GitGuardian

All You Can Leak

Real Tales of **Publicly Leaked Kubernetes Secrets**

Guillaume VALADON | @guedou

www.gitguardian.com

$ whoami

Guillaume

Cybersecurity Researcher

Scapy maintainer

previously at Quarkslab, **ANSSI**…

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

looking for secrets in unusual places

🔬 **Network Packets Manipulation**

`https://www.scapy.net/`

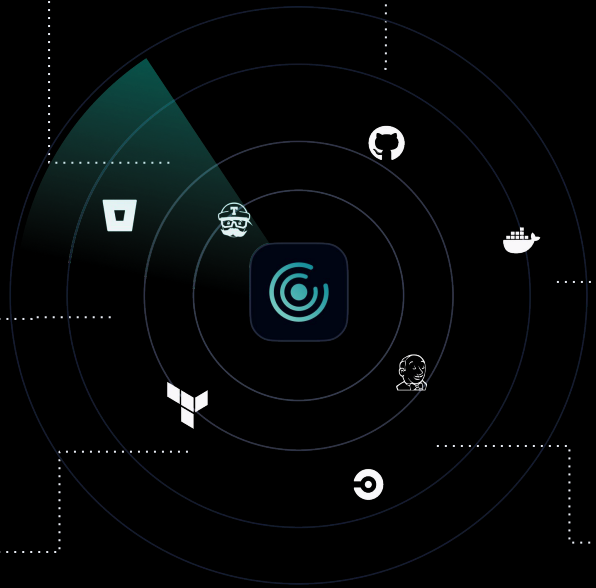
Secrets Categories Exploited by Attackers

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…

MITRE ATT&CK Paths Examples

Container Registries

01 T1078 - Valid Accounts

02 T0613 - Container Discovery

03 T1552 - Unsecured Credentials

Kubernetes Cluster

01 T1078 - Valid Accounts

02 T1578 - Modify Cloud Infrastructure

03 T1496 - Resource Hijacking

https://www.virusbulletin.com/conference/vb2023/abstracts/tales-cloud-csirt-lets-deep-dive-kubernetes-k8s-infection/

Initial Research Questions

Are Kubernetes clusters **publicly accessible**?

What are Kubernetes authentication secrets?

Are they **publicly leaked**?

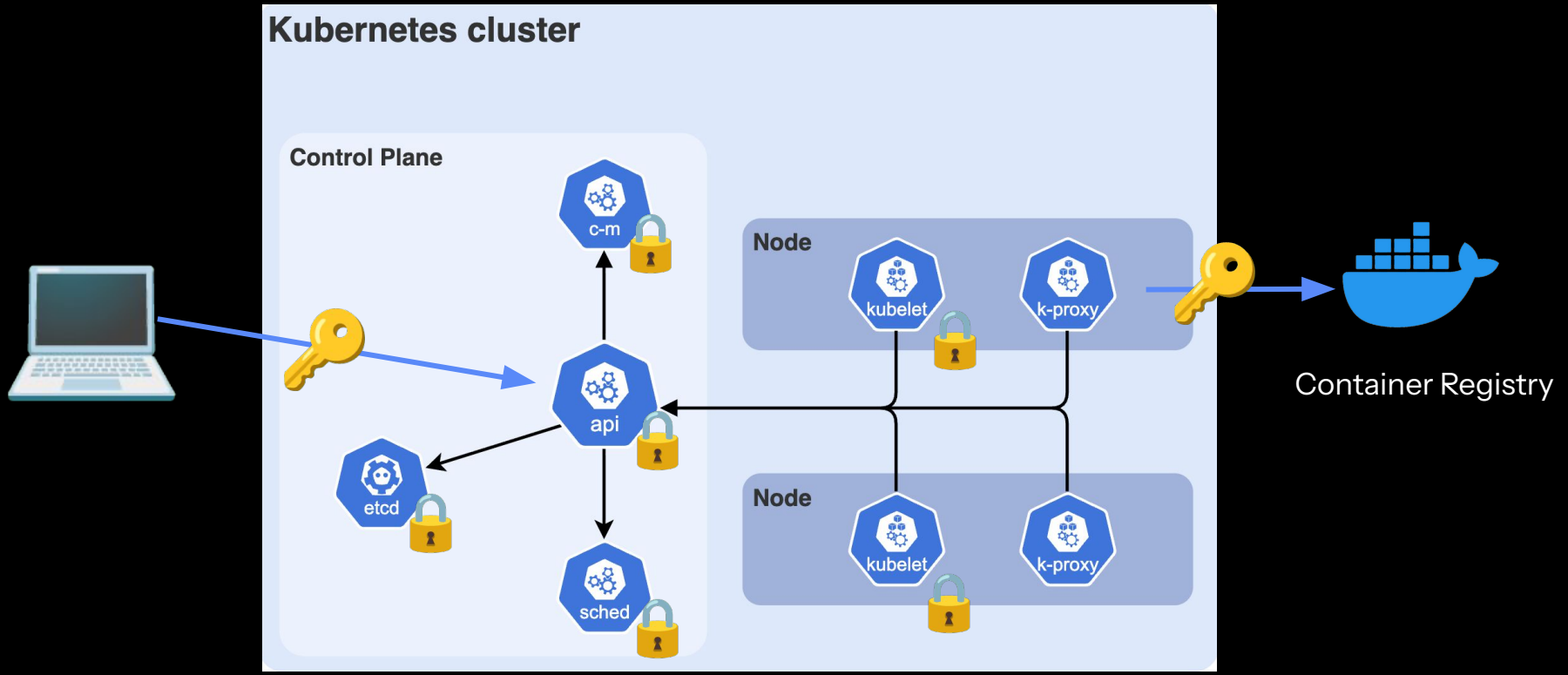
What is the **impact of these leaks**?

