

# ソフトウェア署名のこれからと 今後とSigstoreと

Press Space for next page →

# はじめに

- サプライチェーンセキュリティ周りで **Sigstore** が話題
  - Linux Foundation傘下のOpenSSFにより進められている ソフトウェア署名関連 のプロジェクト
    - ソフトウェア成果物に対する署名(および検証)を推進する取り組み
    - HTTPS推進におけるLet's Encryptのような位置付け
  - 自分たちで開発しているソフトウェアをセキュアにするためにすぐに使える技術ではない(と思う)
    - ただ、Sigstoreが普及した場合、利用しているOSSの検証技術としてお世話になる可能性がある

# 背景

- ソフトウェアサプライチェーンにおいて、各構成物の来歴/出所(Provenance)を知ることは重要
  - **完全性:** 構成物を取得する過程において意図せぬ改竄等が行われていないか
  - **真正性:** 構成物の作者/配布者が意図したもの入手しているか
- 今まででは「成果物のダイジェスト」や「署名」によって担保してきた(?)



# 背景

## Maven Centralの例

Maven CentralやGitHub Releasesなどでは、利用者が配布物を検証するための情報も一緒に配布されていることがある。（任意）

- **XXXX.jar**: 配布物
- **XXXX.jar.md5**: 配布物のMD5 Checksum
- **XXXX.jar.sha1**: 配布物のSHA1 Checksum
- **XXXX.jar.asc**: 配布物の署名

## [org/apache/logging/log4j/log4j-core/2.21.1](#)

..../			
log4j-core-2.21.1-sources.jar	2023-10-20 19:49	1342740	
log4j-core-2.21.1-sources.jar.asc	2023-10-20 19:49	853	
log4j-core-2.21.1-sources.jar.md5	2023-10-20 19:49	32	
log4j-core-2.21.1-sources.jar.sha1	2023-10-20 19:49	40	
log4j-core-2.21.1.jar	2023-10-20 19:49	1895892	
log4j-core-2.21.1.jar.asc	2023-10-20 19:49	853	
log4j-core-2.21.1.jar.md5	2023-10-20 19:49	32	
log4j-core-2.21.1.jar.sha1	2023-10-20 19:49	40	
log4j-core-2.21.1.pom	2023-10-20 19:49	10939	
log4j-core-2.21.1.pom.asc	2023-10-20 19:49	853	
log4j-core-2.21.1.pom.md5	2023-10-20 19:49	32	
log4j-core-2.21.1.pom.sha1	2023-10-20 19:49	40	

図: Maven Central (log4j-core)

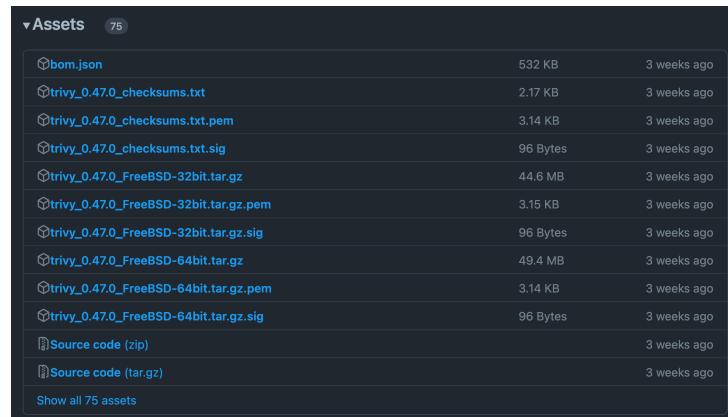


図: GitHub Release - Trivy (参考)

# 背景

Checksumの検証

- **XXXX.jar**: 配布物
- **XXXX.jar.md5**: 配布物のMD5 Checksum
- **XXXX.jar.sha1**: 配布物のSHA1 Checksum
- **XXXX.jar.asc**: 配布物の署名

---

Maven Centralから配布されている成果物が攻撃者によって書き換えられた場合に、改竄を検知できる。

→ 成果物と共にChecksumファイルも書き換えられた場合、改竄を検知できない。

```
1 $ cat log4j-core-2.21.0.jar.md5
2 1024daad23bbd97c630e8df1f73cb026
3 $ md5sum log4j-core-2.21.0.jar
4 1024daad23bbd97c630e8df1f73cb026  log4j-...
5
6 $ cat log4j-core-2.21.0.jar.sha1
7 122e1a9e0603cc9eae07b0846a6ff01f2454bc49
8 $ sha1sum log4j-core-2.21.0.jar
9 122e1a9e0603cc9eae07b0846a6ff01f2454bc49  log4j-...
```

