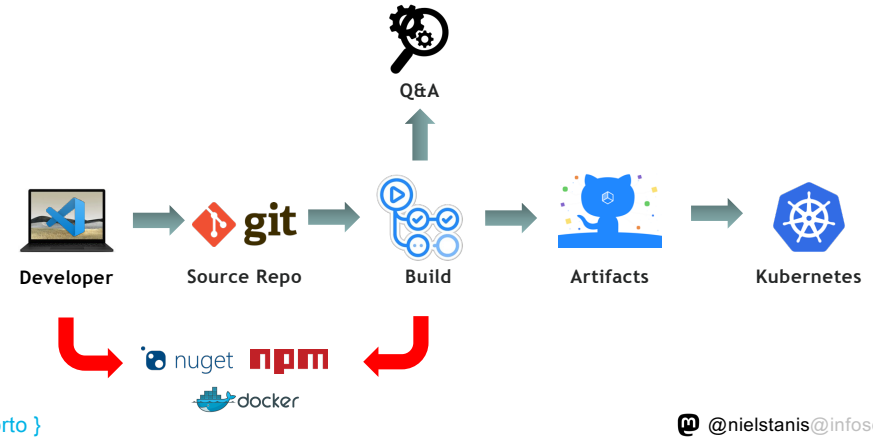
Divider slide optio
please left justify t
text. Delete Sub titl
not needed. Delet
presenter if not need

Reproducible Build Validation

- Design to validate NuGet packages & .NET binaries
  - Does linked source code match binaries?
  - Capable to compare at IL level
  - Ability to rebuild reproducible based on given inputs
  - .NET CLI Validate tool `dotnet validate`

https://github.com/dotnet/designs/blob/main/accepted/2020/reproducible-builds.md

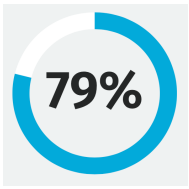
3rd Party Libraries

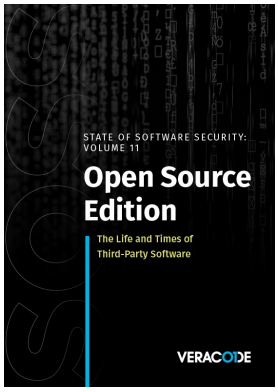


@nielstanis@infosec.exchange

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.”

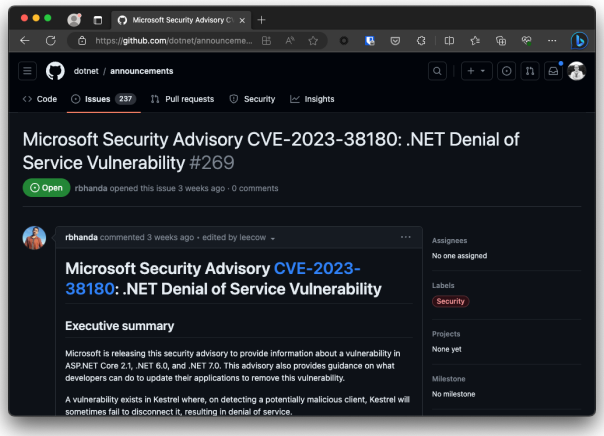




@nielstanis@infosec.exchange

https://info.veracode.com/fy22-state-of-software-security-v11-open-source-edition.html

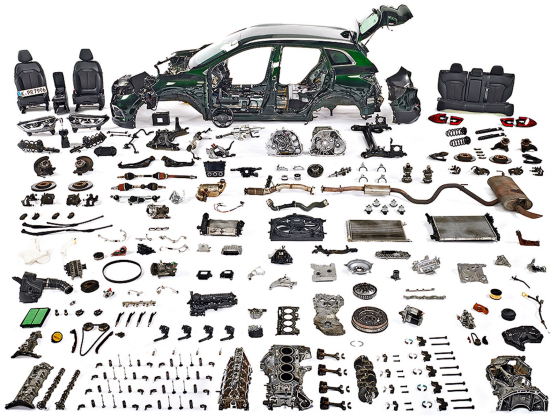
Vulnerabilities in libraries



@nielstanis@infosec.exchange

https://github.com/dotnet/announcements/issues/269

Automotive Industry



@nielstanis@infosec.exchange

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

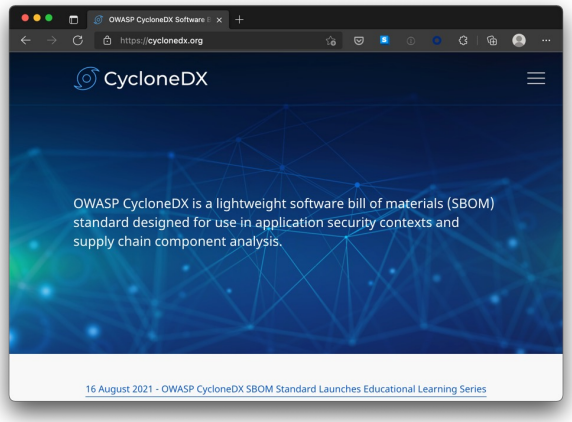
Renault Manufacturing

- 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)



