**DeviceBridge Security Scan Guide**

**Overview**

There are two versions of DeviceBridge -- On-Prem and SAAS -- and we need to manage security scans for each of these.

For code produced by Baxter, we need to perform Static Application Security Testing (SAST) scans, using Coverity, and Software Composition Analysis scans, using FOSSA.

For all container images, we also need to perform Software Composition Analysis scans, but for container scans we use Snyk, rather than FOSSA, because FOSSA's container scans fall short of scanning the contents of the entire container, currently.

We'll determine each version to be scanned by examining the container image versions to be included in the DeviceBridge release, and we'll track scans of code and container images produced by Baxter separate from 3rd-party container images.

Finally, I’ve created a spreadsheet named DeviceBridge\_Scan\_Tracking.xlsx that’s available to download from my OneDrive: [DeviceBridge\_Scan\_Tracking.xlsx](https://myfiles.baxter.com/:x:/g/personal/james_mallmann_baxter_com/ES5jB2FUZ5BItmBE6sXvIzsB-s0EozXbZQNT6YJGoWgEDw?e=jG0gkQ)

In this spreadsheet, I track Coverity, FOSSA, and Snyk scans needed for DeviceBridge On-Prem and DeviceBridge SAAS.

Scans for Baxter container images are tracked separately from scans for 3rd-party container images that are part of the release, where the container images – both Baxter container images and 3rd-party container images – included in the On-Prem release differ from the container images included in the SAAS release.

Baxter’s codebase and container scans are tracked in the same sheet – “BDHP Scan Tracking” – for both On-Prem and SAAS, while there is a separate sheet for each the list of On-Prem 3rd-party container images and the list of SAAS 3rd-party container images.

Currently, the 3rd-party container images sheets are named for the DeviceBridge version – “On-Prem 4.5.0 3rd-party images” and “SaaS 4.5.0 3rd-party images” – but I’d like to create/maintain a “Deployed Versions Archive” sheet for the 3rd-party images deployed per DeviceBridge version and reuse those two sheets for future DeviceBridge versions.

**Determining Versions to be Scanned**

**On-Prem**

For a list of images used for DeviceBridge On-Prem, start by visiting this page:

<https://lva-build.stg.baxter.com/offline/>

For preliminary scans, we're interested in filenames that use the following format:

<version>-beta.<YYYYMMDDhhmmss>-7.<number>.<number>-images.txt

For RC scans, the filenames use this format:

<version>-7.<number>.<number>-images.txt

*(I don't know what* <number>.<number> *represents, but I'm not sure that it matters.)*

Select all from that page and copy/paste the list into a text file in Notepad++ (or other text editor)

Sort the list alphabetically, ignoring case.

Separate the list into two parts: image paths that begin with "bdhpcr.azurecr.io" and image paths that don't.

The image paths that begin with "bdhpcr.azurecr.io" describe images that are composed from code written by Baxter.

Each image path is constructed as follows: <registry>/<repository>:<tag>

Consider this image path: bdhpcr.azurecr.io/images/bdhp-account:4.22.0

For that image path, the Registry is bdhpcr.azurecr.io, the Repository is images/bdhp-account, and the tag is 4.22.0

From the repository, we find the image name, which often -- but not always -- matches the name of the GitHub repository that produces the image.

The image tag is correlated with the tag in the GitHub repository, so the bdhpcr.azurecr.io/images/bdhp-account:4.22.0 image was created by the code at the 4.22.0 tag in the bdhp-account GitHub repo.

For On-Prem, there are currently 40 images from the bdhpcr.azurecr.io registry reported to be used; however, the following image is mistakenly reported:

bdhpcr.azurecr.io/images/**bdhp-k8s-cluster-config-sync-operator**:2.6.4

Other "bdhp-" repositories to be considered for scanning include **bdhp-dotnet-base** and **bdhp-dbmigration-base**, each of which produces a base container image for other BDHP container images to use, while the project that produces the bdhp-dbmigration-baseimage references the BDHP.Core NuGet package and, therefore, needs to be scanned by FOSSA.

After removing the "bdhpcr.azurecr.io" images from the list of On-Prem images, the images that remain are the 3rd-party images that will be deployed for the On-Prem release, and these images should be tracked in the “On-Prem 3rd-Party Images” sheet (currently named “On-Prem 4.5.0 3rd-Party Images”.

**SaaS**

To determine the versions of DeviceBridge (BDHP) repos to be scanned for DeviceBridge SAAS, one must start by connecting to the Kubernetes cluster for the environment to be examined, where the DeviceBridge Kubernetes clusters are named as follows: bdhp-#env#-eus-k8s.

Often, when approaching a release, the release candidate is deployed to STAGE, so this example will work with stage.

To connect to the Kubernetes cluster, go to the bdhp-stage-eus-k8s Kubernetes cluster page in Azure portal, and follow the instructions on the following page: <https://hill-rom.atlassian.net/wiki/spaces/BDHP/pages/4100554753/Login+to+the+Azure+Kubernetes+cluster>

Once connected, a call to kubectl get namespaces should return the following namespaces, among others:

* bdhp-ingress-haproxy
* bdhp-ingress-nginx
* bdhp-monitoring
* bdhp-platform
* bdhp-utilities
* bdhp-vault

To pull a list of container images deployed to these namespaces, one will need to execute two commands to pull details of containers and initialization containers (initContainers).

Here's the command for pulling a list of container images in all namespaces:

kubectl get pods ^

-Ao jsonpath="{range .items[\*]} ^

{@.metadata.namespace}{'\t'} ^

{@.spec.containers[\*].name}{'\t'} ^

{@.spec.containers[\*].image}{'\n'}{end}" ^

> BDHP\_SAAS\_ALL\_namespaces\_kubectl\_containers\_YYYYMMDDhhmm.txt

And here's the command for pulling a list of initialization container images in all namespaces:

kubectl get pods ^

-Ao jsonpath="{range .items[\*]} ^

{@.metadata.namespace}{'\t'} ^

{@.spec.initContainers[\*].name}{'\t'} ^

{@.spec.initContainers[\*].image}{'\n'}{end}" ^

> BDHP\_SAAS\_ALL\_namespaces\_kubectl\_initContainers\_YYYYMMDDhhmm.txt

This data is separated into three tab-separated columns:

* Namespace (@.metadata.namespace)
* Container Name (@.spec.containers[\*].name or @.spec.initContainers[\*].name)
* Image (@.spec.containers[\*].image or @.spec.initContainers[\*].image)

To understand the data being pulled, it helps to examine the source JSON, which may be pushed to a local file using the following command:

kubectl get pods -Ao json > BDHP\_SAAS\_all\_namespaces\_YYYYMMDDhhmm.json

In that JSON file, the second object, named "items", is an array of Kubernetes pod objects, and each pod object contains, a "metadata" object and a "spec" object.

From the **metadata** object, the earlier commands are pulling the values of the **namespace** field.

Working with the **spec** object is trickier, because the desired data is in the **containers** object, which is an array of container data -- even if there's only one container in the pod.

Within each container object in the **spec.containers** array, the desired fields are **name** and **image**.

To request that data for each object in the **spec.containers** array, one must drill down using @.spec.containers[\*], so name and image are requested using @.spec.containers[\*].name and @.spec.containers[\*].image, respectively.

And because it's possible for a pod to have multiple containers, the data in the columns I've named **Container Name** and **Image** may have ***multiple*** space-separated values.

Most of the information needed is in the container images file, which is typically hundreds of lines long.

From that file, copy the lines for the namespaces listed above into a new Excel spreadsheet at Cell A2.

I recommend using the following for column headers (Cells A1-C1):

|  |  |  |
| --- | --- | --- |
| **Namespace  @.metadata.namespace** | **Container Name  @.spec.containers[\*].name** | **Image  @.spec.containers[\*].image** |

Adjust the column width for all columns to match the contents of the columns.

Select the Container Name and Image columns and click the Wrap Text button (Home tab, Alignment section).

Find rows having a Container Name with "helloworld" in the name and delete those rows; then delete the Namespace and Container Name columns.

Next, sort the remaining Image column, and then go to the Excel Data tab, select Remove Duplicates from the Data Tools section, and click OK; usually, this removes several duplicate images.

Now, let's address the multi-valued fields that contain spaces.

First, select Cell A1, copy them, move to Cell B1, and Paste Column Widths (Ctrl + Alt + V, then W, then Enter), and then hit Enter to paste the column header text, itself.

Then, add a formula to Cell B2 to find instances of multi-valued fields in the Image column and replace each space with a new line (and if no space is found, then copy the data from the Image column):

=IF(NOT(ISERROR(FIND(" ", A2))), SUBSTITUTE(A2, " ", "

"), A2)

Select Cell B2, click the Wrap Text button, then double-click the fill handle (the + in the lower-right corner of the cell) to copy down the column, and then hit Ctrl + B twice to ensure that any cells that now have multiple lines are now expanded.