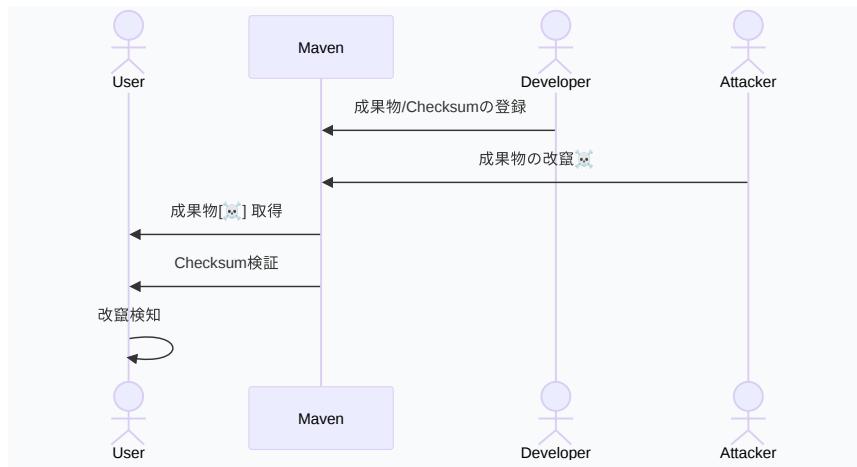


図: Checksumによる改竄検知フロー

# 背景

GPG署名の検証

- **XXXX.jar**: 配布物
- **XXXX.jar.md5**: 配布物のMD5 Checksum
- **XXXX.jar.sha1**: 配布物のSHA1 Checksum
- **XXXX.jar.asc**: 配布物の署名

---

攻撃者がMaven Centralの成果物と署名ファイルを改竄しても、公開鍵を使った検証が失敗するため、改竄を検知できる。

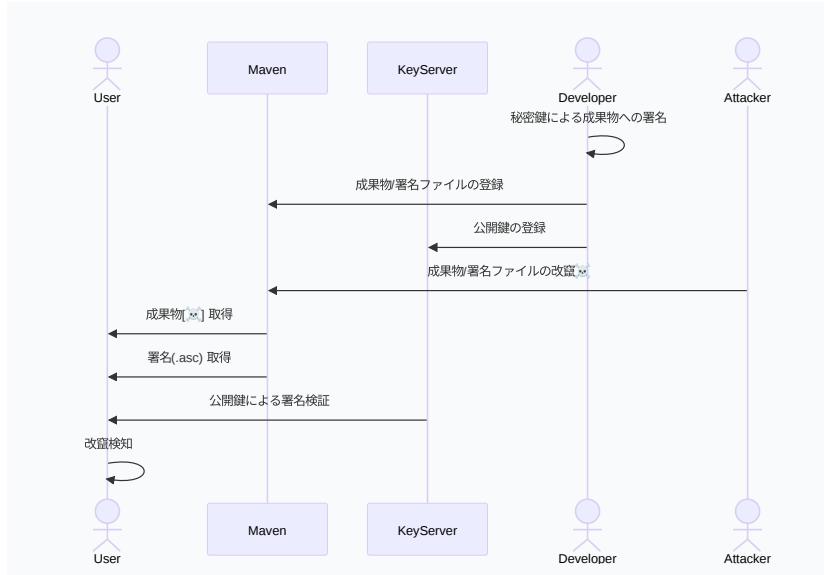


図: 署名ファイル(.asc)による改竄検知フロー

# 背景 (Maven Central)

## GPG署名の検証

### 1. 署名ファイル(.asc)と成果物だけでは署名の検証ができない

```
1 $ gpg --verify log4j-core-2.21.0.jar.asc log4j-core-2.21.0.jar
2 gpg: Signature made 2023年10月13日 00時35分31秒 JST
3 gpg:                 using RSA key 077E8893A6DCC33DD4A4D5B256E73BA9A0B592D0
4 gpg: Can't check signature: No public key
```

### 2. Keyserver (もしくは別の場所) から署名に使った秘密鍵に対する公開鍵を取得する

```
1 $ gpg --keyserver keyserver.ubuntu.com --recv-keys 077E8893A6DCC33DD4A4D5B256E73BA9A0B592D0
2 gpg: key 56E73BA9A0B592D0: public key "ASF Logging Services RM <private@logging.apache.org>" imported
3 gpg: Total number processed: 1
4 gpg:                 imported: 1
```

※ 公開鍵が信頼できるものであるか検証する仕組みは必要...

