CM3070 Final Project - Preliminary Report

Esteban Garcia

Contents

1 Introduction

For my project I'm going to be using the non-profit web application template.

In the modern era of software development, the rapid adoption of third-party open-source libraries and containerised applications has revolutionised how software is built, deployed, and maintained. This shift has brought significant improvements on how we build software. However, it has also introduced new security challenges. As today's software supply chain increasingly depends on external dependencies, the potential attack surface for malicious actors has expanded dramatically.

Software supply chain security aims to address the risks associated with this by ensuring that the integrity, provenance, and authenticity of software artifacts are safeguarded. High-profile supply chain attacks, such as SolarWinds and the exploitation of vulnerabilities like Log4Shell, underscore the critical need to secure software delivery and have demonstrated how a single compromised artifact can cascade through an ecosystem, affecting millions of users and causing substantial damage.

To mitigate such risks, organisations and communities are adopting artifact management tools that enforce stricter security standards. These tools centralise the third-party dependencies and allow organisations to verify their origins, cryptographically sign artifacts and block software with known vulnerabilities.

The Open Container Initiative (OCI) [1] compliance plays a pivotal role in this effort, as it promotes interoperability and consistency in how software can be packaged, distributed, and executed.

OCI is a set of open-source standards designed to create a unified framework for container technology, focusing on interoperability and consistency across different container platforms.

These standards were developed mostly with container technology (docker) in mind but starting with the OCI v1.1 specification, now it can be used to store any type of artifact as long as the specs are followed.

Making it suitable to support different artifact formats beyond just Container Images, bringing then the benefits of the OCI standard to them.

Motivated by these considerations, my project aims to provide a secure platform for storing, managing, and distributing OCI artifacts.

Using OCI as a storage layer allows for my project to leverage its capabilities to store any kind of software, this reduces compatibility issues and fosters consistency in development and deployment environments.

It ensures that artifacts are verifiable and tamper-proof through tools like digital signatures and hash-based verification.

OCI-compliant tools integrate seamlessly with various ecosystem tools, such as container registries, Kubernetes, and CI/CD platforms. This interoperability makes it easier to share, store, and retrieve artifacts across diverse environments

By following these standards, artifacts become more portable and adaptable, allowing organizations to easily move and deploy artifacts across cloud environments, on-premise systems, or hybrid infrastructures

As there is a wide selection of different package formats in the ecosystem to prevent scope creep I will focus only on storage and distribution of Docker Images [2], Helm Charts [3] and PyPi Packages [4]. This last one is the only format that doesn't support the OCI protocol so a translation layer would need to be developed but it will serve as proof of the standard's capabilities to store and distribute any type of software.

2 Literature Review

2.1 Similar Projects

This section provides an analysis of existing solutions, the two projects I'm focusing in here are JFrog Artifactory [5] and Harbor [6]. These platforms have two different approaches to artifact management and container registry functionality.

JFrog Artifactory as a commercial, enterprise-grade solution with extensive support for different artifact formats, and Harbor as an open-source OCI compliant registry.

This analysis will provide a foundation for understanding the design and implementation decisions behind the project, demonstrating how it aims to bridge existing gaps and contribute to the evolution of secure artifact management practices.

2.1.1 JFrog Artifactory (commercial product)

JFrog Artifactory is a commercial, universal artifact repository manager. It supports a wide range of package formats (e.g., Docker images, Maven, npm, PyPI) and integrates into CI/CD pipelines. It is well-known for its scalability, enterprise-grade features, and extensive integrations.

The product offers no free version and is offered with both a cloud and a self-hosted solution. Its cloud-based product is not multi-tenant, meaning that JFrog needs to deploy standalone infrastructure for each customer, and its self-hosted solution is deployed and managed by the customer.

Due to its closed-source nature, and lack of multi-tenancy features, this means that organisations can't be aware of any security vulnerabilities in the platform and patching of these would require JFrog to implement the fix, notify their self-hosted customers and then apply the patch to each Cloud instance administered by them.

