CD.FOUNDATION

ORTELIUS

Open-Source Vulnerability Management Platform

Proof of Concept

Why Use Ortelius

Developers, DevOps, and Security teams struggle to confirm that the software they deliver to end users is safe for consumption. Despite their incredible capabilities, cloud-native systems are often ill-prepared to combat code-level vulnerabilities. These environments involve large clusters of interconnected nodes running in parallel, making it more difficult to detect and patch vulnerabilities across the entire system without causing significant downtime. Ortelius is an open-source, centralized vulnerability management platform that clarifies these complexities, making what's hard about managing vulnerabilities easy. Ortelius is governed by the Continuous Delivery Foundation a part of the Linux Foundation.

Table of Contents

Ortelius POC Success Criteria

Implementing the Ortelius open source vulnerability management platform will ensure that teams can deliver secure, high-quality cloud-native applications at scale by exposing the following:

1 Versioning and Component to Application Dependency Management

Ortelius will track updates and create new versions of components, artifacts, that are being continuously pushed across the supply chain.

Ortelius will automatically create new logical application versions based on changes occurring at the lower component dependency level.

Ortelius will show the 'many-to-many' relationships between components and the logical applications that consume them.

2 Supply Chain Security

Ortelus will integrate into the DevOps pipeline consuming component-level Software Bill of Materials (SBOM) reports, deployment endpoint inventory, and real-time vulnerability reports for each new version of a component released across all assets. Ortelius will aggregate this information to the "logical" application level, sharing SBOM and deployment insights from all of the components the Application consumes.

3 Continuous Vulnerability Reporting

Leveraging the SBOM and deployment endpoint information, Ortelius will continuously scan for new vulnerabilities, using OSV.dev, mapping all vulnerabilties to the Component, Application, and run-time Endpoints an all system assets within the infrastructure.

4 Component and Open-Source Usage and Inventory

Ortelius will provide the ability to search for open-source packages across all logical applications.

Installing Ortelius One-Premise

Ortelius can be installed into your own cloud environment, or onto a hosted cloud environment. Ortelius uses Helm to manage and perform the installation. The process includes the installation of multiple containers. The Ortelius on-premise Helm chart and instructions can be found at ArtifactHub. This is the location for the most up to date instructions for downloading and running the Ortelius Helm chart. (https://artifacthub.io/packages/helm/ortelius/ortelius)

A SaaS Option

DeployHub, the core contributors of Ortelius, offers a free SaaS option. Sign-up for the DeployHub SaaS option at DeployHub.com/deployhub-team. You will be asked to enter a UserID/Password, Company and Project name. Your UserID/Password and Company name are unique. Once you login, your Project will be a found under your Company's high-level Domain.

Note: If another user signs up with the same Company name, they will be informed that they must contact the Administrator for access to your DeployHub account. The Administrator is the first person who signed up to DeployHub with that Company name.

Installing the CI/CD CLI for Pipeline Automation

Regardless if you are running the SaaS version or an Ortelius on-premise version, you will need to install the Oretelius CI/CD Command Line Interface (CLI) to automate the gather of supply chain data from your pipeline workflows.

The Ortelius CLI gathers supply chain data based on a single pipeline workflow at the build and deploy steps. The CLI will support any CI/CD engine, but does require Python. The build step gathers Swagger, SBOM, Read-me, licenses, Git data, Docker image, and other build output. The deploy step records when a release occurs, what was sent and where the objects were sent to.

To complete your POC you will need to install the Ortelius CLI where your CI/CD server is running. Refer to the Ortelius GitHub CLI Documentation (https://github.com/Ortelius/cli/blob/main/doc/dh.md) for installation instructions.

Ortelius CLI Data Gathering using the .toml File

The Ortelius CLI reads from a .toml file. The .toml file contains non-derived information for each artifact that you create at your build step. In Ortelius, an artifact is referred to as a Component. A Component is a Container, DB Object, or file object (.jar, Lamda Function, Apex file, etc.). The .toml file will provide the 'non-derived' data for the Component your are tracking in Ortelius which includes the Component name, owner, Component type, and owner contact details. The Ortelius CLI will read the .toml file from the Git Repository associated to your pipeline. If you are using a mono repository for your entire codebase, you will need a separate Component.toml file for each Component, managed in sub-directories.