# 背景 (Maven Central)

## GPG署名の検証

### 3. Step 2で公開鍵をImportしたことでの署名の検証が成功する

```
1 $ gpg --verify log4j-core-2.21.0.jar.asc log4j-core-2.21.0.jar
2 gpg: Signature made 2023年10月13日 00時35分31秒 JST
3 gpg:           using RSA key 077E8893A6DCC33DD4A4D5B256E73BA9A0B592D0
4 gpg: Good signature from "ASF Logging Services RM <private@logging.apache.org>" [unknown]
5 gpg: WARNING: This key is not certified with a trusted signature!
6 gpg:           There is no indication that the signature belongs to the owner.
7 Primary key fingerprint: 077E 8893 A6DC C33D D4A4  D5B2 56E7 3BA9 A0B5 92D0
```

### 4. 異なるファイルでは署名の検証が失敗する

```
1 $ gpg --verify log4j-core-2.21.0.jar.asc ../trivy/trivy_0.46.0_Linux-64bit.tar.gz
2 gpg: Signature made 2023年10月13日 00時35分31秒 JST
3 gpg:           using RSA key 077E8893A6DCC33DD4A4D5B256E73BA9A0B592D0
4 gpg: BAD signature from "ASF Logging Services RM <private@logging.apache.org>" [unknown]
```

# 背景

署名による検証の課題

## Checksumによる検証

- 配布物の完全性を検証
- 真正性の検証は困難
  - 配布サーバー攻撃され、配布物とChecksumを改竄されると検知不能
- Checksum生成は容易

## 署名による検証

- 配布物・署名ファイル・公開鍵による検証
- 配布サーバーの攻撃に耐性(公開鍵検証)
  - 完全性・真正性の検証が可能
  - 秘密鍵管理が大変(鍵ファイル、パスフレーズ等)
  - 有効期限の管理も大変(期限切れ等の対応)
  - 公開鍵の配布が面倒
  - →署名プロセスの課題を解決するためのSigstore

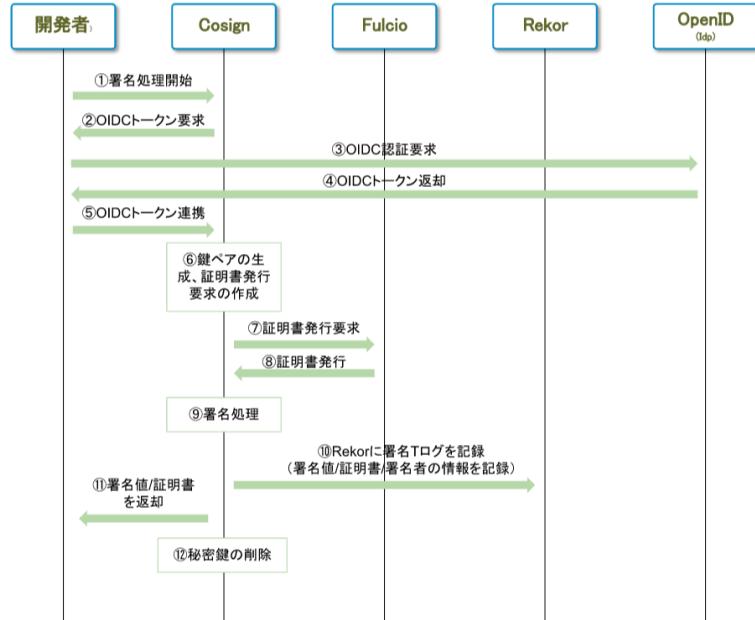
# Sigstore?

- ソフトウェア成果物に対する署名・検証・保護を提供する標準規格及びサービス
- Linux Foundation傘下のOpenSSFにより主導されているプロジェクト
  - 2021/03: Red Hat, Google, Purdue University主導でスタート
  - 2022/10: General Availability (v1.0) リリース
- 以下の3つのツール/サービスによりKeyless Signingを実現する
  - **Cosign**: ソフトウェア成果物やOCIイメージの署名及び検証を行うクライアントツール
    - Flucioと連携することでKeyless Signiningを実現可能
  - **Fulcio**: OIDC Tokenを元に署名ファイルに対する短命の証明書を生成するための認証局
  - **Rekor**: 署名プロセスの透明性を担保するために、証明書発行・署名等の情報やメタデータを保管するサービス (Transparency Logs)

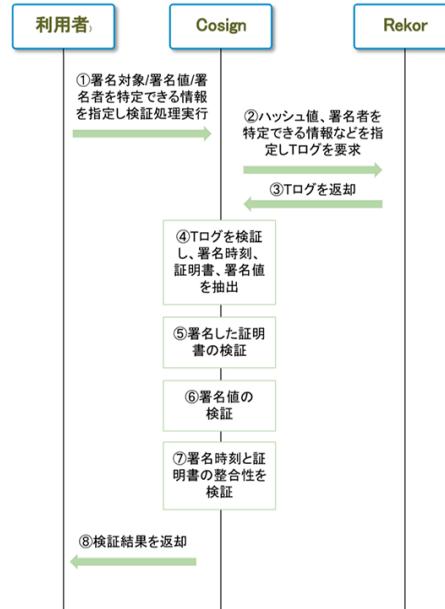
# Sigstore?

