Kubernetes Threat Modeling

Securing Cloud Native Applications

November 2023

James Callaghan

james@control-plane.io
https://control-plane.io/training

Introduction

- 👋 Instructor: James Callaghan, Security Architect at ControlPlane
- ⚔️ DevSecOps, Cloud Native and Open Source security consultancy
- 🧐 We work with regulated organisations: banks, governments, healthcare, energy suppliers, insurance

This workshop shares our experience Threat Modeling Kubernetes-based systems

- Slides: https://shorturl.at/uKXY4

Course goals

- By the end of this course, you will understand:
  - How to build **effective lightweight threat models**
  - **Key controls** to implement on **Kubernetes**
  - Cloud Native **roles and responsibilities**
  - **Best practices** to start your team with Threat Modeling

- And you'll be able to:
  - Lead effective **threat modeling workshops**
  - **Identify threats** to your systems
  - **Progressively harden** Kubernetes and your infrastructure
  - **Mitigate identified threats** and risks and understand how good a job you've done

Prerequisites

- To participate in this course, you will need:
    - **Pen and paper** for exercises
    - Access to materials in **git repo**:
        - https://github.com/controlplaneio/threat-modelling-labs
- We expect you know **Kubernetes basic concepts and terminology**
    - But we can help if you don't!
- Threat Modeling is **collaborative**
    - Unsure of a threat? We'll teach you **how to ask!**
- Learn Kubernetes security in more detail with **control**plane:
    - Hacking Kubernetes [Book]
    - Kubernetes and Container Security, Advanced Kubernetes Security [Training]

Poll

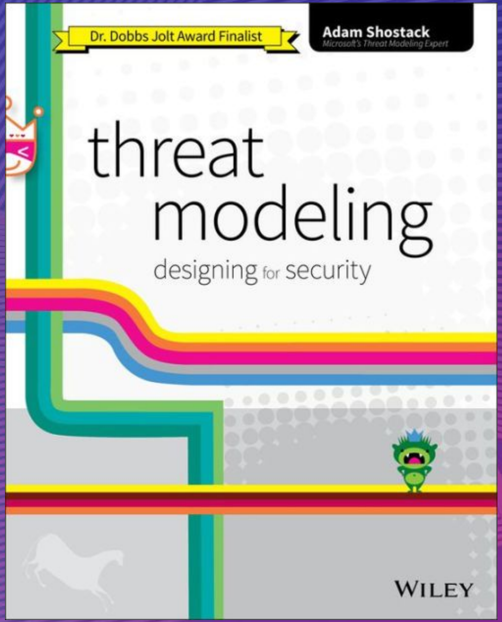
Have you carried out a threat model before?

- Yes
- No

What is Threat Modeling?

- Threat Modeling is:
  - Identifying and enumerating **threats and vulnerabilities**
  - Devising **mitigations**
  - Prioritising **residual risks**
  - **Escalating the most important risks**

- Why Threat Model?
  - Identify **security flaws early**
  - **Save money and time** consuming redesigns
  - **Focus your security requirements**
  - Identify **complex risks and data flows** for critical assets



Everyone can (and should!) Threat Model - **not just security teams**

How to Threat Model

- Four steps:
  - What are we **building?**
  - What can **go wrong** once it's built?
  - What are we **going to do about** the things that can go wrong?
  - Are we doing a **good job of analysis?**
- Run workshops:
  - Structure your workshops around the four questions
  - Get as many **different views** as possible - development, operations, QA, product, business stakeholders, security
  - Workshops can be run to
    - **document system architecture**
    - **generate threats**
    - **devise controls**

Course Summary

- This course will focus on each of the **four threat modeling questions** in a series of 40-50 minute modules.
- Where possible, the modules will address the **threat modeling theory** before applying this to Kubernetes and cloud native systems.
- Throughout the course we will follow an example company, **Boats, Cranes & Trains Logistics (BCTL)**, as they try to defend their systems against the evil pirate **Captain Hashjack**.

The Threat Modeling Process

1 — 2 — 3 — 4

What Are We Building?

What Can Go Wrong?

What Will We Do About It?

Did We Do a Good Job?

Module 1: What Are We Building?

Module 1:
Process - Documenting your system

Goals of this Process

- Understand your business's **data and value**
- Understand your **adversary**
- Decide on the **Threat Model's scope**
- Agree on the desired **threat mitigation level**
    - Linked to **Risk Appetite**
- Identify applicable **infosec policies, compliance standards**
- Agree on the **architecture**
    - **Architecture diagrams**
    - **Sequence diagrams**
    - **Data flow diagrams**

Understand your data - case studies

- Impact variance examples:
  - a **financial institution** would be likely to grade impact levels based on **financial loss**
  - in a **military context**, impact could be related to **loss of life** and **operational failures**
- Impact may be different across the **fundamental properties**: **C**onfidentiality **I**ntegrity and **A**vailability, e.g.
  - When providing bank details to **receive payment**, the **integrity of the information** is essential
  - When **making a card payment** to someone else, **confidentiality** is key

Understand your data - BCTL

Data BCTL processes, and CIA impact assessment using a simple 1-3 scale:

Data description	C	I	A
Customer details, including PII, bank details, addresses, shipment details	**3**	*2*	*2*
Shipment tracking information	**3**	*2*	*1*
Government compliance information	*1*	**2**	**2**
Application and infrastructure code	*2*	**3**	*2*

Understand your adversary

- **Threat landscape** will vary on a case by case basis
- Threat sources profiles:
  - **Skills and resources** available to carry out attacks
  - **Interest level and motivation** to compromise different categories of data
- Consider defending:
  - **Government Intelligence Agency**
    - attacked by highly-capable Foreign Intelligence Services, interested in **obtaining sensitive data**
  - **Game development startup**
    - targeted by rival companies trying to **steal proprietary information**, or script kiddies trying to **gain free access** to a paid service

WANTED: DECOMPILED OR ALLOCATED

DREAD *Pirate*
CAPTAIN HASHJACK

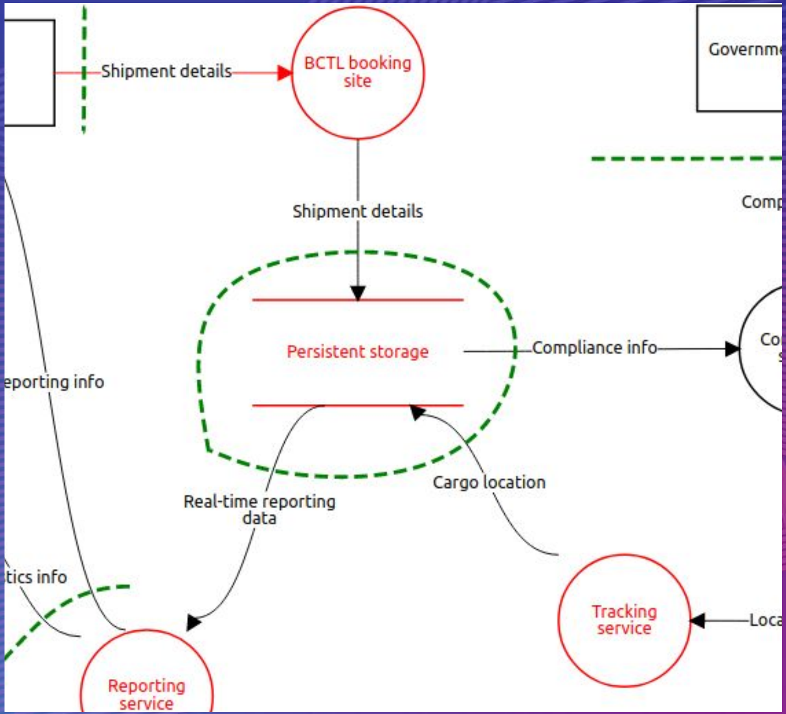
- 8bit monstrosity @captainhashjack
- Scourge of the high internet seas
- CP's archetypal adversary

