Fresh Secrets From The Docks

Lessons Learnt From Analysing **15 Million** Public DockerHub Images

Guillaume VALADON - @guedou

GitGuardian

$ whoami

Guillaume

Cybersecurity Researcher

Scapy maintainer

previously at Quarkslab, **ANSSI**…

editor-in-chief of the french **MISC magazine**

looking for secrets in unusual places

scapy

🔬 **Network Packets Manipulation**

https://www.scapy.net/

Secrets Security 🤝 NHI Governance

0% LEAKED SECR

Secrets Sec

INTEGRATED SOURCES

- Package Registries
- Container Registries
- Code Repositories
- CI/CD Pipelines
- Messaging Systems
- Ticketing Systems
- Logs
- Knowledge Database

RITY

100% MANAGED IDENTITIES & SECRETS

NHI Governance

INTEGRATED SOURCES

- IAM Cloud Providers
- Secret Managers
- Cloud Infrastructure
- Deployment Tools
- Third-Party Apps

te Lifecycle

Vaulting,
Ownership

Check it out !

GitGuardian

Attackers Are Looking For Secrets

APPLICATION SECURITY

PyPI Packages Found to Expose Thousands of Secrets

GitGuardian discovered roughly 4,000 secrets in nearly 3,000 PyPI packages, including Azure, AWS, and GitHub keys.

By Ionut Arghire
November 14, 2023

Home » Security

Websites exposing over a million secrets, leaving visitors at risk

Last updated: 29 May 2024

Home > News > Security > Nearly 12,000 API keys and passwords found in AI training dataset

Nearly 12,000 API keys and passwords found in AI training dataset

By Ionut Ilascu

March 2, 2025 10:23 AM 0

MITRE ATT&CK Paths Examples

Container Registries

01 T1078 - Valid Accounts

02 T0613 - Container Discovery

03 T1552 - Unsecured Credentials

Cloud Platform

01 T1078 - Valid Accounts

02 T1578 - Modify Cloud Infrastructure

03 T1496 - Resource Hijacking

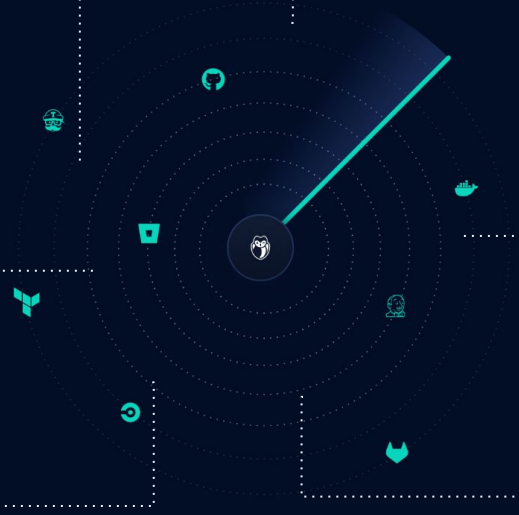
Secrets Categories Exploited by Attackers

Package & Container Registries

Artifactory, **Docker Registry**, NPM…

Version Control Systems

GitHub, GitLab…



Cloud Platforms

AWS, GCP, Digital Ocean…

Secrets Management

HashiCorp Vault, LDAP…

Storage Services

Azure Blob Storage, Dropbox…

Databases

MongoDB, MySQL…

01

Scanning a Docker Image

Everything you need to know about the structure of an image to look for secrets.

Interacting With A Docker Registry

```
$ docker build -t gvaladon/kubecon:latest .
```

```
$ docker push gvaladon/kubecon:latest
```

```
$ docker pull gvaladon/kubecon:latest
```

Steps?

0. build
1. push
2. **pull**

From a Dockerfile to a Docker Image

```
FROM ubuntu:latest

COPY . /src

RUN pip install -r requirements.txt

EXPOSE 8080

ENV NAME=KubeCon

CMD ["python", "app.py"]
```

Image X-ray

JSON manifest
- list of layers (aka tarballs)
- JSON config

everything stored in **blobs**
- SHA256 identify the content

🔬 **Inspecting images**

https://github.com/containers/skopeo

Scanning Docker images for secrets means accessing **JSON files and tar archives** (aka blobs).

Manual Secrets Scanning

```
$ ggshield secret scan docker gvaladon/kubecon:latest

>> Secret detected: AWS Keys
   Validity: Invalid
   Occurrences: 1
   Known by GitGuardian dashboard: NO
   Incident URL: N/A
   Secret SHA: 4d3bb359d157287dbc19a5af9e1019e2547ee8aad651a88973dcf62d0776dd88

 8  8 |
 9  9 |  def aws_upload(data: Dict):
   10 | +    database = aws_lib.connect("AKIA*******WSF5", "hjshuk5****************89sjkja")
...
10    | -    database = aws_lib.connect("AKIA*******WSZ5", "hjshnk5****************89sjkja")
                                          |___client_id___|
10    | -    database = aws_lib.connect("AKIA*******WSZ5", "hjshnk5****************89sjkja")
                                                             |________client_secret_________|
11 11 |      database.push(data)
```

🔬 **Scanning for secrets**

https://github.com/GitGuardian/ggshield

Validity Checks

Test the secret with a known endpoint

and check the **HTTP status code**.