If not already selected, select all the data in Column B, copy it, and Paste Values (Ctrl + Alt + V, then V, then Enter), and then resize Column B to fit the updated data and delete Column A.

Next, each row that contains multiple images needs to be converted to having one line per image.

Frankly, the easiest way to do this is to copy the data from Column A into Notepad++, replace all double-quotes with nothing, Select All, Copy, go back to the spreadsheet, select Cell A2, and Paste.

As we did earlier, sort the column and Remove Duplicates, and we’re left with a distinct list of container images for the DeviceBridge SaaS deployment.

This distinct list of images should be split into two lists: images from the "bdhpcr.azurecr.io" registry should be tracked in the “BDHP Scan Tracking” sheet, and all other images should be tracked in the “SAAS 3rd-Party Images” sheet (currently named “SAAS 4.5.0 3rd-Party Images”).

**Convert Images to Registry, Repository, and Tag**

We’re still working in the spreadsheet we created for the DeviceBridge SaaS container images, and in that spreadsheet, add the following column headers to Cells B2 through E2:

|  |  |  |  |
| --- | --- | --- | --- |
| **Registry** | **Repository** | **Tag** | **Digest** |

***NOTE:*** *In DeviceBridge SAAS, for whatever reason, the registry.k8s.io/ingress-nginx/controller image has typically been declared as tag@digest.*

*For example, this image is currently declared as follows:*

registry.k8s.io/ingress-nginx/controller:v1.11.3@sha256:d56f135b6462cfc476447cfe564b83a45e8bb7da2774963b00d12161112270b7

In Cell B2, add the following formula to extract the Registry:

=IF(NOT(ISERROR(FIND(".io", A2))),LEFT(A2,FIND(".io", A2)+2),"docker.io")

In Cell C2, add this formula to extract the Repository:

=IF(NOT(ISERROR(FIND(".io", A2))), MID(A2, FIND("/", A2)+1, FIND(":", A2)-FIND("/", A2)-1), LEFT(A2, FIND(":", A2)-1))

In Cell D2, this formula will extract the Tag:

=IF(ISERROR(FIND("@", A2)), RIGHT(A2, LEN(A2)-FIND(":", A2)), MID(A2, FIND(":", A2)+1, FIND("@", A2)-FIND(":", A2)-1))

And in Cell E2, the following formula will extract the Digest:

=IF(ISERROR(FIND("@", A2)), "", RIGHT(A2, LEN(A2)-FIND("@", A2)))

Select Cells B2 through E2, and double-click the fill handle to copy down.

If not already selected, select all the data in Columns B, C, D, and E, copy it, and Paste Values (Ctrl + Alt + v, then v, then Enter).

**Scan Tracking**

I've created a spreadsheet named DeviceBridge\_Scan\_Tracking.xlsx to both track the execution of the various scans and to help generate commands for executing those scans.

In this spreadsheet, there's a sheet named "Common Data", and it contains the following:

|  |  |
| --- | --- |
| **Name** | **Value** |
| DeviceBridge Version | 4.5.0 |
| On-Prem Snyk Slug | bdhp-v4.5.0 |
| SAAS Snyk Slug | bdhp-v4.5.0-saas |
| DeviceBridge Container Registry | bdhpcr.azurecr.io |
| DeviceBridge Container Repository | images |
| GLOBAL Domain Username | mallmaj |
| On-Prem SBOM Base Output Path | \Users\mallmaj\Desktop\BDHP-4.5.0\SBOMs\On-Prem |
| SAAS SBOM Base Output Path | \Users\mallmaj\Desktop\BDHP-4.5.0\SBOMs\SAAS |
| Local path to BDHP repos | \repos\github-hillrom\bdhp |
| SBOM Format | cyclonedx1.6+json |

You'll need to review/update the values with names highlighted in green: "DeviceBridge Version", "GLOBAL Domain Username", and "Local path to BDHP repos".

In the sheet named "BDHP Scan Tracking", I've listed the "bdhp-" repositories that are deployed as part of the DeviceBridge On-Prem and/or SAAS solutions, with the first two columns describing the **Repo Name** and the **Container Image Name**.

The next several columns contain data for the following filters:

* On-Prem?
* SaaS?
* On-Prem or SaaS?
* SAST (Coverity)?
* SCA (FOSSA)?
* Produces Container Image?
* SBOM?
* Language

Then there are columns for "Helm Chart Name" -- rarely, if ever, needed -- and "Dockerfile Path" -- used for Snyk container scans.

The values in the columns mentioned thus far should rarely need to change.

After that are two important columns: "On-Prem Version to Test" and "SaaS Version to Test".

And they’re followed by a column titled “On-Prem matches SaaS?”.

Of course, the version of each repository to be tested should be updated to match the version deployed for the version in question.

**Executing Scans**

**Coverity (SAST) Scans**

Coverity scans for DeviceBridge are executed using a Jenkins server, with all pipelines for DeviceBridge found here: <https://mdap-test-jenkins-prod.aws.baxter.com/job/SDC/>

**##### TO BE COMPLETED #####**

**Preparing Local Repos for SCA Scans**

For both SCA and Container SCA scans, it’s important that one’s local copy of each repository being scanned is at the version tag matching the version being deployed for the DeviceBridge version being released.

For example, for DeviceBridge version 4.5.x, at the time of writing, version 5.78.2 of the bdhp-catalog repository/container is to be deployed, but if I pull the latest code on the dev branch and the latest tag is newer than 5.78.2, then the code I have locally won’t truly represent version 5.78.2.

In addition, if the FOSSA scan is executed from the DEV branch, it’s more difficult to know which version was scanned.

So, I always create a local branch that matches the code for the Git tag that matches the version being deployed.

For bdhp-catalog, this would involve executing the following commands from the base of the local bdhp-catalog repository, starting from the dev branch:

git pull origin dev

git fetch

git checkout -b 5.78.2-tag-as-branch 5.78.2

Usually, I perform this work assembly line style, in stages.

First, I run through all the repos and make sure each is on the dev branch.

Next, I run through the repos again to pull and fetch the latest and list the branches (because sometimes I’ve already created a tag-based branch for the repo version).

In the DeviceBridge Scan Tracking spreadsheet, I have a column titled “Git Pull Latest and Fetch” that constructs a command for each repo, like this one for bdhp-catalog:

cd ../bdhp-catalog && git pull origin dev && git fetch && git branch

When I executed this command locally, the following output was displayed:

A computer screen shot of a computer program

AI-generated content may be incorrect.

In this output, one can see that there’s no branch named “5.78.2-tag-as-branch”, so it needs to be created.

The spreadsheet also contains commands for creating the tag-based branch for each repo in columns named “On-Prem Git Checkout Tag as Branch” and “SAAS Git Checkout Tag as Branch”.

Note that only one of the two commands should be needed, per repo, as the version deployed to On-Prem should match that deployed for SAAS; however, some repos are deployed to only one or the other; thus, the two columns.

The command for creating the new branch in the local copy of bdhp-catalog is as follows:

cd ../bdhp-catalog && git checkout -b 5.78.2-tag-as-branch 5.78.2

And when I executed this command locally, this output was displayed:

A screenshot of a computer

AI-generated content may be incorrect.

**FOSSA (SCA) Scans**

FOSSA scans are executed from one’s own computer.

To execute these scans, one needs to have the FOSSA CLI, and I’ve found that the FOSSA CLI works best from a Linux-based OS.

Thus, on Windows machines, I highly recommend installing the FOSSA CLI on an instance of Ubuntu installed on WSL2.

Instructions for installing the FOSSA CLI can be found here: <https://docs.fossa.com/docs/importing-a-project>

Also, to execute FOSSA CLI commands, one must have a FOSSA API Token, preferably with Full Access, rather than Push Only.

API Tokens are managed here: <https://app.fossa.com/account/settings/integrations/api_tokens>

On that page, click the “Add new token” button, and in the modal window that’s displayed, give the token a name, make sure the “Push Only” checkbox is unchecked, and click the Confirm button.

Copy the key and create an environment variable named FOSSA\_API\_KEY with value set to the value of the new token.

**.NET/C# Projects**

In the DeviceBridge Scan Tracking spreadsheet, filtering the Language column by C# results in 35 rows being displayed, so C# is the most common language being used, by far, and each of these projects must be built before executing a FOSSA scan.

This is done by executing the following commands using the dotnet CLI:

dotnet clean -c Release

dotnet build -c Release

In addition, there must be a file at the root of each repository named **.fossa.yml** (note the leading dot).

Unfortunately, the .fossa.yml file that’s currently checked in to each .NET/C# repository is incorrect, as the exclude section should ***not*** exclude the obj folder under each project.

For example, the .fossa.yml file checked into the bdhp-catalog repository looks like this:

version: 3

project:

id: bdhp-catalog

team: BDHP

paths:

only:

- CatalogService

- CatalogService.Core

- CatalogService.Domain

- CatalogService.Repository

exclude:

- CatalogService/obj

- CatalogService.Core/obj

- CatalogService.Domain/obj

- CatalogService.Repository/obj

However, in this case, the entire exclude section should be removed, so the file looks like this:

version: 3

project:

id: bdhp-catalog

team: BDHP

paths:

only:

- CatalogService

- CatalogService.Core

- CatalogService.Domain

- CatalogService.Repository

For each repository, once the .fossa.yml file is corrected, and the projects are built, the FOSSA scan may be executed.