Kubernetes 101

*"Kubernetes, also known as K8s, is an open source system for **automating deployment,** scaling, and management of containerized **applications**."*

https://kubernetes.io/

Kubernetes Architecture



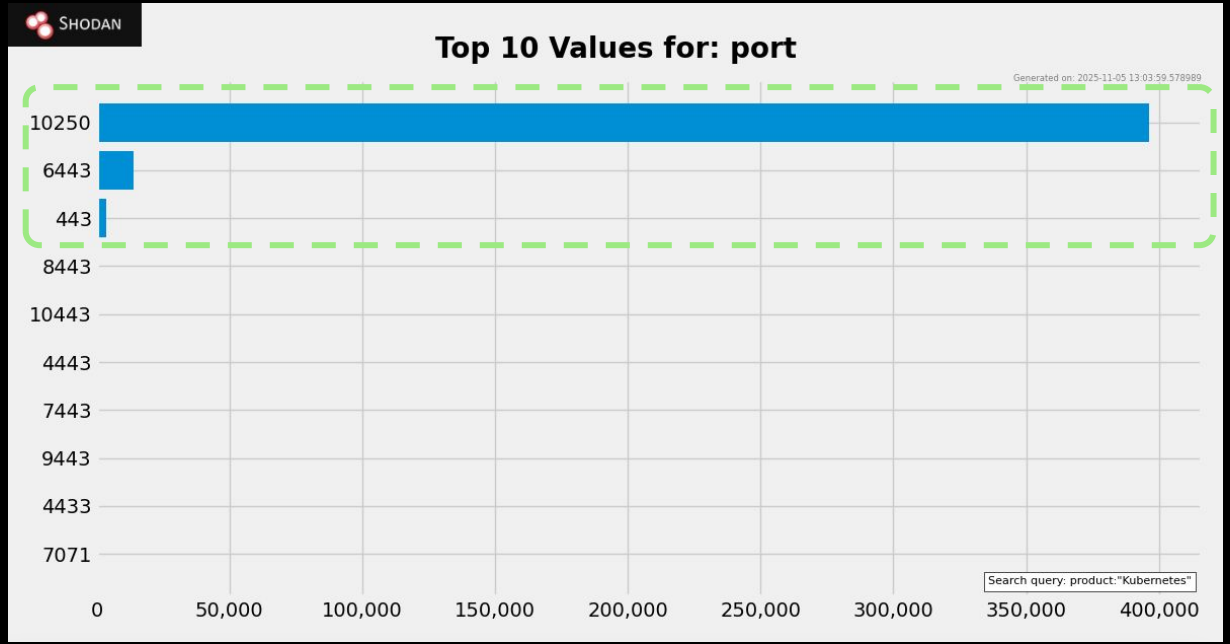
https://kubernetes.io/docs/concepts/overview/components/