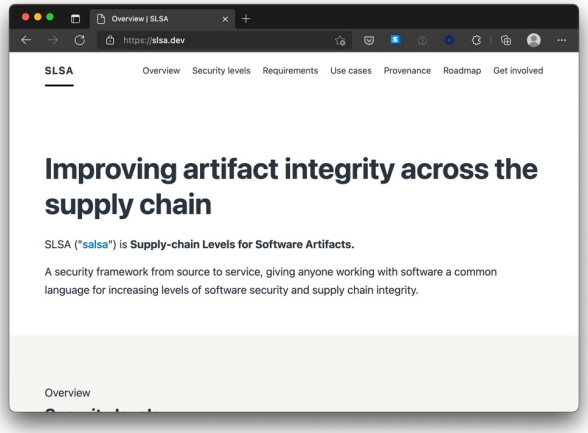
https://cyclonedx.org

Lab 5

CycloneDX .NET
Any vulnerabilities?

Divider slide optio
please left justify t
text. Delete Sub titl
not needed. Delet
presenter if not need

Google SLSA



https://slsa.dev

Google SLSA Levels

Level	Description	Example
1	Documentation of the build process	Unsigned provenance
2	Tamper resistance of the build service	Hosted source/build, signed provenance
3	Extra resistance to specific threats	Security controls on host, non-falsifiable provenance
4	Highest levels of confidence and trust	Two-party review + hermetic builds

@nielstanis@infosec.exchange

https://slsa.dev

Google SLSA Levels

1. The build process must be fully scripted/automated and generate provenance.
2. Requires using version control and a hosted build service that generates authenticated provenance.
3. The source and build platforms meet specific standards to guarantee the auditability of the source and the integrity of the provenance respectively.
4. Requires two-person review of all changes and a hermetic, reproducible build process.

https://slsa.dev

SLSA GitHub Action

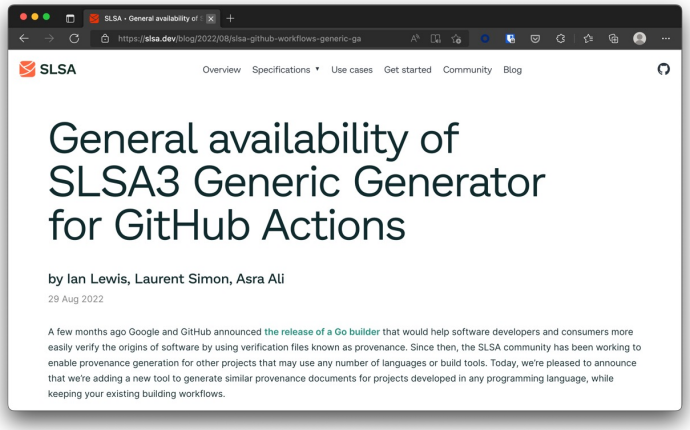
- Released April 2022

- SLSA level 2 provenance generator in GitHub Action
- SLSA level 3+ provenance generator for Go binaries

- GitHub Hosted Runner
- Uses SigStore to do keyless signing with GitHub OIDC
- Verifier included

https://security.googleblog.com/2022/04/improving-software-supply-chain.html

SLSA3 Generator GitHub Actions



https://slsa.dev/blog/2022/08/slsa-github-workflows-generic-ga

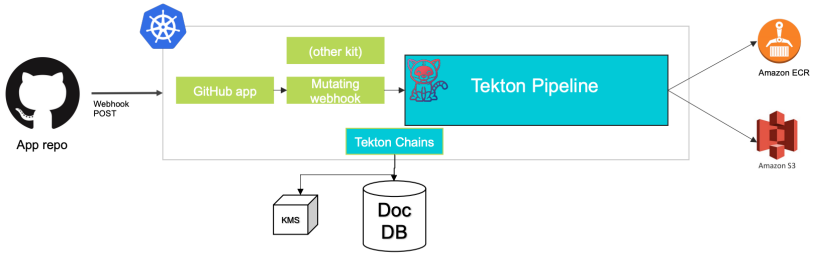
Lab 6

SLSA Level 3 Provenance

Divider slide optio
please left justify t
text. Delete Sub titl
not needed. Delet
presenter if not need