Adversary Matrix

Actor	Motivation	Capability	Sample attacks
Vandal: Script Kiddie, Trespasser	Curiosity, Personal Fame	Uses publicly available tools and applications (Nmap, Metasploit, CVE PoCs)	Small scale DOS / Launches prepackaged exploits / cryptomining
Motivated individual: Political activist, Thief, Terrorist	Personal Gain, Political or Ideological	May combine publicly available exploits in a targeted fashion. Modify open source supply chains	Phishing / DDOS / Exploit known vulnerabilities
Insider: employee, external contractor, temporary worker	Disgruntled, Profit	Detailed knowledge of the system, understands how to exploit/conceal	Exfiltrate data (to sell on) / Misconfiguration / ”codebombs”
Organised crime: syndicates, state-affiliated groups	Ransom, Mass extraction of PII/credentials/PCI data, financial gain	Devotes considerable resources, writes exploits, can bribe/coerce, can launch targeted attacks	Social Engineering / Phishing / Ransomware / Coordinated attacks
Cloud Service Insider: employee, external contractor, temporary worker	Personal Gain, Curiosity	Depends on segregation of duties and technical controls within cloud provider	Access to or manipulation of datastores
Foreign Intelligence Services (FIS): nation states	Intelligence gathering, Disrupt Critical National Infrastructure	Disrupt or modify supply chains. Infiltrate organisations. Develop multiple zero-days. Highly targeted.	Stuxnet, SUNBURST

Scope the Threat Model

- Multiple techniques for **documenting a system,** e.g. Data Flow Diagrams (DFDs) and information matrices
- DFDs should:
  - Describe the **complete set of data flows**, where **process logic** occurs & **data stores**
  - Describe **trust boundaries**, all **user roles** and **network interfaces**
  - Be **self contained** and most importantly, **accurate**.
- DFDs can be drawn at various levels
  - Level 0: high-level system view
  - Subsequent levels (L1/L2 etc.) drill down into more detail on system components
- Tools for diagrams/documentation:
  - OWASP Threat Dragon etc.



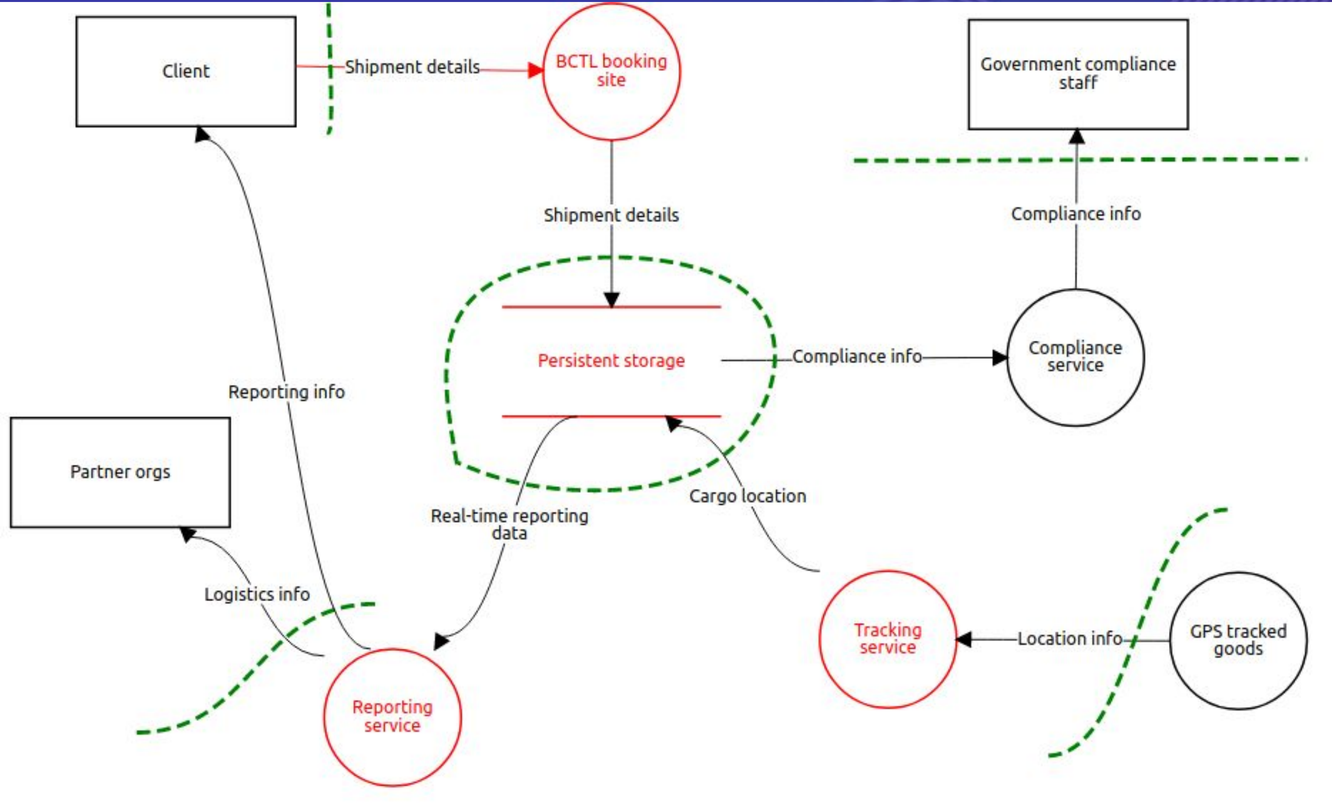
Understand the System

- For BCTL, an **information flow matrix** will be used to understand data flows (in a later lab)

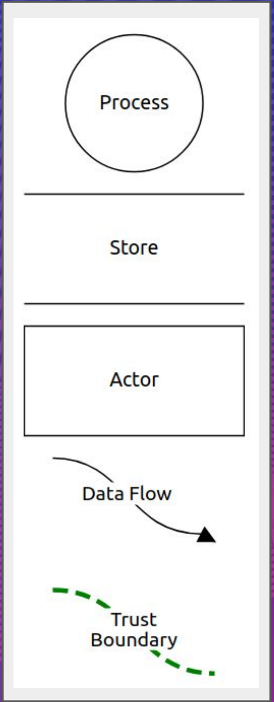
- Tools for **generating diagrams** from cloud environments/infrastructure:
    - Cloudmapper
    - Lucidchart cloud (Lucidscale)
    - Cloudcraft

Data	Highest Impact Level (C/I/A)	Source	Destination	Protocol	Encryption	Authentication	Network route
Example: customer details	*Confidentiality*	*customer*	*Ingress controller*	BCTL responsibility	BCTL responsibility	BCTL responsibility	*public*
Developer K8s API calls		kubectl	API server	HTTPS	yes		public
Customer details	Confidentiality	Ingress controller	Booking pod	BCTL responsibility	BCTL responsibility		cluster-internal
Customer details	Confidentiality	Booking pod	PostgreSQL Pod			BCTL responsibility	cluster-internal
Customer details	Confidentiality	PostgreSQL Pod	Persistent storage	AWS responsibility	AWS responsibility	AWS responsibility	

BCTL Level 0 Data Flow Diagram



Legend



@

controlplane

DFD: Intro to Cloud Native Terminology

- Cloud native systems can be thought about in terms of:
  - **Workloads**
    - **Virtual machines**, **Containers,** or **FaaS products**, e.g. AWS Lambda
  - **Networking**
    - **Overlay** networks, e.g. Calico, Flannel, **cloud provider networking** services, e.g. AWS VPCs
  - **Storage**
    - **Block storage**, e.g. AWS EBS, **object** storage, e.g. AWS S3, **file storage** or **network file** shares, e.g. AWS EFS
  - **Control Plane**
    - Controls where **workloads are scheduled** to run, e.g. **Kubernetes**, **AWS EC2**, **GCP GCE**
- Public cloud providers have **APIs** allowing customers to interact with these types of services
- **Identity and access management (IAM)** is crucial
- For the rest of this course we will be interested in **workloads running in containers on a Kubernetes cluster**

What are we building? - Kubernetes detail

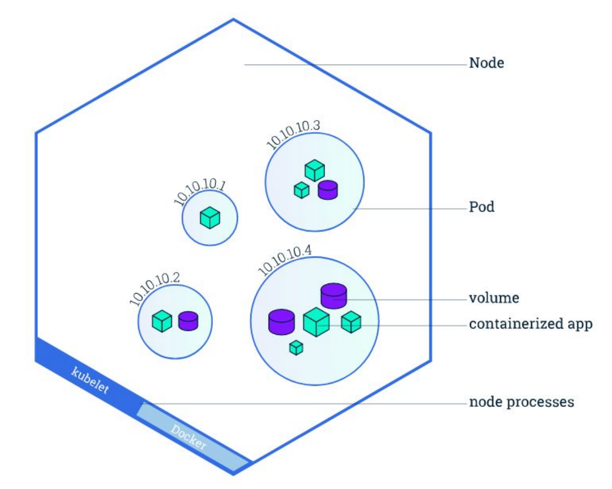
Poll

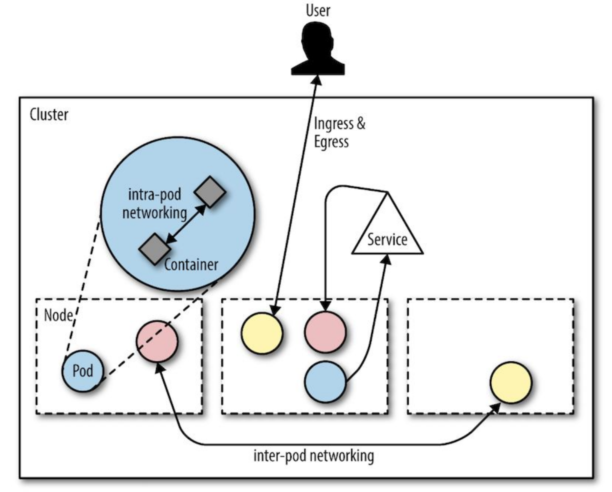
Have you worked with Kubernetes in Production before?

