SBOM to VEX - Discovering What's in the Box and How Badly it Can Hurt You

From the creators of the fastest growing open source kubernetes security platform

Kubescape
Ben Hirschberg

Co-founder & CTO @ARMO

Kubescape maintainer

Whitehat in the past (unofficially still ;-) )

Fluent in Hebrew, Hungarian, C, ASM and Go

Contributor in CNCF + organizer of CNCF Jerusalem

Father of 4 <3
Kubernetes security

Scanning and monitoring

From dev to production

Misconfigurations and vulnerability finding

Operator and CLI
Cancel noise
Apply fix
Kubernetes_secured

Actionable, contextual, end-to-end Kubernetes-native security.
By Security standards, at DevOps pace.

Get a Demo
/this time

BEFORE THIS TALK

AFTER THIS TALK
SBOMs and VEXs

Exploitability evaluation with eBPF

Automation of VEX generation with Kubescape
- Modern software contains **80-90%** open source software
- At least **70%** of the containerized workloads are coming from external sources
- **90%** of the first level dependencies have dependencies themselves
If the sausage is your software, the SBOM is the list of the ingredients.
- Licensing issues in an organization (software composition analysis)
- Security posture/exposure (software posture management)
- Strategic exposures in organizational software
Can only find what they are looking for

Near 100% true positives
/vulnerabilities-and-sbom

SBOM

Vulnerability scanner

Vulnerability DB

NVD

Pypp ypi

linux dist

pkg

language specific

ask for input

Vulnerability DB

NVD

Pypp ypi

linux dist

pkg

language specific

ask for input

![Diagram of SBOM and vulnerability scanner process]

The diagram illustrates the process of vulnerability management and software bill of materials (SBOM) creation. It shows how a vulnerability scanner interacts with a vulnerability database (NVD) and a package manager (like Pypp ypi) to assess and manage Linux distribution components. The flow includes steps for asking for input, analyzing vulnerabilities, and ensuring that the SBOM is up-to-date and comprehensive. The diagram emphasizes the importance of tracking vulnerabilities and keeping software components secure.
How to exploit my software?

Take the SBOM

Cross-query vulnerability database for each package

100+ criticals
State of vulnerabilities

Comparing the whole sample to the sub-sample of graduated projects

- Reviewing the distribution of severities
- Reviewing top CVEs in both
- Reachability
### Top count of repo

<table>
<thead>
<tr>
<th>Repo</th>
<th># workload image scans</th>
</tr>
</thead>
<tbody>
<tr>
<td>quay.io/argoproj/argocd</td>
<td>19,426</td>
</tr>
<tr>
<td>docker.io/bitnami/redis</td>
<td>13,308</td>
</tr>
<tr>
<td>quay.io/argoproj/argoexec</td>
<td>11,427</td>
</tr>
<tr>
<td>quay.io/prometheus-operator/prometheus-config-reloader</td>
<td>11,275</td>
</tr>
<tr>
<td>quay.io/kiwigrid/k8s-sidecar</td>
<td>6,581</td>
</tr>
<tr>
<td>quay.io/prometheus/prometheus</td>
<td>6,390</td>
</tr>
<tr>
<td>docker.io/bitnami/mongodb</td>
<td>6,312</td>
</tr>
<tr>
<td>quay.io/prometheus/node-exporter</td>
<td>5,569</td>
</tr>
<tr>
<td>gcr.io/datadoghq/agent</td>
<td>5,404</td>
</tr>
</tbody>
</table>
# Image tags with most scans in the graduated sample

<table>
<thead>
<tr>
<th>Top count of repo</th>
<th># workload image scans</th>
</tr>
</thead>
<tbody>
<tr>
<td>quay.io/argoproj/argocd</td>
<td>19,426</td>
</tr>
<tr>
<td>quay.io/argoproj/argoexec</td>
<td>11,427</td>
</tr>
<tr>
<td>quay.io/prometheus-operator/prometheus-config-reloader</td>
<td>11,275</td>
</tr>
<tr>
<td>quay.io/prometheus/prometheus</td>
<td>6,390</td>
</tr>
<tr>
<td>quay.io/prometheus/node-exporter</td>
<td>5,569</td>
</tr>
<tr>
<td>quay.io/prometheus/alertmanager</td>
<td>4,172</td>
</tr>
<tr>
<td>quay.io/prometheus-operator/prometheus-operator</td>
<td>4,088</td>
</tr>
<tr>
<td>registry.k8s.io/kube-proxy</td>
<td>3,530</td>
</tr>
<tr>
<td>registry.k8s.io/kube-state-metrics/kube-state-metrics</td>
<td>3,039</td>
</tr>
</tbody>
</table>
### TOP vulnerabilities in general population