In a cloud-native architecture there are many, if not hundreds, of Components. Organizing your Components within Ortelius is done in two ways. They are grouped based on a subject Domain and assigned to a logical Application. Not all Components need to be assigned to an Application, but they should be stored in a subject matter Domain so they can be easily found and reused.

A logical Application is a collection of Components that make up a complete software systems consumed by an end user. Applications are composed of shared Components and Application specific Components, and are a logical representation of what Components need to be deployed in order for the software system to run.

Note: Once created, your .toml file does not need to be updated unless the non-derived information changes, or you want to reorganize to which Applications or Domains the Component has been assigned. For example, a Component has been reassigned to a new owner and new team represented by a Domain or Application.

Steps for Running the Proof of Concept

To automate Ortelius, you will need to add it's data gathering to your CI/CD pipeline. The following steps will guide you through the process of implementing the Ortelius CLI to implement your Proof of Concept. Be sure you have installed the Ortelius CLI before you start.

Note: This POC does not include data gathering of the deployment for inventory tracking.

Step 1 - Define Your Ortelius Pipeline Variables

The following variables should be set at the beginning of your Pipeline.

DHURL - URL to Ortelius Login
DHUSER - The ID used to log into Ortelius
DHPASS - The password used to log into Ortelius. This can encrypted based on the CI/CD solution.
DOCKERREPO -Name of your Docker Repository .For Components that are Docker Images. Not needed for non-docker objects.
IMAGE_TAG - Tag for the Docker Image if used . For Components that are Docker Images. Not needed for non-docker objects.

Example:

```
export DHURL=https://console.ortelius.com
export DHUSER=Stella99
export DHPASS=chasinghorses
export DOCKERREPO=quay.io/ortelius/hello-world
export IMAGE_TAG=1.0.0
```

Step 2 - Create your Component.toml file

Cut and paste the following into a component.toml file, update 'your' information, and commit/push it to your Git Repository.

```
# Application Name and Version - not required. If not used the Component will not be associated to an Application
Application = "GLOBAL."your Application Name"
Application_Version = "your Application Version"

# Define Component Name, Variant and Version - required
Name = "GLOBAL.your Component Name"
Variant = "${GIT_BRANCH}"
Version = "vyour Component Version.${BUILD_NUM}-g${SHORT_SHA}"

# Key/Values to associate to the Component Version
[Attributes]
   DockerBuildDate = "${BLDDATE}"
   DockerRepo = "${DOCKERREPO}"
   DockerSha = "${DIGEST}"
   DockerTag = "${IMAGE_TAG}"
   DiscordChannel = "your Discord channel" or SlackChannel = "your Slack Channel"
   ServiceOwner = "${DHUSER}"
   ServiceOwnerEmail = "your Component Owner Email"
```

Example:

```
# Application Name and Version
Application = “GLOBAL.Santa Fe Software.Online Store Company.Hipster Store.Prod.helloworld app”
Application_Version = “1”

# Define Component Name, Variant and Version
Name = “GLOBAL.Santa Fe Software.Online Store Company”
Variant = “${GIT_BRANCH}”
Version = “v1.0.0.${BUILD_NUM}-g${SHORT_SHA}”

# Key/Values to associate to the Component Version
[Attributes]
   DockerBuildDate = “${BLDDATE}”
   DockerRepo = “${DOCKERREPO}”
   DockerSha = “${DIGEST}”
   DockerTag = “${IMAGE_TAG}”
   DiscordChannel = “ttps://discord.gg/wM4b5yEFzS”
   ServiceOwner= “${DHUSER}”
   ServiceOwnerEmail = “stella@DeployHub.io”
```

Note: For SaaS users, you will have a second high-level qualifier that was created as part of your sign-up. This second high-level qualifier must be used as the start of your Application Name and Component Name. For example: GLOBAL.Santa Fe Software.Online Store.