GitHub Token

```
$ curl  --write-out  "%{http_code}" --silent --output /dev/null \
  --header "Authorization: Bearer $VALID_TOKEN" \
  https://api.github.com/user

200
```

```
$ curl  --write-out  "%{http_code}" --silent --output /dev/null \
  --header "Authorization: Bearer $INVALID_TOKEN" \
  https://api.github.com/user

401
```

02

Scanning a Docker Registry

Methodology and API calls to retrieve configuration files and layers content.

Methodology

Four simple steps

combined to **scan all Docker images** from a single registry.

01 **enumerating repositories**

02 **listing tags**

03 **retrieving manifests**

04 **scan configs & layers**

Corresponding Docker Registry API

Pulling an image manually

leaving authentication aside, these API calls retrieve Docker images content.

01 `/v2/_catalog`

02 `/v2/repository/tags/list`

03 `/v2/repository/manifest/tag`

04 `/v2/repository/blobs/blob`

03

Scanning Docker Hub

Techniques to retrieve blobs, and look for secrets.

Enumerating Docker Hub Repositories

/v2/_catalog is not available

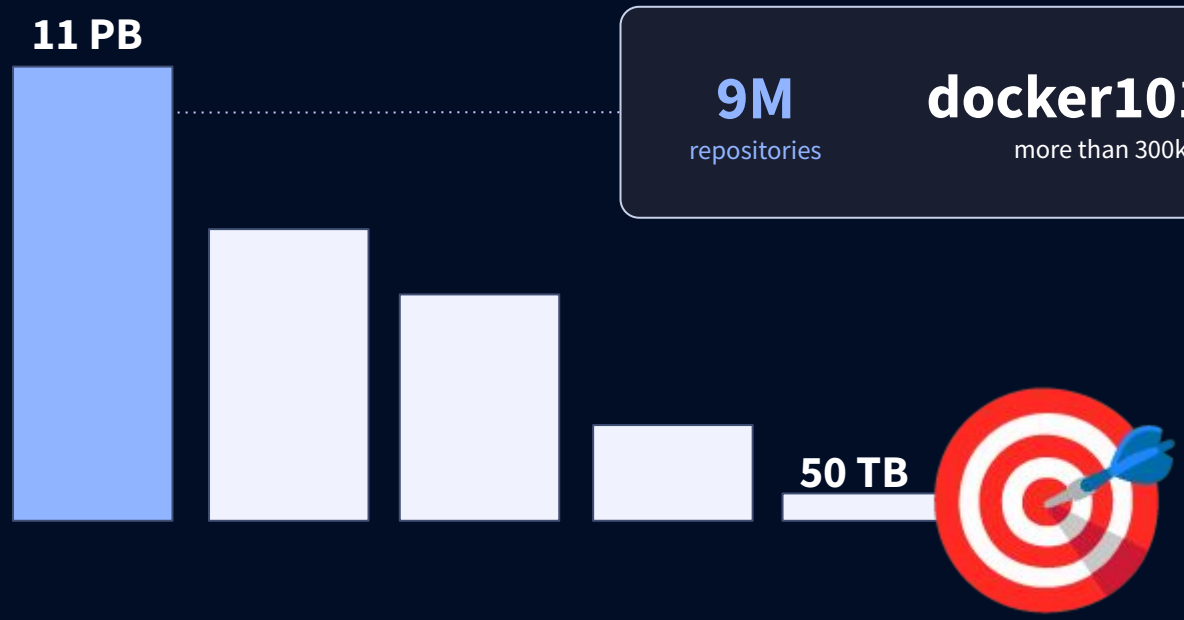
Docker Hub web search returns a maximu
10k results per keyword

~1 day

sko?

bruceejacobs/**sko**
rancher/mirrored-**sko**peo-skopeo
skorochkin/gradle-node-phantomjs2
opsani/**sko**pos
ivaldidk/**sko**leit_apcupsd
skola/kolawebserver
skottaramcs/shields-rail
skopciewski/dnsmasq