<table>
<thead>
<tr>
<th>CVE</th>
<th>Count of images</th>
<th>severity</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2022-28391</td>
<td>36,579</td>
<td>High</td>
<td>BusyBox through 1.35.0 allows remote attacker to escalate privilege.</td>
</tr>
<tr>
<td>CVE-2021-33560</td>
<td>14,561</td>
<td>High</td>
<td>Libgcrypt before 1.8.8 and 1.9.x before 1.9.3 might allow an attacker to crosstrain an</td>
</tr>
<tr>
<td>CVE-2019-8457</td>
<td>14,543</td>
<td>Critical</td>
<td>SQLite3 from 3.6.0 to and including 3.27.2 is vulnerable to a SQL injection attack.</td>
</tr>
<tr>
<td>CVE-2022-29458</td>
<td>14,531</td>
<td>High</td>
<td>ncurses 6.3 before patch 20220416 has an out-of-bounds read/write vulnerability.</td>
</tr>
<tr>
<td>CVE-2020-16156</td>
<td>14,391</td>
<td>High</td>
<td>CPAN 2.28 allows Signature Verification Bypass attacks.</td>
</tr>
<tr>
<td>CVE-2022-1304</td>
<td>14,224</td>
<td>High</td>
<td>An out-of-bounds read/write vulnerability was found in zlib through 1.2.12.</td>
</tr>
<tr>
<td>CVE-2022-37434</td>
<td>12,159</td>
<td>Critical</td>
<td>zlib through 1.2.12 has a heap-based buffer overflow vulnerability.</td>
</tr>
<tr>
<td>CVE-2021-46848</td>
<td>10,783</td>
<td>Critical</td>
<td>GNU Libtasn1 before 4.19.0 has an ETYPE_OKPNP in the libtasn1 library.</td>
</tr>
<tr>
<td>CVE-2022-0778</td>
<td>10,480</td>
<td>High</td>
<td>The BN_mod_sqrt() function, which computes an</td>
</tr>
</tbody>
</table>
**CVE-2022-28391**

**CVSS vector:** AV:N/AC:L/PR:N/UI:R/S:U/C:H/I:H/A:H

**Description:**
BusyBox through 1.35.0 allows remote attackers to execute arbitrary code if netstat is used to print a DNS PTR record's value to a VT compatible terminal. Alternatively, the attacker could choose to change the terminal's colors.

**Cloud native environment:**
If someone is running netstat in a Pod from a terminal while the attack controls the DNS entry the terminal is prone to the attack. Not a common scenario.
CVE-2021-33560

**CVSS vector:** AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:N/A:N

**Description:**
Libgcrypt before 1.8.8 and 1.9.x before 1.9.3 mishandles ElGamal encryption because it lacks exponent blinding to address a side-channel attack against mpi_powm, and the window size is not chosen appropriately. This, for example, affects use of ElGamal in OpenPGP.

**Cloud native environment:**
Libgcrypt is around in many images for GPG signature verification of APT/YUM packages. It is mostly not in use during deployment + uo private key in the image
CVE-2019-8457

**CVSS vector:** AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:H

**Description:**
SQLite3 from 3.6.0 to and including 3.27.2 is vulnerable to heap out-of-bound read in the rtreenode() function when handling invalid rtree tables.

**Cloud native environment:**
If the attacker can inject arbitrary SQL statements then the attacker can get arbitrary code execution. SQLite is part of Centos/RH base images.
/Opinion: **these are the vulnerabilities** has some probability to be exploited

<table>
<thead>
<tr>
<th></th>
<th>CVE</th>
<th>Count of images</th>
<th>severity</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>CVE-2022-37434</td>
<td>12,159</td>
<td>Critical</td>
<td>zlib through 1.2.12 has a heap-based buffer overf</td>
</tr>
</tbody>
</table>
## TOP vulnerabilities in graduated projects

