



Chainguard.

Software Supply Chain Security
Assessment

Cosmos

Jul 14, 2022

Version	Author	Date	Notes
0.0.1	James Strong	May 16, 2022	Template Creation
0.0.2	Jason Lutz	May 16, 2022	Template Modification
0.0.3	Jason Lutz	June 23, 2022	Engagement Started w/ Updates to Document
0.1	Jason Lutz	July 14, 2022	Release to client.

Table Of Contents

Table Of Contents	2
Chainguard Company Overview	6
Executive Summary	6
Synopsis	6
Summary of Repository Observations	7
Summary of Image Observations	7
Software Supply Chain Security Background	9
Using ko to harden the build process for cosmos/gaia	11
Distroless images	11
ko	11
Conversion process	12
SLSA/OpenSSF Scorecards Overview	13
Repository Reviews	13
Introduction	13
1. strangelove-ventures/heighliner	13
Overview	13
Observations and Recommendations	14
Make Build System Transparent	14
Update Base Images Used in Builds	14
Sign Images	15
Add SBOMs to Releases	15
Enforce Code Reviews	16
Consider Pinning Builder Images	16
Verify Downloads in Images	17
Use Minimal Production Images	17
Scan Images for Vulnerabilities	18
Automated SLSA/OpenSSF Review	18
Manual SLSA Review	22
Source Requirements	22
Version Controlled (Required for: L1, L2, L3, L4)	22
Verified History (Required for: L3, L4)	22
Retained Indefinitely (Required for: L3 [for 18.mos], L4)	22
Two-person Reviewed (Required for: L4)	22
Build Requirements	23
Scripted Build (Required for: L1, L2, L3, L4)	23

Build Service (Required for: L2, L3, L4)	23
Build as Code (Required for: L3, L4)	23
Ephemeral Environment (Required for: L3, L4)	23
Isolated (Required for: L3, L4)	23
Parameterless (Required for: L4)	23
Hermetic (Required for: L4)	23
Reproducible (Required for: L4)	23
Provenance Requirements	24
Available (Required for: L1, L2, L3, L4)	24
Authenticated (Required for: L2, L3, L4)	24
Service Generated (Required for: L2, L3, L4)	24
Non-falsifiable (Required for: L3, L4)	24
Dependencies Complete (Required for: L4)	24
Common Requirements	24
Security (Required for: L4)	24
Access (Required for: L4)	24
Superusers (Required for: L4)	24
2. Cosmos/Gaia Repo	26
Overview	26
Observations and Recommendations	26
Sign Images	26
Releases Should Produce an SBOM & SLSA Provenance Attestation	26
Artifacts Should be Signed	27
Consider Using distroless.dev/static Base Image	27
Scan Images for Vulnerabilities	27
Fix the Container Image Build	28
Automated SLSA/OpenSSF Review	28
Manual SLSA Review	31
Source Requirements	31
Version Controlled (Required for: L1, L2, L3, L4)	31
Verified History (Required for: L3, L4)	32
Retained Indefinitely (Required for: L3 [for 18.mos], L4)	32
Two-person Reviewed (Required for: L4)	32
Build Requirements	32
Scripted Build (Required for: L1, L2, L3, L4)	32
Build Service (Required for: L2, L3, L4)	32
Build as Code (Required for: L3, L4)	32
Ephemeral Environment (Required for: L3, L4)	32
Isolated (Required for: L3, L4)	32
Parameterless (Required for: L4)	32
Hermetic (Required for: L4)	32

Reproducible (Required for: L4)	33
Provenance Requirements	33
Available (Required for: L1, L2, L3, L4)	33
Authenticated (Required for: L2, L3, L4)	33
Service Generated (Required for: L2, L3, L4)	33
Non-falsifiable (Required for: L3, L4)	33
Dependencies Complete (Required for: L4)	33
Common Requirements	33
Security (Required for: L4)	33
Access (Required for: L4)	33
Superusers (Required for: L4)	33
3. Cosmos/Cosmos-SDK	36
Overview	36
Observations and Recommendations	36
SDK Signing	36
SDK Archive: Provide SBOM to Users	36
SDK is Installing Dependencies via Curl without Verification	36
Automated SLSA/OpenSSF Review	37
Manual SLSA Review	41
Source Requirements	41
Version Controlled (Required for: L1, L2, L3, L4)	41
Verified History (Required for: L3, L4)	41
Retained Indefinitely (Required for: L3 [for 18.mos], L4)	41
Two-person Reviewed (Required for: L4)	41
Build Requirements	42
Scripted Build (Required for: L1, L2, L3, L4)	42
Build Service (Required for: L2, L3, L4)	42
Build as Code (Required for: L3, L4)	42
Ephemeral Environment (Required for: L3, L4)	42
Isolated (Required for: L3, L4)	42
Parameterless (Required for: L4)	42
Hermetic (Required for: L4)	42
Reproducible (Required for: L4)	42
Provenance Requirements	42
Available (Required for: L1, L2, L3, L4)	42
Authenticated (Required for: L2, L3, L4)	43
Service Generated (Required for: L2, L3, L4)	43
Non-falsifiable (Required for: L3, L4)	43
Dependencies Complete (Required for: L4)	43
Common Requirements	43
Security (Required for: L4)	43

Access (Required for: L4)	43
Superusers (Required for: L4)	43
Github Actions Signing, Hardening and Event Monitoring	45
GitHub Actions (Build) Key Recommendations	45
Appendix:	47
A.4 Checks for Hardening GitHub Actions	48
A.5 Events to Monitor for Hardening GitHub Actions	50
A.6 Image Scans	55
Sources	59
Related Reading	59

Chainguard Company Overview

Chainguard is the industry's premiere Software Supply Chain leader with a mission to make the software supply chain secure by default. Our teams consist of the brightest minds in the industry with experience across containers & orchestration, cloud computing, information security, DevOps and all things software supply chain. We have a strong commitment to building and scaling secure open-source technologies across the cloud-native landscape.

Executive Summary

Synopsis

Orijtech enlisted Chainguard to engage with Cosmos to validate the Chainguard Image product and prove that Chainguard images reduce vulnerability findings, improve SLSA compliance and help to implement best practices prescribed by the National Institute of Standards and Technology (NIST) Secure Software Development Framework (SSDF) and Open Source Security Foundation (OpenSSF).

The scope of the assessment included three Orjitech/Cosmos related repos:

- cosmos/cosmos-csk
- cosmos/gaia
- strangelove-ventures/heighliner

This assessment resulted in 18 observations with a focus on signing artifacts. SLSA framework observations are also provided in individual tables to provide guidance on achieving the four levels of compliance. The assessment of those repos and build systems provides recommendations that can be implemented immediately to strengthen the software supply chain security posture and help safeguard the consumers of both the Comos & Strangelove-ventures builds. We also recommend the docker images over binaries that are available to end users as an additional measure of security.

The use of Chainguard images and implementation of recommendations around build processes provide for an improvement of Cosmos client security and security posture of the builds. Integration of Chainguard recommended images included this assessment. This will result in increased provenance of artifacts produced in the image builds. Run-time compute and container orchestration is out of scope as the repos do not deploy workloads.

Lastly, the use of Github Actions is a step forward in achieving maturing SLSA levels. The final piece of this assessment is an overview of how configurations and hardening of Github Actions can help produce a more comprehensive security posture while providing end users attestations which, in turn, provides provenance of artifacts. As noted Heighliner does not use an in-toto compliance build process (it appears to be a script executed by cron job).

Summary of Repository Observations

Repository	High Risk/Notable Observations
strangelove-ventures/heighliner	Lack of visibility into how images are built. No scanning of images for vulnerabilities. No Software Bill of Materials (SBOMs) or signatures for built images. Dockerfile downloads unverified artifacts. Out-of-date base images with known vulnerabilities are being used in builds.
cosmos/cosmos-sdk	No scanning of images for vulnerabilities. No Software Bill of Materials (SBOMs) or signatures for built images.
cosmos/gaia	Lack of verification of base images No scanning of images for vulnerabilities. No Software Bill of Materials (SBOMs) or signatures for built images.

Summary of Image Observations

- The `interchainio/simapp` image is based on `alpine:edge` and the vulnerabilities were present in the base image. They could be addressed by running `apk update` in the Dockerfile, or moving to a more regularly updated base image like `distroless.dev/alpine-base`.
- We expected to find the `simapp` image at `cosmos-sdk/simapp`. However this repo appears abandoned in favor of `interchainio`. This opens the project to a potential squatting attack if a malicious actor gains control of the `cosmos-sdk` repo, or points users to out-of-date images on the existing repo.
- The 15 vulnerabilities in most of the `heighliner` images is caused by use of an out-of-date base image. Changing the build system to pull the image everyday, or moving back to GitHub actions would fix this issue.
- The `cosmos/gaia` image build hasn't run successfully for 3 months, resulting in an out-of-date image.

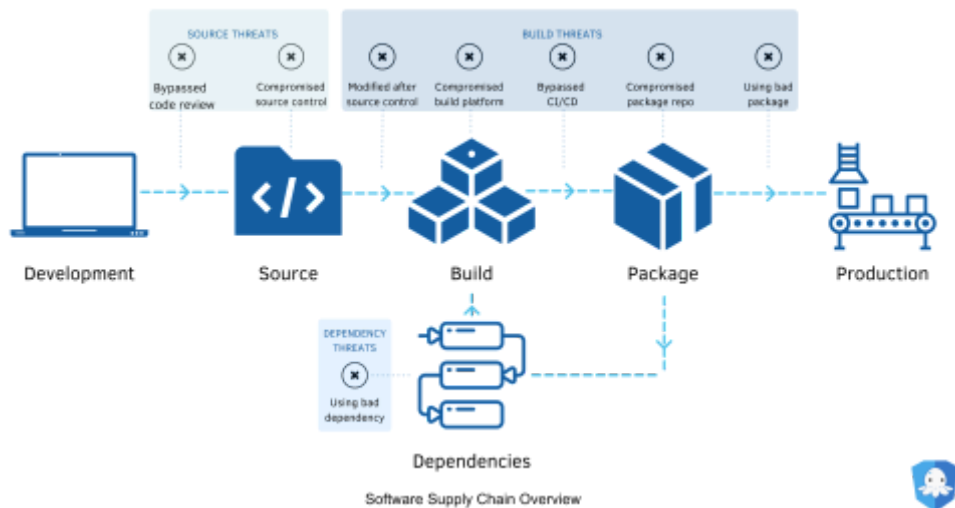
- There are 2 images - polkadot and penumbra - that have a separate build process based on Rust that results in 132 vulnerabilities. The base image used here is debian:bullseye, which is a relatively large image and again out-of-date. We strongly recommend moving these to a smaller base image.