Continuing with version 5.78.2 of the bdhp-catalog as an example, the FOSSA scan would be executed using the following command:

fossa analyze -T BDHP -t bdhp-catalog -r 5.78.2

**IMPORTANT: fossa analyze Command Details**

Now, the Team Name (-T) parameter probably wouldn’t need to be provided in the command, as it’s declared in the .fossa.yml file, but I highly recommend including the Project Title (-t) parameter, explicitly, rather than relying on the value in that file.

Why? Well, currently, some “bdhp-\*” Git repositories are configured to execute a FOSSA scan upon merging a pull request, and for the bdhp-catalog repo, such scans are pushed to the FOSSA project with the following Project Locator: git+github.com/Hillrom-Enterprise/bdhp-catalog

And if the Project Title (-t) parameter is ***not*** provided in the fossa analyze command, then the scan will be added to that git+github.com/Hillrom-Enterprise/bdhp-catalog FOSSA project.

However, when the Project Title (-t) parameter ***is*** provided in the fossa analyze command, then the scan will be added to the FOSSA project with this Project Locator: custom+19518/bdhp-catalog

At this time, I prefer to keep the scans executed manually via the FOSSA CLI separate from the scans executed upon merging a pull request in Git.

In the DeviceBridge Scan Tracking spreadsheet, there are columns titled **On-Prem fossa analyze Command** and **SAAS fossa analyze Command** that, for .NET/C# projects, contain the dotnet and fossa CLI commands, like this for bdhp-catalog:

cd ../bdhp-catalog

dotnet clean -c Release

dotnet build -c Release

fossa analyze -T BDHP -t bdhp-catalog -r 5.78.2

Note that the revision parameter revision (-r) parameter of each fossa analyze command should be changed to match the needs of the release.

**JavaScript Projects**

Currently, DeviceBridge has two repositories containing JavaScript projects: bdhp-enterprise-configuration and bdhp-manual-adt-ui.

Normally, working with a JavaScript project would be easy; however, FOSSA does not yet work correctly when the lockfileVersion in the **package-lock.json** file is greater than or equal to three (3), as FOSSA includes many devDependencies in the analysis when the lockfile version matches one of the newer versions.

Therefore, instead of performing a simple installation of npm packages, using npm i, we need to perform a ***special*** installation for which the lockfileVersion is declared to be 2:

npm i --lockfile-version 2 --package-lock-only

Once this command has been executed, the command to have FOSSA analyze the repo may be executed.

For the **bdhp-enterprise-configuration** repo, the package.json and package-lock.json files live in the EnterpriseConfiguration directory, so the sequence of calls, starting in the root of the repository, should be as follows:

cd EnterpriseConfiguration

npm i --lockfile-version 2 --package-lock-only

cd ..

fossa analyze -T BDHP -t bdhp-enterprise-configuration -r 5.4.8

And for the bdhp-manual-adt-ui repo, the package.json and package-lock.json files live in the ManualAdtUIService\web directory, so the sequence of calls, starting in the root of the repository, should be as follows:

cd ManualAdtUIService\web

npm i --lockfile-version 2 --package-lock-only

cd ..\..

fossa analyze -T BDHP -t bdhp-manual-adt-ui -r 4.9.1

Of course, the value provided for the revision (-r) parameter of each fossa analyze command should be changed to match the needs of the release.

**PHP Projects**

Currently, DeviceBridge has two repositories containing PHP projects: bdhp-iqecs and bdhp-provisioner.

FOSSA is reasonably accurate when determining which packages are devDependencies that should be excluded; however, based on my own analysis of which packages should be included, FOSSA may include a few dev dependency packages.

**Python Projects**

Currently, DeviceBridge has only one repository containing a Python project: bdhp-metrics.

Unfortunately, FOSSA doesn’t handle Python projects very well.

In the bdhp-metrics repository, there’s a file named requirements.txt in the Metrics directory.

The contents of this file, at the 4.2.4 tag, are as follows:

requests==2.32.3

jsonpath-ng==1.6.0

kubernetes==30.1.0

And if we call fossa analyze from the root of our local bdhp-metrics directory, then FOSSA will analyze those three direct dependencies, but it won’t scan the transitive dependencies (the dependencies of the direct dependencies).

Therefore, what I’ve had to do to get FOSSA to scan all the dependencies for a Python project is update my local copy of that requirements.txt file to include the exact version of each transitive dependency required by the project.

To determine the exact versions of the transitive dependencies to be added, we must look at the Azure build pipeline output for the version we want FOSSA to analyze.

Let’s say that version is 4.2.4, for which the Azure pipeline is **bdhp-metrics-build**, and the output for the 4.2.4 build may be found here: <https://dev.azure.com/hrc-zenith/IAC%20-%20BDHP/_build/results?buildId=202812&view=logs&j=3dc8fd7e-4368-5a92-293e-d53cefc8c4b3&t=440a145c-148e-58c1-7881-71912bd6595e>

In the build output, find the stage labeled “Build an image with argument”, click on it to view its output, and then click on the “View raw log” button in the upper right of the output.

Here’s the URL for the raw log of the “Build and image with argument” stage for version 4.2.4 of the bdhp-metrics-build pipeline: <https://dev.azure.com/hrc-zenith/c5175c51-7c21-4b51-b6ac-bae6756e2297/_apis/build/builds/202812/logs/14>

In the raw log, there’s a call that includes pip install, after which the Python packages are installed.

When I search for the word “Collecting”, I find 19 log entries that describe the Python packages to be downloaded for this project, where the first looks like this:

2024-10-24T12:53:00.5289366Z #7 7.911 Collecting requests==2.32.3 (from -r requirements.txt (line 1))

If I strip “Collecting ”, along with everything before it, from each of those 19 lines, the list of log entries looks like this:

requests==2.32.3 (from -r requirements.txt (line 1))

jsonpath-ng==1.6.0 (from -r requirements.txt (line 2))

kubernetes==30.1.0 (from -r requirements.txt (line 3))

charset-normalizer<4,>=2 (from requests==2.32.3->-r requirements.txt (line 1))

idna<4,>=2.5 (from requests==2.32.3->-r requirements.txt (line 1))

urllib3<3,>=1.21.1 (from requests==2.32.3->-r requirements.txt (line 1))

certifi>=2017.4.17 (from requests==2.32.3->-r requirements.txt (line 1))

ply (from jsonpath-ng==1.6.0->-r requirements.txt (line 2))

six>=1.9.0 (from kubernetes==30.1.0->-r requirements.txt (line 3))

python-dateutil>=2.5.3 (from kubernetes==30.1.0->-r requirements.txt (line 3))

pyyaml>=5.4.1 (from kubernetes==30.1.0->-r requirements.txt (line 3))

google-auth>=1.0.1 (from kubernetes==30.1.0->-r requirements.txt (line 3))

websocket-client!=0.40.0,!=0.41.\*,!=0.42.\*,>=0.32.0 (from kubernetes==30.1.0->-r requirements.txt (line 3))

requests-oauthlib (from kubernetes==30.1.0->-r requirements.txt (line 3))

oauthlib>=3.2.2 (from kubernetes==30.1.0->-r requirements.txt (line 3))

cachetools<6.0,>=2.0.0 (from google-auth>=1.0.1->kubernetes==30.1.0->-r requirements.txt (line 3))

pyasn1-modules>=0.2.1 (from google-auth>=1.0.1->kubernetes==30.1.0->-r requirements.txt (line 3))

rsa<5,>=3.1.4 (from google-auth>=1.0.1->kubernetes==30.1.0->-r requirements.txt (line 3))

pyasn1<0.7.0,>=0.4.6 (from pyasn1-modules>=0.2.1->google-auth>=1.0.1->kubernetes==30.1.0->-r requirements.txt (line 3))

Notice that each of these log entries consists of the requirement being satisfied, along with a parenthetical explaining why the package is being installed.

For the first three lines, the parenthetical begins “from -r requirements.txt”, and those three packages match the packages declared in the requirements.txt file!

For each of the other packages, the parenthetical begins with the requirement that depends on the package to be downloaded, and we can use this information to construct a dependency tree.