Kubernetes Control Plane Components

Protocol	Direction	Port Range	Purpose	Used By
TCP	Inbound	6443	Kubernetes API server	All
TCP	Inbound	2379-2380	etcd server client API	kube-apiserver, etcd
TCP	Inbound	10250	Kubelet API	Self, Control plane
TCP	Inbound	10259	kube-scheduler	Self
TCP	Inbound	10257	kube-controller-manager	Self

https://kubernetes.io/docs/reference/networking/ports-and-protocols/

K8s Components Exposure



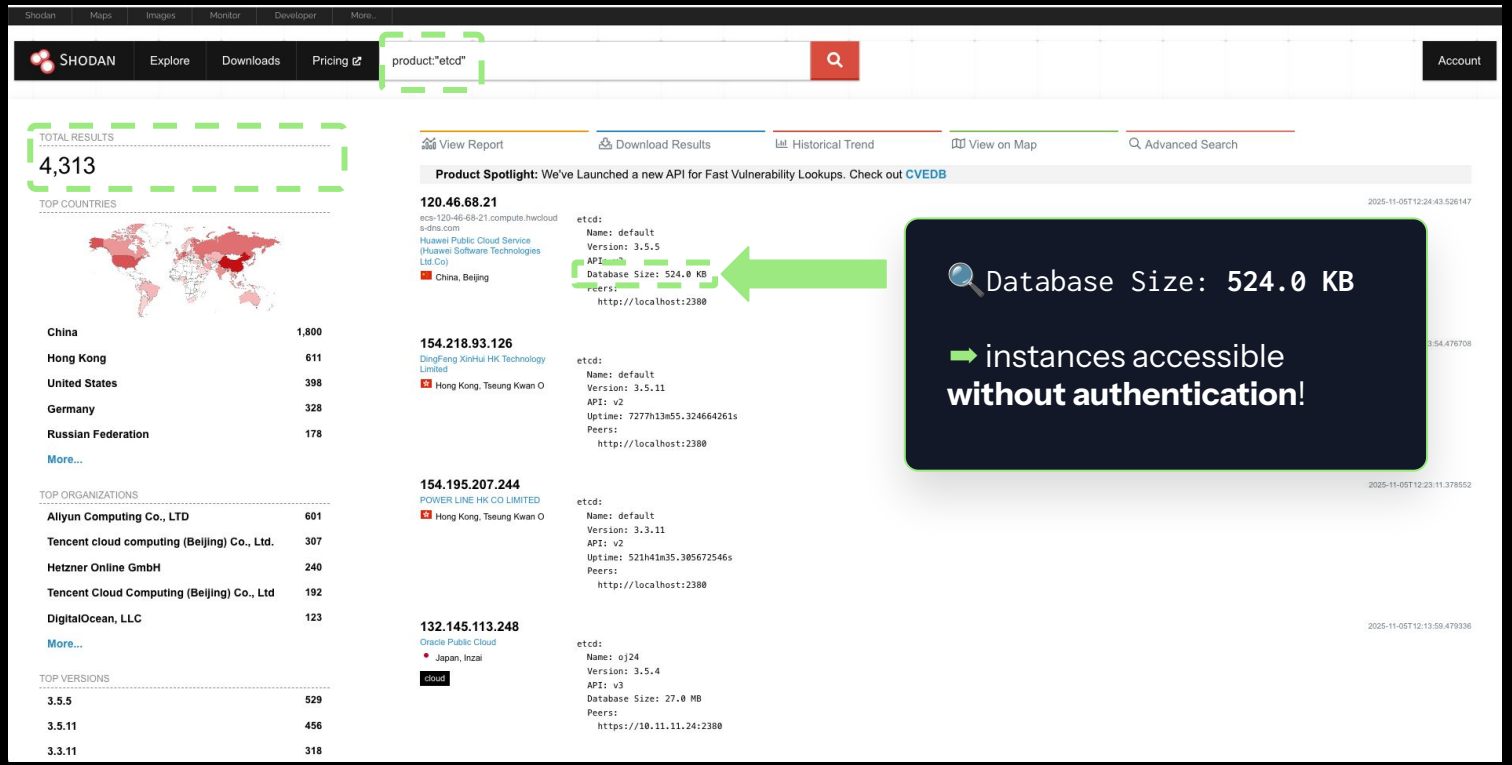
ℹ️ less than 20 kube-scheduler and kube-controller-manager ports exposed.

https://www.shodan.io/search/facet.png?query=product%3A%22Kubernetes%22&facet=port

01
etcd

Public Exposures Analysis

Instances Publicly Exposed



101

a distributed key value store

etcd maps keys to values

01 Kubernetes datastore

etcd persists cluster state, and configuration data
backup it to restore a cluster

02 Hierarchical organization

/config/database
/dashboard/debug_flag

03 Not only Kubernetes

Other cloud native projects: apisix, calico...