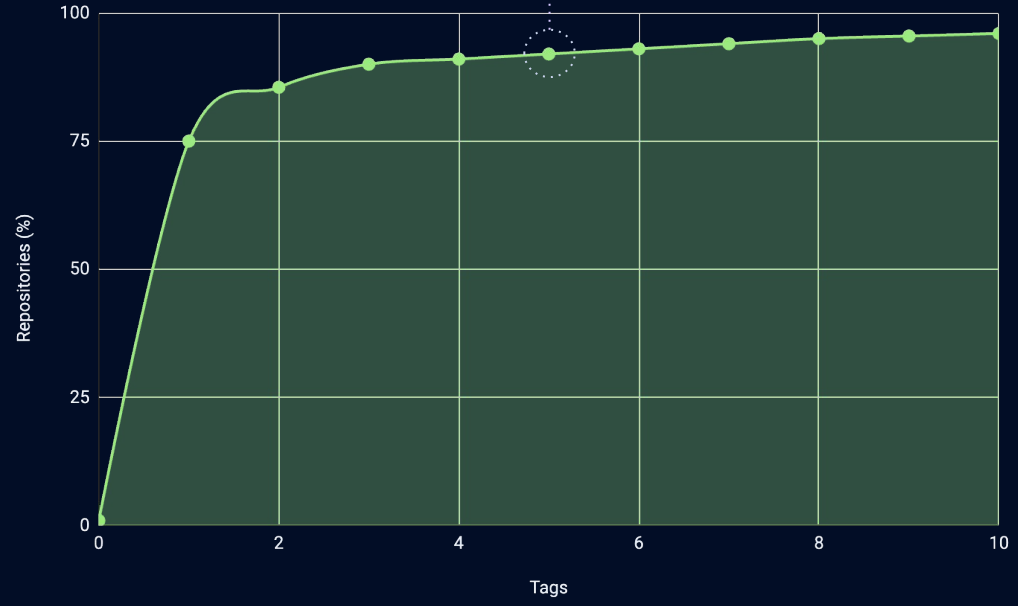
Refining The Scope of the Scan



9M
repositories

docker101tutorial
more than 300k occurrences

Listing Tags



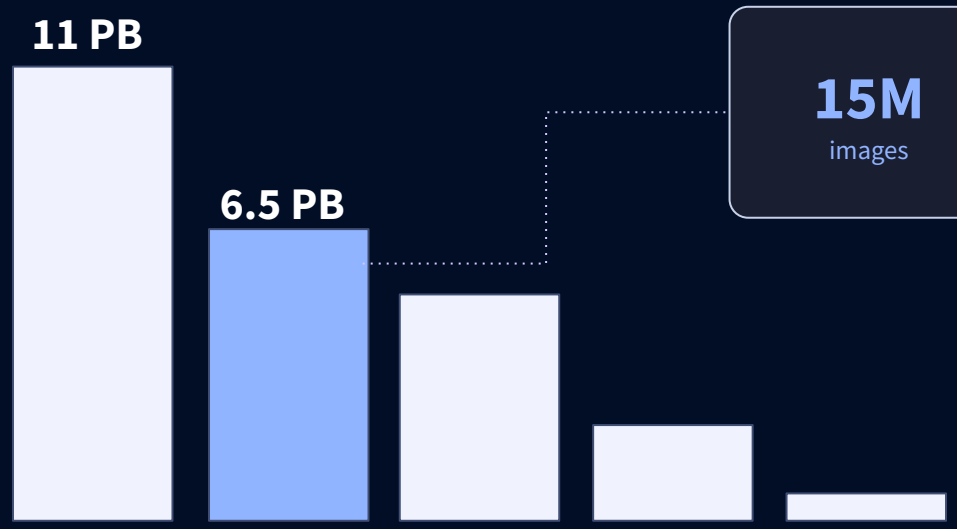
5
tags

93%
of repositories

~3 days

Refining The Scope of the Scan

At Most 5 Tags Per Repository



15M
images

latest
is the most used tag

dockerhub

Search Docker Hub

Explore / liuzzyg/trajdeleter / latest

liuzzyg/trajdeleter:latest

MANIFEST DIGEST sha256:3b3170498d184cf1f07711f8c0ca4fe10baebbbf2617c196ee0561e338f66c5d

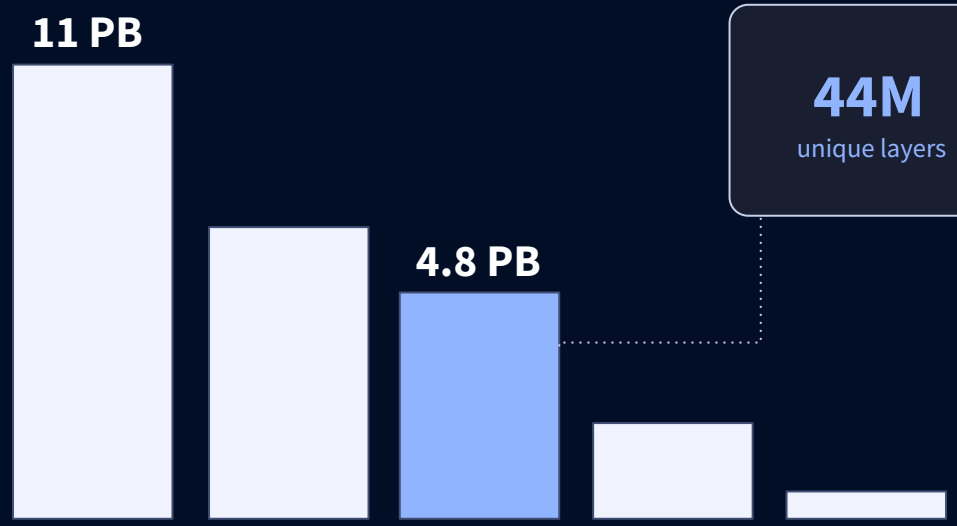
OS/ARCH	COMPRESSED SIZE	LAST PUSHED	TYPE	MANIFEST DIGEST
linux/amd64	170.08 GB	8 months by liuzzyg	Image	sha256:3b317049...