* **requests**==2.32.3 (from -r requirements.txt (line 1))
  + **certifi**>=2017.4.17 (from requests==2.32.3->-r requirements.txt (line 1))
  + **charset-normalizer**<4,>=2 (from requests==2.32.3->-r requirements.txt (line 1))
  + **idna**<4,>=2.5 (from requests==2.32.3->-r requirements.txt (line 1))
  + **urllib3**<3,>=1.21.1 (from requests==2.32.3->-r requirements.txt (line 1))
* **jsonpath-ng**==1.6.0 (from -r requirements.txt (line 2))
  + **ply** (from jsonpath-ng==1.6.0->-r requirements.txt (line 2))
* **kubernetes**==30.1.0 (from -r requirements.txt (line 3))
  + **google-auth**>=1.0.1 (from kubernetes==30.1.0->-r requirements.txt (line 3))
    - **cachetools**<6.0,>=2.0.0 (from google-auth>=1.0.1->kubernetes==30.1.0->-r requirements.txt (line 3))
    - **pyasn1-modules**>=0.2.1 (from google-auth>=1.0.1->kubernetes==30.1.0->-r requirements.txt (line 3))
      * **pyasn1**<0.7.0,>=0.4.6 (from pyasn1-modules>=0.2.1->google-auth>=1.0.1->kubernetes==30.1.0->-r requirements.txt (line 3))
    - **rsa**<5,>=3.1.4 (from google-auth>=1.0.1->kubernetes==30.1.0->-r requirements.txt (line 3))
  + **python-dateutil**>=2.5.3 (from kubernetes==30.1.0->-r requirements.txt (line 3))
  + **pyyaml**>=5.4.1 (from kubernetes==30.1.0->-r requirements.txt (line 3))
  + **oauthlib**>=3.2.2 (from kubernetes==30.1.0->-r requirements.txt (line 3))
  + **requests-oauthlib** (from kubernetes==30.1.0->-r requirements.txt (line 3))
  + **six**>=1.9.0 (from kubernetes==30.1.0->-r requirements.txt (line 3))
  + **websocket-client**!=0.40.0,!=0.41.\*,!=0.42.\*,>=0.32.0 (from kubernetes==30.1.0->-r requirements.txt (line 3))

Back in the raw log, for each “Collecting” log entry there are two corresponding “Downloading” log entries: one for downloading the \*.whl.metadata file and a second for downloading the \*.whl file.

Fortunately, the set of Downloading log entries for downloading the \*.whl files are listed together, directly after the last \*.whl.metadata file is downloaded. (I don’t know that this will always be true, but it’s likely going to be easier to grab the log entries for downloading the \*.whl files.)

That list of log entries looks like this:

Downloading requests-2.32.3-py3-none-any.whl (64 kB)

Downloading jsonpath\_ng-1.6.0-py3-none-any.whl (29 kB)

Downloading kubernetes-30.1.0-py2.py3-none-any.whl (1.7 MB)

━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.7/1.7 MB 90.2 MB/s eta 0:00:00

Downloading certifi-2024.8.30-py3-none-any.whl (167 kB)

Downloading charset\_normalizer-3.4.0-cp313-cp313-musllinux\_1\_2\_x86\_64.whl (144 kB)

Downloading google\_auth-2.35.0-py2.py3-none-any.whl (208 kB)

Downloading idna-3.10-py3-none-any.whl (70 kB)

Downloading oauthlib-3.2.2-py3-none-any.whl (151 kB)

Downloading python\_dateutil-2.9.0.post0-py2.py3-none-any.whl (229 kB)

Downloading PyYAML-6.0.2-cp313-cp313-musllinux\_1\_1\_x86\_64.whl (751 kB)

━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 751.6/751.6 kB 68.2 MB/s eta 0:00:00

Downloading six-1.16.0-py2.py3-none-any.whl (11 kB)

Downloading urllib3-2.2.3-py3-none-any.whl (126 kB)

Downloading websocket\_client-1.8.0-py3-none-any.whl (58 kB)

Downloading ply-3.11-py2.py3-none-any.whl (49 kB)

Downloading requests\_oauthlib-2.0.0-py2.py3-none-any.whl (24 kB)

Downloading cachetools-5.5.0-py3-none-any.whl (9.5 kB)

Downloading pyasn1\_modules-0.4.1-py3-none-any.whl (181 kB)

Downloading rsa-4.9-py3-none-any.whl (34 kB)

Downloading pyasn1-0.6.1-py3-none-any.whl (83 kB)

Each of the \*.whl files follow this naming convention:

{distribution}-{version}(-{build tag})?-{python tag}-{abi tag}-{platform tag}.whl

Thus, the package versions downloaded for version 4.2.4 of bdhp-metrics are as follows:

|  |  |
| --- | --- |
| **Package Name** | **Version** |
| requests | 2.32.3 |
| jsonpath\_ng | 1.6.0 |
| kubernetes | 30.1.0 |
| certifi | 2024.8.30 |
| charset\_normalizer | 3.4.0 |
| google\_auth | 2.35.0 |
| idna | 3.10 |
| oauthlib | 3.2.2 |
| python\_dateutil | 2.9.0.post0 |
| PyYAML | 6.0.2 |
| six | 1.16.0 |
| urllib3 | 2.2.3 |
| websocket\_client | 1.8.0 |
| ply | 3.11 |
| requests\_oauthlib | 2.0.0 |
| cachetools | 5.5.0 |
| pyasn1\_modules | 0.4.1 |
| rsa | 4.9 |
| pyasn1 | 0.6.1 |

Of course, the first three packages are listed in the requirements.txt file, but we need to add a line for each of the other 16 packages – dependencies of the three direct dependencies – to our local copy of requirements.txt before asking FOSSA to analyze the project!

This results in the local copy of requirements.txt looking like this:

requests==2.32.3

jsonpath-ng==1.6.0

kubernetes==30.1.0

certifi==2024.8.30

charset\_normalizer==3.4.0

google\_auth==2.35.0

idna==3.10

oauthlib==3.2.2

python\_dateutil==2.9.0.post0

PyYAML==6.0.2

six==1.16.0

urllib3==2.2.3

websocket\_client==1.8.0

ply==3.11

requests\_oauthlib==2.0.0

cachetools==5.5.0

pyasn1\_modules==0.4.1

rsa==4.9

pyasn1==0.6.1

With the changes to the requirements.txt file saved, the FOSSA command may now be executed:

fossa analyze -T BDHP -t bdhp-metrics -r 4.2.4

**Snyk (Container SCA) Scans**

Unfortunately, the Baxter-approved tool for scanning container images, FOSSA, falls woefully short of reporting everything included in certain container images, so the DeviceBridge team uses Snyk to scan for and monitor container image vulnerabilities.

In Snyk, the names of their groupings are odd, as the top level is **Tenant**, followed by **Group**, and **Organization**.

Currently, Baxter’s Tenant is named “Baxter International Pilot”, and the Group has that same name.

However, each “Organization” represents a ***release*** of DeviceBridge (BDHP), and each Organization has many Projects, where each project, the way we’re using Snyk, is a container.

For DeviceBridge, I’ve usually created an Organization named for the version and a second that has the same name with “-SAAS” appended.

For example, for version 4.5.0 of DeviceBridge I created “BDHP v4.5.0” and “BDHP v4.5.0-SAAS”, and the “slugs” for these – used for the --org parameter in CLI commands – are bdhp-v4.5.0 and bdhp-v4.5.0-saas, respectively.

In preparation for using Snyk for container scans, please take the following steps

* Start Docker, if not already running
* Login to Hillrom’s Azure (az login --tenant myhillrom.onmicrosoft.com)
* Login to the bdhpcr Container Registry (az acr login -n bdhpcr)
* Authenticate with Snyk (snyk auth)

To ask Snyk to scan *and monitor* a container image, one can use the following command (at a minimum):

snyk container monitor --org=<org-name> <image-path>

Thus, continuing with version 5.78.2 of bdhp-catalog, the command to request a container scan of the bdhp-catalog image for version 4.5.0 of DeviceBridge On-Prem would be, *at a minimum*, the following:

snyk container monitor --org=bdhp-v4.5.0 bdhpcr.azurecr.io/images/bdhp-catalog:5.78.2

However, for our own container images, we should also provide Snyk with the path to the Dockerfile, which gives Snyk further insight into the details of the container image, and the updated command looks like this:

snyk container monitor ^

--org=bdhp-v4.5.0 ^

--file=\repos\github-hillrom\bdhp\bdhp-catalog\CatalogService\Dockerfile ^

bdhpcr.azurecr.io/images/bdhp-catalog:5.78.2

|  |  |
| --- | --- |
| Now, we provide a tag for that image that matches the tag/version in Git, and that tag is reported in the Snyk UI, as shown to the right.  However, I have found no way to make that image tag value appear in a vulnerability report. | A screenshot of a computer  AI-generated content may be incorrect. |

Therefore, we must add a tag to the Snyk scan, which may be done by adding a “version=*<image-version>*” tag via the CLI command:

snyk container monitor ^

--org=bdhp-v4.5.0 ^

--tags=version=5.78.2 ^

--file=\repos\github-hillrom\bdhp\bdhp-catalog\CatalogService\Dockerfile ^

bdhpcr.azurecr.io/images/bdhp-catalog:5.78.2

Because DeviceBridge consists of many container images, I highly recommend performing container scans in batches, using PowerShell.

Now, one can do this by simply putting the snyk container monitor commands in a PowerShell file and executing the file in PowerShell; however, the output won’t include the commands being executed.