Step 3 - Add a step in your pipeline to run Syft if you are not generating SBOMS (Optional)

Ortelius can consume any SPDX and CycloneDX formatted SBOM. If you are already generating SBOMs, you will pass the name of the SBOM results to Ortelius is step 4 below. If you are not generating SBOMs as part of your pipeline process, you will need to add SBOM generation to collect the lower dependency data. Following is how to add Syft to your workflow to include the collection of SBOM data.

Syft SBOM tool (https://github.com/anchore/syft) will generate Software Bill of Material Reports for popular coding languages and package managers, including Docker images.

The following code example scans a Docker Image to generate the SBOM. See Syft Options (https://github.com/anchore/syft#supported-sources) to scan other objects and coding languages.

```
# install Syft
curl -sSfL https://raw.githubusercontent.com/anchore/syft/main/install.sh | sh -s -- -b $PWD

# create the SBOM
./syft packages $DOCKERREPO:$IMAGE_TAG --scope all-layers -o cyclonedx-json > cyclonedx.json

# display the SBOM
cat cyclonedx.json
```

Step 4 - Run the Ortelius CLI to add Your Component and Create an Application

Execute the following calls to the Ortelius CLI as part of your workflow. It should be called after the build and SBOM generation:

With CycloneDX SBOM

dh updatecomp --rsp component.toml --deppkg “cyclonedx@name of your SBOM file”

Example:
dh updatecomp --rsp component.toml --deppkg “cyclonedx@cyclonedx.json”

With SPDX SBOM

dh updatecomp --rsp component.toml --deppkg “spdx@name of your SBOM file. “

Example:
dh updatecomp --rsp component.toml --deppkg “spdx@spdx.json”

Without SBOM

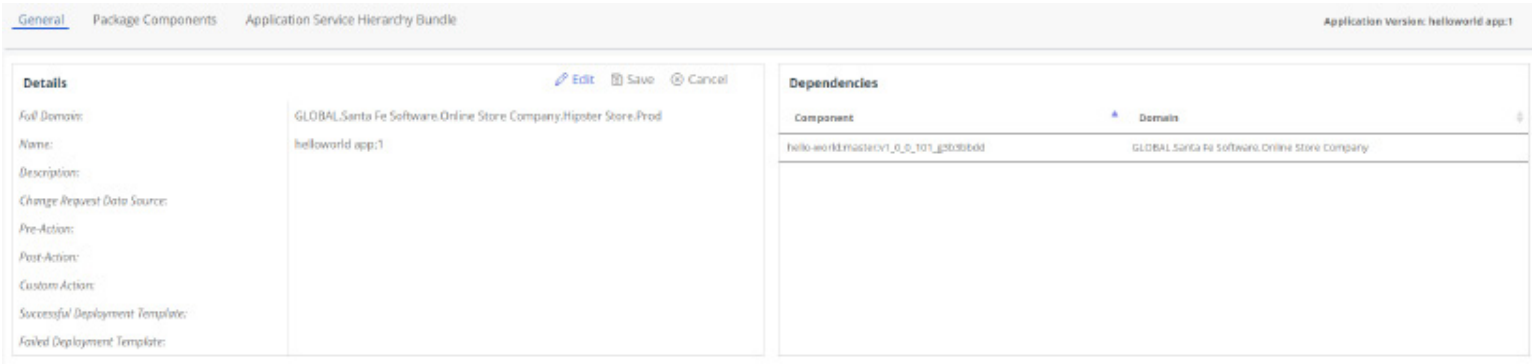
dh updatecomp --rsp component.toml

Expected Results

Bring up your Ortelius URL and login using the DHUSER and DHPASS from Step 1.

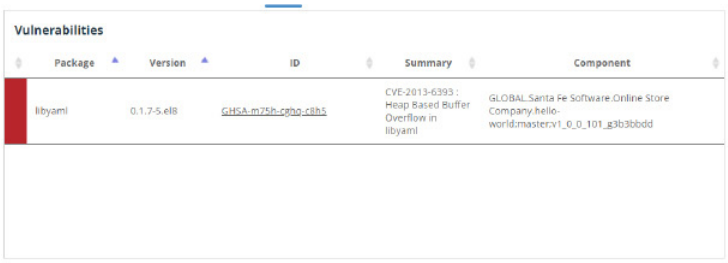
Application to Component Dependencies

Select Your Application from the 'Application View.' It should show you one Component as a dependency.