Software Supply Chain Security Background

Software supply chain security compromises have been growing in frequency over the past decade. In 2021, the popular logging library log4j, used by more than 35,000 Java packages, was found to be susceptible to a remote code execution vulnerability, CVE-2021-44228, more commonly known as log4shell. This CVE set off a flurry of activities inside organizations to understand if they were vulnerable to this specific exploit. In March 2021, someone snuck a backdoor to gain unauthorized access to the open-source programming language PHP's source code. These are just a few of the high-profile issues that software supply chains face. This report reviews the current state of Cosmos' software supply security and security posture.

The software development life cycle has become increasingly complex. One way to deal with that complexity is for developers to rely more and more on open source software as part of the build product. This reliance on third party components has opened an attack vector for malicious actors as well as introducing unintentional vulnerabilities. In 2020 alone, there was a 430% growth in next-generation cyber-attacks that actively targeted open-source software projects¹; compromising open source software components in the software supply chain that downstream projects depend on.

Software build systems are also under attack providing another attack vector for malicious actors (Enck, 2020). This constant threat was notably highlighted by the prolific Sunburst attack of 2020. This attack compromised the Solarwinds build process to inject malicious code into their IT monitoring system, distributed to downstream customers unbeknownst to Solarwinds. There are many vulnerable points in the software supply chain of an organization, and any good defensive strategy requires diligence, defense in depth, and observability of the entirety of the software supply chain.



Software supply chains and the processes involved can be conceptually divided into five phases:

- Development: The act of writing software
- Source: Artifact created via software development and related processes.
- Build: Code, technologies, and processes that transform source code into a usable form.
- Package - The results of the build process.
- Production: The operation and maintenance of code artifacts consumed by end users.

Each of these links in the software supply chain are vulnerable and have been exploited. Fortunately, there are a number of ways to prevent, detect, and remediate software supply chain compromises. A software supply chain security audit is an important start. Other key principles include using known good components produced with security in mind (such as Chainguard Images) and creating and enforcing software supply chain security policies about the software an organization deploys to production (such as Chainguard Enforce).

Using ko to harden the build process for cosmos/gaiad

The primary piece of software in the Cosmos Network ecosystem is Cosmos Hub, also called gaiad. The code for gaiad is hosted on GitHub at <https://github.com/cosmos/gaiad>, and we propose to convert this repository to use `ko` and distroless images as an example of how elements of Chainguard's secure software factory initiative can be useful for hardening the build process for Cosmos Network images.

Chainguard strongly recommends adoption of these secure software factory elements – namely ko and distroless runtime images – across the entire Cosmos Network ecosystem to improve the build processes and provenance of the images they provide to their downstream user community.

Distroless images

First implemented by the Google Container Tools team, distroless images are images which contain only the minimum set of dependencies necessary to support an application. As an evolution of this concept, Chainguard maintains the Distroless project, a reimaged take on so-called distroless image building, built on a pair of componentized Linux distributions: Alpine and Inky, a companion GNU/Linux distribution to Alpine that uses the same tooling but maintains basic compatibility with CentOS Stream.

Chainguard recommends the use of distroless runtime images to support Cosmos Network applications, including and especially Cosmos Network validators. Having minimal dependencies, in many cases not even a shell, these images reduce the possibility of an attacker being able to use a compromised container as an effective tool for obtaining elevated privilege or exfiltrating key data such as wallet files controlled by a validator.

ko

Also originally implemented at Google, now maintained by engineers at Chainguard, ko is a tool for building go applications and distributing them as OCI containers. The default base image for images built with ko is a distroless base image, which will be distroless.dev/static:latest in the next ko release., which is sufficient for supporting the gaiad binary.

We have noticed that there are variants of the gaiad binary which contain components built with cgo, notably when building gaiad with support for cleveldb using the cleveldb build tag. For stability and reduced binary size, Chainguard strongly recommends building these components statically linked against musl. To facilitate this, the Distroless project provides a `distroless.dev/ko` image which we will use to build the final gaiad image.

Conversion process

Assuming that you have already authenticated to a registry, it is possible to use the `distroless.dev/ko` image to build a `gaiad` image using `docker` from the root of the `cosmos/gaia` repository by running:

```
$ KO_DOCKER_REPO=... ko build ./cmd/gaiad
```

The `KO_DOCKER_REPO` variable should be set to the prefix of the repository you want to publish to, for example, `ghcr.io/cosmos` for Cosmos Network. When `ko` completes, it will print the `sha256` reference that can be used with commands such as `docker pull`, etc.

SLSA/OpenSSF Scorecards Overview

SLSA is a set of standards and technical controls you can adopt to improve artifact integrity, and build towards completely resilient systems. It's not a single tool, but a step-by-step outline to prevent artifacts being tampered with and tampered artifacts from being used, and at the higher levels, hardening up the platforms that make up a supply chain. The SLSA framework can be found here: <https://slsa.dev/>.

Scorecards is an automated tool that assesses a number of important heuristics checks associated with software security and assigns each check a score of 0-10. You can use these scores to understand specific areas to improve in order to strengthen the security posture of your project. You can also assess the risks that dependencies introduce, and make informed decisions about accepting these risks, evaluating alternative solutions, or working with the maintainers to make improvements. The OpenSSF project can be found here: <https://github.com/ossf/scorecard>.

Repository Reviews

Introduction

Three GitHub repositories in the Cosmos ecosystem were reviewed in this assessment; the assessments were conducted primarily against the OpenSSF Scorecard and SLSA framework. These are the three repos that are included:

1. strangelove-ventures/heighliner
2. cosmos/cosmos-sdk
3. cosmos/gaia

1. strangelove-ventures/heighliner

Overview

Heighliner is a repository of Docker images for the node software of Cosmos chains. The images are available from the GitHub Container Registry (ghcr). Heighliner builds an image for each of the projects listed in the `chains.yaml` file, using template Dockerfiles. Images are built daily.

A consumer can build the images themselves using the heighliner CLI tool. The tool can be downloaded as a binary from the repository, or built from the Go source code. Users can build individual images or all images, and can choose to build particular releases. Cross compilation for other architectures can be performed via buildkit (<https://github.com/moby/buildkit>).

Images for other projects can be built by adding them to [chains.yaml](#) and submitting a pull request.

It should be noted that this build system is bespoke and takes an unusual approach; it will clone the repository for each chain, build it, and copy the result into an appropriate base image. For most repositories this is straightforward, but several have different build systems that require bespoke changes reflected in `chains.yaml` settings.

Observations and Recommendations

1. Make Build System Transparent

Description	Build system is not transparent.
Observation/Risk	<p>Heighliner builds were moved away from GitHub actions apparently due to problems with usage limits. We understand the images are now being built on a GCP instance which pushes the results to GHCR.</p> <p>This forces users to trust the build system without insight into what it is doing. If a malicious actor was to compromise the build, it would be very hard for the user to detect. With GitHub Actions, it is possible to audit what happens in the build (with the caveat that the user still has to trust GitHub).</p> <p>This also makes it impossible to assess the status of several SLSA requirements.</p>
Recommendation	For public images like these, we would recommend using a more transparent system like GitHub Actions or Circle CI. If that's not possible, releasing some details on how the images are built, plus using signing and attestations would increase trust in the system and allow users to verify the provenance of images.
Supply Chain Risk	Build

2. Update Base Images Used in Builds

Description	The images pushed to
-------------	----------------------

	ghcr.io/strangelove-ventures/heighliner/ are being built on out-of-date base images.
Observation/Risk	The heighliner images e.g. ghcr.io/strangelove-ventures/heighliner/gaia:v7.0.1 contain a number of vulnerabilities that have already been fixed in the base image. These vulnerabilities could potentially be exploited by an attacker. This is presumably because the build system is not pulling fresh images before building.
Recommendation	The build system should be modified to always pull fresh base images before building.
Supply Chain Risk	Build

3. Sign Images

Description	Images are not signed.
Observation/Risk	The Heighliner images are not signed.
Recommendation	<p>The build process for heighliner images should include signing of the image.</p> <p>Signing would allow end users to verify the provenance of the images. Users would have more confidence that they had come from the cosmos project and had not been tampered with.</p> <p>Signing can be performed with the Sigstore project and easily implemented as a GitHub action.</p>
Supply Chain Risk	Provenance

4. Add SBOMs to Releases

Description	Binaries (releases) do not include SBOMs.
Observation/Risk	Binaries (releases) should include SBOMs to provide visibility into the components included in binary artifacts to support CVE detection and remediation.
Recommendation	Build an SBOM on the binaries (releases) & sign using SLSA GitHub actions runner that can be plugged into the workflow, giving them provenance & attestations for artifacts.
Supply Chain Risk	Dependency Threats

5. Enforce Code Reviews

Description	Two-person reviews not being performed/documented.
Observation/Risk	Commits are being pushed directly to the main branch without two-person review. Uploader and reviewer need to be two different trusted persons.
Recommendation	Enable branch protection and enforce two-person code reviews.
Supply Chain Risk	Source

6. Consider Pinning Builder Images

Description	Builder images are not pinned, meaning it is difficult to reproduce images which can cause issues with debugging and provenance.
Observation/Risk	The image used to build Go code is <code>golang:alpine</code> , which will always be the latest version of Go. This makes it difficult to reproduce the same image, as it is never clear which version of the Go compiler was used to create the binary. A similar problem exists in the Rust image and also in libraries installed via apk.
Recommendation	<p>Consider pinning to a particular version of the Go compiler e.g. <code>1.18.3-alpine3.16</code> or using a digest to pin to an exact image e.g. <code>golang:alpine@sha256:7cc62574fcf9c5fb87ad42a9789d5539a6a085971d58ee75dd2ee146cb8a8695</code>. The versions should still be updated regularly and the changes tested to make sure nothing has broken.</p> <p>Note that libraries installed via apk can be pinned in a similar way.</p> <p>An alternative approach would be to add the version of the compiler and image used as metadata to the image, which would make it easier to reproduce the artifact if needed.</p> <p>Note that the daily production image should probably remain as “latest” to ensure the image is up-to-date and without vulnerabilities.</p>
Supply Chain Risk	Develop