- Yes
- No

Kubernetes: Lightspeed Intro 1

- Workloads:
    - **Nodes** must run the **Kubelet** and a **container runtime**
    - **Pods** run on Nodes in the cluster
    - **Pods** are groups of **containers** which share resources and run your applications
- Networking:
    - **Ingress** exposes HTTP and HTTPS routes from outside the cluster to services within the cluster
    - **Services** expose pods (running your apps) outside the cluster
    - An **overlay network** allows pods on any node to communicate with each other (over an underlay network)





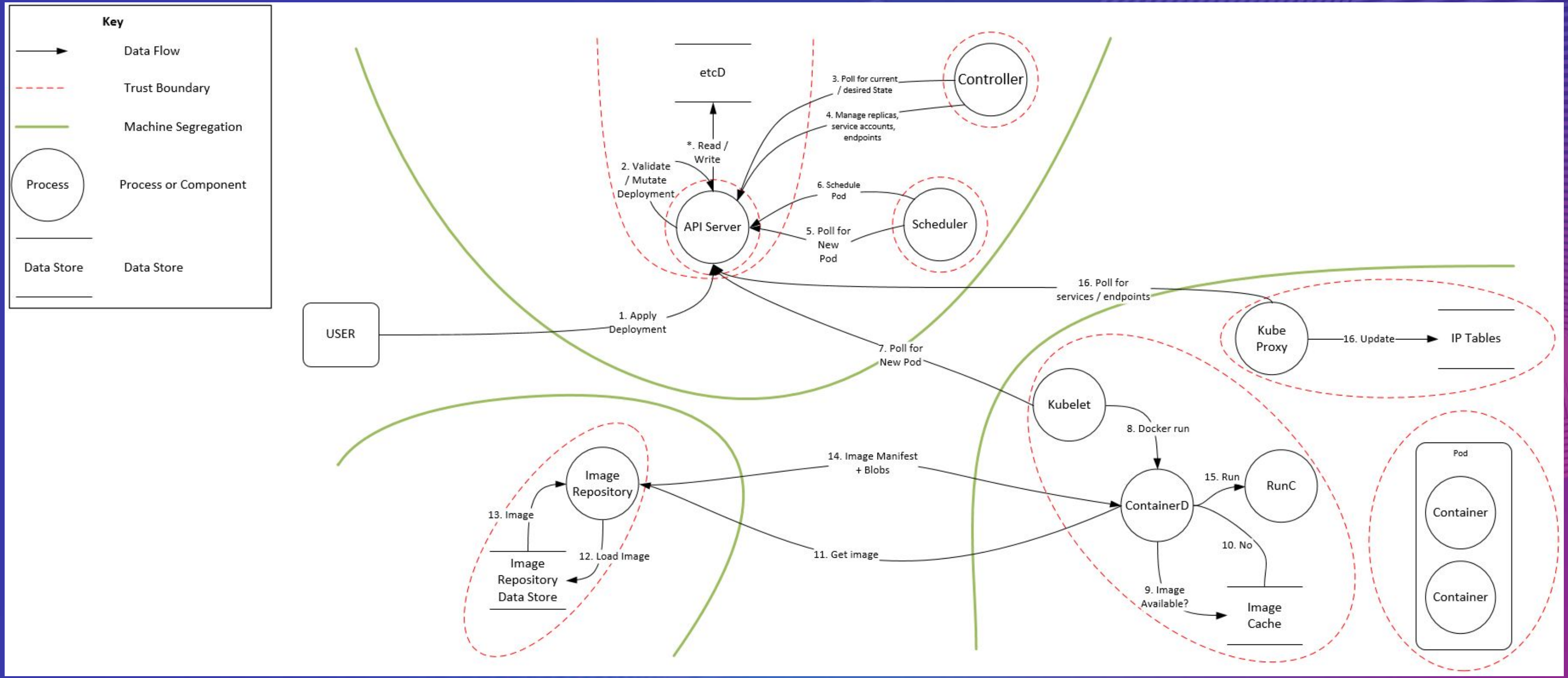
Source

@controlplaneio

Kubernetes: Lightspeed Intro 2

- Control Plane:
  - The Kubernetes **API server** provides the frontend to the cluster's state
    - RESTful API, kubectl is a convenience wrapper for humans
  - **Scheduler** - matches Pods to Nodes so that Kubelet can run them
  - **Controller manager** - runs multiple **controller** processes
    - **Controllers** manage Kubernetes resource lifecycles
- Storage:
  - Containers in a pod do not share a mount namespace by default
    - Optional **shared volume(s)** - a filesystem mounted into the container's local filesystem
  - To persist data outside of the pod's lifecycle, **Persistent Volumes** are used
  - The Kubernetes API server persists its state in **etcd** - a distributed key-value store
    - etcd is only available through the API server

Kubernetes Data Flow Diagram



@controlplaneio

controlplane

[CNCF financial-services-user-group K8s DFD]

Kubernetes Multitenancy (1/2)

- By default, **Kubernetes is not configured to host multiple tenants**
- Work is needed to make it secure
  - **Soft multitenancy**: assumes **partially trusted** clients
    - Often "tenant per namespace"
    - Flexible operational policy optimised for tenants
    - Not resilient to serious or zero-day attacks

  - **Hard multitenancy**: assumes **hostile tenants**
    - Tenants only perform a **restricted** set of operations
    - Optimised for isolation
    - Likely to resist some zero days through **layered controls**
    - More **human and computationally expensive**

Kubernetes Multitenancy (2/2)

- Visibility of **K8s RBAC** resources is scoped to one of:
  - **to a namespace** (e.g. Pods, ServiceAccounts)
  - **to the whole cluster** (e.g. ClusterRoles, Nodes)

- **Segregate a tenant** application into **multiple namespaces**
  - Policy and controls are often namespace-bound (network policy, admission control, RBAC, etc)

- For hard multitenancy, consider:
  - Security **guardrails in CI/CD pipelines**
  - **Admission control** to enforce policy at runtime
  - **Intrusion detection**
  - **Continuous compliance**