Exposure Impacts

Securing etcd clusters

Access to etcd is equivalent to root permission in the cluster so ideally only the API server should have access to it. Considering the sensitivity of the data, it is recommended to grant permission to only those nodes that require access to etcd clusters.

To secure etcd, either set up firewall rules or use the security features provided by etcd. etcd security features depend on x509 Public Key Infrastructure (PKI). To begin, establish secure communication channels by generating a key and certificate pair. For example, use key pairs `peer.key` and `peer.cert` for securing communication between etcd members, and `client.key` and `client.cert` for securing communication between etcd and its clients. See the example scripts provided by the etcd project to generate key pairs and CA files for client authentication.

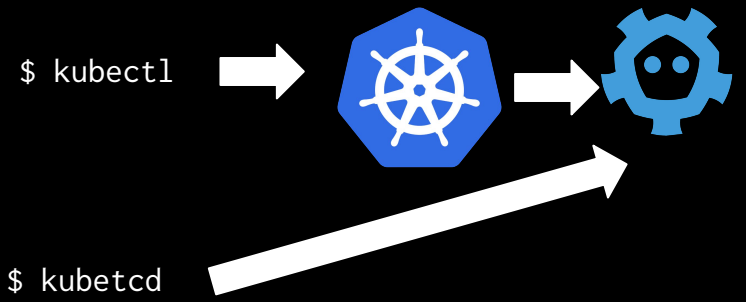
https://kubernetes.io/docs/tasks/administer-cluster/configure-upgrade-etcd/

Exploitation Scenarios

→ **deployments via direct etcd write**
bypass normal API server validation

→ **pods than cannot be deleted**
due to inconsistent data

→ **deploy a remote shell**
via privileged pods



https://github.com/nccgroup/kubetcd

Communicating with etcd instead of kube-api-server offers **new perspectives** previously explored by **Luis Toro** in kubetcd!

Exposed Instances Analysis

roots of etcd keys are linked to applications

/registry → Kubernetes
/apisix
/tidb
/timescaledb

4,306	1,827	14	**9**
from Shodan	containing keys	with /registry	**with *Running* pods**
	3374 etcd dumped	12 /registry/namespaces 11 /registry/pods	6 clusters with 3 nodes, or more

Takeaways

→ **network isolation**
- --listen-client-urls with private addresses only
- avoid localhost bindings
- use firewalls and security groups

→ **sensitive data protection**
- store secrets in a vault
- encrypting data stored in etcd

→ **no authorization model**
- any certificate has full access to etcd

→ **don't install K8s yourself**
- publicly exposed etcd are likely self-configured installation
- offload etcd security to providers & K8s distributions

https://kubernetes.io/docs/concepts/security/api-server-bypass-risks/

02
Kubernetes API Server

Leaked Secrets Analysis

K8s Secrets Types

Built-in Type	Usage
Opaque	arbitrary user-defined data
kubernetes.io/service-account-token	ServiceAccount token
kubernetes.io/dockercfg	serialized ~/.dockercfg file
kubernetes.io/dockerconfigjson	serialized ~/.docker/config.json file
kubernetes.io/basic-auth	credentials for basic authentication
kubernetes.io/ssh-auth	credentials for SSH authentication
kubernetes.io/tls	data for a TLS client or server
bootstrap.kubernetes.io/token	bootstrap token data

https://kubernetes.io/docs/concepts/configuration/secret/

Kubernetes API Server

Authentication

main types

TLS certificates

JSON Web Tokens
no expiration by default

Used to access clusters as admins or services.