7. Verify Downloads in Images

Description	Libraries and repositories are being downloaded without verification.
Observation/Risk	<p>In multiple places in the Dockerfile packages are downloaded with wget e.g. https://github.com/strangelove-ventures/heighliner/blob/main/dockerfile/sdk/Dockerfile#L12</p> <p>As no verification is performed on these packages, we cannot be sure they have not changed or been tampered with.</p>
Recommendation	<p>Verify the GPG signature of binaries where available. Here is an example where the Redis Dockerfile verifies the gosu binary with GPG: https://github.com/docker-library/redis/blob/c81f977ff974b4f0fe69d14fdf7127ad8b59d07d/7.0/Dockerfile#L15-L19</p> <p>In other cases where a signature is unavailable, or as an additional measure, the hash of a file can be checked. This ensures that the file has not been modified since the hash was taken (and presumably the file was verified or tested). Here is an example where the Redis binary is checked against a known SHA https://github.com/docker-library/redis/blob/c81f977ff974b4f0fe69d14fdf7127ad8b59d07d/7.0/Dockerfile#L49-L50</p>
Supply Chain Risk	Develop

8. Use Minimal Production Images

Description	Images produced by heighliner include unnecessary packages.
Observation/Risk	<p>The final images produced by the heighliner Dockerfiles include a lot of development packages (see https://github.com/strangelove-ventures/heighliner/blob/main/dockerfile/rust/Dockerfile#L64) such as gcc, git and nano. These are significantly increasing the size of the image, but won't be used by most users. In addition, they may pull in vulnerabilities that could be targeted by attackers.</p>

	The use of alpine as the base image results in a small image, but it is still possible to go smaller by using distroless images.
Recommendation	Remove the packages from the final image. If some of the packages are required for debugging, create a separate debug variant of the image. Even better, consider using distroless images (e.g. https://github.com/distroless), which would remove even more packages and potential attack surface. A simple way to do this for Go based images is to use https://github.com/google/ko . See guide: Using ko to harden the build process .
Supply Chain Risk	Build

9. Scan Images for Vulnerabilities

Description	Scan images for known vulnerabilities.
Observation/Risk	The images being currently produced contain known vulnerabilities (see A.4 Image Scans) due to out of date dependencies. These vulnerabilities could result in users being exposed to attacks.
Recommendation	Add an image scan step to the build process, using a well known scanning tool such as snyk, trivy or grype. Developers should be alerted when new vulnerabilities occur. The results should also be available to users e.g. via a GitHub badge.
Supply Chain Risk	Build

Automated SLSA/OpenSSF Review

SLSA Targeted Repo:
<https://github.com/strangelove-ventures/heighliner>

We used a tool to perform an automated SLSA analysis. The results are used to suggest areas of focus for the manual analysis and should not be taken as a guarantee of SLSA (non) conformance. The automated analysis indicated:

- Very few of the last 20 commits were signed, (10% signed).
- Very few of the last 20 commits were approved, (5% approved).
- Very few of the last 20 commits were reviewed, (5% reviewed).
- Very few of the last 20 commits had an associated PR, (10% with PRs).
- Very few of the last 20 commits had two party reviews, (5 % two party reviews).
- At the time of the last analysis the last commit was 27 days ago.
- There is no evidence of SBOM usage on main or releases page.
- The releases were automated with GitHub Actions.
- No private keys were found in the repository.

OpenSSF Scorecard Targeted Repo:

<https://github.com/strangelove-ventures/heighliner>

Score	Name	Reason	Details	Remediation
10 / 10	Binary-Artifacts	No binaries found in the repo.	None	Link
0 / 10	Branch-Protection	Branch protection not enabled on development/release branches.	Warn: branch protection not enabled for branch 'main'.	Link
10 / 10	CI-Tests	2 out of 2 merged PRs checked by a CI test.	None	Link
0/10	CII-Best-Practices	No badge detected.	None	Link
0/10	Code-Review	GitHub code reviews found for 2 commits out of the last 30.	Warn: no reviews found for commits: 3c5ac5291e05d89d0f0b7d7413d19f6ba5e05d7e 7966e45eb373bca8b1be55782842b48bb7ef4d1d 5d2cd37ba9f3bdd4d9f83c2b63023150fcfd4c82 <continued output>	Link
10/10	Contributors	3 different organizations found.	Info: contributors work for: OLSF,cosmos,strangelove-ventures	Link
10/10	Dangerous-Workflo	No dangerous	None	Link

	w	workflow patterns detected.		
0/10	Dependency-Update-Tool	No update tool detected.	Warn:dependabot config file not detected in source location. We recommend setting this configuration in code so it can be easily verified by others. Warn: renovatebot config file not detected in source location. We recommend setting this configuration in code so it can be easily verified by others.	Link
0/10	Fuzzing	Project is not fuzzed.	None	Link
10/10	License	License file detected.	Info: LICENSE:1	Link
10/10	Maintained	30 commit(s) out of 30 and 3 issue activity out of 3 found in the last 90 days.	None	Link
10/10	Packaging	Publishing workflow detected.	Info: GitHub publishing workflow used in run https://api.github.com/repos/strangelove-ventures/heighliner/actions/runs/2244209836:.github/workflows/release.yml:11	Link
0/10	Pinned-Dependencies	Dependency not pinned by hash - detected.	Warn:GitHub-owned GitHubAction not pinned by hash: .github/workflows/lint.yml:15	Link

			<p>Warn: third-party GitHubAction not pinned by hash: .github/workflows/lin.t.yml:17</p> <p>Warn:GitHub-owned GitHubAction not pinned by hash: .github/workflows/release.yml:15</p> <p><output concatenated></p>	
0/10	SAST	SAST tool is not run on all commits.	<p>Warn: 0 commits out of 2 are checked with a SAST tool.</p> <p>Warn: CodeQL tool not detected.</p>	Link
0/10	Security-Policy	Security policy file not detected.	None	Link
0/10	Signed-Releases	0 out of 5 artifacts are signed.	<p>Warn: release artifact v0.0.5 not signed:</p> <p>https://api.github.com/repos/strangelove-ventures/heighliner/releases/65655155</p> <p>Warn: release artifact v0.0.4 not signed:</p> <p>https://api.github.com/repos/strangelove-ventures/heighliner/releases/65197561</p> <p>Warn: release artifact v0.0.3 not signed:</p> <p>https://api.github.com/repos/strangelove-ventures/heighliner/releases/61107799</p> <p><output concatenated></p>	Link
0/10	Token-Permissions	Non read-only tokens detected in GitHub workflows.	Warn: no top level permission defined: .github/workflows/li	Link

			nt.yml:1 Warn: no top level permission defined: .github/workflows/release.yml:1 Info: candidate goyang publishing workflow: .github/workflows/release.yml:11 Info: candidate goyang.	
10/10	Vulnerabilities	No vulnerabilities detected.	None	Link
Aggregate Score	4.3/10			

Manual SLSA Review

Source Requirements

Version Controlled (Required for: L1, L2, L3, L4)

Strangelove-ventures/Heighliner is version controlled with the use of Git. Version controlled via Git including: change history and immutable references (with branches, tags, commitIDs).

Verified History (Required for: L3, L4)

Every change in the revision's history is strongly authenticated with an authenticated actor identity, although we cannot verify that two-step verification is in place.

Retained Indefinitely (Required for: L3 [for 18.mos], L4)

The revision history is preserved indefinitely and cannot be deleted, except for obliteration. For SLSA level 3 the retention can be limited to 18 months.

Two-person Reviewed (Required for: L4)

It appears that two-person reviews are not occurring. To achieve L4, every change in the revision's history will need to be agreed-to by two trusted persons prior to submission with both of the entities strongly authenticated.

Build Requirements

[Scripted Build](#) (Required for: L1, L2, L3, L4)

All build steps are fully defined in a build script, although there is a lack of visibility of the build triggers and build steps: GitHub actions are not producing the container images (but are being used for binary releases).

[Build Service](#) (Required for: L2, L3, L4)

Build steps for binaries are run using GitHub Actions as the build service and not on the developers' workstations. This can not be verified for image builds, which are run in GCP on compute instances.

[Build as Code](#) (Required for: L3, L4)

The build definition and configuration can be verified for the binaries, but lack of visibility into the image builds on GCP precludes verification of build definitions/configurations..

[Ephemeral Environment](#) (Required for: L3, L4)

GitHub Actions provide for an ephemeral environment in the binary builds, but visibility into the VM used for image builds precludes validating an ephemeral environment.

[Isolated](#) (Required for: L3, L4)

GitHub Actions provide for an isolated environment in the binary builds, but visibility into the VM used for image builds precludes validating an isolated environment.

[Parameterless](#) (Required for: L4)

GitHub Actions provide for parameterless binary builds. Even though the build configurations are based on tags, providing parameterless builds, there is no visibility into the actual build configurations used on the VM in GCP.

[Hermetic](#) (Required for: L4)

Build steps, source and dependencies are declared up front, but network access during the build cannot be verified due to lack of visibility in the compute-instance used.

[Reproducible](#) (Required for: L4)

Image builds are not binary reproducible. Running the build twice will result in a different digest. Binary reproducibility helps validate that artifacts haven't been tampered with and can simplify debugging. Addressing this requires the use of a build system that supports reproducibility, such as apko or bazel.

Provenance Requirements

Available (Required for: L1, L2, L3, L4)

Provenance is not available for the consumer in any format; see provenance details: <https://slsa.dev/provenance/v0.2>

Authenticated (Required for: L2, L3, L4)

The provenance authenticity and integrity cannot be verified by the consumer; see provenance details: [link](#).

Service Generated (Required for: L2, L3, L4)

Provenance is not being generated by the build service.

Non-falsifiable (Required for: L3, L4)

Provenance is not being generated by the build service.