This last point is concerning as reported by Google in their latest report [7] we've been seeing the Time-To-Exploit metric being reduced from 32 days in 2021 & 2022 to 5 days in 2023, and 97% of them being exploited as zero-days vulnerabilities.

Artifactory classifies itself as an 'universal' artifact management platform, meaning it provides the capabilities to store any type of package format and new ones are being implemented all the time. But it uses its own Storage Architecture Layer and API that doesn't adhere to any open-source standard.

Artifactory is OCI-compliant but the standard is not used as the storage layer for all of its supported format but rather only for OCI-artifacts only.

So this means that custom JFrog-developed libraries have to be used to interface with its APIs for administering artifacts outside of each format's native tooling.

JFrog uses their own close-sourced static analysis vulnerability scanner called XRay [8], as it's a separate product it needs to be payed separately.

At the same time signature [9] of uploaded packages is an 'enterprise' only feature.

Artifactory is a very complete solution but only accesible to those with a big budget and requires a lot of administrative effort to create a secure supply chain for an organization's software.

2.1.2 Harbor (open-source project)

Harbor is an open-source container image registry, built by VMware [10] and now part of the Cloud Native Computing Foundation [11]. It is designed to enhance container image security and improve efficiency in managing containerized environments.

As it's an open-source project, Harbor benefits from community-driven development and transparency.

In contrast to Artifactory, Harbor is fully self-hosted and requires the operator to deploy and administer its infrastructure without an multi-tenant cloud-hosted option.

It's an fully OCI-compliant registry, this enables seamless integration with OCI-complaint tooling allowing developers/devops engineers to manage artifacts with either Harbor's Custom Tooling or native OCI tools. It only supports the upload of OCI artifacts, package formats that don't adhere to the standard can't be uploaded without additional custom automation.

It provides vulnerability scanning by connecting to Trivy [12] or Clair [13], two open-source static analysis tools, these tools have to be deployed separately by the administrator.

2.1.3 Conclusion

Both projects have extensive support and can be used to secure an organisation's software supply chain. Even though Artifactory is OCI-compliant its underlying storage architecture is not based on that technology, increasing the complexity for its users and its commercial nature limits its usage only to those with deep enough pockets to pay for it.

Harbor is fully OCI-compliant and open-source, it benefits from its transparency with the community but it focuses only on OCI/Container artifacts, doesn't support other package formats so it can't be used as a full solution to secure the supply chain.

2.2 Research Studies

In this section I evaluate different reserach studies that justify the development and usage of a project like the one I'm developing.

William Enck and Laurie Williams define the Three of the Top Five Challenges in Software Supply Chain Security [14] as:

- updating vulnerable dependencies
- choosing trusted supply chain dependencies
- securing the build process

For a developer to be able to secure their software, they need to be aware of vulnerabilities on their dependencies, this process has to be automated so a platform that is aware of a software's dependencies has to be used for this.

Choosing trusted supply chain dependencies is very important, what if a library your software is dependent on is deleted and someone takes the name? You don't only need to establish trust with the dependency you are using but also with the Package Index is coming from, most of the programming languages out there have some kind of public package index, PyPi for Python, NPM for NodeJS, there's good intentention behind these services but there's no guarantee that anything being uploaded there is safe as there is no vetting nor barrier to entry.

Package Typo-squatting, a form of social-engineering attack that targets users who incorrectly type a package's name, is becoming more normal in public package indexes, a good example discovered by a member of the Python community named Bertus [15] who reviewed a malicious package named 'pytz3-dev' seen in the Python package index (PyPi). This package targeted developers looking for the very popular 'pytz' package, it executed malicious code that searched for a SQLite database used by Discord the popular chat application, this database is used by Discord to store user information locally but also it includes tokens used to authenticate the user against the service.

So a developer should upload their dependencies to a trusted store where they have control over what goes in and what goes out.