Clusters in the wild

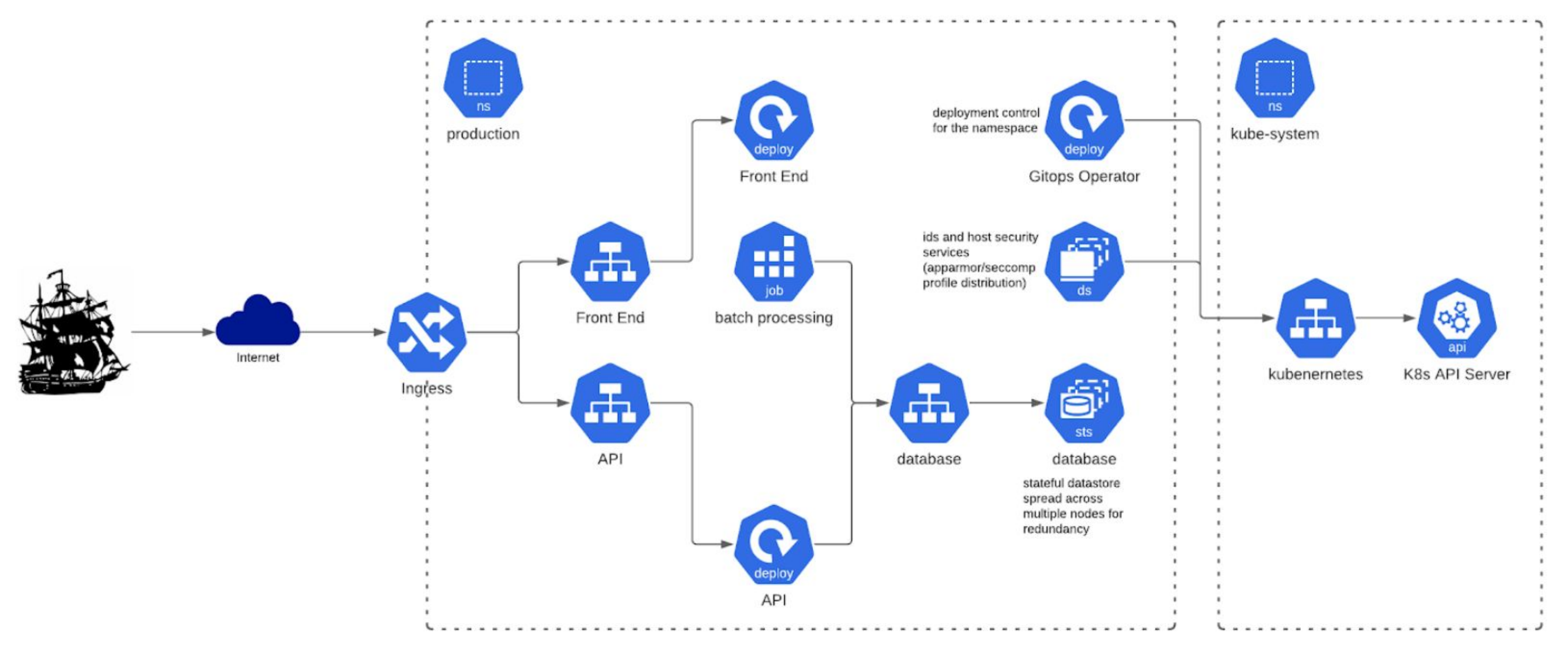
- Rory McCune (@raesene) has blogged about finding publicly accessible Kubernetes cluster on the Internet
- Using the censys search facility, valuable information for attackers can be found
- Many clusters will make the **`/version` endpoint available** on the API server **without authentication**
- Armed with the public IP address and the version of Kubernetes being run, Captain Hashjack can start to plan their assault!

Understand your technical architecture

- BCTL run their booking, tracking, reporting and compliance services:
    - on a **Kubernetes cluster**
    - **running on AWS EC2** instances
    - managed by the BCTL cloud infrastructure team

- The team are not Kubernetes experts, and have used `kubeadm` with default settings and the following choices:
    - **Multiple master configuration** for control plane components
    - **Private network segments for nodes**, with internet traffic coming from a load balancer
    - **etcd is run in a stacked topology** (i.e. on the control plane nodes, not hosted in K8s itself)
    - **EC2 EBS for persistent volumes** (providing storage to pods)
    - **Publicly accessible Kubernetes API server**, to allow staff to work from their home networks

BCTL system architecture



Lab 1

- BCTL Engineers have populated an **information flow matrix**, capturing some of their business data flows, as well as some Kubernetes internals
- Some fields have been **left blank**
  - There is not always a undisputable correct answer
  - The intention of the lab is to provoke thought ahead of our threat identification session in the next module
- https://github.com/controlplaneio/threat-modelling-labs

Lab 1 walkthrough

The Threat Modeling Process

1 — 2 — 3 — 4

1. What Are We Building?
2. What Can Go Wrong?
3. What Will We Do About It?
4. Did We Do a Good Job?

Module 2: What Can Go Wrong?

Module 2:
Process - Identifying Threats

Summary of this Process

- **Inputs**:
  - Outcomes from steps discussed in Module 1: “What are we building?”

- **Actions**:
  - **Brainstorming threats** independently, then pool the results
  - Run through **STRIDE against the DFD**
  - **Annotate existing architecture diagrams** with threats
  - **Create Attack Trees** to enumerate threats
  - If required, **prioritise threats** using consensus or quantitative risk scoring.

Gathering Techniques and Threat Sources

- You and your team's **experience**, **gut instincts**, and **intuition**!
- Threat intelligence sources
  - MITRE ATT&CK framework
  - Open Web App Security Project (OWASP)
  - Common Weakness Enumeration (CWE)
  - Common Attack Pattern Enumeration and Classification (CAPEC)
  - CNCF financial services attack trees
  - Microsoft Kubernetes attack matrix
  - Enhanced Kubernetes attack matrix
- Modeling techniques
  - STRIDE
  - Attack Trees

MITRE ATT&CK for containers

Initial Access 3 techniques	Execution 4 techniques	Persistence 4 techniques	Privilege Escalation 4 techniques
Exploit Public-Facing Application	Container Administration Command	External Remote Services	Escape to Host
External Remote Services	Deploy Container	Implant Internal Image	Exploitation for Privilege Escalation
Valid Accounts (2)	Scheduled Task/Job (1)	Scheduled Task/Job (1)	Scheduled Task/Job (1)
	User Execution (1)	Valid Accounts (2)	Valid Accounts (2)

Defense Evasion 7 techniques	Credential Access 3 techniques	Discovery 3 techniques	Lateral Movement 1 techniques	Impact 3 techniques
Build Image on Host	Brute Force (3)	Container and Resource Discovery	Use Alternate Authentication Material (1)	Endpoint Denial of Service
Deploy Container	Steal Application Access Token	Network Service Discovery		Network Denial of Service
Impair Defenses (1)	Unsecured Credentials (2)	Permission Groups Discovery		Resource Hijacking
Indicator Removal				
Masquerading (1)				
Use Alternate Authentication Material (1)				
Valid Accounts (2)				

TODO: link

OWASP Top Ten

*“A primary aim of the OWASP Top 10 is to **educate** developers, designers, architects, managers, and organizations **about the consequences of the most common** and most **important web application security weaknesses**”*

OWASP Top Ten Web Application Security Risks | OWASP

- A01:2021-Broken Access Control
- A02:2021-Cryptographic Failures
- A03:2021-Injection
- A04:2021-Insecure Design
- A05:2021-Security Misconfiguration
- A06:2021-Vulnerable and Outdated Components
- A07:2021-Identification and Authentication Failures
- A08:2021-Software and Data Integrity Failures
- A09:2021-Security Logging and Monitoring Failures
- A10:2021-Server-Side Request Forgery

Microsoft Threat Matrix for Kubernetes

Initial Access	Execution	Persistence	Privilege Escalation	Defense Evasion	Credential Access	Discovery	Lateral Movement	Collection	Impact
Using Cloud credentials	Exec into container	Backdoor container	Privileged container	Clear container logs	List K8S secrets	Access the K8S API server	Access cloud resources	Images from a private registry	Data Destruction
Compromised images in registry	bash/cmd inside container	Writable hostPath mount	Cluster-admin binding	Delete K8S events	Mount service principal	Access Kubelet API	Container service account		Resource Hijacking
Kubeconfig file	New container	Kubernetes CronJob	hostPath mount	Pod / container name similarity	Access container service account	Network mapping	Cluster internal networking		Denial of service
Application vulnerability	Application exploit (RCE)	Malicious admission controller	Access cloud resources	Connect from Proxy server	Applications credentials in configuration files	Access Kubernetes dashboard	Applications credentials in configuration files		
Exposed Dashboard	SSH server running inside container				Access managed identity credential	Instance Metadata API	Writable volume mounts on the host		
Exposed sensitive interfaces	Sidecar injection				Malicious admission controller		Access Kubernetes dashboard		
							Access tiller endpoint		
							CoreDNS poisoning		
							ARP poisoning and IP spoofing		

= **New technique**

= **Deprecated technique**

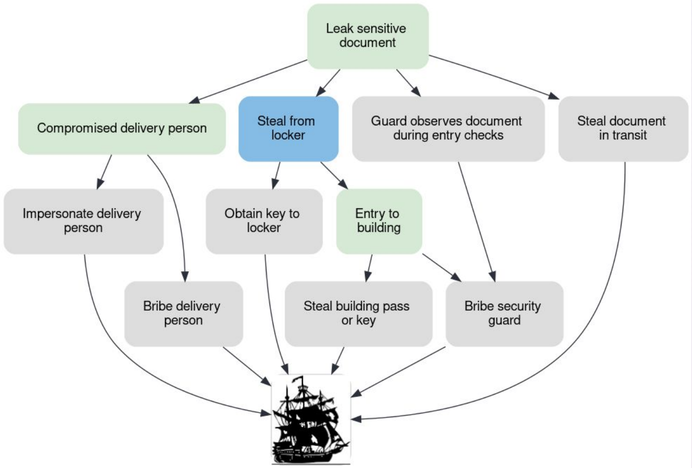
STRIDE

- For each process, data flow, store or actor, identify what might go wrong:
    - **Spoofing**
        - **Pretending** to be something or someone you're not
    - **Tampering**
        - **Modifying** something you're not supposed to modify. This can be on disk, in memory, and/or in transit
    - **Repudiation**
        - **Claiming you didn't do something**, whether or not you actually did
    - **Information Disclosure**
        - **Exposing information** to people who aren't authorised to see it
    - **Denial Of Service**
        - Taking actions to **prevent the system** from providing service to **legitimate users**
    - **Elevation of Privilege**
        - Being able to perform operations you aren't supposed to be able to perform