Dependencies Complete (Required for: L4)

Provenance is not being generated by the build service.

Common Requirements

Security (Required for: L4)

The compute-instance used for image builds cannot be verified as having a baseline security standard. (i.e. patching, vulnerability scanning, user isolation, transport security, secure boot, machine, etc).

Access (Required for: L4)


Access to image-build compute-instance needs to be verified; Github Actions for binaries help compliance for the Access common requirements.


Superusers (Required for: L4)


Platform admin access cannot be validated to the image-build compute-instance.

For more information on requirements see <https://slsa.dev/spec/v0.1/requirements>. The following legends are used:

X - required.

 - present.

 - not present.

 - unknown/Could not be determined.

		Required At			
Requirement		SLSA 1	SLSA 2	SLSA 3	SLSA 4
Source	Version Controlled		x	x	x
	Verified History			x	x
	Retained Indefinitely			18mo	x
	Two-Person Reviewed				x
Build	Scripts	x	x	x	x
	Build Service		x	x	x
	Build as Code			x	x
	Ephemeral Environment			x	x
	Isolated			x	x
	Parameterless				x
	Hermetic				x
	Reproducible				O
Provenance	Available	x	x	x	x
	Authenticated		x	x	x
	Service Generated		x	x	x
	Non-Falsifiable			x	x
	Dependencies Complete				x
Common	Security				x
	Access				x
	Superusers				x

O = required unless there is a justification

2. Cosmos/Gaia Repo

Overview

Gaia is the implementation of Cosmos Hub. Cosmos Hub is a blockchain network that operates on a “Proof of Stake” consensus. The hub is built using the Cosmos SDK and compiled to a binary called “gaiad” which is the Gaia Daemon. Cosmos Hub and other Cosmos SDK blockchains interact via a protocol called IBC that enables Inter-Blockchain communication. These thousands of interconnected blockchains compose the Cosmos Network.

Observations and Recommendations

10. Sign Images

Description	Gaia images should be signed.
Observation/Risk	Gaia images are not signed.
Recommendation	<p>The build process used to produce the cosmos/gaia images should include signing of the image.</p> <p>Signing would allow end users to verify the provenance of the images. Users would have more confidence that they had come from the cosmos project and had not been tampered with.</p> <p>Signing can be performed with the Sigstore project and easily implemented as a GitHub action.</p>
Supply Chain Risk	Provenance

11. Releases Should Produce an SBOM & SLSA Provenance Attestation

Description	Binaries should include SBOMs.
Observation/Risk	Release reports do not include SBOMs.
Recommendation	In the release report with checksums (that help validate the downloads) SBOMs should be included in the releases and build reports used to generate the SLSA provenance attestation
Supply Chain Risk	Provenance

12. Artifacts Should be Signed

Description	GitHub Actions can be used to sign artifacts.
Observation/Risk	Artifacts produced in the builds are not signed.
Recommendation	Use GitHub Actions to sign artifacts produced in the builds. Link to GitHub Actions signing: https://github.blog/2022-04-07-slsa-3-compliance-with-github-actions/
Supply Chain Risk	Provenance

13. Consider Using distroless.dev/static Base Image

Description	Prefer <code>distroless.dev/static</code> to <code>gcr.io/distroless/cc</code>
Observation/Risk	<code>distroless.dev/static</code> offers some advantages over <code>gcr.io/distroless/cc</code> , notably it is a smaller image with signing and is rebuilt nightly.
Recommendation	Replace <code>gcr.io/distroless/cc</code> in the Dockerfile with <code>distroless.dev/static</code> . Full instructions are at https://github.com/distroless/static
Supply Chain Risk	Provenance

14. Scan Images for Vulnerabilities

Description	Scan images for known vulnerabilities.
Observation/Risk	The current <code>gaia</code> image contains known vulnerabilities (see A.4 Image Scans) as it is out-of-date. These vulnerabilities could result in users being exposed to attacks.
Recommendation	Add an image scan step to the build process, using a well known scanning tool such as <code> snyk </code> , <code> trivy </code> or <code> grype </code> . Developers should be alerted when new vulnerabilities occur. The results should also be available to users e.g. via a GitHub badge.

Supply Chain Risk	Build
-------------------	-------

15. Fix the Container Image Build

Description	The gaia image appears out-of-date
Observation/Risk	The current gaia image (https://github.com/cosmos/gaia/pkgs/container/gaia) is built from out-of-date code and dependencies, which places any users of the image at risk. This seems to stem from problems with the current GitHub action.
Recommendation	Fix the build process. Produce new images each day and for releases. Document where the image is and when it is updated.
Supply Chain Risk	Build

Automated SLSA/OpenSSF Review

SLSA Targeted Repo:

<https://github.com/cosmos/gaia>

An automated SLSA analysis was completed but the tool is used to suggest areas of focus and needed to be validated. Here is what the automated analysis indicated:

- All of the last 20 commits were signed, (100% signed).
- Most of the last 20 commits were approved, (80% approved).
- Most of the last 20 commits were reviewed, (85% reviewed).
- Most of the last 20 commits had an associated PR, (95% with PRs).
- Very few of the last 20 commits had two party reviews, (5% two party reviews).
- At the time of the last analysis the last commit was 3 days ago.
- There is no evidence of SBOM usage on main or releases page.
- Releases were found, indicating not fully automated.
- No private keys were found in the repository.

OpenSSF Scorecard Targeted Repo:

<https://github.com/cosmos/gaia>

Score	Name	Reason	Details	Remediation
10/10	Binary-Artifacts	No binaries found in	None	Link

		the repo.		
0/10	Branch-Protection	Branch protection not enabled on development/release branches.	Warn: branch protection not enabled for branch 'main'.	Link
9/10	CI-Tests	28 out of 29 merged PRs checked by a CI test.	None	Link
0/10	CII-Best-Practices	No badge detected.	None	Link
9/10	Code-Review	GitHub code reviews found for 29 commits out of the last 30.	Warn: no reviews found for commit: a9959b644d8f9f2883c8010bf403a3bb876cb7ed	Link
10/10	Contributors	31 different organizations found.	Info: contributors work for: CivicTechTO, CosmosContracts, FourthState, RestoretheFourthSF, Ristretto, TheDawnProject, allinbits, b-harvest, bin-studio, celestiaorg, clovers-network, cosmos, cosmos @tendermint @commercionetwork@kiracore, datagether, dawn-network, folia-app, galaxypi, guild-is, interchainberlin, interchainio, iqlusioninc, ixofoundation, kloudsio, metadefoundation, notional, osmosis-labs, ournetworks, relevant-community, solidity-korea, tendermint, tharsis@cosmos	Link
10/10	Dangerous-Workflow	No dangerous workflow patterns detected.	None	Link
10/10	Dependency-Update-Tool	Update tool detected.	Info: Dependabot detected: .github/dependabot.yml:1	Link
0/10	Fuzzing	Project is not fuzzed.	None	Link
10/10	License	License file detected.	Info: LICENSE:1	Link

10/10	Maintained	30 commit(s) out of 30 and 12 issue activity out of 30 found in the last 90 days.	None	Link
10/10	Packaging	Publishing workflow detected.	Info: GitHub publishing workflow used in run: https://api.github.com/repos/cosmos/gaia/actions/runs/2238033319:.github/workflows/docker-push.yml:13	Link
5/10	Pinned-Dependencies	Dependency not pinned by hash detected.	Warn: GitHub-owned GitHubAction not pinned by hash: .github/workflows/co-deql-analysis.yml:41 Warn: GitHub-owned GitHubAction not pinned by hash: .github/workflows/co-deql-analysis.yml:45 Warn: GitHub-owned GitHubAction not pinned by hash: .github/workflows/co-deql-analysis.yml:56 <output concatenated>	Link
9/10	SAST	SAST tool is not run on all commits.	Warn: 28 commits out of 29 are checked with a SAST tool. Warn: CodeQL tool not detected.	Link
10/10	Security-Policy	Security policy file detected.	Info: security policy detected in current repo:SECURITY.md:1.	Link
0/10	Signed-Releases	0 out of 5 artifacts are signed.	Warn: release artifact v7.0.2 not signed: https://api.github.com/repos/cosmos/gaia/releases/6637289	Link

			<p>2</p> <p>Warn: release artifact v7.0.1 not signed: https://api.github.com/repos/cosmos/gai/releases/64365771</p> <p>Warn: release artifact v7.0.0 not signed: https://api.github.com/repos/cosmos/gai/releases/62699473</p> <p><output concatenated></p>	
0/10	Token-Permissions	Non read-only tokens detected in GitHub workflows.	<p>Warn: no top level permission defined: .github/workflows/co-deql-analysis.yml:1</p> <p>Info: job level 'actions' permission set to 'read': .github/workflows/co-deql-analysis.yml:28</p> <p>Info: job level 'contents' permission set to 'read': .github/workflows/co-deql-analysis.yml:29</p>	Link
10/10	Vulnerabilities	No vulnerabilities detected.	None	Link
Aggregate Score	6.8/10			

Manual SLSA Review

Source Requirements

Version Controlled (Required for: L1, L2, L3, L4)

Every change to the source is tracked in a version control system, including change history and immutable references.

Verified History (Required for: L3, L4)

Every change in the revision's history has at least one strongly authenticated actor identity and timestamp. (Clarification on authentication of actors needed from Cosmos.)

Retained Indefinitely (Required for: L3 [for 18.mos], L4)

The revision and its change history are preserved indefinitely and cannot be deleted, except when subject to an established and transparent policy for obliteration.

Two-person Reviewed (Required for: L4)

Changes in the revision's history are agreed to by two trusted persons prior to submission. (Clarification on authentication of actors needed from Cosmos.)

Build Requirements

Scripted Build (Required for: L1, L2, L3, L4)

Build steps are defined in the build script and invoked with GitHub actions defined in yaml files.

Build Service (Required for: L2, L3, L4)

GitHub Actions are used for the build service.

Build as Code (Required for: L3, L4)

Build definitions and configurations are executed by the build service and are verifiably derived from text file definitions stored in a version control system.

Ephemeral Environment (Required for: L3, L4)

GitHub Actions are used as the build service and are run within an ephemeral environment; not reused from a prior build.