Sigstoreによる署名/検証フロー

SigstoreによるKeyless Signing



Sigstoreによる署名検証



参考: [Sigstore で OSS コード署名 | BLOG | サイバートラスト](#)

# Sigstore?

## メリット

	完全性	真正性	開発者コスト	サービス依存度
Checksum	✓	✗	✓	✓
GPG等の鍵による署名	✓	✓	✗	▲
Sigstoreによるkeyless signing	✓	✓	✓	✗

- SigstoreによるKeyless Signingにより、ソフトウェア署名の仕組みを少ない労力で導入できるようになる
- ただし、Sigstore関連の様々なサービス(`Fulcio`, `Rekor`等)に依存することになる

# Sigstore

Sigstoreの実世界での利用例

- `Cosign`によるソフトウェア署名
- `Cosign`によるDocker/OCI Imageの署名
- `npm provenance`コマンド

# Cosignによる署名

Blobファイル (sign-blob/verify-blob)

```
1 $ cosign generate-key-pair
2 # ==> `cosign.key`(秘密鍵), `cosign.pub`(公開鍵)
3 $ cosign sign-blob --key cosign.key ./log4j-core-2.21.0.jar
4 Using payload from: ./log4j/log4j-core-2.21.0.jar
5 Enter password for private key:
6
7     The sigstore service, hosted by sigstore a Series of LF Projects...
8     (意訳: 署名プロセスに関する情報が、PublicなTransparency Logsに削除不能な形で保存されるから送信情報に含まれるデータに注意な)
9
10 By typing 'y', you attest that (1) you are not submitting the personal data of any other person; and (2) you understand
11 Are you sure you would like to continue? [y/N] y
12 tlog entry created with index: 50905245
13 MEQCIH/9Fu2Nvx/LA0rIFLf4a+UdgGkGTeRcWBtPTxvZ77lNAiANC5fjf0fBiRMjtNIKLzPRXNGnWYRpWw0wl dx3l0yBRA==
```

# Cosignによる署名

Blobファイル(sign-blob/verify-blob)

The screenshot shows the Rekor Search interface for a signed blob entry. At the top, there are three icons: a purple hexagon icon, a GitHub icon, and a gear icon. Below them is the search bar with the placeholder "Rekor Search". Underneath the search bar are two input fields: "Attribute" set to "Log Index" and "Log Index" set to "50903905". To the right of these fields is a blue "SEARCH" button.

Below the search bar, the text "Showing 1 of 1" is displayed. Underneath this, the entry details are shown:

TYPE	LOG INDEX	INTEGRATED TIME
hashedrekord	<a href="#">50903905</a>	25 minutes ago (2023-11-19T10:08:35+09:00)

Under the "Hash" section, the SHA-256 hash value is listed: `sha256:d0f77cecddc269169bef40873e53a9610ba38ca1c4a1cff32f306b3a7ea8a7ea`. The "Signature" section contains a long base64 encoded string: `MEUCIA6rfjBMfn+Hvjj5UthbndusdU7Kd52EneDeIDK407amAiEawyogL3chGoH6moMMxLBxbfDHbgJeMLxH2LhgEVuc00A=`. The "Public Key" section displays a public key block:

```
-----BEGIN PUBLIC KEY-----  
MFkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDQgAEVSaCY/vhmBf1Tn+SBBmpNnZscjfn  
nKWPitzkOpLzrkYpVWxCspm5Zw2vDYUbrsU2j3zT08/AyqqX9/ErpmHaeg==  
-----END PUBLIC KEY-----
```

At the bottom, there are two expandable sections: "Raw Body" and "Verification".

# Cosignによる署名

Blobファイル(sign-blob/verify-blob)

```
1 $ cosign generate-key-pair
2 # ==> `cosign.key`(秘密鍵), `cosign.pub`(公開鍵)
3 $ cosign sign-blob --key cosign.key ./log4j-core-2.21.0.jar
4 Using payload from: ./log4j/log4j-core-2.21.0.jar
5 Enter password for private key:
6
7     The sigstore service, hosted by sigstore a Series of LF Projects...
8     (意訳: 署名プロセスに関する情報が、PublicなTransparency Logsに削除不能な形で保存されるから送信情報に含まれるデータに注意な)
9
10 By typing 'y', you attest that (1) you are not submitting the personal data of any other person; and (2) you understand
11 Are you sure you would like to continue? [y/N] y
12 tlog entry created with index: 50905245
13 MEQCIH/9Fu2Nvx/LA0rIFLf4a+UdgGkGTeRcWBtPTxvZ77lNAiANC5fjf0fBiRMjtNIKLzPRXNGnWYRpWwOwl dx3l0yBRA==
14
15 # 署名検証
16 $ cosign verify-blob --key cosign.pub --signature MEQCIH/9Fu2Nvx... ./log4j-core-2.21.0.jar
17 Verified OK
```