Leak Example

```
apiVersion: v1
kind: Config
clusters:
- name: minikube
  cluster:
    server: https://10.211.55.7:8443
    certificate-authority-data: LS0tLS1CRUd[..]LS0tCg==
users:
- name: minikube
  user:
    client-certificate-data: LS0tLS1CRUd[..]LS0tCg==
    client-key-data: LS0tLS1CRUd[..]LS0tCg==
contexts:
- name: minikube
  context:
    cluster: minikube
    user: minikube
current-context: minikube
```

Testing TLS Certificates Using curl

```
# Step 1 - If needed, convert to PEM
echo LS0tLS1CRUd[..]LS0tCg== | base64 -d > cert.pem
echo LS0tLS1CRUd[..]LS0tCg== | base64 -d > key.pem

# Step 2 - Check RSA modulus
openssl rsa -in key.pem -modulus -noout | sha256
2034187c7d1e2433a63b6c275d5f67957da7c3424e508dfa66281d1b476b095f
openssl x509 -in cert.pem -modulus -noout | sha256
2034187c7d1e2433a63b6c275d5f67957da7c3424e508dfa66281d1b476b095f

# Step 3 - Use credentials with curl
curl --insecure --key key.pem --cert cert.pem https://10.211.55.7:8443/api/v1/namespaces
{
  "kind": "NamespaceList",
  "apiVersion": "v1",
  "metadata": {
    "resourceVersion": "1155"
  },
  "items": [
    {
      "metadata": {
        "name": "default",
[..]
```

Exploitation Scenarios

→ **lateral movement**
deploy pods
elevate privileges to get a shell on a node

→ **persistence**
create or alter service accounts

→ **credentials access**
retrieve secrets and configs

https://insomnihack.ch/talks/beyond-the-surface-exploring-attacker-persistence-strategies-in-kubernetes/

Leaks Chain Reaction

Publicly exposed leaks lead to private DockerHub images, and **other secrets being exposed**.

Leaked Secrets Analysis

Certificates

Unique Hosts:	5,846
Valid credentials:	74
Active clusters:	**47**

JWT

Unique Hosts:	99
Valid credentials:	18
Active clusters:	**14**

44 unique active clusters
4 with more than 10 nodes

30% exposed for more than 2 years
one cluster exposed since 2021

only 10% secrets deleted since the leak

valid JWT barely seen on GitHub

Takeaways

→ **network isolation**

don't expose K8s components publicly
use providers features to restrict traffic

→ **logging & monitoring**

monitor access to the cluster
alert on suspicious activities

→ **public access & long-lived credentials**

use short-lived tokens (i.e. OIDC - AWS IAM, Azure AD, Okta...)
authorization with least privilege principle

03
Kubernetes Container Registries

Leaked Secrets Analysis

K8s Secrets Types

Built-in Type	Usage
Opaque	arbitrary user-defined data
`kubernetes.io/service-account-token`	ServiceAccount token
`kubernetes.io/dockercfg`	serialized `~/.dockercfg` file
`kubernetes.io/dockerconfigjson`	serialized `~/.docker/config.json` file
`kubernetes.io/basic-auth`	credentials for basic authentication
`kubernetes.io/ssh-auth`	credentials for SSH authentication
`kubernetes.io/tls`	data for a TLS client or server
`bootstrap.kubernetes.io/token`	bootstrap token data

https://kubernetes.io/docs/concepts/configuration/secret/

Docker Config Secrets

two types

dockercfg

legacy format, barely seen

dockerconfigjson

```
echo $DOCKERCONFIGJSON | jq -R '@base64d | fromjson | .auths'
{
  "https://index.docker.io/v1/": {
    "username": "blackalps",
    "password": "raclette",
    "auth": "YmxhY2thbHBzOnJhY2xldHRlCg=="
  }
}
```

Leak Example

```
apiVersion: v1
kind: Secret
metadata:
  name: blackalps
data:
  .dockerconfigjson:
eyJhdXRocyI6eyJodHRwczovL2luZGV4LmRvY2tlci5pby92MS8iOnsidXNlcm5hbWUiOiJibGFja2FscHMiLCJwYXNzd29yZCI6InJhY2xldHRlIiwiYXV0aCI6IllteGhZMnRoYkhCek9uSmhZMnhsZEhSbENnPT0ifX19Cg==
type: kubernetes.io/dockerconfigjson
```