Isolated (Required for: L3, L4)

GitHub Actions are used to ensure the build steps are run in an isolated environment free of influence from other build instances.

Parameterless (Required for: L4)

The build is defined through the build script. (Validate with client.)

Hermetic (Required for: L4)

Artifacts can be located with a cryptographic hash to ensure integrity.

[Reproducible](#) (Required for: L4)

Image builds are not binary reproducible. Running the build twice will result in a different digest. Binary reproducibility helps validate that artifacts haven't been tampered with and can simplify debugging. Addressing this requires the use of a build system that supports reproducibility, such as apko or bazel.

Provenance Requirements

[Available](#) (Required for: L1, L2, L3, L4)

Provenance is not available for the consumer in any format; see provenance details: [link](#).

[Authenticated](#) (Required for: L2, L3, L4)

The provenance authenticity and integrity cannot be verified by the consumer; see provenance details: [link](#).

[Service Generated](#) (Required for: L2, L3, L4)

Provenance is not being generated by the build service.

[Non-falsifiable](#) (Required for: L3, L4)

Provenance is not being generated by the build service.

[Dependencies Complete](#) (Required for: L4)

Provenance is not being generated by the build service.

Common Requirements

[Security](#) (Required for: L4)

GitHub Actions provides managed security at the system level (with baseline security and maintenance).

[Access](#) (Required for: L4)

GitHub Actions provides physical and remote security to the trusted system.

[Superusers](#) (Required for: L4)

GitHub Actions provides managed security at the system level with limited platform admins..

For more information on requirements see <https://slsa.dev/spec/v0.1/requirements>. The following legends are used:

X - required.

- present.

- not present.

- unknown/Could not be determined.

		Required At			
Requirement		SLSA 1	SLSA 2	SLSA 3	SLSA 4
Source	Version Controlled		x	x	x
	Verified History			x	x
	Retained Indefinitely			18mo	x
	Two-Person Reviewed				x
Build	Scripts	x	x	x	x
	Build Service		x	x	x
	Build as Code			x	x
	Ephemeral Environment			x	x
	Isolated			x	x
	Parameterless				x
	Hermetic				x
	Reproducible				O
Provenance	Available	x	x	x	x
	Authenticated		x	x	x
	Service Generated		x	x	x
	Non-Falsifiable			x	x
	Dependencies Complete				x

Common	Security				x
	Access				x
	Superusers				x
<i>O = required unless there is a justification</i>					

3. Cosmos/Cosmos-SDK

Overview

Cosmos SDK is a framework for building blockchain applications in Go. The SDK itself is written in the Golang programming language and is used to build Gaia, which is the implementation of Cosmos Hub. Cosmos SDK is used to build complex blockchains on top of Tendermint, an open-source software for launching blockchains that allows application development in any language.

Observations and Recommendations

16. SDK Signing

Description	Artifacts are not signed.
Observation/Risk	Cosmos-SDK is used in builds for Gaia and used within other projects, making it critical that Gaia artifacts are signed.
Recommendation	Include signing of artifacts in the build stages; GitHub Actions to provide verification of releases.
Supply Chain Risk	Provenance

17. SDK Archive: Provide SBOM to Users

Description	Binaries (releases) should include SBOMs
Observation/Risk	Cosmos-SDK is used in builds in Gaia and used within other projects, SBOMs should be provided to downstream projects.
Recommendation	Generate SBOM to help provide provenance of the software compiled.
Supply Chain Risk	Provenance

18. SDK is Installing Dependencies via Curl without Verification

Description	Libraries and repositories are being downloaded without
-------------	---

	verification.
Observation/Risk	There is no validation when installing dependencies are not verified and the artifacts produced can be used downstream in a number of projects.
Recommendation	Verifications and signing should be used within the builds.
Supply Chain Risk	Provenance

Automated SLSA/OpenSSF Review

SLSA Targeted Repo:

<https://github.com/cosmos/cosmos-sdk>

An automated SLSA analysis was completed but the tool is used to suggest areas of focus and needed to be validated. Here is what the automated analysis indicated:

- All of the last 20 commits were signed, (100% signed).
- All of the last 20 commits were approved, (100% approved).
- All of the last 20 commits were reviewed, (100% reviewed).
- All of the last 20 commits had an associated PR, (100% with PRs).
- Few of the last 20 commits had two party reviews, (40 % two party reviews).
- At the time of the last analysis the last commit was 3 days ago.
- There is no evidence of SBOM usage on main or releases page.
- Previous release was fully automated (“github-actions”).
- No private keys were found in the repository.

OpenSSF Scorecard Targeted Repo:

<https://github.com/cosmos/cosmos-sdk>

Score	Name	Reason	Details	Remediation
10/10	Binary-Artifacts	No binaries found in the repo.	None	Link
8/10	Branch-Protection	Branch protection is not maximal on development and all release branches.	Info: 'force pushes' disabled on branch 'main'. Info:'allow deletion'	Link

			<p>disabled on branch 'main' Info: status check found to merge onto on branch 'main'.</p> <p>Warn: number of required reviewers is only 1 on branch 'main'.</p>	
10 / 10	CI-Tests	30 out of 30 merged PRs checked by a CI test.	None	Link
0/10	CII-Best-Practices	No badge detected.	None	Link
10/10	Code-Review	All last 30 commits are reviewed through GitHub.	None	Link
10/10	Contributors	48 different organizations found.	<p>Info: contributors work for:</p> <p>BerkeleyBlockchain, CosmWasm, CosmosContracts, DA0-DA0, DevchainUserGroup, FourthState, PeggyJV, Project-Arda, ROTranslationTools, all in bitsinc, alpha-fi, arkworks-rs, bluecollarcoding, celestiaorg, census-instrumentation, clojure, confio, cosmos, cosmos / @tendermint, cosmos @regen-network. previously@polkadot-js, cosmos@tendermint, cosmos@tendermint@commercionetwork@kiracore, dcrypto, geneva-haskell-group, golang, informalsystems, informal systems, interchainio, ixofoundation, opentelemetry, opencensus-integrations, orijtech, osmosis-labs, peggyjv, reacherhq, regen-network, regen-network, scaleit, scale-it, scipr-lab, shootismoke, stranglove-ventures, sunshine-validation, tendermint, tharsis@cosmos, uc berkeley</p>	Link

			@sikkatech, vitwit, voxel	
10/10	Dangerous-Workflow	No dangerous workflow patterns detected.	None	Link
10/10	Dependency-Update-Tool	Update tool detected.	Info: Dependabot detected: .github/dependabot.yml:1	Link
10/10	Fuzzing	Project is fuzzed with: [OSSFuzz GoBuiltinFuzzer]	Info: func FuzzCryptoHDDerivePrivateKeyForPath(f *testing.F): fuzz/tests/crypto_hd_deriveprivatekeyforpath_test.go:17 Info: func FuzzCryptoHDNewParamsFromPath(f *testing.F): fuzz/tests/crypto_hd_newparamsfrompath_test.go:11 Info: func FuzzCryptoTypesCompactbitarrayMarshalUnmarshal(f *testing.F): fuzz/tests/crypto_types_compactbitarray_marshalunmarshal_test.go:11 Info: func FuzzStoreInternalProofsCreateNonmembershipProof(f *testing.F): <output concatenated>	Link
10/10	License	License file detected.	None	Link
10/10	Maintained	30 commit(s) out of 30 and 19 issue activity out of 30 found in the last 90 days.	None	Link

10/10	Packaging	Publishing workflow detected.	Info: GitHub publishing workflow used in run: https://api.github.com/repos/cosmos/cosmos-sdk/actions/runs/1826713772:github/workflows/cosmover-release.yml:11	Link
5/10	Pinned-Dependencies	Dependency not pinned by hash - detected.	Warn: GitHub-owned GitHubAction not pinned by hash: .github/workflows/atlas.yml:47 Warn: third-party GitHubAction not pinned by hash: .github/workflows/atlas.yml:48 Warn: third-party GitHubAction not pinned by hash: .github/workflows/atlas.yml:53 Warn: GitHub-owned GitHubAction not pinned by hash: .github/workflows/atlas.yml:17 <output concatenated>	Link
8/10	SAST	SAST tool detected but not run on all commits.	Warn: 17 commits out of 30 are checked with a SAST tool Info: SAST tool detected: CodeQL.	Link
10/10	Security-Policy	Security policy file detected.	Info: security policy detected in current repo: SECURITY.md:1	Link
0/10	Signed-Releases	0 out of 1 artifacts are signed.	Warn: release artifact cosmovisor/v1.1.0 not signed: https://api.github.com/repos/cosmos/cosmos-sdk/releases/59286154	Link
0/10	Token-Permissions	Non read-only	Warn: no top level	Link

		tokens detected in GitHub workflows.	permission defined: .github/workflows/atlas.yml:1 Info: top level 'contents' permission set to 'read': .github/workflows/check-docs.yml:11 Warn: no top level permission defined: .github/workflows/clean-artifacts.yml:1 Warn: no top level permission defined: .github/workflows/codeql-analysis.yml:1 <output concatenated>	
10/10	Vulnerabilities	No vulnerabilities detected.	None	Link
Aggregate Score	7.9/10			

Manual SLSA Review

Source Requirements

[Version Controlled](#) (Required for: L1, L2, L3, L4)

Every change to the source is tracked in a version control system, including change history and immutable references.

[Verified History](#) (Required for: L3, L4)

Every change in the revision's history has at least one strongly authenticated actor identity and timestamp.

[Retained Indefinitely](#) (Required for: L3 [for 18.mos], L4)

The regions and its change history are preserved indefinitely and cannot be deleted, except when subject to an established and transparent policy for obliteration.

[Two-person Reviewed](#) (Required for: L4)

Changes in the revision's history are agreed to by two trusted persons prior to submission.

Build Requirements

[Scripted Build](#) (Required for: L1, L2, L3, L4)

Build steps are defined in the build script and invoked with GitHub actions defined in yaml files.

[Build Service](#) (Required for: L2, L3, L4)

GitHub Actions are used for the build service.

[Build as Code](#) (Required for: L3, L4)

Build definitions and configurations are executed by the build service and are verifiably derived from text file definitions stored in a version control system.