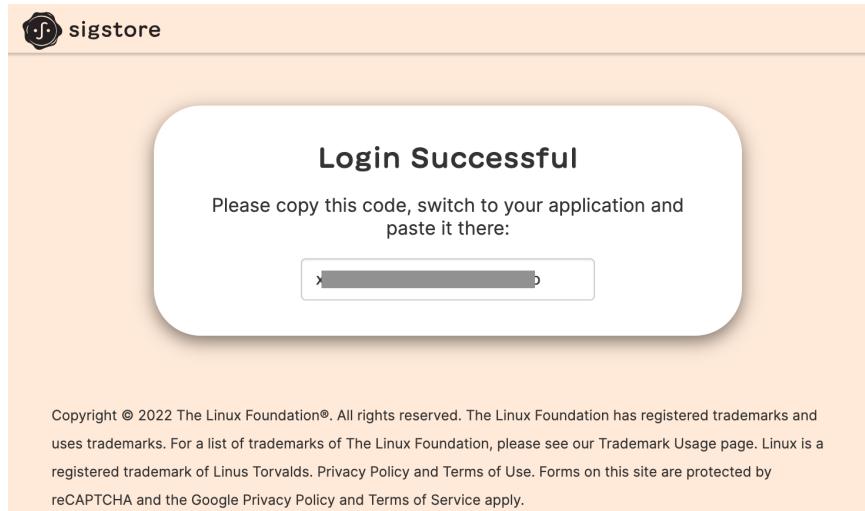
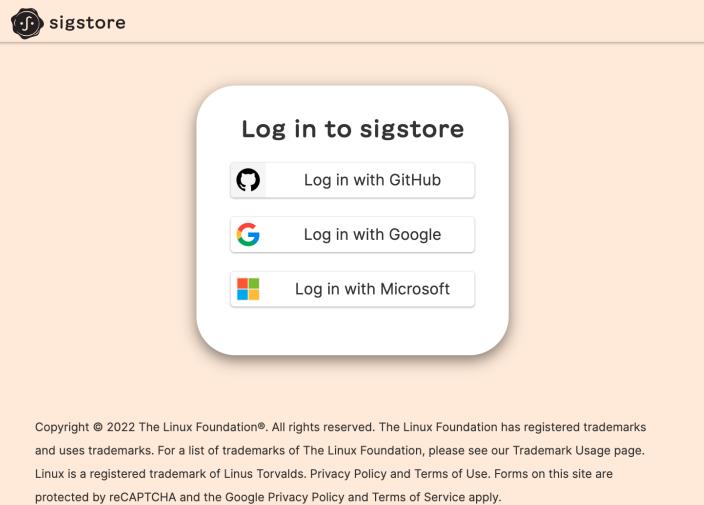
# Cosignによる署名

Docker/OCI Imageのkeyless signing

```
1 # GitHub Container Registryのイメージに署名
2 $ cosign sign ghcr.io/kaakaa/nginx@sha256:6b06964cdbbc517102ce5e0cef95152f3c6a7ef703e4057cb574539de91f72e6
3 Generating ephemeral keys...
4 Retrieving signed certificate...
5
6     The sigstore service, hosted by sigstore a Series of LF Projects...
7 (意訳: 署名プロセスに関する情報が、PublicなTransparency Logsに削除不能な形で保存されるから送信情報に含まれるデータに注意な)
8
9 Are you sure you would like to continue? [y/N] y
10 error opening browser: exit status 3
11 Go to the following link in a browser:
12     https://oauth2.sigstore.dev/auth/auth?access_type=online&client_id=sigstore&code_challenge=...
13     # Microsoft, Google, GitHubのいずれかで認証 (次ページ)
14 Enter verification code: xxxxxxxxxxxxxxxxxxxxxxxxx
15
16 Successfully verified SCT...
17 tlog entry created with index: 12345678
18 Pushing signature to: ghcr.io/kaakaa/nginx
```