Image Layers

1	128.25 GB
2 /bin/bash	41.72 GB
3 /bin/bash	25.26 KB
4 /bin/bash	47.98 KB

Refining The Scope of the Scan

Removing Duplicated Layers Per Repository



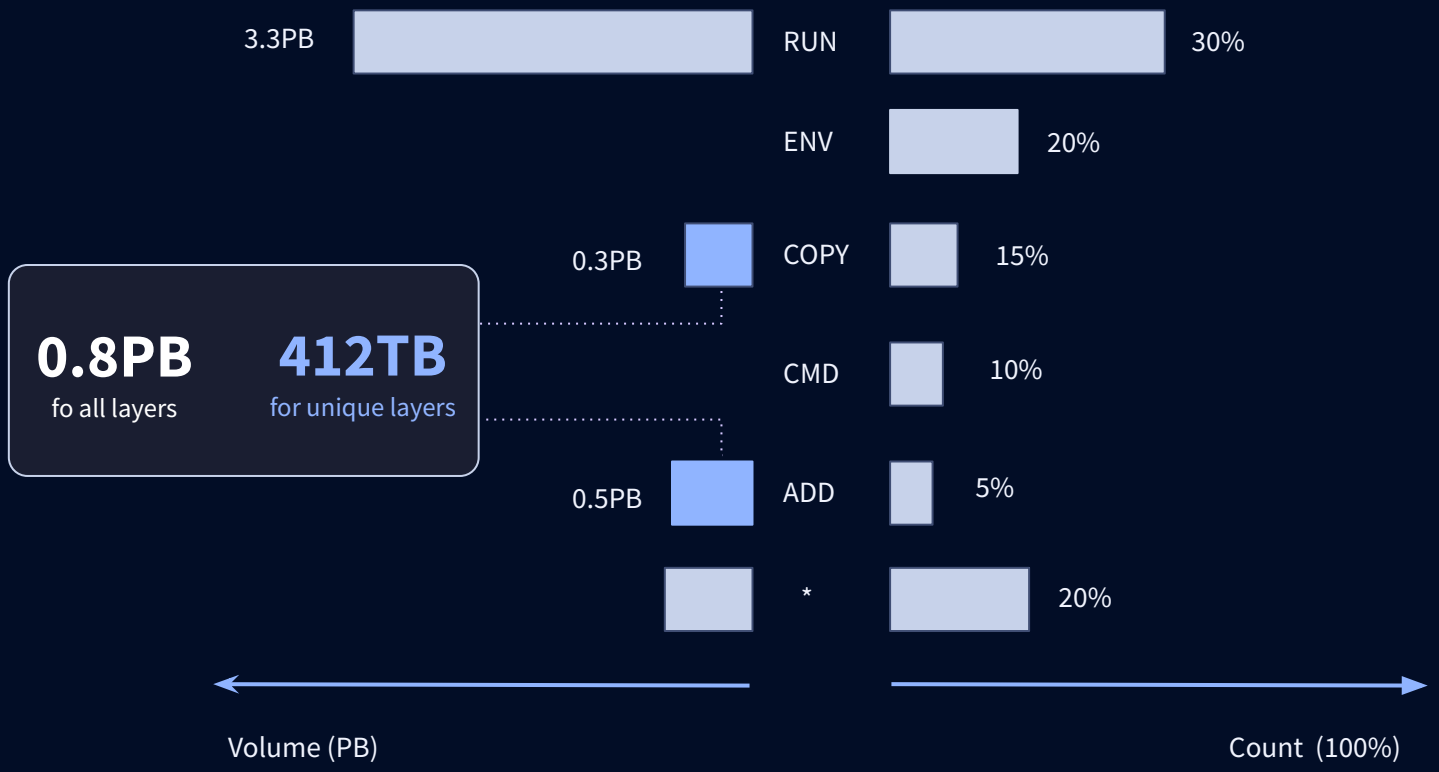
44M
unique layers

148GB
is the largest layer

~10 days

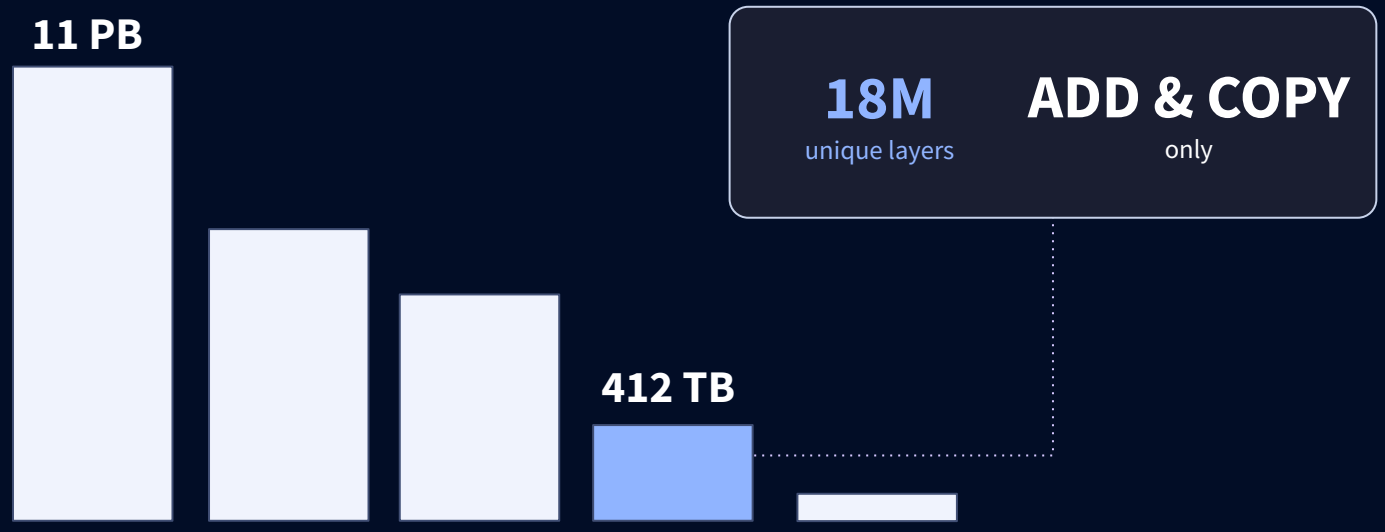
Scanning Docker Hub

Layers Sizes & Instructions

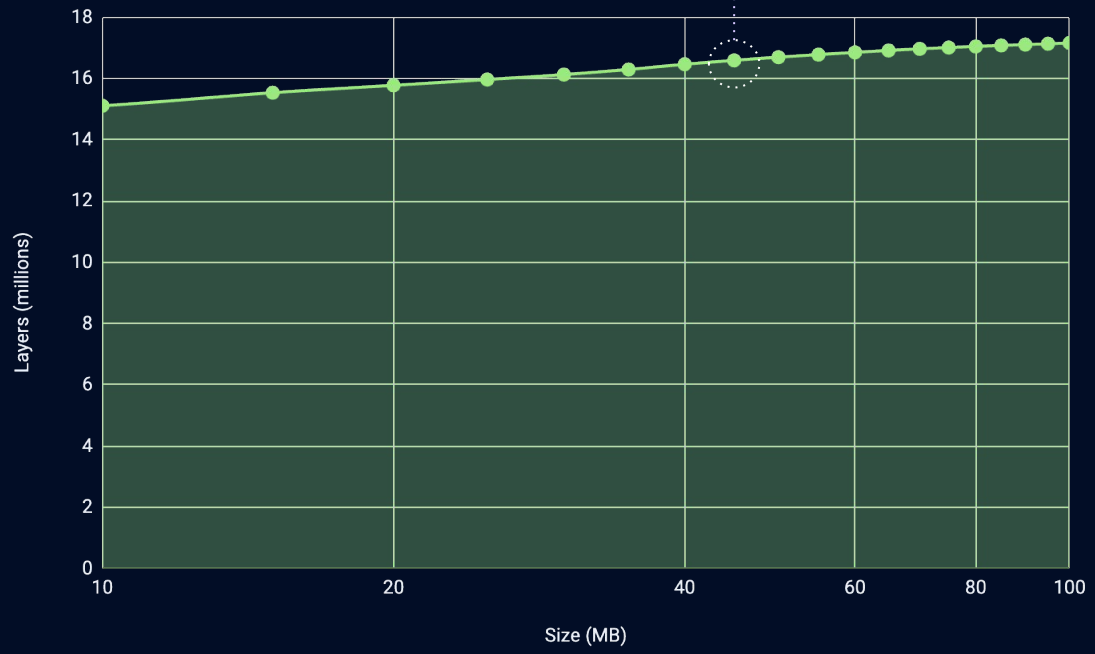


Refining The Scope of the Scan