[Ephemeral Environment](#) (Required for: L3, L4)

GitHub Actions are used as the build service and are run within an ephemeral environment; not reused from a prior build.

[Isolated](#) (Required for: L3, L4)

GitHub Actions are used to ensure the build steps are run in an isolated environment free of influence from other build instances.

[Parameterless](#) (Required for: L4)

The build is defined through the build script. (Validate with client.)

[Hermetic](#) (Required for: L4)

Artifacts can be located with a cryptographic hash to ensure integrity.

[Reproducible](#) (Required for: L4)

Binary reproducibility helps validate that artifacts haven't been tampered with and can simplify debugging. Addressing this requires the use of a build system that supports reproducibility, such as apko or bazel.

Provenance Requirements

[Available](#) (Required for: L1, L2, L3, L4)

Provenance is not available for the consumer in any format; see provenance details: [link](#).

Authenticated (Required for: L2, L3, L4)

The provenance authenticity and integrity cannot be verified by the consumer; see provenance details: [link](#).

Service Generated (Required for: L2, L3, L4)

Provenance is not being generated by the build service.

Non-falsifiable (Required for: L3, L4)

Provenance is not being generated by the build service.

Dependencies Complete (Required for: L4)

Provenance is not being generated by the build service.

Common Requirements

Security (Required for: L4)

GitHub Actions provides managed security at the system level (with baseline security and maintenance).

Access (Required for: L4)


GitHub Actions provides physical and remote security to the trusted system.


Superusers (Required for: L4)


GitHub Actions provides managed security at the system level with limited platform admins..

For more information on requirements see <https://slsa.dev/spec/v0.1/requirements>. The following legends are used:

X - required.

 - present.

 - not present.

 - unknown/Could not be determined.

	Required At			
Requirement	SLSA 1	SLSA 2	SLSA 3	SLSA 4

Source	Version Controlled		x	x	x
	Verified History			x	x
	Retained Indefinitely			18mo	x
	Two-Person Reviewed				x
Build	Scripts	x	x	x	x
	Build Service		x	x	x
	Build as Code			x	x
	Ephemeral Environment			x	x
	Isolated			x	x
	Parameterless				x
	Hermetic				x
	Reproducible				O
Provenance	Available	x	x	x	x
	Authenticated		x	x	x
	Service Generated		x	x	x
	Non-Falsifiable			x	x
	Dependencies Complete				x
Common	Security				x
	Access				x
	Superusers				x

Github Actions Signing, Hardening and Event Monitoring

The use of GitHub Actions is already a step towards achieving SLSA compliance. GitHub Actions allows for secure capture of metadata in the CI/CD pipeline executions as well as integration for Sigstore signing. Most containers available today are vulnerable to supply chain attacks because they can be published with nothing more than a simple API key (Hutchings, 2021). One of the best ways to protect users from these kinds of attacks is by signing the image at creation time so that developers can verify that the code they received is the code that the maintainer authored.

GitHub Actions have integrated Sigstore support for container image and code signing in the GitHub Actions workflows. The integration allows developers to build, distribute, and verify signed software artifacts by including attestations into the signature. Attestations can track the repository the build came from, workflow and commit references. Images are signed with GitHub-provided OIDC tokens in the actions without having to provision private keys. With Sigstore, specifically Rekor, the keyless signing process publishes your username, organization name, repository name and workflow name to the public transparency logs.

Sigstore is comprised of three projects (Palafox, 2022):

Cosign - used to sign software or images.

Fulcio - a certificate authority providing short-lived certificates via OpenID.

Rekor - secure log signing, for provenance verification of software artifacts.

Implementation of SLSA is accomplished by including metadata during the build process (Palafox, 2022). Consumers can use the metadata to make risk-based assessments on what they consume. This metadata provides provenance: information about where it was built, who built it and from what codebase it originated from. With GitHub actions, signing can include the exact repository, commit and Actions workload the build came from.

GitHub Actions (Build) Key Recommendations

This report also details recommendations to add to the roadmap, such as these top three Findings from interviews were and their Description

- Ensure Github Actions best practices are implemented to increase SLSA compliance, [SLSA](#)
 - Secrets & Sensitive Information - ensure Github Actions cannot inject secrets or sensitive information during the build process.
 - Third party Github Actions - ensure marketplace Github Actions are properly vetted. (Only allow enterprise, and select non-enterprise actions & workflows.)
 - Github Standard Actions
 - Verified Creators

- Trusted Vendors
 - By-product from workflows - ensure appropriate access to artifacts created in the build jobs. Reduce retention where possible.
 - “Add-mask” command for values in logs, ensure the use of github secrets when required as opposed to github variables.
 - Forked repositories - workflows generated on forked pull requests may have inappropriate access.
 - Malicious docker images - ensure provenance of images used for containers.
- Expand security focused developer education programs:
 - An ounce of prevention saves time, money and reputation. Having developers understand supply chain attacks, risks, and mitigations will greatly decrease the risk of them happening.
- De-risk 3rd party provided code:
 - Third party closed source code could contain vulnerable or unknown code execution. It should be treated just like any software being introduced into the Cosmos software supply chain.

Appendix:

A.1 Analysis of SLSA-related practices of strangelove-ventures/heighliner using Chainguard internal automated analysis:

Check	Result
Percentage of last 20 commits that were signed	10%
Percentage of last 20 commits that were approved	5%
Percentage of last 20 commits that were reviewed	5%
Percentage of last 20 commits that had an associated PR	10%
Percentage of last 20 commits that had two-party review	5%
Last commit made how many days ago	27 days
Evidence of SBOM usage	No
Evidence that previous release was automated	Yes
Evidence of private keys checked into git	No

A.2 Analysis of SLSA-related practices of Cosmos/Gaia using Chainguard internal analysis:

Check	Result
Percentage of last 20 commits that were signed	100%
Percentage of last 20 commits that were approved	80%
Percentage of last 20 commits that were reviewed	85%
Percentage of last 20 commits that had an associated PR	95%
Percentage of last 20 commits that had two-party review	5%
Last commit made how many days ago	3 days
Evidence of SBOM usage	No
Evidence that previous release was automated	No
Evidence of private keys checked into git	No

A.3 Analysis of SLSA-Related practices of Cosmos/Cosmos-SDK using Chainguard internal analysis:

Check	Result
Percentage of last 20 commits that were signed	100%
Percentage of last 20 commits that were approved	100%
Percentage of last 20 commits that were reviewed	100%
Percentage of last 20 commits that had an associated PR	100%
Percentage of last 20 commits that had two-party review	40%
Last commit made how many days ago	3 days
Evidence of SBOM usage	No
Evidence that previous release was automated	Yes
Evidence of private keys checked into git	No

A.4 Checks for Hardening GitHub Actions

Hardening Check	Short Description	Details	Action
1.0	Using Secrets	Sensitive values should never be stored as plaintext in the workflow files, but rather as secrets.	Secrets can be configured at the organization, repository, or environments level and allows you to store sensitive information in GitHub.
2.0	Using CODEOWNERS to monitor changes	Use the "CODEOWNERS" feature to control how changes are made to your workflow files.	Add workflow files to the code owners list so proposed changes to these files will first require approval from a designated reviewer.
3.0	Risk & Mitigation of script injection attack	Consider whether your code might execute untrusted input from attackers, which can be done by adding malicious commands and scripts to a context. (Stings may be interpreted as code which is then executed on the runner.)	Ensure that values of github contexts (body, default_branch, email, head_ref, label, message, name page_nae, ref and title) do not flow directly into workflows, actions, API calls, or anywhere else that can allow them to be interpreted as executable code.

4.0	Using OpenID Connect for cloud resources	If GitHub Actions workflows need to access resources from a cloud provider that supports OpenID Connect (OIDC), you can configure your workflows to authenticate directly to the cloud provider. (This helps to prevent storing credentials as long-lived secrets.)	Configure workflows to authenticate directly with cloud providers using OIDC.
5.0	Using third-party actions	Individual jobs in a workflow can interact with (and compromise) other jobs. A compromised action can have access to all secrets configured in the repository and may be able to use the GITHUB_TOKEN to write to the repository. There is significant risk in sourcing actions from third-party repositories.	Pin actions to a full length commit SHA. Audit the source code of the action. Pin actions to a tag only if you trust the creator.
6.0	Reusing third-party workflows	Same as above	Same as above
7.0	Prevent GitHub Actions from creating/approving pull requests	Allowing workflows, or any other automation, to create or approve pull requests could be a security risk if the pull request is merged without proper oversight.	Prevent GitHub Action workflows from creating or approving pull requests.
8.0	Using OpenSSF 'Scorecards' to secure workflows	Use OpenSSF 'Scorecards' action and starter workflow to provide guidance on security best practices. When configured, 'Scorecards' action runs automatically on repository changes and alerts developers about risk supply chain practices. Checks include: script injection attacks, token permissions, and pinned actions.	Use OpenSSF 'Scorecards' actions and starter workflows to follow security best practices.
9.0	Cross-repository access	<ol style="list-style-type: none"> 1. GITHUB_TOKEN should be used whenever possible. 2. Repository deploy keys can be used to interact with another repository in a workflow. 3. Github App tokens: GitHub Apps can be installed on select repositories and provide granular permissions on resources within them. GitHub Apps can be used internally within an organization. 4. Personal Access Tokens should 	<p>Use GitHub tokens whenever possible. Repository deploy keys can be used to interact with another repository in a workflow.</p> <p>GitHub App tokens can be used to provide granular permissions on resources within a repository. Actively prevent use of Personal Access Tokens/SSH keys.</p>

		<p>never be used. These tokens provide access to all repositories within the organizations that the person has access to and to personal repositories.</p> <p>5. SSH keys should never be used for workflows, they grant read/write permissions to all of the organizations' repositories and personal repositories.</p>	
10.0	Hardening self-hosted runners	<p>Self-hosted runners should never be used for public repositories because any user can open pull requests against the repository and compromise the environment.</p> <p>Use caution when using self-hosted runners on private or internal repositories: anyone who can fork the repository and open a pull request are able to compromise the self-hosted runner environment, including gaining access to secrets and the GITHUB_TOKEN.</p>	<p>Do not use self-hosted runners for public repositories.</p> <p>Use caution when using self-hosted runners on private/internal repositories. Consider if sensitive information resides on machine configured as a self-hosted runner (i.e. private SSH keys, API access tokens, etc)</p> <p>Consider if the host has access to sensitive services including cloud services.</p> <p>Plan your self-hosted runner strategy to the organization's practices. (Centralized Vs. Decentralized)</p>