To demonstrate this, I created a file named execute\_snyk\_container\_monitor\_commands\_directly.ps1 that has the following content:

snyk container monitor `

--org=bdhp-v4.5.0 `

--tags=version=4.22.0 `

--file=\repos\github-hillrom\bdhp\bdhp-account\AccountService\Dockerfile `

bdhpcr.azurecr.io/images/bdhp-account:4.22.0

snyk container monitor `

--org=bdhp-v4.5.0 `

--tags=version=5.9.0 `

--file=\repos\github-hillrom\bdhp\bdhp-adt\ADTService\Dockerfile `

bdhpcr.azurecr.io/images/bdhp-adt:5.9.0

And the output from executing this file is as follows:

A screenshot of a computer screen

AI-generated content may be incorrect.

Alternatively, I created a custom PowerShell function named Invoke-Snyk-Container-Monitor (found in Appendix B) for which the call to perform the same snyk container monitor command for bdhp-catalog would be executed using this command:

Invoke-Snyk-Container-Monitor `

-organization "bdhp-v4.5.0" `

-tags "version=5.78.2" `

-image "bdhpcr.azurecr.io/images/bdhp-catalog:5.78.2" `

-dockerfilePath "\repos\github-hillrom\bdhp\bdhp-catalog\CatalogService\Dockerfile"

I created a file named execute\_invoke-snyk-container-monitor\_cmds.ps1 that has the following content:

Try

{

Invoke-Snyk-Container-Monitor `

-organization "bdhp-v4.5.0" `

-tags "version=4.22.0" `

-image "bdhpcr.azurecr.io/images/bdhp-account:4.22.0" `

-dockerfilePath "\repos\github-hillrom\bdhp\bdhp-account\AccountService\Dockerfile"

Invoke-Snyk-Container-Monitor `

-organization "bdhp-v4.5.0" `

-tags "version=5.9.0" `

-image "bdhpcr.azurecr.io/images/bdhp-adt:5.9.0" `

-dockerfilePath "\repos\github-hillrom\bdhp\bdhp-adt\ADTService\Dockerfile"

}

Catch

{

Write-Host "ERROR OCCURRED" -f Yellow

Write-Host $\_ -f Red

}

And the output from executing this file is as follows:

A screenshot of a computer screen

AI-generated content may be incorrect.

In the DeviceBridge\_Scan\_Tracking.xlsx spreadsheet, I created columns that produce Invoke-Snyk-Container-Monitor commands for the container images that need to be scanned. (See the sheets named “BDHP Scan Tracking”, “On-Prem 4.5.0 3rd-Party Images”, and “SaaS 4.5.0 3rd-Party Images”.)

**Producing Vulnerability Reports**

**Coverity**

**FOSSA**

Honestly, the Vulnerability Report provided by FOSSA is lousy.

When I requested a Vulnerability Report for version 4.22.0 of bdhp-account, FOSSA generated a four-page PDF to report three vulnerabilities.

**Snyk**

In Snyk, start by making sure the “Organization” being viewed is the one for which a vulnerability report is needed, and then click on the Reports option in the menu on the left.

|  |  |
| --- | --- |
| The default report is named “Issues Detail”, but one may change the report using the “Change Report” link to the right of the report name 🡪 | A close up of a computer screen  AI-generated content may be incorrect. |

The default view of this page looks like this:

A screenshot of a computer

AI-generated content may be incorrect.

Of particular interest are the items in the red boxes, which are the “Add filter” option near the top left, and the “Modify Columns” and “Download CSV” buttons to the right

|  |  |  |
| --- | --- | --- |
| To create the desired report, we’ll start by clicking the “Add filter” option.  In the panel that’s displayed, click the “Show all issues filters” link to display the rest of the “Issues” filters, and scroll down that list to find and select the “Type” filter.  In the Issue Type filter panel, select only the **Vulnerability** filter. |  |  |

Next, click the “Modify Columns” button and making the following changes (with the columns listed in the order in which they appear in the dropdown box):

* **Remove** INTRODUCED
* Add INTRODUCTION CATEGORY
* Add FIXED IN AVAILABLE
* Add FIXED IN VERSION
* Add ATTACH VECTOR
* Add SNYK CVSS CODE
* Add NVD SCORE
* Add NVD SEVERITY
* Add PACKAGE NAME AND VERSION
* Add PROBLEM ID
* Add PROJECT TAGS
* Add PROJECT TARGET
* Add PROJECT TYPE
* Add REACHABILITY
* Add SEMVER VULNERABILITY RANGE
* Add VULNERABILITY PUBLICATION DATE

Once the Issue Type filter has been applied and the columns to include in the report have been configured, click on the Download CSV button.

When I did this while writing, Snyk produced a CSV with the following name: “snyk\_issues\_detail\_03\_25\_2025\_89221135-b715-441b-bd62-d37943dc96c2.csv”.

Notice that there is no obvious indication of the “Organization” with which the data in the CSV is associated, so I highly recommend renaming the file upon downloading it: I named mine “bhdp-v4.5.0\_snyk\_vulnerability\_report\_202503250908.csv”, where the name before the first underscore matches the Snyk slug for the “Organization”.

Upon opening the downloaded CSV in Excel, one should see columns with the following names (and because some columns don’t correlate to an option in the “Modify Columns” dropdown, I note differences in parentheses):

* ISSUE\_SEVERITY\_RANK (automatic – provided even when SEVERITY is unchecked)
* ISSUE\_SEVERITY (automatic – provided even when SEVERITY is unchecked)
* SCORE
* PROBLEM\_TITLE (automatic)
* CVE
* CVE\_URL (CVE)
* CWE
* PROJECT\_NAME (PROJECT)
* PROJECT\_URL (PROJECT)
* EXPLOIT\_MATURITY
* COMPUTED\_FIXABILITY
* PRODUCT\_NAME (SNYK PRODUCT)
* PROJECT\_TAGS
* PROJECT\_TYPE
* PROBLEM\_ID
* INITIAL\_ISSUE\_TYPE (PROBLEM ID)
* VULN\_DB\_URL (PROBLEM ID)
* PACKAGE\_NAME\_AND\_VERSION
* PROJECT\_TARGET
* ATTACK\_VECTOR
* SNYK\_CVSS\_SCORE
* NVD\_SEVERITY
* NVD\_SCORE
* FIXED\_IN\_AVAILABLE
* FIXED\_IN\_VERSION
* SEMVER\_VULNERABLE\_RANGE
* INTRODUCTION\_CATEGORY
* VULNERABILITY\_PUBLICATION\_DATE
* REACHABILITY
* ISSUE\_URL (automatic)
* ISSUE\_STATUS\_INDICATOR (automatic)
* ISSUE\_TYPE (automatic)

Historically, I’ve done my best to update this CSV into a proper Excel spreadsheet that’s easy for others to navigate and use.

Please start by activating filtering (Ctrl + Shift + L), auto-sizing all columns, making the column headers **bold**, and freezing the header row (**View** tab, **Freeze Panes** button, then **Freeze Top Row** option).

Also, for better readability, Select All and vertically center all cells

A screenshot of a computer

AI-generated content may be incorrect.

**IMPORTANT**

Check the filter options in the ISSUE\_TYPE column, which should be the last column to the right.

The only filter option should be “Vulnerability”, but I know I sometimes forget to add the Issue Type filter to my report before downloading.

Thus, if there’s an ISSUE\_TYPE option named “License”, then select it to view all license rows, and delete those rows.

Now, let’s **remove** the following columns:

* CVE\_URL
* PROJECT\_URL
* INITIAL\_ISSUE\_TYPE
* ISSUE\_URL
* ISSUE\_STATUS\_INDICATOR (which is always “Open”)
* ISSUE\_TYPE

Find the VULN\_DB\_URL column and move it to be the last column.

Next, find the PROJECT\_TARGET column and move it to be the first column, and then find the PROJECT\_TAGS column and move it to appear after PROJECT\_TARGET.

Columns C, D, and E should now be ISSUE\_SEVERITY\_RANK, ISSUE\_SEVERITY, and SCORE, respectively.

Please center the data in these columns, but not the headers. (Please know that centering column headers with filtering activated often forces auto-sized columns to be wider.)

Also, center the data in the ATTACK\_VECTOR, SNYK\_CVSS\_SCORE, NVD\_SEVERITY, and NVD\_SCORE columns (which should appear consecutively).

Find all the VULNERABILITY\_PUBLICATION\_DATE column, select all data in that column, and apply the following custom number format: yyyy-mm-dd.

**Data Cleanup**

**CVE & CWE**

Unfortunately, some of the data needs to be cleaned up a bit, and we’ll start with the CVE and CWE columns.

Please insert two columns to the right of the CWE column, as we’ll use these columns to reformat the data in the CVE and CWE columns.

Please execute the following steps:

Filter the CVE column to display only rows containing [] and delete that text from this column in those rows.

Clear that filter, and then do the same thing in the CWE column.