Filtering Dockerfile Instructions



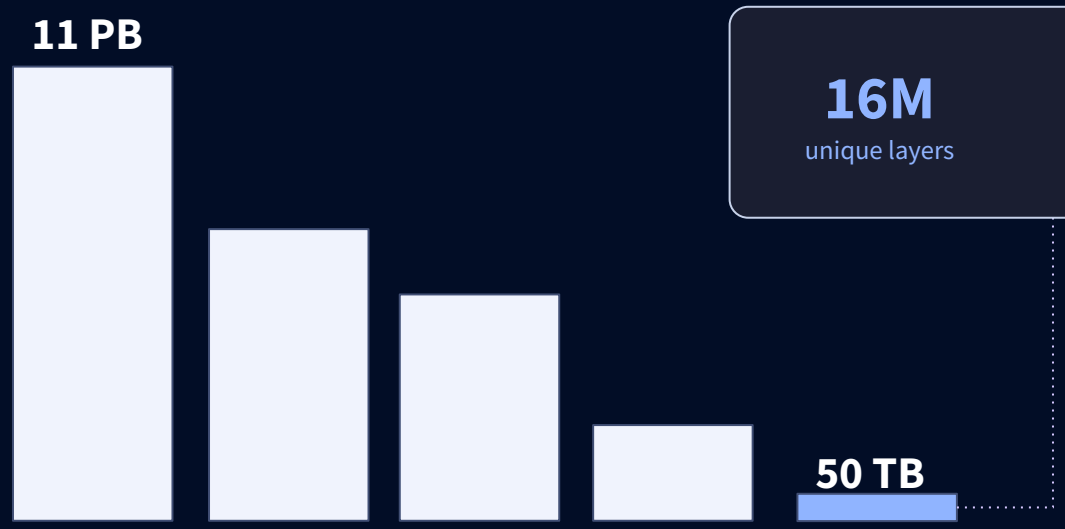
ADD & COPY Layers Sizes



99%
below 200MB

90%
below 45MB

Refining The Scope of the Scan



16M
unique layers

45MB
max layer size

Scanning Docker Hub

15 days

3 days

20 days

Discovery

9 millions repositories
26 millions images

Scanning Configs

15 millions images

Scanning Layers

16 millions unique layers

GitGuardian

04
Secrets Analysis

Charts & Numbers.