# Cosignによる署名

OCI Image



# Cosignによる署名

Docker/OCI Imageのkeyless signing

```
1 # GitHub Container Registryのイメージに署名
2 $ cosign sign ghcr.io/kaakaa/nginx@sha256:6b06964cdbbc517102ce5e0cef95152f3c6a7ef703e4057cb574539de91f72e6
3 Generating ephemeral keys...
4 Retrieving signed certificate...
5
6     The sigstore service, hosted by sigstore a Series of LF Projects...
7     (意訳: 後から削除できないPublicなTransparency Logsに保存されるから送信情報に含まれるデータに注意な)
8
9 Are you sure you would like to continue? [y/N] y
10 error opening browser: exit status 3
11 Go to the following link in a browser:
12     https://oauth2.sigstore.dev/auth/auth?access_type=online&client_id=sigstore&code_challenge=...
13     # Microsoft, Google, GitHubのいずれかで認証 (次ページ)
14 Enter verification code: xxxxxxxxxxxxxxxxxxxxxxxxx
15
16 Successfully verified SCT...          # SCT = Signed Certificate Timestamp
17 tlog entry created with index: 12345678 # Rekorに格納されたTransparency LogのID
18 Pushing signature to: ghcr.io/kaakaa/nginx # 署名情報はOCI Registryに格納される(次ページ参照)
```

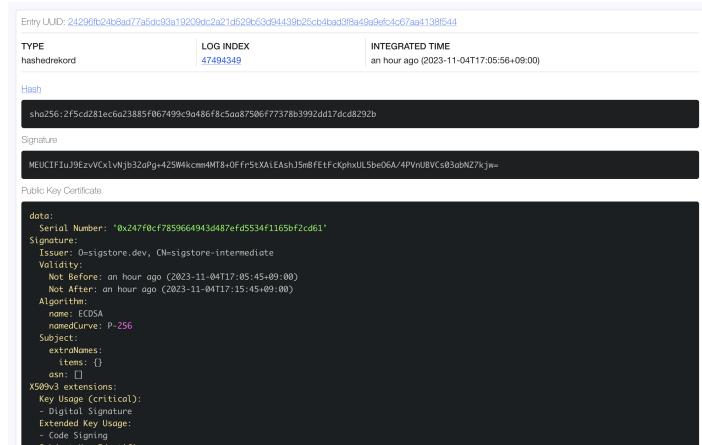
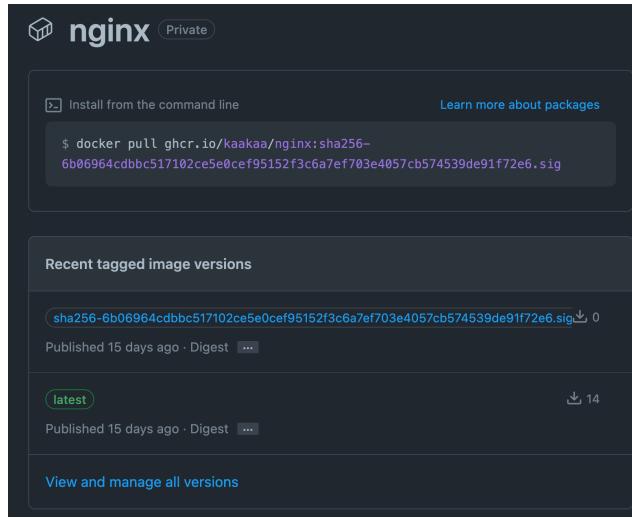
# Cosignによる署名

Docker/OCI Imageのkeyless signing

- 署名情報がOCI RegistryにOCI Artifactとして登録
  - tag: `sha256-\${IMAGE\_DIGEST}.sig`

参考: GitHub Actions で distroless イメージのコンテナ署名を検証する

- RekorにTransparency Logが追加される
  - Rekor Search



# Cosignによる署名

Docker/OCI Imageのkeyless signing

「鍵による署名」と「Keyless Signing」でtlogの公開鍵部分が異なる

Entry UUID: 24296fb24b8ad7745a43738005fb0a9afc11298d9bbfae396cc2a0bea9d68aae401fe5dec343

TYPE	LOG INDEX	INTEGRATED TIME
hashekrekord	50903905	25 minutes ago (2023-11-19T10:08:35+09:00)

Hash

```
sha256:d0f77cecdcc260160bef40873e53a9610ea38ca1c4a1cff32f380b3a7ea8a7ea
```

Signature

```
MEUCIAerfjBMfn+HvJ5UthbnusdU7Kd52EneDk407amA1EwyogL3chGef6m0MxLBxfDhbgJeMLxH2LhgEvUc00A=
```

Public Key

```
-----BEGIN PUBLIC KEY-----MFkwEwYKIZyZj9GMQYTKzIzj80A0QcDqgAEV5aC/YvhMBf1Tr5SBmpNzscJfnnKMr1tzK0plzrKpWkCjPm5zN2vDyubrsUzjSz108/AyqqK9/Erpmlaeq==-----END PUBLIC KEY-----
```

Raw Body

Verification

Entry UUID: 24296fb24b8ad7745a43738005fb0a9afc11298d9bbfae396cc2a0bea9d68aae401fe5dec343