At this point, I have the CVE column in Column G and the CWE column in Column H, with two empty columns in Columns I & J.

In Cell I2, enter the following formula:

=SUBSTITUTE(SUBSTITUTE(SUBSTITUTE(G2, "[""", ""), """]", ""), """,""", "

")

This removes the double-quotes and square brackets from the value in the CVE column, specifically Cell G2, and replaces any comma with a new line.

Copy the formula to Cell J2, so the same substitutions are made on the value in the CWE column, specifically Cell H2.

To finish the data transformations, please proceed with the following steps:

1. Filter Column G (the CVE column) to exclude blank cells
2. Copy Cell I2
3. Paste in all data cells in Column I
4. Select all data cells in Column I (if not already selected)
5. Remove the filter from Column G
6. ***With all cells in Column I still selected***, copy those cells (Ctrl +C), paste their values (Ctrl + Alt + V, then V, then Enter), and change the selected cells to have Wrapped Text (**Home** tab, **Alignment** section, **Wrap Text** button).

Then perform the same procedure using Columns H & J.

Once both CVE and CWE data has been transformed, cut the headers from Columns G & H, move them to Columns I & J, auto-resize Columns I & J, and then delete Columns G & H.

**Package Name and Version**

The PACKAGE\_NAME\_AND\_VERSION column should now be Column N, and some rows in that column contain multiple comma-separated values that I’d rather see displayed on separate rows (because that makes the column much narrower).

Please start by inserting a column to the right of the PACKAGE\_NAME\_AND\_VERSION column, which should produce an empty Column O.

In Cell O2, add the following formula:

=SUBSTITUTE(N2, ",", "

")

This formula replaces each comma with a new line.

With Cell O2 selected, double-click the fill handle to copy the formula to the remaining rows.

Next, select all data in Column O, copy those cells (Ctrl +C), paste their values (Ctrl + Alt + V, then V, then Enter), and change the selected cells to have Wrapped Text (**Home** tab, **Alignment** section, **Wrap Text** button).

Finally, move the header from Column N to Column O, auto-resize Column O, and delete Column N.

**“Fixed In” Version & SemVer Vulnerable Range**

The data in the FIXED\_IN\_VERSION and SEMVER\_VULNERABLE\_RANGE columns is like the data that was in the CVE & CWE columns in that square brackets surround one or more values in double-quotes, with multiple values being comma-separated.

Here again, I prefer each value to appear in its own row.

I have these two columns in Columns T & U, so please adjust the instructions, if needed.

Please start by inserting two columns to the right of Column U.

In Cell V2, enter the following formula:

=SUBSTITUTE(SUBSTITUTE(SUBSTITUTE(T2, "[""", ""), """]", ""), """,""", "

")

Again, this removes the double-quotes and square brackets from the value in the FIXED\_IN\_VERSION column, specifically Cell T2, and replaces any comma with a new line.

Copy the formula to Cell W2, so the same substitutions are made on the value in the CWE column, specifically Cell U2.

To finish the data transformations, please proceed with the following steps:

1. Filter Column T (the FIXED\_IN\_VERSION column) to exclude blank cells
2. Copy Cell V2
3. Paste in all data cells in Column V
4. Select all data cells in Column V (if not already selected)
5. Remove the filter from Column T
6. ***With all cells in Column V still selected***, copy those cells (Ctrl +C), paste their values (Ctrl + Alt + V, then V, then Enter), and change the selected cells to have Wrapped Text (**Home** tab, **Alignment** section, **Wrap Text** button).
7. Clear the filter from Column T

Then perform the same procedure using Columns U & W.

Once both FIXED\_IN\_VERSION and SEMVER\_VULNERABLE\_RANGE data has been transformed, cut the headers from Columns T & U, move them to Columns V & W, auto-resize Columns V & W, and then delete Columns T & U.

**Add NIST URL**

Finally, please add a column to the right of the existing columns with the header **NIST URL**.

This should be Column Z.

Before adding any data to this column, please filter the CVE column (Column G) to exclude any rows where that column contains a blank.

Next, back in the new NIST URL column, add the following formula to the first cell under the header, which, for me, is Cell Z2:

="https://nvd.nist.gov/vuln/detail/"&G2

Copy Cell Z2, and paste it into the remaining cells in Column Z.

Clear the filter from Column G

Then copy Column Z and paste its values in its place.

**Update Column Headers**

Please update the column headers to the following:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Project Target** | **Project Tag(s)** | **Severity  Rank** | **Snyk Severity** | **Snyk Score** | **Problem Title** | **CVE** | **CWE** | **Project Name** |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Exploit Maturity** | **Computed  Fixabiliity** | **Product Name** | **Project Type** | **Package Name and Version** | **Attack Vector** |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Snyk  CVSS  Score** | **NVD  Severity** | **NVD  Score** | **"Fixed In"  Available?** | **"Fixed In" Version** | **Semantic Version  Vulnerable Range** | **Introduction Category** | **Vulnerability  Publication  Date** |

|  |  |  |
| --- | --- | --- |
| **Reachability** | **Snyk Issue Analysis URL** | **NIST URL** |

**Producing SBOMs**

**Codebase (FOSSA)**

For each DeviceBridge codebase repository, we must produce an SBOM, and the Baxter-approved tool for this is FOSSA.

While there’s a command in the FOSSA CLI to request an SBOM (fossa report attribution), this command doesn’t provide options (at the time of writing) to allow one to *customize* the contents of the SBOM.

However, the FOSSA UI in the browser provides the options needed to create the desired SBOM.

Continuing with version 5.78.2 of bdhp-catalog as an example, the FOSSA scan results for this version are found at this URL:

<https://app.fossa.com/projects/custom%2b19518%2fbdhp-catalog/refs/branch/5.78.2-tag-as-branch/5.78.2>

On that Summary page, there’s an Actions menu on the right, in which there’s a link to Generate SBOM Report: click that link.

|  |  |  |
| --- | --- | --- |
| Once on the SBOM page, find the Format dropdown in the Customization column on the right, select CycloneDX (JSON) for the format, and click the Apply button in the modal window that appears below: | A screenshot of a computer  AI-generated content may be incorrect. | |
| Once the list of Customization checkboxes has been updated for CycloneDX (JSON), update those checkboxes so only the following boxes are checked:  **Included *(heading)***   * + Direct Dependencies   + Transitive Dependencies   + Support Status   + First Party Licenses   + Full License List   + Copyrights   + Copyrights from non-license text   **Dependency Metadata *(heading)***   * + Package   + Authors   + Discovered License(s)   + Package Download URL   Once the checkboxes are configured to match the list above, click the Apply button in the modal window, and once the customization has been applied, click the Download button (above the Customization section header).  The downloaded file will have a ridiculous name like “project-attribution-custom+19518\_bdhp-catalog$5.78.2-1742317795017.bom.json”.  I updated this one to “FOSSA\_SBOM\_bdhp-catalog\_5.78.2.json”, and I recommend that each of these files be named “FOSSA\_SBOM\_*[repo-name]*\_*[version]*.json”. | | A screenshot of a computer  AI-generated content may be incorrect. |

**Correcting FOSSA SBOMs**

|  |  |
| --- | --- |
| Unfortunately, each SBOM that FOSSA produces has a metadata section that must be updated.  Please review the beginning of a FOSSA SBOM produced for version 6.9.2 of the bdhp-provisioner repository, shown to the right. 🡪  Notice that "FOSSA, Inc." is declared as the name of both the manufacture and the supplier metadata.  But when Snyk produces SBOMs, Snyk is only mentioned in the tools metadata object:  "metadata": {  "timestamp": "2025-03-21T13:43:14Z",  "tools": {  "services": [  {  "provider": {  "name": "Snyk"  },  "name": "SBOM Export API",  "version": "v1.110.0"  }  ]  },  I was curious what each of these metadata objects is supposed to contain, so I turned to the CycloneDX v1.6 JSON Reference (<https://cyclonedx.org/docs/1.6/json/#metadata>).  Here are the descriptions of the relevant metadata objects:   * tools: “The tool(s) used in the creation, enrichment, and validation of the BOM.” * manufacture: [Deprecated] This will be removed in a future version. Use the @.component.manufacturer instead. The organization that manufactured the component that the BOM describes. * supplier: The organization that supplied the component that the BOM describes. The supplier may often be the manufacturer, but may also be a distributor or repackager. | {  "$schema": "http://cyclonedx.org/schema/bom-1.6.schema.json",  "bomFormat": "CycloneDX",  "specVersion": "1.6",  "version": 1,  "serialNumber": "urn:uuid:3be4209a-bf4e-4a5e-8830-6a5f6d7c1d47",  "metadata": {  "timestamp": "2025-03-21T14:34:41.804Z",  "authors": [  {  "name": "Baxter"  }  ],  "component": {  "type": "application",  "name": "bdhp-provisioner\_new",  "version": "6.9.2",  "bom-ref": "custom+19518/bdhp-provisioner\_new$6.9.2",  "supplier": {  "name": "Organization: Baxter"  },  "description": "Project uploaded via Provided Builds from fossa-cli"  },  "manufacture": {  "name": "FOSSA, Inc.",  "url": [  "https://app.fossa.com"  ],  "contact": [  {  "name": "FOSSA, Inc.",  "email": "support@fossa.com"  }  ]  },  "supplier": {  "name": "FOSSA, Inc.",  "url": [  "https://app.fossa.com"  ],  "contact": [  {  "name": "FOSSA, Inc.",  "email": "support@fossa.com"  }  ]  }  }, |
| Well, FOSSA is *neither* the organization that manufactured the component that the BOM describes *nor* the organization that supplied the component that the BOM describes, but FOSSA *is* the tool used in creation of the BOM.  Thus, we need to update the metadata in the FOSSA SBOMs by adding the tools object and removing both the manufacture and supplier objects.  What I’ve done is replace the authors metadata object with new tools and manufacturer objects and deleted the manufacture and supplier objects. | "metadata": {  "timestamp": "2025-03-21T14:34:41.804Z",  "tools": {  "services": [  {  "provider": {  "name": "FOSSA",  "url": [  "https://app.fossa.com"  ],  "contact": [  {  "name": "FOSSA, Inc.",  "email": "support@fossa.com"  }  ]  },  "name": "FOSSA CLI",  "version": "3.9.38"  }  ]  },  "manufacturer": [  {  "name": "Baxter"  }  ],  "component": {  "type": "application",  "name": "bdhp-provisioner\_new",  "version": "6.9.2",  "bom-ref": "custom+19518/bdhp-provisioner\_new$6.9.2",  "supplier": {  "name": "Organization: Baxter"  },  "description": "Project uploaded via Provided Builds from fossa-cli"  }  }, |