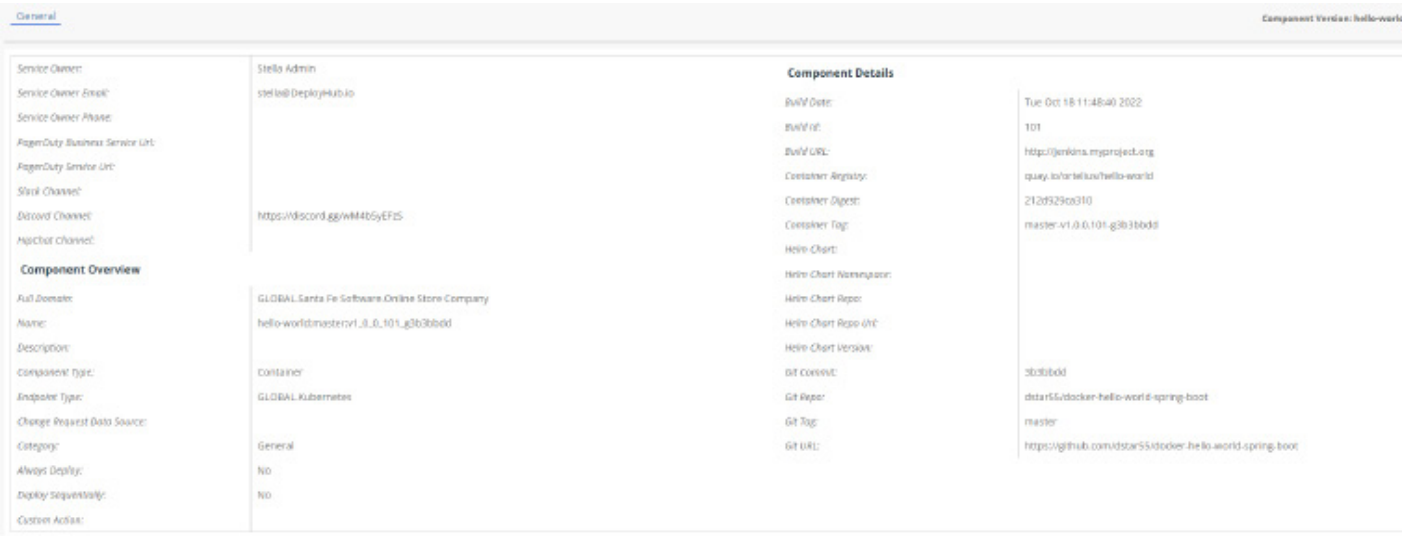
Application Level SBOM and CVE

Review the Applicaiton SBOM and vulnerabilities. *Note: CVE Results may vary depending on the time of the scan.*