<table>
<thead>
<tr>
<th>#</th>
<th>CVE</th>
<th>Count of images</th>
<th>Severity</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>CVE-2015-5237</td>
<td>119</td>
<td>High</td>
<td>It was discovered that the protobuf library and codebase has an issue.</td>
</tr>
<tr>
<td>2</td>
<td>CVE-2022-21698</td>
<td>17</td>
<td>High</td>
<td>In client_golang prior to version 1.11.1, HTTP server is vulnerable.</td>
</tr>
<tr>
<td>3</td>
<td>CVE-2022-31836</td>
<td>16</td>
<td>Critical</td>
<td>Function leafInfo.match() use path.join() to deal with dangerous input.</td>
</tr>
<tr>
<td>4</td>
<td>CVE-2022-46146</td>
<td>13</td>
<td>High</td>
<td>Prometheus Exporter Toolkit is a utility package to listen to Kubernetes.</td>
</tr>
<tr>
<td>5</td>
<td>CVE-2022-31054</td>
<td>7</td>
<td>High</td>
<td>Argo Events is an event-driven workflow automation tool.</td>
</tr>
<tr>
<td>6</td>
<td>GHSA-qpgx-64h2-gc3c</td>
<td>7</td>
<td>High</td>
<td>The package github.com/argoproj/argo-events/sensor is vulnerable.</td>
</tr>
<tr>
<td>7</td>
<td>CVE-2020-16156</td>
<td>6</td>
<td>High</td>
<td>CPAN 2.28 allows Signature Verification Bypass.</td>
</tr>
<tr>
<td>8</td>
<td>CVE-2021-33560</td>
<td>6</td>
<td>High</td>
<td>Libgcrypt before 1.8.8 and 1.9.x before 1.9.3 mishandles the input.</td>
</tr>
<tr>
<td>9</td>
<td>CVE-2019-8457</td>
<td>6</td>
<td>Critical</td>
<td>SQLite3 from 3.6.0 to and including 3.27.2 is vulnerable.</td>
</tr>
</tbody>
</table>
CVE-2015-5237

**CVSS vector:** AV:N/AC:L/PR:L/UI:N/S:U/C:H/I:H/A:H

**Description:**
protobuf allows remote authenticated attackers to cause a heap-based buffer overflow

**Cloud native environment:**
It is indeed a vulnerability in protobuf C/C++ package. But not in the Golang package!

https://github.com/anchore/grype/issues/558
/Opinion: **these are the vulnerabilities** has some probability to be exploited

*gut feeling :-/

<table>
<thead>
<tr>
<th>CVE</th>
<th>Count of image</th>
<th>severity</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2022-21698</td>
<td>17</td>
<td>High</td>
<td>In client_golang prior to version 1.11.1, HTTP server returns sensitive</td>
</tr>
<tr>
<td>CVE-2022-31836</td>
<td>16</td>
<td>Critical</td>
<td>Function leafInfo.match() use path.join() to deal with</td>
</tr>
<tr>
<td>CVE-2022-46146</td>
<td>13</td>
<td>High</td>
<td>Prometheus Exporter Toolkit is a utility package to</td>
</tr>
<tr>
<td>CVE-2022-31054</td>
<td>7</td>
<td>High</td>
<td>Argo Events is an event-driven workflow automation</td>
</tr>
<tr>
<td>GHSA-qpgx-64h2-gc3c</td>
<td>7</td>
<td>High</td>
<td>The package github.com/argoproj/argo-events/sens</td>
</tr>
</tbody>
</table>
Looking at general results

Average vulnerability count per severity

- Critical
- High
- Medium
- Low
- Negligible
- Other

General sample
Graduated sample
CNCF IS AWESOME
WAIT A SECOND...
It's Rewind time.
Vulnerability in image ≠ Application exploit
Vulnerabilities in common images
Kubescape reachability

SBOM (full)

Files actually used by the container

SBOM (filtered)

Vulnerability scanner

Scan image  eBPF  Compare against SBOM  Feed to Vulnerability scanner
/Kubescape reachability

Relevant vulnerabilities

Filtered SBOM

grype

pod

System calls

eBPF

Kernel

INPEKTOR GADGET
Kubescape reachability results
what-is-kubescape

- CNCF Project
- Kubernetes security side-kick
- Configuration & Vulnerability analysis
- Runtime detection
- 10k GitHub starts
- Widely adopted tool (both CLI and service)
Looking only at filtered results

Average relevant vulnerability count per severity

- Critical
- High
- Medium
- Low
- Negligible
- Other
If SBOM is like the ingredient list of a sausage, then VEX is like the list of allergens.
{
  "@context": "https://openvex.dev/ns/v0.2.0",
  "@id": "https://openvex.dev/docs/example/vex-9fb3463de1b57",
  "author": "Wolfi J Inkinson",
  "role": "Document Creator",
  "timestamp": "2023-01-08T18:02:03.647787998-06:06",
  "version": "1",
  "statements": [
    {
      "vulnerability": {
        "name": "CVE-2014-123456"
      },
      "products": [
        {
          "@id": "pkg:apk/distro/git@2.39.0-r1?arch=armv7"
        },
        {
          "@id": "pkg:apk/distro/git@2.39.0-r1?arch=x86_64"
        }
      ],
      "status": "fixed"
    }
  ]
}
I AM GOING TO TELL YOU

WHICH VULNERABILITIES REALLY MATTER
Preparing and maintaining reliable VEX is time consuming.