Of course, executing this change 30+ times for all the FOSSA SBOMs would be tedious, so I recommend using Notepad++ to perform Replace operations on all files in the codebase SBOM directories.

In Notepad++, do the following for the first Replace:

1. Open the **Find in Files** dialog (Ctrl + Shift + F)
2. Set the **Search Mode** to **Extended** (Alt + X).
3. Enter \*.json for **Filters**
4. Select the directory containing FOSSA SBOMs for **Directory**
5. Set the **“Find what”** box to the following:  
   "authors": [\n {\n "name": "Baxter"\n }\n ],
6. Set the **“Replace with”** text to this:  
   "tools": {\n "services": [\n {\n "provider": {\n "name": "FOSSA",\n "url": [\n "https://app.fossa.com"\n ],\n "contact": [\n {\n "name": "FOSSA, Inc.",\n "email": "support@fossa.com"\n }\n ]\n },\n "name": "FOSSA CLI",\n "version": "3.9.38"\n }\n ]\n },\n "manufacturer": [\n {\n "name": "Baxter"\n }\n ],
7. Click the **Find All** button to check how many replacements to expect.
8. If the expected number of results are returned, re-open Find in Files, make sure the “Find what” box contains the desired text, click the **“Replace in Files”** button, and click OK on the “Are you sure?” dialog that appears.

Then, execute the second Replace, clearing the value from the “Replace with” box, and setting the “Find what” box to the following:

,\n "manufacture": {\n "name": "FOSSA, Inc.",\n "url": [\n "https://app.fossa.com"\n ],\n "contact": [\n {\n "name": "FOSSA, Inc.",\n "email": "support@fossa.com"\n }\n ]\n },\n "supplier": {\n "name": "FOSSA, Inc.",\n "url": [\n "https://app.fossa.com"\n ],\n "contact": [\n {\n "name": "FOSSA, Inc.",\n "email": "support@fossa.com"\n }\n ]\n }

**Containers (Snyk)**

For each container included in the DeviceBridge release – both Baxter-created and 3rd-Party – we must produce an SBOM.

Unfortunately, the Baxter-approved tool for producing SBOMs, FOSSA, falls woefully short of reporting everything included in certain container images, so **the DeviceBridge team uses Snyk to produce SBOMs for container images**.

Here’s an example of how to request an SBOM for a container image from Snyk:

snyk container sbom `

--org=bdhp-v4.5.0 `

--format=cyclonedx1.6+json `

bdhpcr.azurecr.io/images/bdhp-catalog:5.78.2 > `

\Users\mallmaj\Desktop\BDHP-4.5.0\On-Prem\SBOMs\bdhp\_container\Snyk\_container\_SBOM\_bdhpcr.azurecr.io\_\_images\_bdhp-catalog\_\_5.78.2.json

The --org option, here is the same as for the snyk container monitor command, and the --format command declares the SBOM format to use.

The output of the snyk container sbom command is pushed to a local file, and this is important – otherwise, the SBOM output is displayed in the console!

As with the snyk container monitor command, I’ve found it better to use a custom PowerShell function to execute the snyk container monitor command, and the custom function named New-Snyk-Container-Sbom can be found in Appendix B.

And, here too, I highly recommend requesting container SBOMs in batches, using PowerShell.

In the DeviceBridge\_Scan\_Tracking.xlsx spreadsheet, I created columns that produce New-Snyk-Container-Sbom commands for the container images that need to be scanned. (See the sheets named “BDHP Scan Tracking”, “On-Prem 4.5.0 3rd-Party Images”, and “SaaS 4.5.0 3rd-Party Images”.)

Please note that I use the following pattern for naming the Snyk SBOM files to be produced: Snyk\_container\_SBOM\_*<registry>*\_\_*<repository>*\_\_*<tag>*.json

**Merging SBOMs**

While it would be nice to simply pass along all the individual SBOMs produced for each codebase repository and each container image, project managers need to archive a ***single*** SBOMs as part of the release documentation (though it’s good to pass along the individual SBOMs, too.)

Fortunately, there’s a CycloneDX command line tool to help us merge SBOMs, and that can be downloaded here: <https://github.com/CycloneDX/cyclonedx-cli/releases>

I downloaded the cyclonedx-win-x64.exe file and put it in the “C:\Program Files\CycloneDX” directory on my computer.

I also added “C:\Program Files\CycloneDX” to my PATH environment variable to allow me to easily call the cyclonedx-win-x64 on the command line from anywhere.

|  |  |  |
| --- | --- | --- |
| By the time I’ve produced all the SBOMs for DeviceBridge version 4.5.0, I have the directory structure shown to the right.  Typically, I start by merging the SBOMs in each of the 3rd-Party\_container, bdhp\_container, and codebase directories, so I end up with the following files in the directories under BDHP-4.5.0:   * On-Prem\SBOMs   + 3rd-Party\_container\BDHP\_4.5.0\_On-Prem\_Snyk\_SBOM\_ALL\_3rd-Party\_container.json   + bdhp\_container\BDHP\_4.5.0\_On-Prem\_Snyk\_SBOM\_ALL\_BDHP\_container.json   + codebase\BDHP\_4.5.0\_On-Prem\_FOSSA\_SBOM \_ALL\_codebase.json * SAAS\SBOMs   + 3rd-Party\_container\BDHP\_4.5.0\_SAAS\_Snyk\_SBOM\_ALL\_3rd-Party\_container.json   + bdhp\_container\BDHP\_4.5.0\_SAAS\_Snyk\_SBOM\_ALL\_BDHP\_container.json   + codebase\BDHP\_4.5.0\_SAAS\_FOSSA\_SBOM \_ALL\_codebase.json | |  |
| I’ll demonstrate the use of the CycloneDX merge command by merging the SBOMs in the On-Prem\SBOMs\codebase directory, which contains “only” 32 SBOMs.  I start by opening this directory in Windows Explorer (Win + e), selecting all the SBOM files, holding Shift while right-clicking on the first file, and selecting “Copy Path” from the context menu. 🡪  Then I paste the list in Notepad++ and remove everything but the filenames, leaving me with a list of 32 filenames.  Next, I replace \r\n (CRLF) with a space, leaving a space-separated list of SBOM files to be merged.  At the beginning of this string, I add the following:  cyclonedx-win-x64 merge --input-files  And at the end of the string, I add this:  --output-file BDHP\_4.5.0\_On-Prem\_FOSSA\_SBOM \_ALL\_codebase.json |  | |