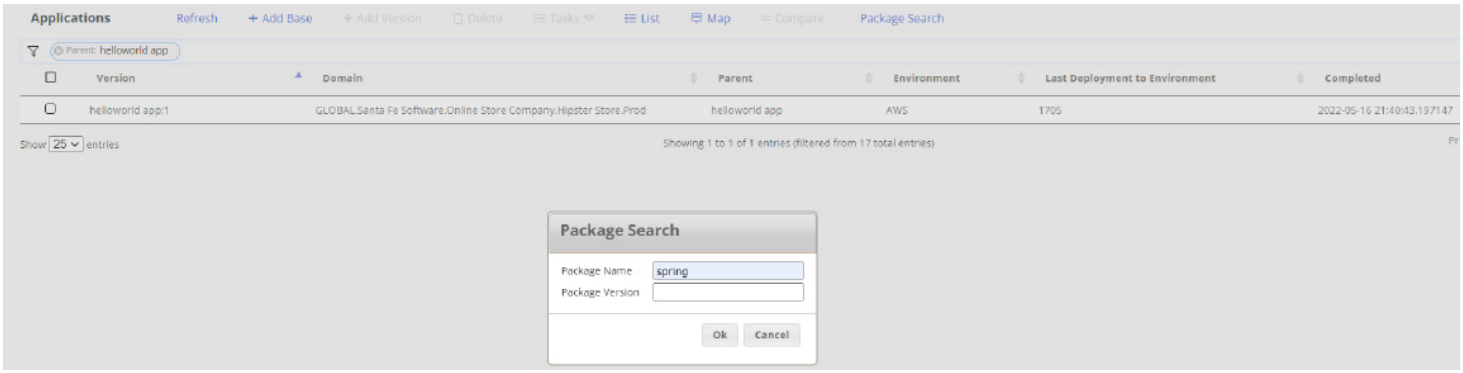
Component Ownership

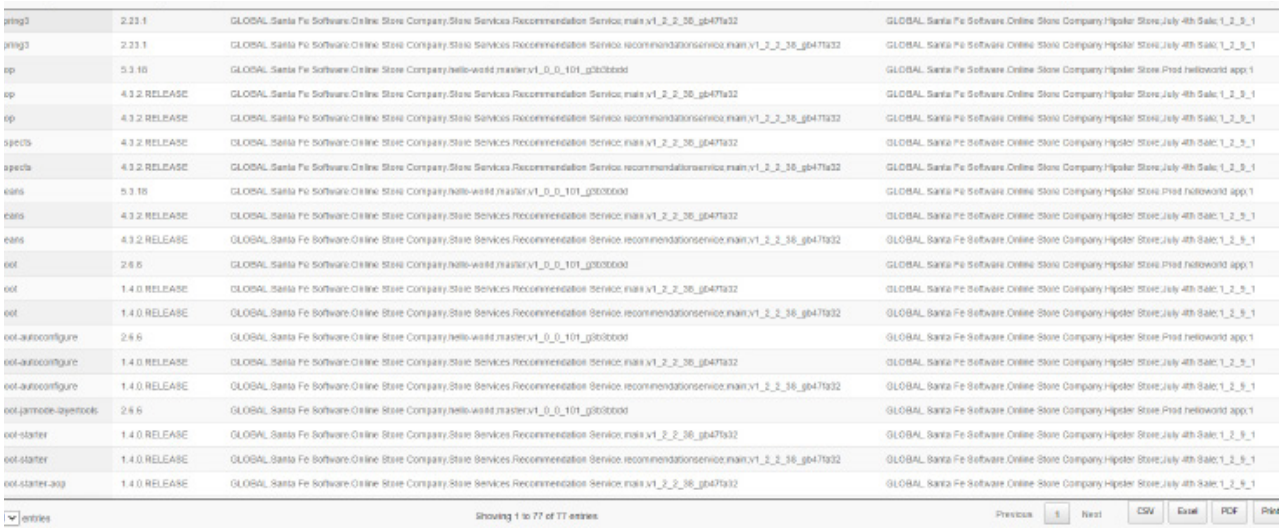
Go to the 'Component View'. You should see your Component Ownership and Detail, including its SBOM and vulnerabilities.



Package Search

Go to the 'Application View.' Select 'Package Search' from the high-level menu. Enter a package name such as 'spring' to identify all locations where the package is used.





Next Steps

After completing these initial POC steps, you can add additional Components to your Application, update them via your pipeline, and watch how Ortelius discovers new vulnerabilities on a daily basis. You can also add OpenSSF Scorecard reporting to track security compliance levels to Components and Applications.

Thank you for your interest in Ortelius.

Get Help

Report an Issue: https://github.com/ortelius/ortelius/issues

Community Discord Channel: https://discord.gg/wM4b5yEFzS

Ortelius Documentation: https://docs.ortelius.io/guides/

Get Involved in Open-Source

Help us create the best, open source software vulnerability management platofrm available at ortelius.io. We believe everyone has something to offer in solving the vulnerability management puzzle. We would love to have you on board.

CD.FOUNDATION

About the CD Foundation

The Continuous Delivery Foundation (CDF) serves as the vendor-neutral home of many of the fastest-growing projects for continuous integration/continuous delivery (CI/CD). It fosters vendor-neutral collaboration between the industry's top developers, end users and vendors to further CI/CD best practices and industry specifications. Its mission is to grow and sustain projects that are part of the broad and growing continuous delivery ecosystem.

Learn more about the Continuous Delivery Foundation at CD.Foundation