A.5 Events to Monitor for Hardening GitHub Actions

Events for Environments	Event Details
environment.create_actions_secret	<u>Triggered when a secret is created in an environment. For more information, see "Environment secrets."</u>
environment.delete	<u>Triggered when an environment is deleted. For more information, see "Deleting an environment."</u>
environment.remove_actions_secret	<u>Triggered when a secret is removed from an environment. For more information, see "Environment secrets."</u>
environment.update_actions_secret	<u>Triggered when a secret in an environment is updated. For more information, see "Environment secrets."</u>
Events for configuration changes	Event Details
repo.actions_enabled	<u>Triggered when GitHub Actions is enabled for a repository. Can be viewed using the UI. This event is not visible when you access the audit log using the REST API. For more information,</u>

	see " Using the REST API. "
repo.update_actions_access_settings	Triggered when the setting to control how your repository is used by GitHub Actions workflows in other repositories is changed.
Events for secret management	Event Details
org.create_actions_secret	<u>Triggered when a GitHub Actions secret is created for an organization.</u> For more information, see " Creating encrypted secrets for an organization. "
org.remove_actions_secret	Triggered when a GitHub Actions secret is removed.
org.update_actions_secret	Triggered when a GitHub Actions secret is updated.
repo.create_actions_secret	<u>Triggered when a GitHub Actions secret is created for a repository.</u> For more information, see " Creating encrypted secrets for a repository. "
repo.remove_actions_secret	Triggered when a GitHub Actions secret is removed.
repo.update_actions_secret	Triggered when a GitHub Actions secret is updated.
Events for self-hosted runners	Event Details
enterprise.register_self_hosted_runner	<u>Triggered when a new self-hosted runner is registered.</u> For more information, see " Adding a self-hosted runner to an enterprise. "
enterprise.remove_self_hosted_runner	Triggered when a self-hosted runner is removed.
enterprise.runner_group_runners_updated	<u>Triggered when a runner group's member list is updated.</u> For more information, see " Set self-hosted runners in a group for an organization. "
enterprise.self_hosted_runner_online	<u>Triggered when the runner application is started.</u> Can only be viewed using the REST API; not visible in the UI or JSON/CSV export. For more information, see " Checking the status of a self-hosted runner. "
enterprise.self_hosted_runner_offline	<u>Triggered when the runner application is stopped.</u> Can only be viewed using the REST API; not visible in the UI or JSON/CSV export. For more information, see " Checking the status of a self-hosted runner. "
enterprise.self_hosted_runner_updated	Triggered when the runner application is updated. Can be viewed using the REST API and the UI. This event is not included when you export the audit log as JSON data or a CSV file. For more information, see " About self-hosted runners " and

	"Reviewing the audit log for your organization."
org.register_self_hosted_runner	<u>Triggered when a new self-hosted runner is registered. For more information, see "Adding a self-hosted runner to an organization."</u>
org.remove_self_hosted_runner	<u>Triggered when a self-hosted runner is removed. For more information, see Removing a runner from an organization.</u>
org.runner_group_runners_updated	<u>Triggered when a runner group's list of members is updated. For more information, see "Set self-hosted runners in a group for an organization."</u>
org.runner_group_updated	<u>Triggered when the configuration of a self-hosted runner group is changed. For more information, see "Changing the access policy of a self-hosted runner group."</u>
org.self_hosted_runner_online	<u>Triggered when the runner application is started. Can only be viewed using the REST API; not visible in the UI or JSON/CSV export. For more information, see "Checking the status of a self-hosted runner."</u>
org.self_hosted_runner_offline	<u>Triggered when the runner application is stopped. Can only be viewed using the REST API; not visible in the UI or JSON/CSV export. For more information, see "Checking the status of a self-hosted runner."</u>
org.self_hosted_runner_updated	<u>Triggered when the runner application is updated. Can be viewed using the REST API and the UI; not visible in the JSON/CSV export. For more information, see "About self-hosted runners."</u>
repo.register_self_hosted_runner	<u>Triggered when a new self-hosted runner is registered. For more information, see "Adding a self-hosted runner to a repository."</u>
repo.remove_self_hosted_runner	<u>Triggered when a self-hosted runner is removed. For more information, see "Removing a runner from a repository."</u>
repo.self_hosted_runner_online	<u>Triggered when the runner application is started. Can only be viewed using the REST API; not visible in the UI or JSON/CSV export. For more information, see "Checking the status of a self-hosted runner."</u>
repo.self_hosted_runner_offline	<u>Triggered when the runner application is stopped. Can only be viewed using the REST API; not visible in the UI or JSON/CSV export. For more information, see "Checking the status of a self-hosted runner."</u>
repo.self_hosted_runner_updated	<u>Triggered when the runner application is updated. Can be viewed using the REST API and the UI; not visible in the</u>

	JSON/CSV export. For more information, see " About self-hosted runners ."
Events for self-hosted runner groups	Event Details
enterprise.runner_group_created	<u>Triggered when a self-hosted runner group is created.</u> For more information, see " Creating a self-hosted runner group for an enterprise ."
enterprise.runner_group_removed	<u>Triggered when a self-hosted runner group is removed.</u> For more information, see " Removing a self-hosted runner group ."
enterprise.runner_group_runner_removed	<u>Triggered when the REST API is used to remove a self-hosted runner from a group.</u>
enterprise.runner_group_runners_added	<u>Triggered when a self-hosted runner is added to a group.</u> For more information, see " Moving a self-hosted runner to a group ."
enterprise.runner_group_updated	<u>Triggered when the configuration of a self-hosted runner group is changed.</u> For more information, see " Changing the access policy of a self-hosted runner group ."
org.runner_group_created	<u>Triggered when a self-hosted runner group is created.</u> For more information, see " Creating a self-hosted runner group for an organization ."
org.runner_group_removed	<u>Triggered when a self-hosted runner group is removed.</u> For more information, see " Removing a self-hosted runner group ."
org.runner_group_updated	<u>Triggered when the configuration of a self-hosted runner group is changed.</u> For more information, see " Changing the access policy of a self-hosted runner group ."
org.runner_group_runners_added	<u>Triggered when a self-hosted runner is added to a group.</u> For more information, see " Moving a self-hosted runner to a group ."
org.runner_group_runner_removed	<u>Triggered when the REST API is used to remove a self-hosted runner from a group.</u> For more information, see " Remove a self-hosted runner from a group for an organization ."
Events for workflow activities	Event Details
cancel_workflow_run	<u>Triggered when a workflow run has been canceled.</u> For more information, see " Canceling a workflow ."
completed_workflow_run	<u>Triggered when a workflow status changes to completed.</u> Can only be viewed using the REST API; not visible in the UI or the JSON/CSV export. For more information, see " Viewing workflow run history ."
created_workflow_run	<u>Triggered when a workflow run is created.</u> Can only be viewed

	<u>using the REST API; not visible in the UI or the JSON/CSV export. For more information, see "Create an example workflow."</u>
delete_workflow_run	<u>Triggered when a workflow run is deleted. For more information, see "Deleting a workflow run."</u>
disable_workflow	Triggered when a workflow is disabled.
enable_workflow	Triggered when a workflow is enabled, after previously being disabled by disable_workflow.
rerun_workflow_run	<u>Triggered when a workflow run is re-run. For more information, see "Re-running a workflow."</u>
prepared_workflow_job	<u>Triggered when a workflow job is started. Includes the list of secrets that were provided to the job. Can only be viewed using the REST API. It is not visible in the GitHub web interface or included in the JSON/CSV export. For more information, see "Events that trigger workflows."</u>
approve_workflow_job	<u>Triggered when a workflow job has been approved. For more information, see "Reviewing deployments."</u>
reject_workflow_job	<u>Triggered when a workflow job has been rejected. For more information, see "Reviewing deployments."</u>

A.6 Image Scans

We scanned several of the Cosmos/Orijtech images for vulnerabilities. In all cases the arm64 version of the images was used. Scanning was performed on 8 July 2022 with the following version of docker scan:

```
$ docker scan --version
Version:    v0.17.0
Git commit: 061fe0a
Provider:   Snyk (1.827.0)
```

The findings can be summarized as:

- The `interchainio/simapp` image is based on `alpine:edge` and the vulnerabilities were present in the base image. They could be addressed by running `apk update` in the Dockerfile, or moving to a more regularly updated base image like `distroless.dev/alpine-base`.
- We expected to find the `simapp` image at `cosmos-sdk/simapp`. However this repo appears abandoned in favor of `interchainio`. This opens the project to a potential squatting attack if a malicious actor gains control of the `cosmos-sdk` repo, or points users to out-of-date images on the existing repo.
- The 15 vulnerabilities in most of the heighliner images is caused by use of an out-of-date base image. Changing the build system to pull the image everyday, or moving back to GitHub actions would fix this issue.
- The `cosmos/gaia` image build hasn't run successfully for 3 months, resulting in an out-of-date image.
- There are 2 images - `polkadot` and `penumbra` - that have a separate build process based on Rust that results in 132 vulnerabilities. The base image used here is `debian:bullseye`, which is a relatively large image and again out-of-date. We strongly recommend moving these to a smaller base image.