TYPE	LOG INDEX	INTEGRATED TIME
hashekrekord	47494349	15 days ago (2023-11-04T17:05:56+09:00)

Hash

```
sha256:f5cd281ec6a23885f067490c9a486f8c5aa87586f77378b3992dd17dcdb292b
```

Signature

```
MEUCIfIuJ9EzvVcx1vNjb32aPg+25W46cmw4MT8+Offr5tXAlEashJ5n6fEtFcpxhUL5be06A/4PVnUBVcs03abNz7kjw=
```

Public Key Certificate

```
-----  
data:  
  Serial Number: 0x247f0cf7853664943d487efd5534f1165bf2cd61'  
Signature:  
  Issuer: <sigstore.dev, CN=sigstore-intermediate  
  Validity:  
    Not Before: 15 days ago (2023-11-04T17:05:45+09:00)  
    Not After: 15 days ago (2023-11-04T17:15:45+09:00)  
  Algorithm:  
    name: EDDSA  
    namedCurve: P-256  
  Subject:  
    extraNames:  
      items: {}  
    asn: []  
  X509v3 extensions:  
    Key Usage (critical):  
      - Digital Signature  
    Extended Key Usage:  
      - Code Signing  
    Subject Alternative Name:  
      - Authority Key Identifier:  
        keyid: DF:D3:E9:CF:56:24:11:96:19:A8:D8:E9:28:55:A2:C6:2E:18:64:3F  
    Subject Alternative Name (critical):  
      email:  
        - stooner.hoe@gmail.com  
    OIDC Issuer: https://github.com/login/oauth  
    OIDC Issuer (v2): https://github.com/login/oauth  
-----  
Raw Body  
Verification
```

図: tlog比較 (左:sign-blob / 右:sign)

# Cosignによる署名

Docker/OCI Imageのkeyless signing

```
1 $ cosign verify \
2   --certificate-oidc-issuer https://github.com/login/oauth \ # 署名を使ったIDP
3   --certificate-identity example@gmail.com \           # 署名者のOIDCアカウント
4   ghcr.io/kaakaa/nginx@sha256:6b06964cdbbc517102ce5e0cef95152f3c6a7ef703e4057cb574539de91f72e6
5
6 Verification for ghcr.io/kaakaa/nginx@sha256:6b06964cdbbc517102ce5e0cef95152f3c6a7ef703e4057cb574539de91f72e6 --
7 # OCI Registryに格納された署名情報が正しいこと(改竄されていないこと)を検証
8 The following checks were performed on each of these signatures:
9   - The cosign claims were validated
10  - Existence of the claims in the transparency log was verified offline
11  - The code-signing certificate was verified using trusted certificate authority certificates
12 [
13   "critical": {
14     "identity": {"docker-reference": "ghcr.io/kaakaa/nginx"},
15     # 注意: 検証済みの署名情報に格納されたImage Digestと手元のOCI Imageが同一であることを追加で検証する必要がある
16     "image": {"docker-manifest-digest": "sha256:6b06964cdbbc517102ce5e0cef95152f3c6a7ef703e4057cb574539de91f72e6"},
17     "type": "cosign container image signature"
18   },
19   ...
20 ]
```

# npm provenance

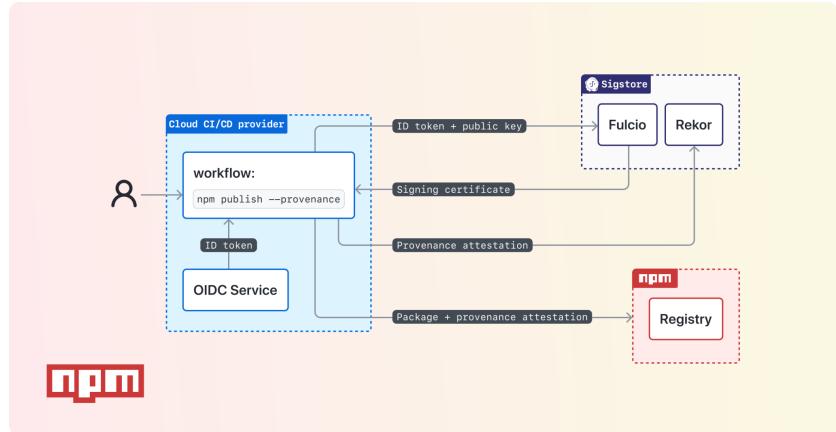
## NPMモジュールの来歴情報登録

- 2023/09/26 にGAとなった`npm`の機能
  - 2023/09/26: npm provenance general availability - The GitHub Blog
  - (2023/04/19: Public Beta)

```
$ npm publish --provenance
```