Two Types of Secrets Detectors

Specific

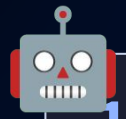
well-known patterns & formats
mapped to services and providers
may be **automatically validated**

500k unique secrets

Generic

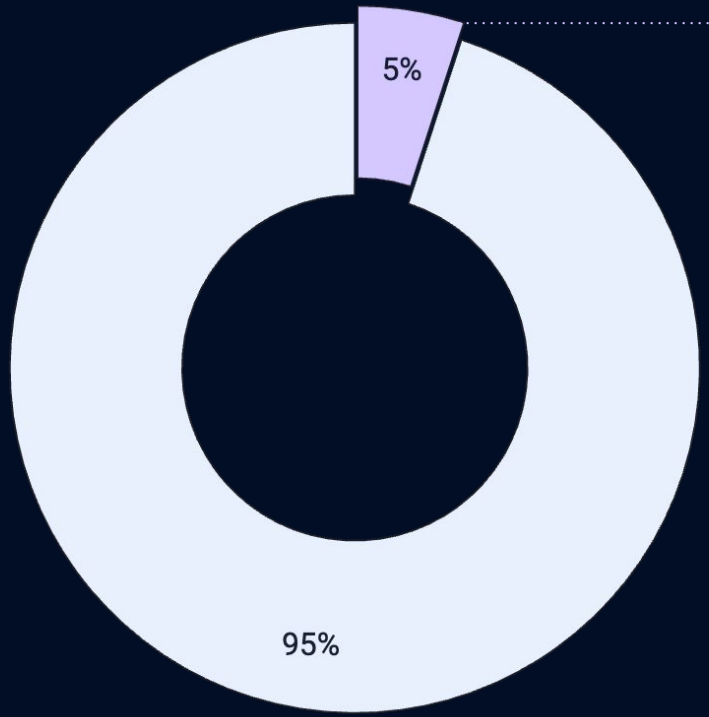
random looking
unknown service or provider
lack context