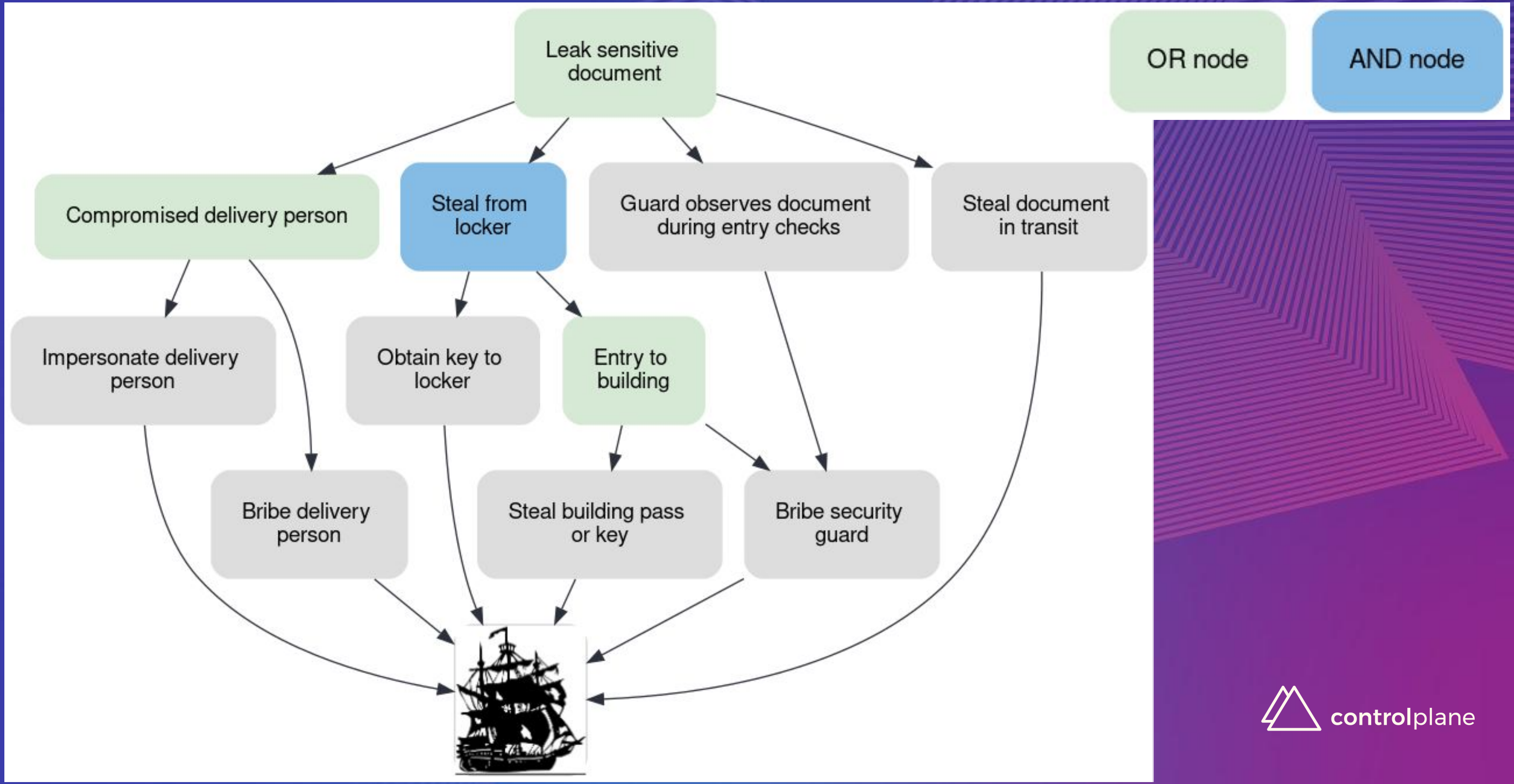
Attack Trees

*"**represent attacks against a system in a tree structure**, with the goal as the root node and different ways of achieving that goal as leaf nodes"*

- Can **annotate** in a number of ways to provide **extra context** e.g.:
  - **Attack costs** in order to understand attack **likelihood and required resource**.
  - **Mitigating security controls** to the security of the system and threat events not covered.
- Attack trees should **enhance** the threat modeling process and **not overwhelm**
  - E.g. focus on areas of most interest



Example Attack Tree ("carry a briefcase to a locker")



Attack Trees as code

- **Graphviz** can be used to **represent attack trees as code**:
  - Kelly Shortridge leads the way https://swagitda.com/blog/posts/security-decision-trees-with-graphviz/
- **Deciduous** is a web application that makes this easier:
  - https://swagitda.com/deciduous/

- We have provided a Dockerfile generate trees in PNG format:
  - `git clone https://github.com/controlplaneio/threat-modelling-labs`
  - `cd threat-modelling-labs/course-materials`
  - `docker build -t graphviz-render .`
  - `cat example_tree.dot | docker run --rm -i graphviz-render > example_tree.png`
- Our sample tree captures a **subsection of confidentiality threats** to BCTL's system

Applying this to Cloud & Kubernetes

What Can Go Wrong: Cloud Native vs On-Prem

- BCTL run a **Kubernetes-based**, **Cloud Native** system
- There are **fundamental differences** between threat modeling these:
  - **on-premise Microsoft enterprise systems**, most common, lowest common denominator
  - **Cloud Native systems**, newer technologies
- On-premise systems have been the de-facto standard for longer
  - many **more attacks have been observed**
  - **more threat research** has been performed

What Can Go Wrong: STUXNET

- **Stuxnet** is one of the most infamous computer worms of all time
  - discovered in 2010
  - **targeted industrial control machinery** and processes
  - responsible for **significant damage to Iran's nuclear programme**

- Point of entry was via a **malicious USB stick**
  - USB stick entered into computer on air-gapped network
  - network connected Windows machines could then be infected

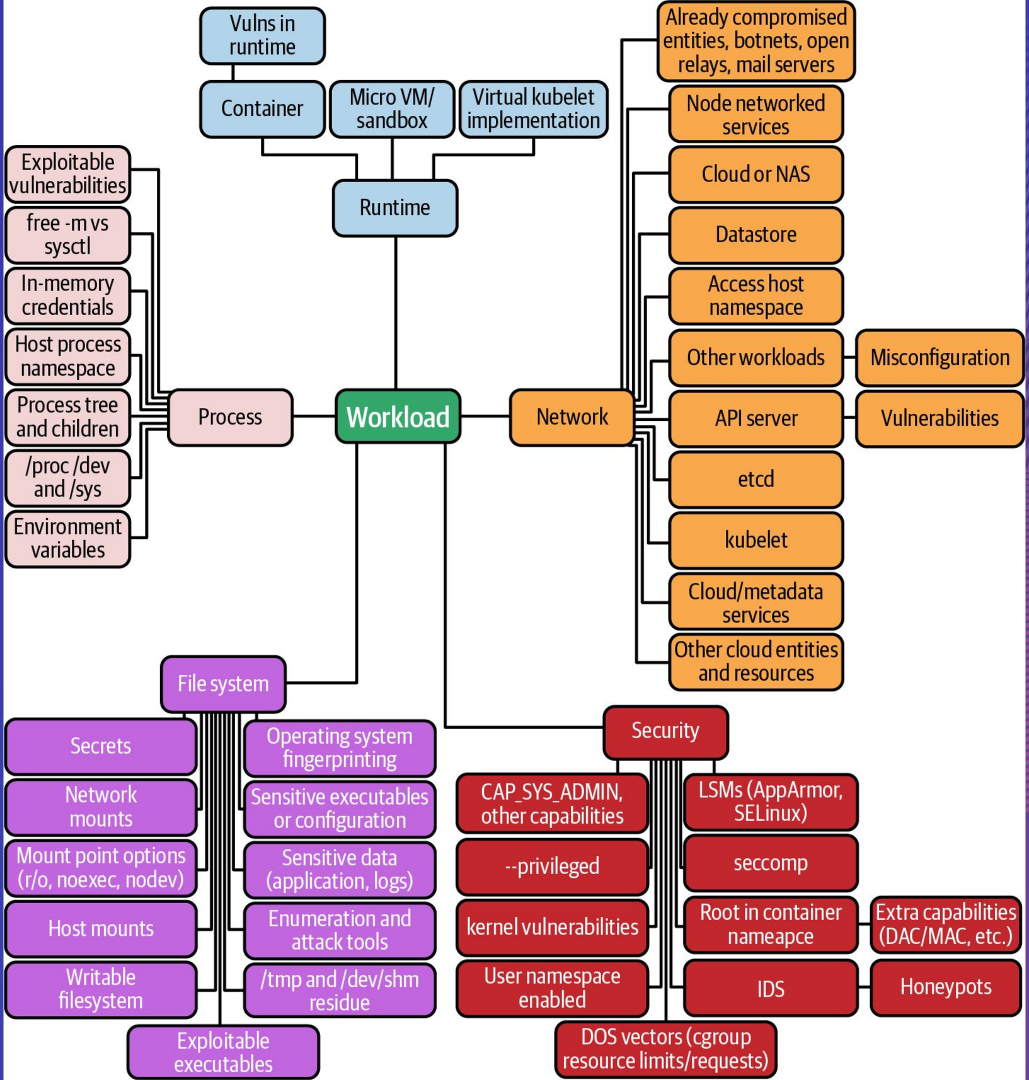
- The worm sought out machines running the **Siemens Step7** industrial control software
- **Zero-day vulnerabilities** were exploited
- **False feedback** was provided to monitoring systems to ensure Machinery failed **before the attack was spotted**