Using Registries Credentials

login

retrieve a bearer token using credentials

endpoints are provider dependant

pull / push

standard Docker registry API

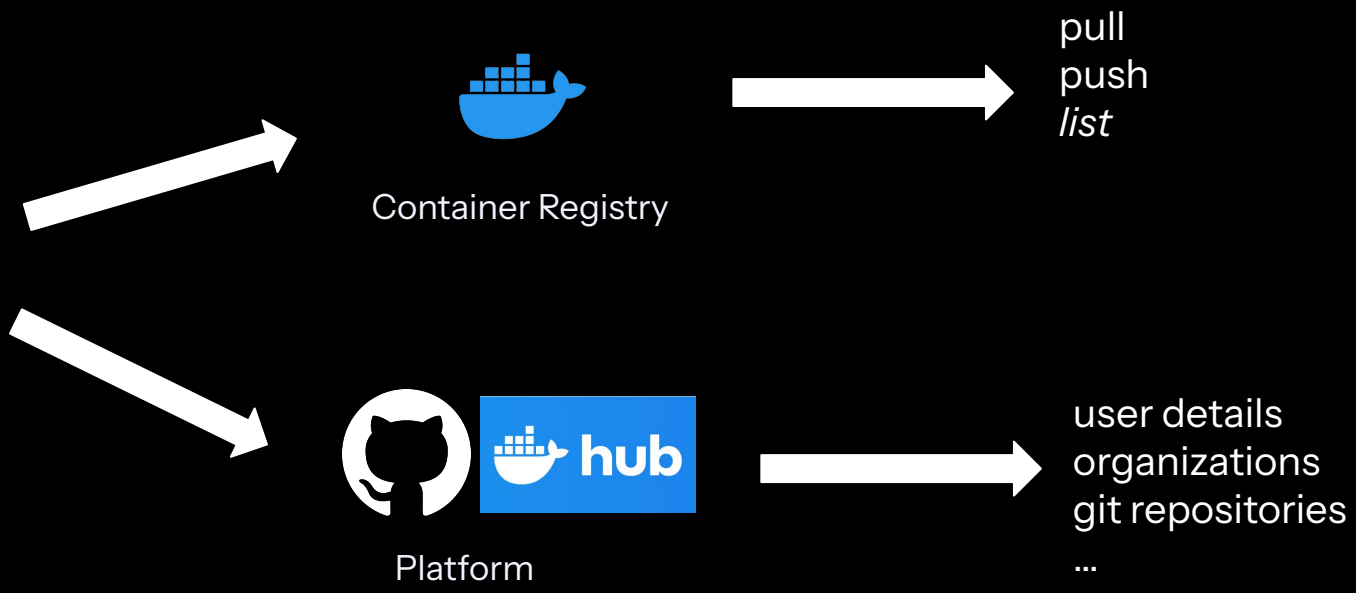
enumeration

standard (i.e. /v2/_catalog) or provider dependant

Login & Pull
Example

```
$ docker login -u $USERNAME -p $PASSWORD
$ docker pull $USERNAME/image:$TAG
```

More Than Registries



Exploitation Scenarios

→ **discovery**
access other Docker images
enumerate git repositories

→ **lateral movement**
compromise Docker images, pulled by the privileged pods

→ **credentials access**
more secrets in private registries

Leaked Secrets Analysis

4,055 auths

2,034 resolved auths

- **1,025** DockerHub
- **480** GitHub
- **276** Quay
- **249** GitLab
- **59** Azure

46% valid

309 DockerHub contain **private images**

730 private GitHub **repositories exposed**

some leaks as old as 2022

Takeaways

→ **avoid using public container registries**
private content and public services don't mix well

→ **least privileges credentials**
aim for read-only (i.e. pull)
not possible with GitHub Container Registry!

→ **registries credentials outlive K8s clusters**
many still valid years after the initial leak
revoke them when the cluster is decommissioned

Responsible Disclosures

16 companies contacted

Identification based on secrets metadata, email addresses, pods names, containers names, hostnames...

01

etcd

4 owners identified
2 invalid contacts
2 pending answers

02

Kubernetes API Server

7 owners identified and contacted
3 pending answers

03

Kubernetes Container Registries

4 owners identified
1 pending answer

Final Thoughts

→ **leaked K8s secrets are diverse**

misconfigurations make them worse
lateral movements scenarios are real

→ **post raclette checklist**

1/ limit K8s components exposures
2/ enforce short lived & scoped credentials
3 /use private containers registries
4/ rotate leaked credentials
5/ audit your images for hard-coded secrets!
6/ implement RFC9116 (i.e. /.well-known/security.txt)

https://www.lemonde.fr/pixels/article/2019/03/02/pourquoi-vous-n-aurez-jamais-d-emoji-raclette-dans-votre-smartphone_5430435_4408996.html

GitGuardian

Thank you!

Question time 🔥

gitguardian.com

Guillaume VALADON | @guedou