Demystifying Code signing and its role in DevSecOps

•••

February 7, 2023

Gaurav Kamathe

Introduction

- Product Security @ Red Hat
- Interests - Security, Linux, Malware, Emerging tech etc
- Correspondent @ opensource.com

@kamatheg

https://www.redhat.com/en/blog/channel/security
https://twitter.com/RedHatSecurity

Agenda

- SLSA
- DevSecOps
- Introduction to Code signing
- How does Code signing work
- Past challenges in implementation of Code signing
- Introduction to the sigstore project
- Why use sigstore
- Conclusion

Predictions ?

“Gartner predicts that by 2025, 45% of organizations worldwide will have experienced attacks on their software supply chains, a three-fold increase from 2021.”

https://www.gartner.com/en/articles/7-top-trends-in-cybersecurity-for-2022
https://www.sonatype.com/state-of-the-software-supply-chain/open-source-supply-demand-security

A (reasonable) rant

- Perfect is the enemy of good in Cybersecurity
- No Security silver bullet
- Mitigating even a single threat is still a win
- Make incremental progress
- **Don’t be the clown in this meme**

Credits - https://www.linkedin.com/in/danlorenc/

Supply Chain Levels for Software Artifacts (SLSA)

SLSA (pronounced salsa)

- Security framework
- Checklist of standards and controls (Specification, not a tool)
- Prevent tampering
- Improve integrity
- Secure packages and Infrastructure
- 4 incremental levels of Assurances (higher levels = more security)

https://slsa.dev/
https://security.googleblog.com/2021/06/introducing-slsa-end-to-end-framework.html

Key terminology

Term	Description	Example
Artifact	File produced as a result of a build pipeline	Container image, compiled binaries etc
Provenance	Metadata about how an artifact was built	Build process, top-level source, dependencies
Digest	Result of a cryptographic hash function	Produces fixed size value to uniquely identify artifact
Attestation	Cryptographically signed file	Provenance of build pipeline at specific time
Attestor	Any system or process that produces attestation	

Summary of 4 levels

Level	Description	Example
1	Documentation of build process	Documentation, Unsigned provenance
2	Tamper resistance of build service	Signed provenance, use of signatures
3	Extra resistance to specific threats	Non-falsifiable provenance
4	Highest level of confidence and trust	Two-party review

https://slsa.dev/spec/v0.1/levels

Requirements

Source code

- Version controlled
- Verifiable history
- Retention
- Two person reviewed
- etc

Software builds

- Automated
- Reproducible
- Built in isolated environments
- Built in hermetically sealed environments
- etc

Dependency provenance

- Authenticated by digital signature
- Generated by build service
- Non-falsifiable
- Complete list of build dependencies
- etc

https://slsa.dev/spec/v0.1/requirements

Remember him ?

How SLSA could have helped.. ?

	Threat	Known example	How SLSA could have helped
A	Submit bad code to the source repository	Linux hypocrite commits: Researcher attempted to intentionally introduce vulnerabilities into the Linux kernel via patches on the mailing list.	Two-person review caught most, but not all, of the vulnerabilities.
B	Compromise source control platform	PHP: Attacker compromised PHP's self-hosted git server and injected two malicious commits.	A better-protected source code platform would have been a much harder target for the attackers.
C	Build with official process but from code not matching source control	Webmin: Attacker modified the build infrastructure to use source files not matching source control.	A SLSA-compliant build server would have produced provenance identifying the actual sources used, allowing consumers to detect such tampering.
D	Compromise build platform	SolarWinds: Attacker compromised the build platform and installed an implant that injected malicious behavior during each build.	Higher SLSA levels require stronger security controls for the build platform, making it more difficult to compromise and gain persistence.
E	Use bad dependency (i.e. A-H, recursively)	event-stream: Attacker added an innocuous dependency and then updated the dependency to add malicious behavior. The update did not match the code submitted to GitHub (i.e. attack F).	Applying SLSA recursively to all dependencies would have prevented this particular vector, because the provenance would have indicated that it either wasn't built from a proper builder or that the source did not come from GitHub.
F	Upload an artifact that was not built by the CI/CD system	CodeCov: Attacker used leaked credentials to upload a malicious artifact to a GCS bucket, from which users download directly.	Provenance of the artifact in the GCS bucket would have shown that the artifact was not built in the expected manner from the expected source repo.
G	Compromise package repository	Attacks on Package Mirrors: Researcher ran mirrors for several popular package repositories, which could have been used to serve malicious packages.	Similar to above (F), provenance of the malicious artifacts would have shown that they were not built as expected or from the expected source repo.
H	Trick consumer into using bad package	Browserify typosquatting: Attacker uploaded a malicious package with a similar name as the original.	SLSA does not directly address this threat, but provenance linking back to source control can enable and enhance other solutions.