700k unique secrets



13%
categorized

Repositories



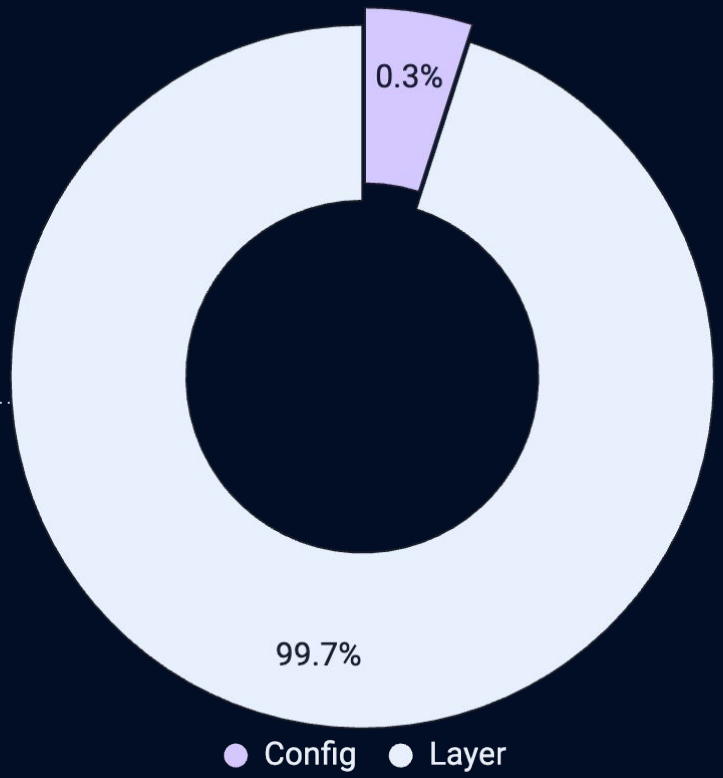
9M
repositories

450k
repositories with a secret

Origin

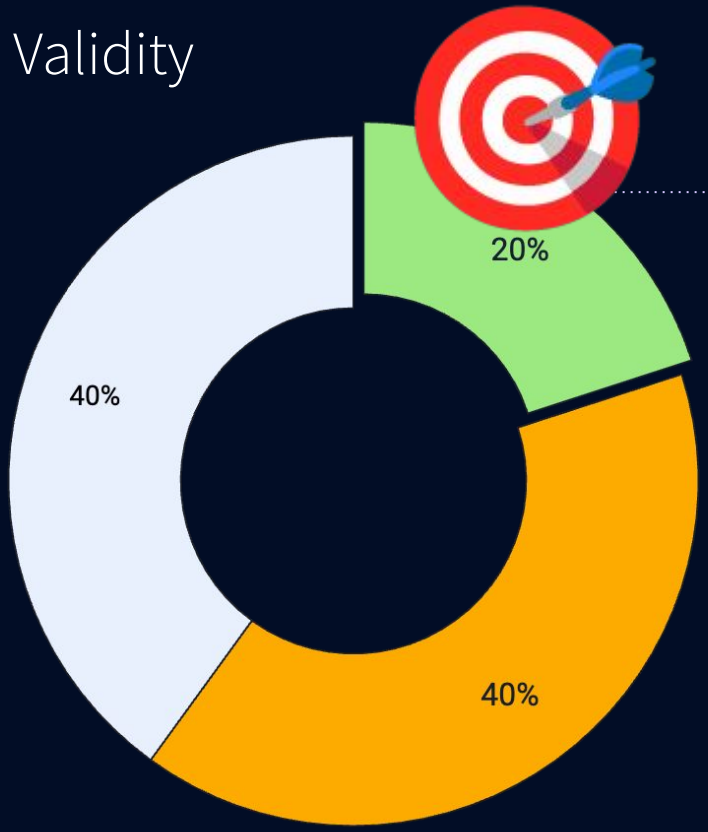
most secrets are in layers

good Dockerfile hygiene?



Secrets Analysis

Validity



Valid Cannot Check Not Valid

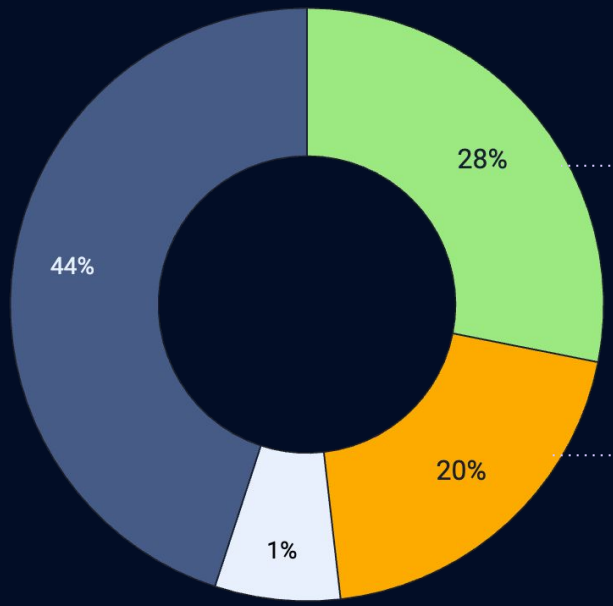
100k
valid secrets

230
detector types

GitGuardian

Secrets Analysis

Types



13% are valid
out of 140k

50% are valid
out of 100k

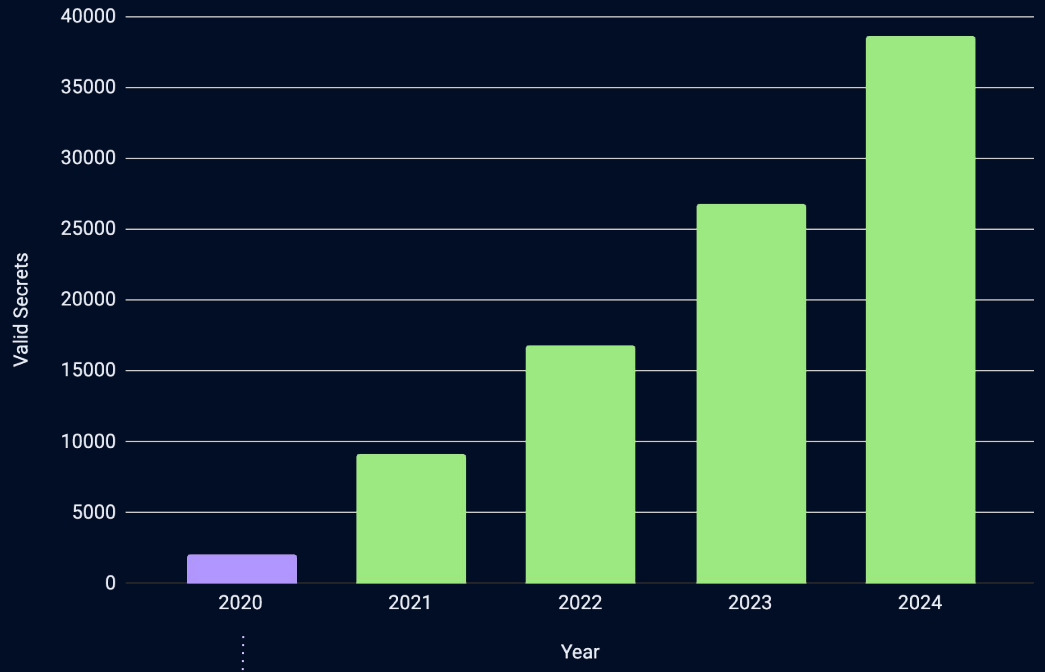
Data Storage · Cloud Provider · Version Control System · Other