Recently the NSA released a paper on recommended practices for developers on how to secure the software supply chain [16], as part of their recommended Secure Development Practices, they recommend that all third-party libraries are uploaded to a secure repository where they can be analysed for vulnerabilities on an on-going basis and an audit log of upload and downloads of libraries is kept.

2.3 OCI v1.1 Distribution Specification

Before I can start defining the design for my project there needs to be an understanding of the OCI v1.1 Distribution Specification [17]. I'm going to explain the basic functionality needed for my project to be conformant with the specification, without going into too much detail as the document referred is a better source for it.

This specification defines the API protocol to facilitiate and standardize the distribution of OCI artifacts, and it is design to be agnostic of content type. Container Images is the most prominent type but the protocol is open to any.

There are four different types of workflows as part of the specification:

- Pull - Clients are able to pull from the registry
- Push - Clients are able to push to the registry
- Content Discovery - Clients are able to list or otherwise query the content stored in the registry
- Content Management - Clients are able to control the full life-cycle of the content stored in the registry

2.3.1 Terms Definitions

Before going in detail into the different categories, there are certain terms that warrant basic definitions:

- Repository: a scope for API calls on a registry for a collection of content (including manifests, blobs, and tags).
- Blob: the binary form of content that is stored by a registry, addressable by a digest
- Digest: a unique identifier created from a cryptographic hash of a Blob's content, normally SHA-256
- Manifest: A JSON document uploaded to the registry, it serves as an index, providing information needed to retrieve and assemble the layers and configurations that make up an artifact.
- Layers: A list of descriptors for the artifact, each pointing to a specific uploaded blob
- Push: the act of uploading blobs and manifests to a registry
- Pull: the act of downloading blobs and manifests from a registry
- Tag: a custom, human-readable pointer to a manifest. A manifest digest may have zero, one, or many tags referencing it.

2.3.2 Pull

This is the process of pulling manifest and blobs from the registry and typically the first step in pulling an artifact is to retrieve its manifest as it contains all the information on what blobs need to be downloaded.

2.3.2.1 Pulling manifests

This is done by performing a 'GET' request to the registry to the URL `/v2/<name>/manifests/<reference>`

Where 'name' is the name of the repository and 'reference' either the digest of the manifest or a tag.

2.3.2.2 Pulling blobs

This is done by performing a 'GET' request to the registry to the URL `/v2/<name>/blobs/<digest>`

Where 'name' is the name of the repository and 'digest' is the blob's digest

2.3.2.3 Checking for content existance

To verify for the existance of either a blob or a manifest a 'HEAD' request to either the pull manifest or blobs URLs. A successful response would indicate that the content exists in the registry.

2.3.3 Push

This is the process of pushing manifests and blobs to the registry, and it works in the opposite order described previously, blobs are pushed first and the manifest last.

2.3.3.1 Pushing blobs

There are two ways to push blobs: chunked or monolithic. A monolithic upload happens as a single request to the registry, a chunked one consists of multiple upload request each one with a chunk of the blob. Each chunk is uploaded in order from start to finish.

The client decides which type of push mechanism wants to use, but as a general rule of thumb, monolithic pushes will be used to upload small blobs and chunked to upload very big ones.

There are two ways to push a blob monolithically, a POST request followed by a PUT request [18] or a single POST request [19].

Chunk uploads are similar to the first option for a monolithic upload, the first POST request starts an upload session, then multiple PUT requests are made to the registry each one with a chunk of the blob, afterwards the upload is finalised by a last PUT request that also includes the cryptographic hash of the blob, this signals to the registry that the upload session is finished.

2.3.3.2 Pushing manifests

To push a manifest a PUT request to the following URL is performed: `/v2/<name>/manifests/<reference>`

Where 'name' is the name of the repository and 'reference' either the digest of the manifest or a tag.

A manifest can have a different 'Content-Type' depending on the type of artifact being pushed and it is used to identify the type of artifact server-side.

Example 'application/vnd.oci.image.manifest.v1+json' determines that the pushed manifest is for a OCI Container Image.