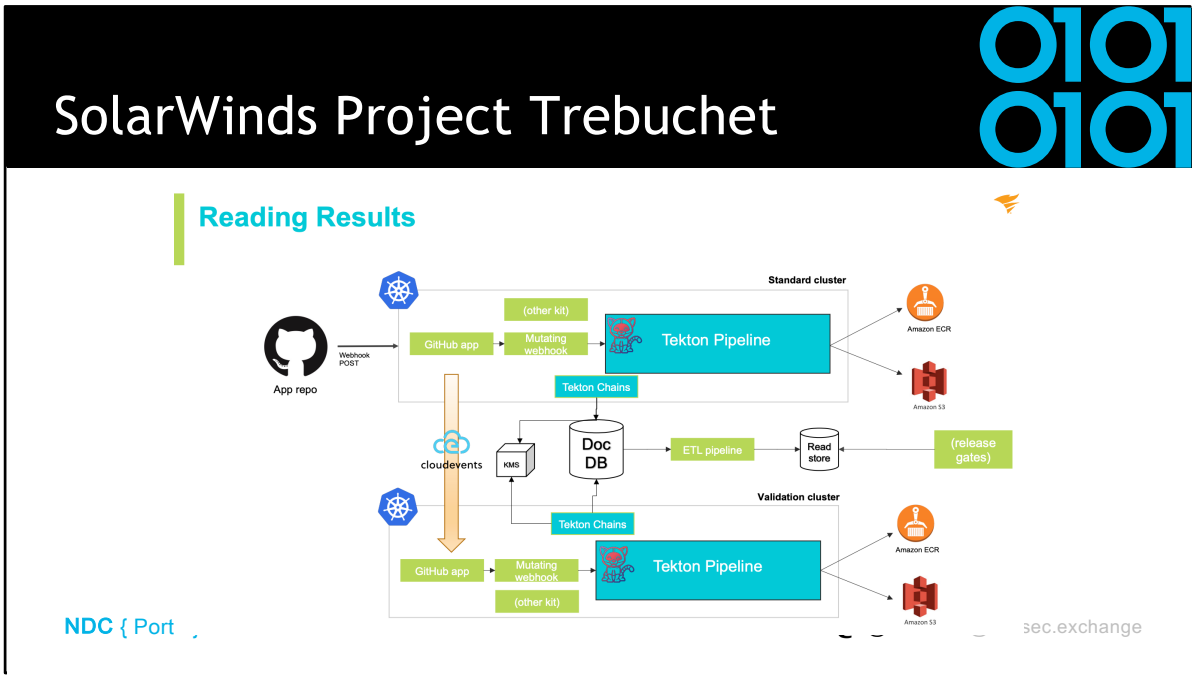
SolarWinds Project Trebuchet

Pipeline With Attestations



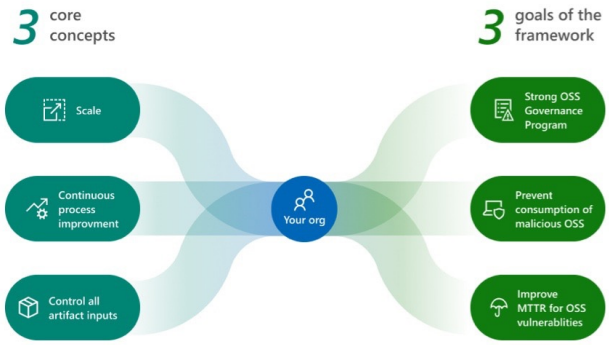
https://static.sched.com/hosted_files/supplychainsecurityconna21/df/SupplyChainCon-TrevorRosen-Keynote.pdf
https://www.youtube.com/watch?v=1-tMRxqMwTQ

SolarWinds Project Trebuchet



https://static.sched.com/hosted_files/supplychainsecurityconna21/df/SupplyChainCon-TrevorRosen-Keynote.pdf
https://www.youtube.com/watch?v=1-tMRxqMwTQ

Secure Supply Chain Consumption Framework (S2C2F)



@nielstanis@infosec.exchange

https://www.microsoft.com/en-us/securityengineering/opensource/osssscframeworkguide

Secure Supply Chain Consumption Framework Practices

1. Ingest It – I can ship any existing asset if external OSS sources are compromised or unavailable
2. Scan It - I know if any OSS artifact in my pipeline has vulnerabilities or malware
3. Inventory It - I know where OSS artifacts are deployed in production.
4. Update It - I can deploy updated external artifacts soon after an update becomes publicly available.
5. Audit It - I can prove that every OSS artifact in production has a full chain-of-custody from the original artifact source and is consumed through the official supply chain
6. Enforce It - I can rely on secure and trusted OSS consumption within my organization.
7. Rebuild It - I can rebuild from source code every OSS artifact I'm deploying.