https://security.googleblog.com/2021/06/introducing-slsa-end-to-end-framework.html

DevSecOps

DevSecOps

- Security function within the DevOps
- Defining and implementing Security policies
- Increase Security at all levels of SDLC
- Spread security responsibilities to all stakeholders
- SAST, DAST, Threat Modeling, privacy, SCA, Code signing, etc

Introduction to Code Signing

Code signing

- Process of applying a digital signature to a software binary or file
- Ties an identity (company/person) to an artifact
- Validates the identify of software author/publisher
- Verifies that the file has not been tampered/altered
- Indicator of trust for a recipient
- Tells you about Security posture of a publisher/company

How does Code signing work ?

- Public/Private key pair, Certificate Authority (CA), Digital certificate
- **Signed Software** - Publisher's software, Code signing certificate, digital certificate
- **Code signing certificate** - Identity and public key of the publisher, CA verifies the identity
- **Digital signature** - Signed hash of software using publishers private key

https://en.wikipedia.org/wiki/Public_key_infrastructure

Why Code signing ? Benefits

- Identity of developer/publisher
- Integrity of the underlying software
- Improved user confidence, promotes trust
- Proves software is not tampered/modified
- Protect fraudulent use of brand/name

Past challenges in implementation of Code signing

Weaknesses of Code signing

- Managing security of private keys is difficult, time consuming, expensive
- Tooling has not evolved and is still arcane
- Difficult to correctly use
- Managing key rotations
- No centralized key management
- Unable to enforce Security policies consistently
- Challenges with key compromise

Introduction to sigstore

What is sigstore ?

- Linux Foundation project
- The open source software signing service (community managed public service)
- Enable widespread software signing; simple and ergonomic approach; can be adopted by project of different sizes
- Make signatures and Infra “**frictionless**” and “**invisible**”
- For open source maintainers, by open source maintainers

https://www.sigstore.dev/
https://twitter.com/decodebytes
https://twitter.com/lorenc_dan

Let's Encrypt vs sigstore

"to be software signing and provenance, what Let's encrypt is to HTTPS/SSL"