Key Differences: Cloud Native vs On-Prem

- **Shared responsibility model** places responsibility for certain controls on the **cloud provider**
- Management access to cloud systems is through public portals
  - Protection of **credentials** and robust **multi-factor authentication** is crucial
  - Management access to on-prem infrastructure often involves further steps, e.g. **VPN access**
- Cloud provider APIs make **automation much easier**
- The speed of change is much higher for cloud native systems, due to:
  - the introduction of **new services** by cloud providers, and the **evolution** of existing ones
  - the ability for teams to release products **more quickly**
  - Automated roll out leads to increased **focus on misconfiguration**
- Cloud native encourages **Infrastructure as Code** deployments
  - The **integrity** of infrastructure code is paramount
- Layers of abstraction
  - On prem: **virtual machines**
  - Cloud native: **containers** and **serverless**

What Can Go Wrong: BCTL Kubernetes Workloads

- **RBAC misconfigurations** for either Kubernetes or AWS services could lead to unauthorised access to workloads
- **Remote code execution**: if a network facing workload contains a vulnerability that can be exploited to run untrusted code
- **Container breakout:** could occur if a pod has a weak security context, e.g. a privileged pod, any shared host namespaces, absent seccomp, &c.
- **Compromised credentials:** stolen workload identity
- **Application and supply chain attacks:** a springboard for other types of compromise, such as exfiltration of sensitive data, e.g. if public images and/or libraries contain vulnerabilities



@controlplaneio

controlplane

A note on Kubernetes Service Accounts

- https://kubernetes.io/docs/tasks/configure-pod-container/configure-service-account/
- By default, Kubernetes **does not assign identities to services**
- **A namespace's default service account** is assigned to a pod if not specified
- Service account credentials are **automatically mounted** (unless specified otherwise) into a pod and do not expire
- Service account credentials give access to the Kubernetes API (from inside a pod, or anywhere the API is available)
- Cloud provider APIs are often called by containerised workloads, however:
  - In a **multi-tenant environment**, multiple tenants' containers will **run on the same node**
  - IAM roles attached to hosts allow malicious tenants to impersonate a second tenant by calling the EC2 metadata API to retrieve the appropriate credentials

What Can Go Wrong: BCTL Kubernetes Networking

- Using a default `kubeadm` installation, BCTL will have the following setup:
    - **Flat topology**: every pod can see and talk to every other pod in the cluster
    - **No security context**: workloads can escalate to host network interface controller (NIC)
    - **No environmental restrictions**: workloads can query their host and cloud metadata
    - **No encryption on the wire**: between pods and cluster-externally
- As BCTL have made the Kubernetes API **publicly accessible**, compromising a developer's `kubeconfig` file would be very attractive for Captain Hashjack
- As **no network policy or egress control** is in place, exfiltration of data could be possible from a compromised workload
- With no security context, a compromised pod (with hostNetwork or privilege) could **sniff unencrypted traffic on any interface on the host**

What Can Go Wrong: BCTL Kubernetes Storage

- Access to other pod's persistent volumes is a **danger to the confidentiality of sensitive workloads**
  - Persistent volumes persist data outside of the pod's lifecycle
- A Kubelet mounts volumes into a pod, **which may hold plaintext secrets**, mounted into pods for use at runtime
  - Root on the node has **access to all the secrets** of every workload running on it
- A **container runtime bug** means **container breakouts could be possible**, e.g. runc /proc/self/exe vulnerability
  - Recall container filesystems exist on the host's filesystem
- With BCTL's stacked etcd topology, if a privileged pod is run on a control-plane node, **data stored in etcd** may be accessible

What Can Go Wrong: Summary

- Lots!
  - Workloads
  - Networking
  - Storage
- Kubernetes Control Plane
  - Supply chain
  - Insider threats
  - CVEs
  - Misconfiguration
  - Stolen credentials
  - Compromised underlying infrastructure
  - ...and much more!

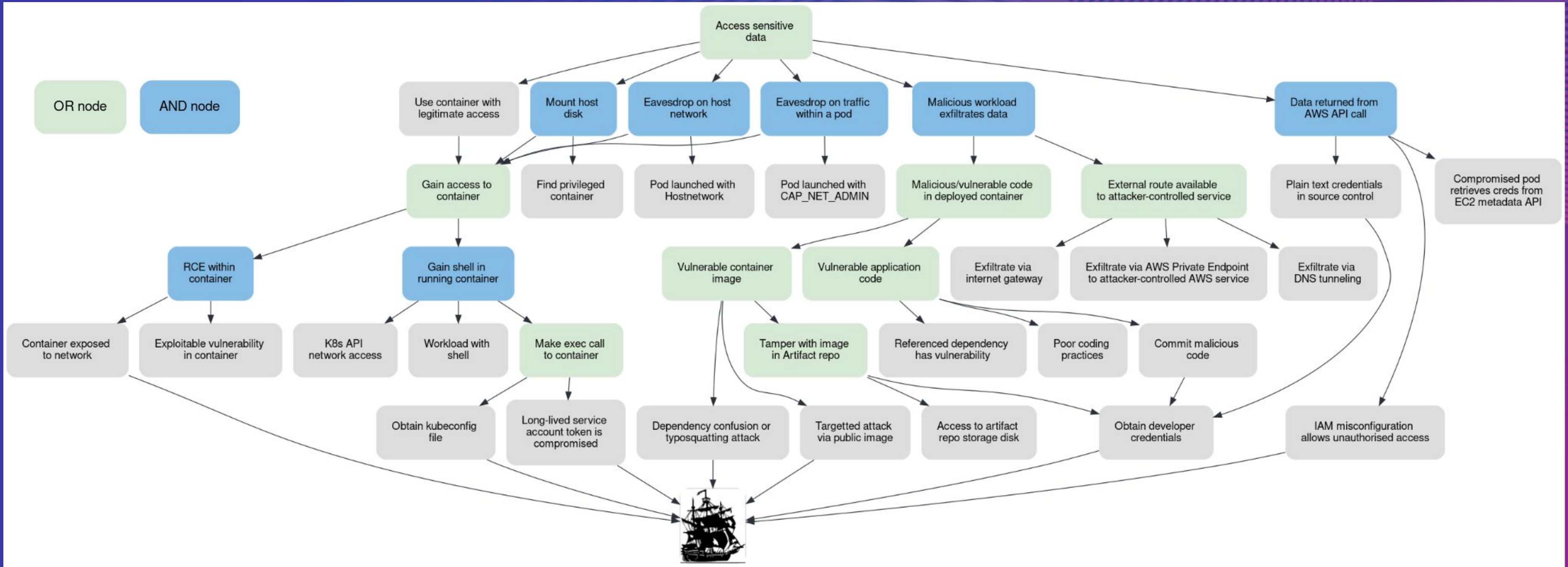
Exercise - STRIDE for BCTL (Lab 2)

- https://github.com/controlplaneio/threat-modelling-labs

A	B	C	D	E	F	G
	Spoofing	Tampering	Repudiation	Information Disclosure	Denial of Service	Escalation of Privilege
Workloads						
Networking						
Storage						
Control Plane						

Lab 2 walkthrough

- A list of consolidated threats has been provided in a sample spreadsheet
- These are mapped to the below attack tree
  - This is not a complete attack tree, and only shows a few branches for demonstration purposes
- The spreadsheet link and attack tree code are in the course-materials folder in the GitHub repo: https://github.com/controlplaneio/threat-modelling-labs



The Threat Modeling Process

1 — What Are We Building?