The entire command looks like this:

cyclonedx-win-x64 merge --input-files FOSSA\_SBOM\_bdhp-account\_4.22.0.json FOSSA\_SBOM\_bdhp-adt\_5.9.0.json FOSSA\_SBOM\_bdhp-alarm-reporter\_4.25.1.json FOSSA\_SBOM\_bdhp-allergy-intolerance\_5.14.0.json FOSSA\_SBOM\_bdhp-bridge\_1.5.1.json FOSSA\_SBOM\_bdhp-catalog\_5.78.2.json FOSSA\_SBOM\_bdhp-catalog-configuration\_5.16.1.json FOSSA\_SBOM\_bdhp-catalog-service-jobs\_4.3.1.json FOSSA\_SBOM\_bdhp-communication\_4.26.0.json FOSSA\_SBOM\_bdhp-core\_5.26.0.json FOSSA\_SBOM\_bdhp-dbmigration-base\_4.12.0.json FOSSA\_SBOM\_bdhp-device\_4.26.0.json FOSSA\_SBOM\_bdhp-device-gateway\_4.19.0.json FOSSA\_SBOM\_bdhp-device-gateway-engine\_5.13.10.json FOSSA\_SBOM\_bdhp-emr-vitals-service\_4.17.0.json FOSSA\_SBOM\_bdhp-encounter\_4.24.0.json FOSSA\_SBOM\_bdhp-enterprise-configuration\_5.4.8.json FOSSA\_SBOM\_bdhp-hl7-bed-outbound-service\_4.23.0.json FOSSA\_SBOM\_bdhp-iqecs\_6.24.20.json FOSSA\_SBOM\_bdhp-location\_4.30.1.json FOSSA\_SBOM\_bdhp-manual-adt-backend\_4.16.0.json FOSSA\_SBOM\_bdhp-manual-adt-ui\_4.9.1.json FOSSA\_SBOM\_bdhp-metrics\_4.2.4\_\_CORRECTED.json FOSSA\_SBOM\_bdhp-observation\_4.37.2.json FOSSA\_SBOM\_bdhp-patient\_4.41.0.json FOSSA\_SBOM\_bdhp-practitioner\_4.29.0.json FOSSA\_SBOM\_bdhp-procedure-request\_5.13.0.json FOSSA\_SBOM\_bdhp-provisioner\_6.9.2.json FOSSA\_SBOM\_bdhp-publisher\_4.27.0.json FOSSA\_SBOM\_bdhp-reference-updater\_4.14.0.json FOSSA\_SBOM\_bdhp-terminology-service\_4.62.0.json FOSSA\_SBOM\_bdhp-vitals-outbound\_4.21.0 .json --output-file BDHP\_4.5.0\_On-Prem\_FOSSA\_SBOM \_ALL\_codebase.json

In a command line window, I switch to the directory from which I pulled the list of files (cd \Users\mallmaj\Desktop\BDHP-4.5.0\On-Prem\SBOMs\codebase).

And then I paste the merge command and hit Enter.

**Converting Encoding of Snyk SBOMs**

After that, it’s time to merge the container SBOMs produced by Snyk.

Unfortunately, the Snyk SBOMs are created using “UTF-16 LE BOM” encoding; however, this causes problems when attempting to merge SBOMs using the cyclonedx-win-x64 command line tool, as the tool expects these files to have “UTF-8” encoding.

Now, one can use Notepad++ to change the encoding of a single file, but there are hundreds of Snyk SBOM files to convert!

Thus, I created a file named **convert-files-to-utf8.ps1** that contains the following four lines:

dir \*.json -Recurse | ForEach { #($file in Get-ChildItem \*.json)

echo $\_.name

(Get-Content $\_) | Set-Content -Encoding utf8 $\_

}

The easiest way I’ve found to convert the encoding of the Snyk SBOM files is to add a copy of this convert-files-to-utf8.ps1 file to each of the directories containing Snyk SBOMs, and then, from the command line, I switch to each directory to execute each of those files:

cd \Users\mallmaj\Desktop\BDHP-4.5.0\On-Prem\SBOMs\3rd-Party\_container

.\convert-files-to-utf8.ps1

cd \Users\mallmaj\Desktop\BDHP-4.5.0\On-Prem\SBOMs\bdhp\_container

.\convert-files-to-utf8.ps1

cd \Users\mallmaj\Desktop\BDHP-4.5.0\SAAS\SBOMs\3rd-Party\_container

.\convert-files-to-utf8.ps1

cd \Users\mallmaj\Desktop\BDHP-4.5.0\SAAS\SBOMs\bdhp\_container

.\convert-files-to-utf8.ps1

**Appendix A: Add Custom Function to PowerShell Profile**

To add a custom function to one’s PowerShell profile, start by executing the following in PowerShell to help determine the proper profile file:

PS> $PROFILE | Select-Object \*

This command will produce output like the following:

AllUsersAllHosts : C:\Windows\System32\WindowsPowerShell\v1.0\profile.ps1

AllUsersCurrentHost : C:\Windows\System32\WindowsPowerShell\v1.0\Microsoft.PowerShell\_profile.ps1

CurrentUserAllHosts : C:\Users\mallmaj\Documents\WindowsPowerShell\profile.ps1

CurrentUserCurrentHost : C:\Users\mallmaj\Documents\WindowsPowerShell\Microsoft.PowerShell\_profile.ps1

Length : 77

Choose the file for either the CurrentUserAllHosts or CurrentUserCurrentHost option and add custom functions to that file.

**Appendix B: Custom PowerShell Functions**

**Invoke-Snyk-Container-Monitor**

Function Invoke-Snyk-Container-Monitor {

<#

.SYNOPSIS

Invokes the `snyk container monitor` command, which captures the container image layers and dependencies in a project and monitors that snapshot for vulnerabilities, sending the results to snyk.io

IMPORTANT: One must be authenticated with Snyk via a call to `snyk auth` first!

#>

param(

[Parameter(

Mandatory=$true,

HelpMessage="The name of the Snyk 'Organization' to which the container monitor results should be posted. For example, ""bdhp-v4.5.0"".")]

[string]$organization,

[Parameter(

Mandatory=$true,

HelpMessage="The container image path, including [registry]/repository:tag. For example, ""docker.io/bitnami/redis:7.0.15"" OR ""bitnami/kubectl:1.31.3"".")]

[string]$image,

[Parameter(

Mandatory=$false,

HelpMessage="The local path to the Dockerfile used to create the container image.")]

[string]$dockerfilePath,

[Parameter(

Mandatory=$false,

HelpMessage="The tags to apply to the container monitor results in Snyk. NOTE: Please include a 'version' tag, so the version of the container image may be included in Snyk vulnerability reports by including the Project Tags column. For example, for the 'docker.io/bitnami/rabbitmq:3.13.4' image, the value for -tags should be ""version=3.13.4"".")]

[string]$tags

)

Try

{

if ($PSBoundParameters.ContainsKey('tags'))

{

if ($PSBoundParameters.ContainsKey('dockerfilePath'))

{

Write-Host "snyk container monitor --org=$organization --tags=$tags $image --file=$dockerfilePath"

snyk container monitor --org=$organization --tags=$tags $image --file=$dockerfilePath

}

else

{

Write-Host "snyk container monitor --org=$organization --tags=$tags $image"

snyk container monitor --org=$organization --tags=$tags $image

}

}

else

{

if ($PSBoundParameters.ContainsKey('dockerfilePath'))

{

Write-Host "snyk container monitor -org=$organization $image --file=$dockerfilePath"

snyk container monitor --org=$organization $image --file=$dockerfilePath

}

else

{

Write-Host "snyk container monitor --org=$organization $image"

snyk container monitor --org=$organization $image

}

}

}

Catch

{

Write-Host "Error occurred in Snyk-Container-Monitor" -f Yellow

Write-Host $\_ -f Red

}

}

**New-Snyk-Container-Sbom**

Function New-Snyk-Container-Sbom {

<#

.SYNOPSIS

Invokes the `snyk container sbom` command, which command generates an SBOM for a container image.

IMPORTANT: One must be authenticated with Snyk via a call to `snyk auth [<API\_TOKEN>]` first!

#>

param(

[Parameter(

Mandatory=$true,

HelpMessage="The name of the Snyk 'Organization' to which the container monitor results should be posted. For example, ""bdhp-v4.5.0"".")]

[string]$organization,

[Parameter(

Mandatory=$true,

HelpMessage="The SBOM Format, where the options are currently as follows: ""cyclonedx1.6+json"", ""cyclonedx1.4+xml"", ""cyclonedx1.5+json"", ""cyclonedx1.5+xml"", ""cyclonedx1.6+json"", ""cyclonedx1.6+xml"", or ""spdx2.3+json"".")]

[ValidateSet("cyclonedx1.6+json", "cyclonedx1.4+xml", "cyclonedx1.5+json", "cyclonedx1.5+xml", "cyclonedx1.6+json", "cyclonedx1.6+xml", "spdx2.3+json", IgnoreCase=$true)]

[string]$sbomFormat,

[Parameter(

Mandatory=$true,

HelpMessage="The container image path, including [registry]/repository:tag. For example, ""docker.io/bitnami/redis:7.0.15"" OR ""bitnami/kubectl:1.31.3"".")]

[string]$image,

[Parameter(

Mandatory=$true,

HelpMessage="The local path, including filename, to which the SBOM should be saved. Remember, the extension should match the requested format (.json or .xml).")]

[string]$outputPath

)

Try

{

Write-Host "snyk container sbom --org=$organization --format=$sbomFormat $image > $outputPath"

snyk container sbom --org=$organization --format=$sbomFormat $image > $outputPath

}

Catch

{

Write-Host "Error occurred in Snyk-Container-Sbom" -f Yellow

Write-Host $\_ -f Red

}

}