Secrets Analysis

Longevity

60%
of valid secrets
before 2024

2053
valid secrets
from 2020



05

Docker Pitfalls & Takeaways

Docker Remembers

```
FROM ubuntu:latest

COPY . /app

RUN /app/do_things.sh

RUN npm ci --production && rm .npmrc

ENTRYPOINT ["docker-entrypoint.sh"]"
```

.npmrc **not removed**

still in the COPY layer

Leaks in Dockerfiles?

```
$ skopeo inspect docker://■■■/■■■:latest --config |jq -rc '.history[].created_by'
[..]
/bin/sh -c echo ■■■:■■■ > ~/.passwd-s3fs
/bin/sh -c chmod 600 ~/.passwd-s3fs
```

in practice, there are **few hard-coded secrets in Dockerfiles**

Leaks in Configs?

```
$ skopeo inspect docker://■ ■ ■/■ ■ ■:latest --config |jq -rc '.history[].created_by'
[..]
/bin/sh -c #(nop)  ARG BUILD_ARG_GITHUB_ACCESS_TOKEN
/bin/sh -c #(nop)  ARG BUILD_ARG_GITHUB_USERNAME
/bin/sh -c #(nop)  ARG BUILD_ARG_CONCOURSE_URL
/bin/sh -c #(nop)  ARG BUILD_ARG_CONCOURSE_USER
/bin/sh -c #(nop)  ARG BUILD_ARG_CONCOURSE_PASSWORD
/bin/sh -c #(nop)  ENV GITHUB_ACCESS_TOKEN=ghp_■ ■ ■
/bin/sh -c #(nop)  ENV GITHUB_USERNAME=■ ■ ■
/bin/sh -c #(nop)  ENV CONCOURSE_URL=http://■ ■ ■:8080
/bin/sh -c #(nop)  ENV CONCOURSE_USER=concourse
/bin/sh -c #(nop)  ENV CONCOURSE_PASSWORD=con@123
```

leaks appear in configs, as a result of **`docker build`**

Leaking Build Arguments

`docker build` leaks secrets into the config

```
FROM ubuntu:latest

ARG SECRET
ENV TEST=$SECRET

RUN /bin/bash -c "/bin/echo ${TEST}"
```

```
$ docker build -t leak:latest --build-arg SECRET=Ku83c0n@ .
```

```
ARG SECRET=Ku83c0n@
ENV TEST=Ku83c0n@
RUN |1 SECRET=Ku83c0n@ /bin/sh -c /bin/bash -c "/bin/echo ${TEST}"
```

Build Arguments Leaks Are Documented

Home / Manuals / Docker Build / Building / Variables

Build variables

In Docker Build, build arguments (`ARG`) and environment variables (`ENV`) both serve as a means to pass information into the build process. You can use them to parameterize the build, allowing for more flexible and configurable builds.

Warning

Build arguments and environment variables are inappropriate for passing secrets to your build, because they're exposed in the final image. Instead, use secret mounts or SSH mounts, which expose secrets to your builds securely.

See Build secrets for more information.

Similarities and differences

Build arguments and environment variables are similar. They're both declared in the Dockerfile and can be set using flags for the `docker build` command. Both can be used to parameterize the build. But they each serve a distinct purpose.

Edit this page

Request changes

Table of contents

Best Practice: Secrets Mounts

secrets are only accessible at build time

```
FROM ubuntu:latest

RUN --mount=type=secret,id=SECRET,env=SECRET /bin/bash -c "/bin/echo ${SECRET}"
```

```
$ docker build -t leak:latest --secret id=SECRET,env=RANDOM .
```

```
RUN /bin/sh -c /bin/bash -c "/bin/echo ${SECRET}" # buildkit
ENV SECRET=
```

Could Secrets Leak in RUN Layers?

```
FROM ubuntu:latest

COPY . /src

RUN pip install -r requirements.txt

RUN env > .env

EXPOSE 8080

ENV NAME=KubeCon

CMD ["python", "app.py"]
```

similar files

- config.js
- settings.py
- credentials.json
- appsettings.json

Let's Grep This!

```
/bin/sh -c printenv > .env

RUN /bin/bash -o pipefail -c env > .env # buildkit

RUN /bin/sh -c cat /run/secrets/dot_env > .env # buildkit

RUN /bin/sh -c cat envs/standalone.k3s.env > .env # buildkit

RUN /bin/sh -c cat main-env > .env # buildkit

RUN /bin/sh -c cat main.env > .env # buildkit

RUN /bin/sh -c env > .env # buildkit
```

ideal secrets management meets hard-coded secrets

Takeaways

Include Secrets Leaks as a Key Security Threat

you are probably leaking secrets
without realizing it

exposures come from a wide range of Docker pitfalls

audit your images for hard-coded secrets

Prevention is more cost-effective than dealing with a breach!

Thank you

Question Time 🔥

Guillaume VALADON

GitGuardian

Contacts

contact@gitguardian.com

sales@gitguardian.com
www.gitguardian.com