- Free certificates
- Automation tooling
- **HTTPS**

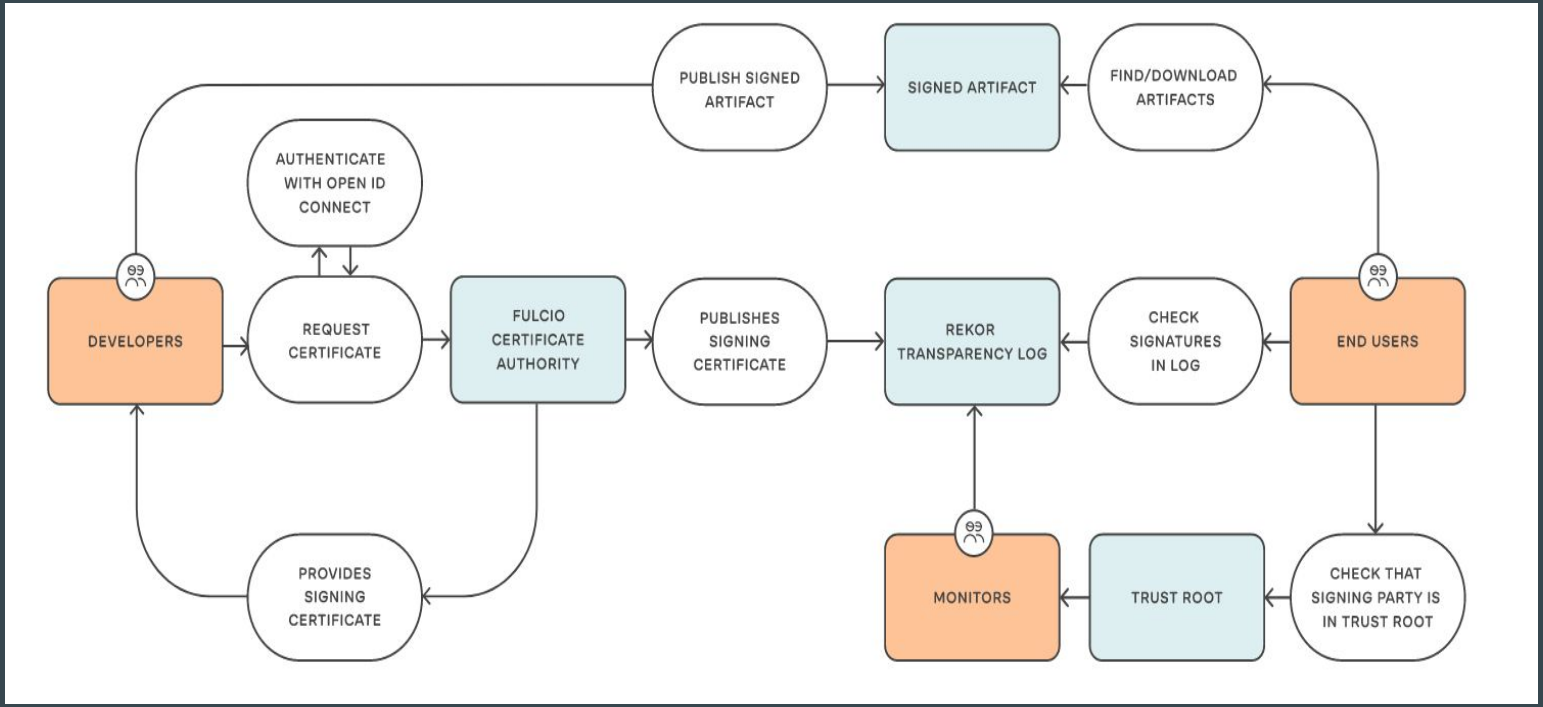
- Free certificates
- Automation tooling
- **Signatures**

sigstore components

- **Fulcio** - free root certification authority
- **Rekor** - built-in transparency and timestamping service
- **Cosign** - tool for signing/verifying containers (and other artifacts)
- **Openid connect** - identity layer

https://www.sigstore.dev/how-it-works

sigstore ecosystem



https://www.sigstore.dev/how-it-works

cosign

- Creates key pair of public and private keys
- Uses private key to create a digital signature of artifacts
- Artifacts like containers etc
- Easy for developers as identity associated with (say Github, Google)
- Which in turn avoid storing the private key

https://github.com/sigstore/cosign

fulcio

- Free to use CA for issuing code signing certificates
- Binds public keys to email addresses using OpenID connect
- Serves as a trusted third party
- Issues short-lived signing certificates
- Commits certificates to transparency log
- Consumers can verify the software artifacts

https://github.com/sigstore/fulcio

rekor

- Transparency and timestamp service for signed artifacts
- Used as storage of artifact metadata
- Immutable data log that stores signed metadata about artifacts
- Provides transparency for signatures
- Allows community to monitor and detect tampering
- Make informed decisions on trust and non-repudiation of an object's lifecycle

https://github.com/sigstore/rekor
https://github.com/google/trillian

Usage/Example ?

Sign (creates .sig, .crt, .rekor files)

- $ python -m pip install sigstore
- $ echo “demo” > cfgmgmt2023
- $ sigstore sign cfgmgmt2023 (Needs OIDC signing)

Verify

- $ sigstore verify identity cfgmgmt2023 –cert-identity <email> –cert-oidc-issuer <Github/Google/etc>

https://github.com/sigstore/sigstore-python

Why use sigstore

Why ?

- Managing keys is painful and insecure
- Makes Code signing and verification easy
- Thriving open source community
- More and more communities adopting sigstore

sigstore adoption

https://blog.sigstore.dev/kubernetes-signals-massive-adoption-of-sigstore-for-protecting-open-source-ecosystem-73a6757da73
https://www.python.org/download/sigstore/
https://internals.rust-lang.org/t/pre-rfc-using-sigstore-for-signing-and-verifying-crates/18115

Thank you

References

- https://www.sigstore.dev/
- https://github.com/sigstore
- https://training.linuxfoundation.org/training/securing-your-software-supply-chain-with-sigstore-lfs182x/
- https://openssf.org/community/sigstore/
- https://blog.trailofbits.com/2023/01/13/sigstore-python/