This will be leveraged to identify what type of artifact we are receiving and also to create our own custom artifacts for those package formats that aren't supported out of the box by OCI.

The request contains the cryptographic hash of the manifest used to reference it and it can optionally contain a tag too.

2.3.4 Content Discovery

These set of endpoints are used to fetch information from the registry about available artifacts.

2.3.4.1 Listing Tags

To fetch the list of tags a GET request to the following URL is performed: `/v2/<name>/tags/list`

Where 'name' is the name of the repository. The registry has to returned all available tags for the specified repository.

2.3.4.2 Listing Referrers

This is a new endpoint released as part of the OCI v1.1 specification [20]. A manifest can refer to a different manifest creating a relationship between the two, this endpoint allows for fetching all related artifacts for a specific manifest.

This is good to attach things like signatures to an OCI artifact.

To fetch the list of referrers a GET request to the URL is performed: `/v2/<name>/referrers/<digest>`

Where 'name' is the name of the repository, and 'digest' is the digest of the manifest.

Upon success, the response must be a JSON body with an image index containing a list of descriptors

2.3.5 Content Management

Content management refers to the deletion of blobs, tags, and manifests. These are optional endpoints that a registry may implement.

3 Design

My project operates in the domain of software artifact management, focusing on creation, storage, management, and distribution of software artifacts, which are essential components of modern DevOps workflows.

The primary users of my project include:

- Software developers
- DevOps Engineers and Site Reliability Engineers
- Security Teams

4 Prototype

5 References

[1] OpenContainer Initiative, https://opencontainers.org/

[2] Docker Inc., https://docs.docker.com/get-started/docker-concepts/the-basics/what-is-an-image/

[3] Helm, https://helm.sh/docs/topics/charts/

[4] PyPi, https://pypi.org/

[5] JFrog Artifactory, https://jfrog.com/artifactory/

[6] Harbor, https://goharbor.io/

[7] 'How Low Can You Go? An Analysis of 2023 Time-to-Exploit Trends', Casey Charrier; Robert Weiner, October 15, 2024, Google, https://cloud.google.com/blog/topics/threat-intelligence/time-to-exploit-trends-2023

[8] JFrog Xray, https://jfrog.com/help/r/get-started-with-the-jfrog-platform/jfrog-xray

[9] 'Security Keys Management', JFrog, July 2024, https://jfrog.com/help/r/jfrog-platform-administration-documentation/security-keys-management

[10] VMWare Inc., https://www.vmware.com/

[11] Cloud Native Computing Foundation, https://www.cncf.io/

[12] Trivy Project, https://github.com/aquasecurity/trivy

[13] Clair Project, https://github.com/quay/clair

[14] 'Top Five Challenges in Software Supply Chain Security: Observations From 30 Industry and Government Organizations', William Enck;Laurie Williams, March 2022, https://ieeexplore.ieee.org/abstract/document/9740718

[15] 'Discord Token Stealer Discovered in PyPI Repository', Bertus, Jan 2019, https://bertusk.medium.com/discord-token-stealer-discovered-in-pypi-repository-e65ed9c3de06

[16] 'Securing the Software Supply Chain', NSA, August 2022, https://media.defense.gov/2022/Sep/01/2003068942/-1/-1/0/ESF_SECURING_THE_SOFTWARE_SUPPLY_CHAIN_DEVELOPERS.PDF

[17] 'Open Container Initiative Distribution Specification', Feb 2024, https://github.com/opencontainers/distribution-spec/blob/v1.1.0/spec.md

[18] 'POST followed by PUT Blob Upload', Feb 2024, https://github.com/opencontainers/distribution-spec/blob/main/spec.md#post-then-put

[19] 'Single POST Blob Upload', Feb 2024, https://github.com/opencontainers/distribution-spec/blob/main/spec.md#single-post

[20] 'Listing referrers', https://github.com/opencontainers/distribution-spec/blob/main/spec.md#listing-referrers