**npm publish**コマンドでパッケージを公開する際にSigstoreによる署名プロセスを実行するオプション。GitHub Actions等のCIと連携することで、ソースコード/ビルドの情報も併せて公開することができる。

→ ソフトウェアを検証するための情報(署名情報)だけでなく、**成果物を生成するプロセス(ソース/ビルド情報)**の情報も公開することができる。



参考: npmパッケージプロベナスを導入 - GitHub ブログ

### Provenance

Built and signed on GitHub Actions	Source Commit Build File Public Ledger	<a href="https://github.com/kaaka/sliderv-addon-rabbit@ffacd60">github.com/kaaka/sliderv-addon-rabbit@ffacd60</a> <a href=".github/workflows/npm-publish.yml">.github/workflows/npm-publish.yml</a> <a href="#">Transparency log entry</a>
---------------------------------------	----------------------------------------------	--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

[Share feedback](#)

参考: sliderv-addon-rabbit - npm

# npm provenance

NPMモジュールの来歴情報登録

```
1  publish-npm:
2    needs: build
3    runs-on: ubuntu-latest
4    permissions:
5      contents: read
6      id-token: write # Sigstoreによる署名のために必要
7    steps:
8      - uses: actions/checkout@v3
9      - uses: actions/setup-node@v3
10     with:
11       node-version: '18.x'
12       registry-url: https://registry.npmjs.org/
13     - run: npm install -g npm
14     - run: npm ci
15     - run: npm publish --provenance --access public # provenanceオプションをつけて実行するだけ
16     env:
17       NODE_AUTH_TOKEN: ${{secrets.npm_token}}
```

Sigstoreにより数行の記述追加のみでソフトウェア署名が実現可能になった。  
ただし、アカウント管理の重要性は増大。

ref: [Top-100 npm package maintainers now require 2FA, and additional security-focused improvements to npm - The GitHub Blog \(2022/2/1\)](#)

# Roadmap

- OSSパッケージマネージャーへの取り込み (Homebrew, PyPI, Mavenなど)
- プライベート環境へのデプロイ手順の簡略化
- アカデミアとのコラボレーション (署名プロセス監視など)
- シームレスな検証プロセス

# Wrap up

- Sigstoreによりソフトウェア署名を運用するまでの鍵管理が不要になった
  - OIDC Tokenベースとなるため、CI等への組み込みが非常に容易となる(gh action/npm provenance等)
    - 現状、npmが先行しているが、他パッケージマネージャー対応(Python, Maven)も進行中
    - [Sigstore: Simplifying Code Signing for Open Source Ecosystems - Open Source Security Foundation](#)
  - OSS開発者がソフトウェア署名を導入しやすくなり、社会全体としてのセキュリティ底上げが期待される
    - HTTPS推進におけるLet's Encryptのような存在
- 個人管理の鍵ファイルから、SaaS(Fulcio, Rekor)依存で、信頼性が向上するかはユースケース次第
  - 「署名を実施されたこと」自体を公開したくない場合(BtoB等)は、Public SaaSを利用できない
    - [Fulcio, Rekor](#)はOSSで開発されているが、これらをセルフホストするなら鍵共有の方が楽そう
    - Privateでの運用もターゲットにはなっている(ref. [Roadmap](#))

# 感想

- 即効性のあるセキュリティ対策ではないが、業界全体の動向としては押さえておいた方が良さそう
  - npmなどは何も考えず`--provenance`オプション指定で良い気がする
  - OSS利用者側(検証側)としてどう動けばいいかはまだよくわからっていない
    - 検証物ごとにアカウント情報を一つ一つ指定するのは現実的ではない
    - Container Image単位であれば理解はできるかも
- 各Registryごとに異なっていた署名管理/検証プロセスがcosignを通じて統一されると嬉しそう

# 参考

- [Sigstore で OSS コード署名 | BLOG | サイバートラスト](#)
  - Cosignによる署名/検証フローについて
- [sigstoreのKeyless Signingでは何を検証しているのか - sometimes I laugh](#)
  - Sigstoreによって防ぐことができる攻撃について
- [GitHub Actions で distroless イメージのコンテナ署名を検証する - ISID テックブログ](#)
  - OCI Registryに格納される署名情報がどのようなものか
- MISC
  - [sigstoreによるコンテナイメージやソフトウェアの署名 - knqyf263's blog](#)
  - [Safeguard your containers with new container signing capability in GitHub Actions - The GitHub Blog](#)
  - [ソフトウェアサプライチェーンセキュリティのための GitHub Actions ワークフロー | 豆蔵デベロッパーサイト](#)
  - [kubernetes 1.24 から提供コンテナイメージが Cosign で署名されるようになったので検証してみよう | by makocchi | Medium](#)