2 — What Can Go Wrong?

3 — What Will We Do About It?

4 — Did We Do a Good Job?

Module 3: What Will We Do About It?

Summary of this Process

- **Inputs**:
  - Outputs from previous stage discussed in Module 2
- **Actions**:
  - For each threat, **generate and map security requirements**
    - On **architecture diagrams**
    - Onto **Attack Trees**
    - Against **threats in the table**

  - **Discuss the new security requirements** with the project and stakeholders:
    - Determine **feasibility of implementation**
    - **Identify unacceptable impacts** on business functionality
    - Identify **responsible parties**

Shared Responsibility Model

- Cloud provider makes **some security and maintenance guarantees**
  - e.g. [AWS Artifact](), [GCP Compliance]()
- Clients are responsible for the **usage and configuration of the system**

controlplane

Risk Management

Risk management strategies: based on your risk tolerance!

- **Avoid**: Remove feature that generates risk
- **Mitigate**: Reducing risk by implementing security controls
  - Enforce design or procedural changes that will reduce risk to an acceptable level
- **Accept**: Risk's impact is minor or its probability of occurrence is very low
- **Transfer**: Disclaimers, insurance policy

All threats and risks should be **explicitly addressed** using one of the risk management strategies. None should be ignored.

Mitigating Risk: Types of Control

- **Preventive** - can be good configuration, or controls in CI/CD e.g. linting/tests that prevent misconfigured/vulnerable clusters from being launched
- **Detective** - detecting bad events within logs
- **Corrective** - e.g. log event triggers corrective Lambda (or equivalent workload), can be a workload/control loop in the clusters
- **“Old school”** controls that are **not effective** in cloud
  - Node segregation: nodes are often multi-tenanted
  - On-prem mindset in cloud
    - Heavily manual change control: restricts deployment
    - Restrictive architectures: difficult to “bin pack” workloads
    - Reliance on detective controls: attacks are implicitly permitted to production!

How to Defend Against our Scoped Attacks?

- **Kubernetes and Cloud IAM** is at the centre of Cloud Native security
  - **Review permissions** regularly
  - **User management** processes are key
  - **Review onboarding/offboarding processes** (Joiners/Movers/Leavers, or JML)
  - Implement **strong access control policies**
- **Default Kubernetes service accounts should not be used**
  - Create a dedicated service account for every workload (it's "workload identity")
  - Workloads will interact with the Kubernetes API, or the cloud via a workload identity integration
- Cloud providers services can help, e.g **IAM roles for Service Accounts (IRSA)** in AWS, **Workload Identity** in GKE
  - IRSA uses **Service Account Token Volume Projections**
  - This is just one benefit of using a managed offering, e.g. EKS, GKE, AKS
- For cryptographically strong workload identities, **SPIRE** can be used
  - SPIRE is a production-ready implementation of the **SPIFFE** APIs

Kubernetes RBAC Overview

- Terms:
  - **Identity**: a human user or service account.
  - **Resource**: something (like a namespace or deployment) we want to provide access to.
  - **Role**: is used to define which Kubernetes **verbs** can be performed on resources.
  - **Role binding**: attaches a role to an identity, effectively representing the permissions of a set of actions concerning specified resources.
- Kubernetes verbs are mapped to **HTTP verbs** on resource API endpoints (https://kubernetes.io/docs/reference/access-authn-authz/authorization/)
- RBAC roles are preventative controls

Kubernetes RBAC Overview

- **Increasing verbosity** when running kubectl commands can show the HTTP verbs used in a given kubectl action
- Actions on **subresources** (e.g. pods/exec) are not inherited from the parent resource (e.g. pods)

```
[james@JamesCP threat-modelling-labs]$ kubectl exec -it nginx --v=6 -- echo roflcopter
I0723 10:59:37.960835  122078 loader.go:375] Config loaded from file:  /home/james/.kube/config
I0723 10:59:37.971177  122078 round_trippers.go:443] GET https://127.0.0.1:42783/api/v1/namespaces/default/pods/nginx 200 OK in 6 milliseconds
I0723 10:59:37.987508  122078 round_trippers.go:443] POST https://127.0.0.1:42783/api/v1/namespaces/default/pods/nginx/exec?command=echo&command=roflcopter&container=nginx&stdin=true&stdout=true&tty=true 101 Switching Protocols in 11 milliseconds
                                                                              roflcopter
```

How to Defend: Workloads

- Least privilege **securityContext** should be set for pods
  - This can be checked in an automated pipeline using static analysis tools such as Kubesec
- **Admission control** should be set up to ensure that non-compliant pods will not run on the cluster
  - Pod Security Admission
  - Validating Admission Policy
    - using Common Expression Language (CEL)
  - Custom policy enforcement can also be achieved using technologies such as OPA (e.g. via Gatekeeper) or Kyverno
- Use **container Intrusion Detection Systems (IDS)**, e.g. Sysdig Falco, to spot malicious behaviour in running containers
- Build **automated security testing** into pipelines
  - do not neglect **governance and processes**
  - these must be in place to **make decisions** based on testing output

How to Defend: Workloads - Supply Chain

- **Scan for CVEs** in dependencies using tools like Trivy
  - In container images
  - In git repositories
- Think about how much an attacker could benefit from seeing your code and **restrict access** appropriately
  - Consider **GPG signing of commits** (e.g. using Yubikeys) and container image signing
  - Implement procedures so that authors **cannot merge their own PRs** etc.
- Build security
  - Perform static analysis using **Hadolint and conftest** to enforce policy
  - Use **hardened images** and multi-stage builds
- Organisations can use tools such as **in-toto** to increase trust in their pipelines and artefacts
  - verifiable signatures at each stage of the chain
- The CNCF Security Technical Advisory Group (tag-security) have published a Software Supply Chain Security Paper

How to Defend: Networking

- Enforce **network policy**, e.g. using Calico or Cilium
- **Restrict Kubernetes API access** to approved networks using cloud provider network controls (e.g. AWS security Groups and NACLs)
- Apply **admission control policies** which do not allow dangerous networking configurations, such as `hostNetwork` or the `CAP_NET_RAW` Linux capability
- **Secure Ingress** to allowlist approved networks only
  - Can be at the cloud firewall level, but also using **mTLS and client certificates** if appropriate
  - Terminating TLS at an ingress controller and inspecting traffic allows for **content inspection**
- **Pod-pod encryption** can be configured at the application layer (e.g. making use of an internal PKI orchestrated by CFSSL)
  - Also a **service mesh** (e.g. Istio) could be considered, which also helps with workload identities

How to Defend: Storage

- **Secrets management**, for example, using:
  - Hashicorp Vault
  - external-secrets, integrating with a cloud provider secrets management solution, e.g. AWS Secrets Manager
- **Admission control** can prevent host mounts entirely, or enforce read-only mount paths
- Persistent Volume **encryption at rest**
- Persistent Volume **access controls**

How to Choose Controls

- Once threats have been identified, **security requirements** can be derived
- These requirements are called **controls** or **countermeasures**
  - They provide mitigations against the enumerated threats
- The mapping between threats/risks and controls is well-suited to a table format, e.g.
  - https://docs.google.com/spreadsheets/d/1xLeh2VMTuypsZlitLukePQU9Z21u9GqZy5VFNc61CM0/
- It will **rarely be possible to implement all controls**, due to
  - **Budget** implications
  - Prioritisation of work with respect to other **business requirements**
  - **Balancing operator, developer, and user requirements and "Total Security"**
- Important to **prioritise controls**
  - Assess the **residual risk level** associated with the final selection

Lab 3

- Prioritise controls in terms of security benefit and ease of implementation
  - This exercise is subjective!
  - It is meant to provoke discussion
- https://github.com/controlplaneio/threat-modelling-labs