"Not good" if not reliable.
Generation of VEX documents by the Kubescape relevancy engine #155

slashben opened this issue on Oct 10 • 4 comments

Overview

Kubescape calculates the relevancy of container image vulnerabilities by monitoring using eBPF the application behavior and produces a filtered list of vulnerabilities. Today the results are stored in the same format as the vulnerabilities, however the VEX seems to be a much better choice to store and publish this information. Kubescape needs to publish the filtered list of vulnerabilities in a VEX format.

Solution

In the current state, the Kubewatch is watching the filtered SBOM objects, every time a new object is created or updated a filtered SBOM is created by the node-agent with only those modules that were loaded into the memory.

When a new filtered SBOM is available, the Kubewatch translates the SBOM to vulnerability list using Gryte to create a filtered vulnerability list.

In the same step when the filtered vulnerability is created, Kubewatch should generate a VEX object. The object contains statements that all these vulnerabilities are loaded into the memory therefore they’re relevant. This object should be stored as an API objects another vulnerability related.

See more at

cc: @craigbox @puerco

puerco commented on Oct 10

Assignees

No one—assign yourself!

Labels

- enhancement

- good first issue

- proposal

Projects

- Name yet

Milestone

No milestone

Development

Create a branch for this issue or link a pull request.

Notifications

You’re receiving notifications because you modified the open/close state.

3 participants
$ helm repo add kubescape https://kubescape.github.io/helm-charts/
$ helm repo update
$ helm upgrade --install kubescape kubescape/kubescape-operator -n kubescape --create-namespace --set clusterName=`kubectl config current-context` --set capabilities.vexGeneration=enable
$ kubectl -n kubescape get pods

<table>
<thead>
<tr>
<th>NAME</th>
<th>READY</th>
<th>STATUS</th>
<th>RESTARTS</th>
<th>AGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>kubescape-6bd764869d-nmk5k</td>
<td>1/1</td>
<td>Running</td>
<td>0</td>
<td>99s</td>
</tr>
<tr>
<td>kubevuln-76bbbdffcd4-8fxcq</td>
<td>1/1</td>
<td>Running</td>
<td>0</td>
<td>99s</td>
</tr>
<tr>
<td>node-agent-dnf6l</td>
<td>1/1</td>
<td>Running</td>
<td>0</td>
<td>99s</td>
</tr>
<tr>
<td>operator-75c999bfc6-dlfj8</td>
<td>1/1</td>
<td>Running</td>
<td>0</td>
<td>99s</td>
</tr>
<tr>
<td>storage-5898d46fd-rmv4x</td>
<td>1/1</td>
<td>Running</td>
<td>0</td>
<td>99s</td>
</tr>
</tbody>
</table>
Generating VEX

```
$ kubectl apply -f https://k8s.io/examples/application/deployment.yaml

$ kubectl -n kubescape get openvulnerabilityexchangecontainer $(kubectl -n kubescape get openvulnerabilityexchangecontainer -o jsonpath='{.items[0].metadata.name}') -o jsonpath='{.spec}' > nginx.json

$ jq "." nginx.json | grep -c ""affected"
58

$ jq "." nginx.json | grep -c ""not_affected"
338
```
$ grype nginx:1.14.2 --vex nginx.json

✔ Vulnerability DB          [no update available]
✔ Loaded image
nginx:1.14.2
✔ Parsed image
sha256:295c7be079025306c4f1d65997fcf7adb411c88f139ad1d34b537164aa060369
✔ Cataloged packages        [111 packages]
✔ Scanned for vulnerabilities [58 vulnerability matches]
  — by severity: 55 critical, 102 high, 85 medium, 52 low, 102 negligible
  — by status:   126 fixed, 270 not-fixed, 338 ignored
Vulnerabilities by scanners are mostly wrong

Good VEX can mitigate this

VEX can be enhanced automatically
As a user

As a developer

As a security expert
Thank you