Image	Digest	Packages	Vulnerabilities
<code>interchainio/simapp:latest</code>	<code>sha256:c2f081ac3d93a22f1c4396bd1aecdcb95b39d97356b1ad79b844ba21b8476ff</code>	16	3
<code>ghcr.io/cosmos/gaia:sha-fca0a63</code>	<code>sha256:8f8b299a893d84b5d8827ec31ea3f3dee0fc196e0f7135d81217e78328c10c35</code>	9	13
ghcr.io/strangelove-ventures/heighliner/sim:latest	<code>sha256:89f24836e3c136293aaffe1433f3f51b46f72c736893d79cff272c2e886e0369</code>	44	15

ghcr.io/strangelove-ventures/heightner/osmosis:latest	sha256:0d9d8826c7ff82f0d2d67d4d0a3cf01e96960c692bc5d77c792af78272c4a6ae	44	15
ghcr.io/strangelove-ventures/heightner/gaia:latest	sha256:1a018c8375ba97d0d598c2bb83c7ae9ac2c0d90042eaa773fa6b3f30edec6eb6	44	15
ghcr.io/strangelove-ventures/heightner/juno:latest	sha256:a136ebcde38bdf079b1dc03a58894e6ff0d7a08391c3ec1cb03ba91e76438b5b	44	15
ghcr.io/strangelove-ventures/heightner/axelar:latest	sha256:b0efc6002c38c080e6e0a0a6fd5db4da8ffd2832a9efd324af98a6af98896d9c	44	15
ghcr.io/strangelove-ventures/heightner/evmos:latest	sha256:8aa302b2659fb7044d93565955a43a12cae8d9308d4cd859bf1ffa40f79af152	44	15
ghcr.io/strangelove-ventures/heightner/sentinel:latest	sha256:e19b46b71f3d5be1ca17acb8f81425f13bce7dda0a0084885be316e48d378209	44	15
ghcr.io/strangelove-ventures/heightner/akash:latest	sha256:39218ad477fdd25ca13ceae861b2bb24b051be7f5a2288b1c00aac5ff7684a8	44	15
ghcr.io/strangelove-ventures/heightner/omniflix:latest	sha256:2caeb47b1ab02fdb181c8149b9001189cef110f0141a0875fe808f928da65419	44	15
ghcr.io/strangelove-ventures/heightner/sommelier:latest	sha256:6c038559589191f2cc17dfaa6d52263b1b8e34a65fbc9c59e5b4743204b956d	44	15
ghcr.io/strangelove-ventures/heightner/penumbra:latest	sha256:02e5acfe38f6279f6d92814ace380870d55b44e5e26323811fd0a92baa41dcca	194	132
ghcr.io/strangelove-ventures/heightner/crescent:latest	sha256:a5b67ef87e9c5aab44b5d0421cc549b342d27a839763e2ae629998d5c367d5ea	44	15
ghcr.io/strangelove-ventures/heightner/tendermint:latest	sha256:852c647711dfef48abed1ec1c115f17f57cf3a3aacb0575ccba3bbac05e2ad9d	44	15
ghcr.io/strangelove-ventures/heightner/polkadot:latest	sha256:68117aa510e51227aef53d57ce44e0d6a89e2b996642f235a507b1b64b50f20b	194	132
ghcr.io/strangelove-ventures/heightner/provenance:latest	sha256:dd50b8fb41ce1831307f67b5e08ce15d07e3860bc609dd957d125bb2fe3ad04c	44	15

ghcr.io/strangelove-ventures/heighliner/bitcanna:latest	sha256:4e3c7fec09d477d8e859f30ab018cc0e4dba4ab729acfb2388db172b8e67d751	44	15
ghcr.io/strangelove-ventures/heighliner/regen:latest	sha256:75b4a4497d3f73a42a3ab1a2b2589d0ff62adf4b30036b279428d473dcae9333	44	15
ghcr.io/strangelove-ventures/heighliner/cerberus:latest	sha256:50e9d49f02d5c33b7742e38efa77af514528f1e4dbe596ee4904aba971e032d8	44	15
ghcr.io/strangelove-ventures/heighliner/cronos:latest	sha256:f91935ea92ab5a643ab0b6e56d3489edf4e588d4329f2ca5ab2543706b4c7c30	44	15
ghcr.io/strangelove-ventures/heighliner/stargaze:latest	sha256:d0127b836cc78395b6f285554b942dbd835bfb8770a8b03a10628168e96aeb59	44	15
ghcr.io/strangelove-ventures/heighliner/starname:latest	sha256:90273d93dd842a3dc9008116623b5b33f3ae4dcd79cdba9a8de812864605f95b	44	15
ghcr.io/strangelove-ventures/heighliner/terra:latest	sha256:8ba521895a3f45335f9a3b211f6724c8dc50b3a96ee2be99505537bb912c5b40	44	15
ghcr.io/strangelove-ventures/heighliner/cryptoorgchain:latest	sha256:0ebd463317fe6ed94db8446bd3be2d506d9f9cdf10089ef45af06fffd2c9910	44	15
ghcr.io/strangelove-ventures/heighliner/decentr:latest	sha256:9391f412ccbd25643931a2923f2f94bbcbfde6b8ed0520dec7747234caf22d20	44	15
ghcr.io/strangelove-ventures/heighliner/desmos:latest	sha256:e03902386175af86c7d1b46ec8f764961c13a881fc851abd37c2df65b5c6ffcb	44	15
ghcr.io/strangelove-ventures/heighliner/gravitybridge:latest	sha256:98ced74578439b5f1495454d96c82715f7e6bbe6a15b756c1a1d8d99e50f2dda	44	15
ghcr.io/strangelove-ventures/heighliner/impacthub:latest	sha256:a1dc65aaa58a46bba986b6e2591f7b0796f5372864a01a149d71ef0493447ef7	44	15
ghcr.io/strangelove-ventures/heighliner/likecoin:latest	sha256:e67a182d3650af78ee741a84ffea06fa4224e93a3105e60c98c5914725b675a2	44	15
ghcr.io/strangelove-ventures/heighliner/lum:latest	sha256:6aaa8c913939e81b65a46087424c4707be5d04627802cc5494b36baf3ccaaa6c	44	15

ghcr.io/strangelove-ventures/heightner/persistence:latest	sha256:94ac450b7b6753d753a866050c7a6c4f64c86238ddf63d56be2458e7c647a299	44	15
ghcr.io/strangelove-ventures/heightner/bitsong:latest	sha256:d6bbbde8ded95e543b6af75a1ff748e1bb30494f9428382b64811800d911f850	44	15
ghcr.io/strangelove-ventures/heightner/bostrom:latest	sha256:035d2660dcbd03ef38cf61ee68848605fc1a14cc7da95c432f24cff986d339c1	44	15

Sources

Jose Palafox, "Achieving SLSA 3 Compliance with GitHub Actions and Sigstore for Go Modules", Github Blog, 2022, available at <https://github.blog/2022-04-07-slsa-3-compliance-with-github-actions/>

Justin Hutchins, "Safeguard Your Containers with New Container Signing Capability in GitHub Actions". Github Blog, 2021, available at <https://github.blog/2021-12-06-safeguard-container-signing-capability-actions/>

Github Documentation, "Security Hardening for Github Actions", available at <https://docs.github.com/en/actions/security-guides/security-hardening-for-github-actions#using-codeowners-to-monitor-changes>, accessed June, 2022.

William Enck, Laurie Williams, "Top Five Challenges in Software Supply Chain Security: Observations From 30 Industry and Government Organizations", 2022, available at <https://ieeexplore.ieee.org/document/9740718>

Related Reading

Aaron Haymore, Iain Smart, Vikto Gazdag, Divya Natesan, and Jennifer Fernick, "10 real-world stories of how we've compromised CI/CD pipelines," NCC Group, 2022, available at <https://research.nccgroup.com/2022/01/13/10-real-world-stories-of-how-weve-compromised-ci-cd-pipelines/>, accessed July 8, 2022.

Adolfo Garcia Veytia, "What an SBOM Can Do for You," Chainguard Blog, 2022, available at <https://blog.chainguard.dev/what-an-sbom-can-do-for-you/>, accessed July 8, 2022.

Dan Geer, Bentz Tozer, and John Speed Meyers, "Counting Broken Links: A Quant's View of Software Supply Chain Security," *USENIX ;login.*, 2020, available at https://www.usenix.org/system/files/login/articles/login_winter20_17_geer.pdf, accessed July 8, 2022.

Executive Order on Improving the Nation's Cybersecurity, May 12, 2021, The White House, available at <https://www.whitehouse.gov/briefing-room/presidential-actions/2021/05/12/executive-order-on-improving-the-nations-cybersecurity/>, accessed July 8, 2022.

Government Accountability Office, "Cybersecurity: Federal Response to SolarWinds and Microsoft Exchange Incidents," 2022, available at <https://www.gao.gov/products/gao-22-104746>, accessed July 8, 2022.

Iosif Arvanitis, Grigoris Ntousakis, Sotiris Ioannidis, Nikos Vasilakis, "A Systematic Analysis of the Event-Stream Incident," 15th European Workshop on Systems Security (EUROSEC), 2022,

available at <https://www.ntousakis.com/p/es-eurosec.pdf>, accessed July 8, 2022.

Julius Musseau, John Speed Meyers, George P. Sieniawski, C. Albert Thompson, and Daniel German, "Is Open Source Eating the World's Software? Measuring the Proportion of Open Source in Proprietary Software Using Java Binaries," IEEE/ACM 19th International Conference on Mining Software Repositories, 2022, available at <https://ieeexplore.ieee.org/document/9796295>, accessed July 8, 2022.

Keith Jarvis and Jason Milletary, "Inside a Targeted Point-of-Sale Data Breach," Dell, 2014, available at <https://krebsonsecurity.com/wp-content/uploads/2014/01/Inside-a-Targeted-Point-of-Sale-Data-Breach.pdf>, accessed July 8, 2022.

Murugiah Souppaya, Karen Scarfone, and Donna Dodson, "[NIST Secure Software Development Framework Version 1.1](https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-218.pdf)," National Institute of Standards and Technology, 2022, available at <https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-218.pdf>, accessed July 8, 2022.

Sonatype, "2020 State of Software Supply Chain," 2020, available at <https://www.sonatype.com/resources/white-paper-state-of-the-software-supply-chain-2020>, accessed July 8, 2022.

SLSA "Supply Chain Threats," SLSA, available at <https://slsa.dev/spec/v0.1/#supply-chain-threats>, accessed July 8, 2022.

Trey Herr, William Loomis, Stewart Scott, and June Lee, "Breaking Trust: Shades of Crisis Across an Insecure Software Supply Chain," The Atlantic Council, 2020, available at <https://www.atlanticcouncil.org/in-depth-research-reports/report/breaking-trust-shades-of-crisis-a-cross-an-insecure-software-supply-chain/>, accessed July 8, 2022.