@nielstanis@infosec.exchange

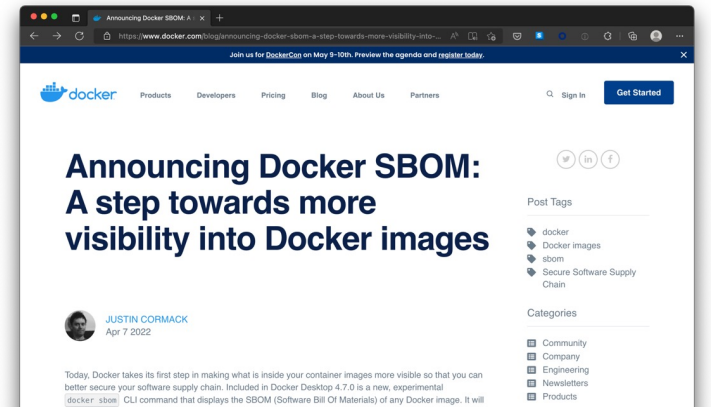
https://www.microsoft.com/en-us/securityengineering/opensource/osssscframeworkguide

Lab 7

DocGenerator on
ASP.NET Core MVC &
Docker

Divider slide optio
please left justify t
text. Delete Sub titl
not needed. Delet
presenter if not need

Docker SBOM



https://www.docker.com/blog/announcing-docker-sbom-a-step-towards-more-visibility-into-docker-images/

Lab 8

Docker SBOM with
Anchore Syft

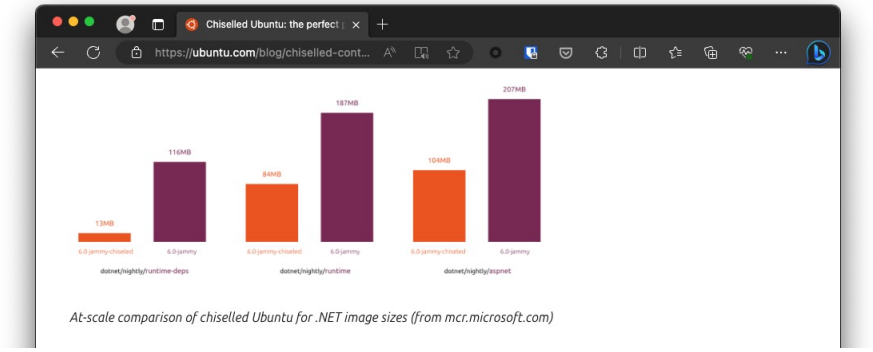
Divider slide optio
please left justify t
text. Delete Sub titl
not needed. Delet
presenter if not need

Lab 9

Cosign on Docker in GitHub Actions

Divider slide optio
please left justify t
text. Delete Sub titl
not needed. Delet
presenter if not need

Ubuntu Chiseled – Distroless Linux



At-scale comparison of chiselled Ubuntu for .NET image sizes (from mcr.microsoft.com)

@nielstanis@infosec.exchange

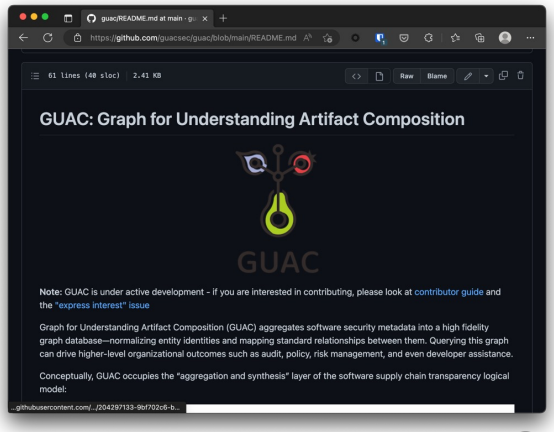
https://ubuntu.com/blog/chiselled-containers-perfect-gift-cloud-applications

Lab 10

You got HACKED!

Divider slide optio
please left justify t
text. Delete Sub titl
not needed. Delet
presenter if not need

GUAC: Graph for Understanding Artifact Composition



@nielstanis@infosec.exchange

https://github.com/guacsec/guac

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.

Conclusion

- Integrate security into your software lifecycle.
- Start working on creating SBOM's and see how SLSA can fit into your process.
- Try to work with SBOM output and use it!


VERACODE
Thanks!
https://github.com/nielstanis/ndcporto2023-supply
ntanis at veracode.com
@nielstanis@infosec.exchange on Mastodon