Lab 3 walkthrough

The Threat Modeling Process

1. What Are We Building?
2. What Can Go Wrong?
3. What Will We Do About It?
4. Did We Do a Good Job?

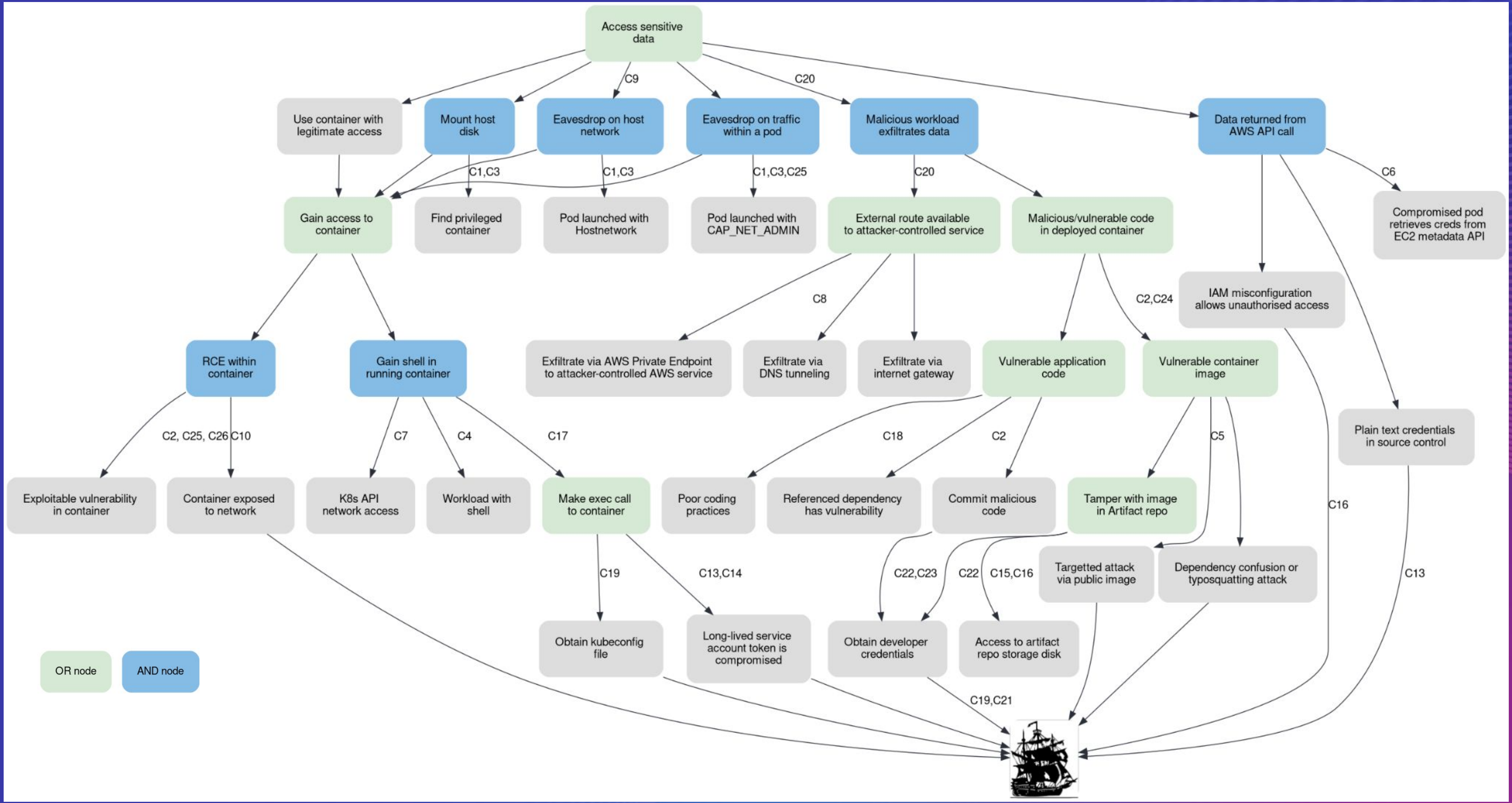
Module 4: Did We Do a Good Job?

Ongoing Threat Modeling Responsibilities

- **Inputs**: outputs of previous stage discussed in Module 3
- **Actions**:
  - **Assess identified threats** and security requirements against reference standards and policy
  - Conduct **internal reviews**
  - Conduct and record the results of **testing**, both automated and external penetration testing
  - Identify, record and **confirm that residual risks are at an acceptable level**
  - **Regularly review and update** the model with new threats and attacks
    - new threats might be mitigated without additional controls being required
- **Outputs**:
  - **Assessment** against compliance standards
  - **Test output, internal automation, external pentest results,** etc
  - Discussion/acceptance of **residual risks**

Mapping Controls to Attack Tree Nodes

- **Select security controls** for implementation
- **Map these controls onto an Attack Tree**
    - This provides a direct means of evaluating effectiveness
    - Visual representations are easier to reason about
- Identify how many nodes and “branches”of the attack tree are covered by the mitigating controls
- **Assess the resulting security of the system**
    - If enough security controls are defined, any **new attacks added to the tree should be mitigated by existing controls**


Access sensitive data
C9
C20
Use container with legitimate access
Mount host disk
Eavesdrop on host network
Eavesdrop on traffic within a pod
Malicious workload exfiltrates data
Data returned from AWS API call
C1,C3
C1,C3
C1,C3,C25
C20
C6
Gain access to container
Find privileged container
Pod launched with Hostnetwork
Pod launched with CAP_NET_ADMIN
External route available to attacker-controlled service
Malicious/vulnerable code in deployed container
Compromised pod retrieves creds from EC2 metadata API
C8
C2,C24
IAM misconfiguration allows unauthorised access
RCE within container
Gain shell in running container
Exfiltrate via AWS Private Endpoint to attacker-controlled AWS service
Exfiltrate via DNS tunneling
Exfiltrate via internet gateway
Vulnerable application code
Vulnerable container image
Plain text credentials in source control
C2, C25, C26
C10
C7
C4
C17
C18
C2
C5
Exploitable vulnerability in container
Container exposed to network
K8s API network access
Workload with shell
Make exec call to container
Poor coding practices
Referenced dependency has vulnerability
Commit malicious code
Tamper with image in Artifact repo
C16
C19
C13,C14
C22,C23
C22
C15,C16
Targetted attack via public image
Dependency confusion or typosquatting attack
C13
Obtain kubeconfig file
Long-lived service account token is compromised
Obtain developer credentials
Access to artifact repo storage disk
C19,C21
OR node
AND node

Scaling

- Jeevan Saini has written about a ‘self-service’ threat modeling ideal - https://segment.com/blog/redefining-threat-modeling/
    - Idea is to **empower Developers and DevOps** engineers to threat model
    - **Feature experts and SMEs** are best placed to discuss potential security issues
- Five stages:
    - Traditional Phase: security do everything
    - Training Phase: security upskill teams
    - Observation Phase: security sit in with teams
    - Review Phase: teams lead, security step back
    - Security Optional Phase: empowered teams
- Centralised threat repo and controls for the enterprise
- Map to tickets and tests

Automated Testing and CI/CD

- **Automated tests** should be devised and run
  - As part of a CI/CD pipeline
  - Every time the **system is deployed** or a **code change made**
- High level test scenarios can be mapped to the **Security Requirements**
- Each test scenario may consist of multiple assertions
  - Document within the test suite itself
  - Add a test pass/fail column if needed
- **Failed tests** indicate a security requirement has not been met
  - **Residual risk** will remain or **remediation** is required
- **Tests should be against threats**
  - Rather than the implementation of a control

Poll

Have you got access to a Linux device or Virtual Machine ?

- Yes
- No

Lab 4 - Automated Testing

- OPA Gatekeeper and bats-detik example
- https://github.com/controlplaneio/threat-modelling-labs

Lab 4 walkthrough

Kubernetes & DevSecOps Training (live / remote)

- Official Kubernetes Training Partner
    - Kubernetes Fundamentals (2 days), Operations (2 days), for Developers (2 days),
    - Kubernetes and Container Security (1 day)
    - Advanced Kubernetes Security: Learn By Hacking (3-4 days)
    - DevSecOps Enablement for Leaders (1 day)
- Certified trainers with production expertise in UK, Europe, North America and Australasia
- Prepares attendees to sit CKA, CKAD, CKS exams
- Informed by our wealth of experience deploying and supporting secure, high compliance, mission-critical distributed systems
- Instructor led hands-on labs, practical examples, and real-world scenarios
- Regularly updated to reflect the latest Kubernetes release
- Includes production debugging and hacking workshops on real clusters