I'm In Your Pipes, Stealing Your Secrets

Securi-Tay 2022

whoami

- Iain Smart
- Former Hacksoc/Securi-Tay bod
- NCC Group Containerisation Practice Lead

Agenda

- Brief intro into CI/CD
- Demo of some attacks
- War stories
- Blue Team advice

Mild Disclaimer

- The examples I'll refer to are skewed towards container-heavy findings
- Minor details have been changed for client confidentiality etc.

CI/CD Overview

nccgroup

CI/CD Introduction

- TL;DR - High levels of automation for testing and deployment
- Allows developers to move faster, and work more centrally
- Actions performed on central compute resources, against central codebase
- (Theoretically) makes devs more efficient

CI - Continuous Integration

- Perform testing against every push/pull request
- Allows testing to be performed before code is merged
- Helps with a "shift-left" mentality
- More testing on smaller changes == faster feedback

CD - Continuous Delivery/Continuous Deployment

- Once tests pass, deploy code to prod
- Devs push once, code automagically ends up running

CI/CD - What's the tech?

- Pipelines
  - Jenkins
  - Github Actions
  - Gitlab CI
  - Azure Devops/AWS CodeCommit/CodeDeploy
- Compute
  - VMs
  - Containers
  - Serverless

CI/CD - An Attacker's View

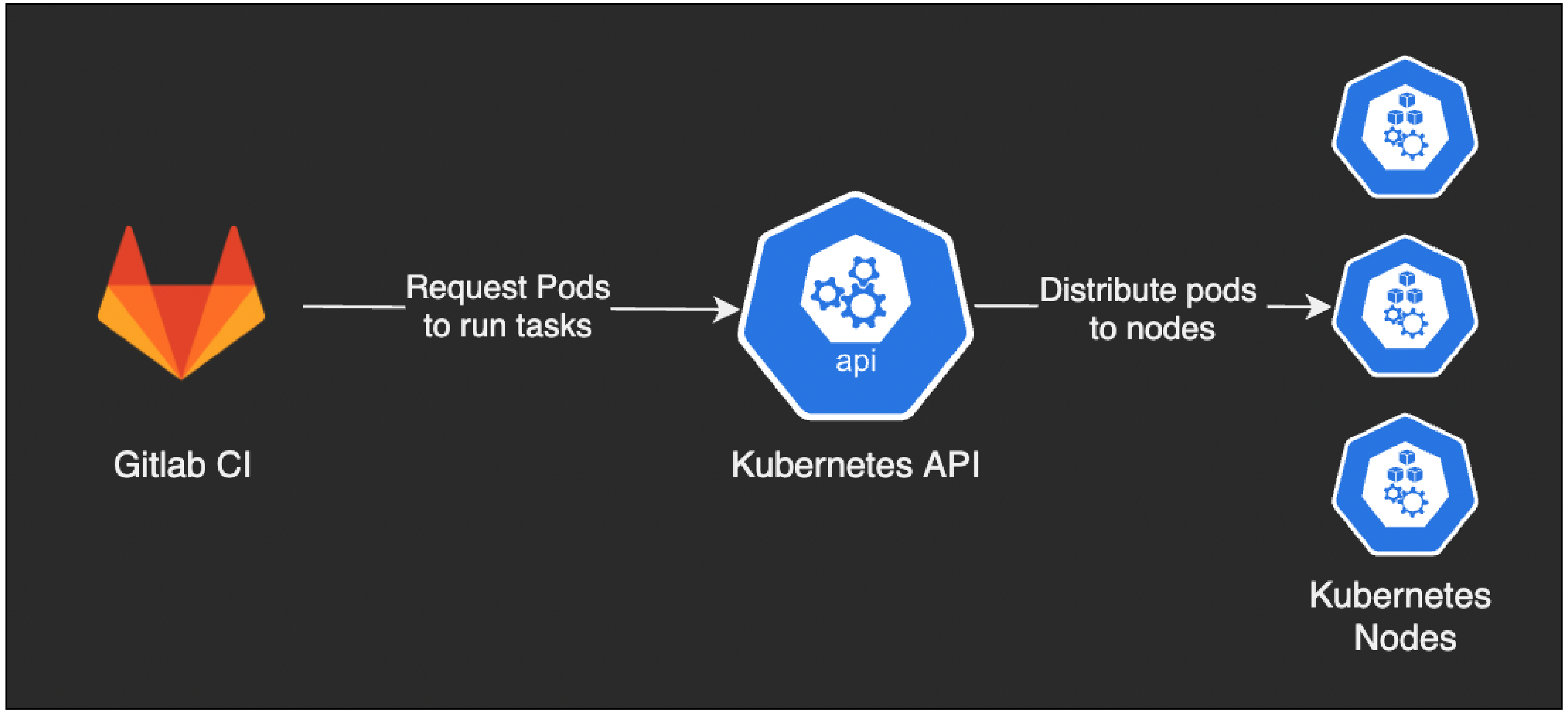
- Pipelines take code, run that code, and have privileged credentials
- Possibly for multiple systems/build environments

Example Pipeline

Quick Example

- Gitlab CI configured with a Kubernetes cluster providing compute resources
- Actions run as `Kubernetes` pods (containers)

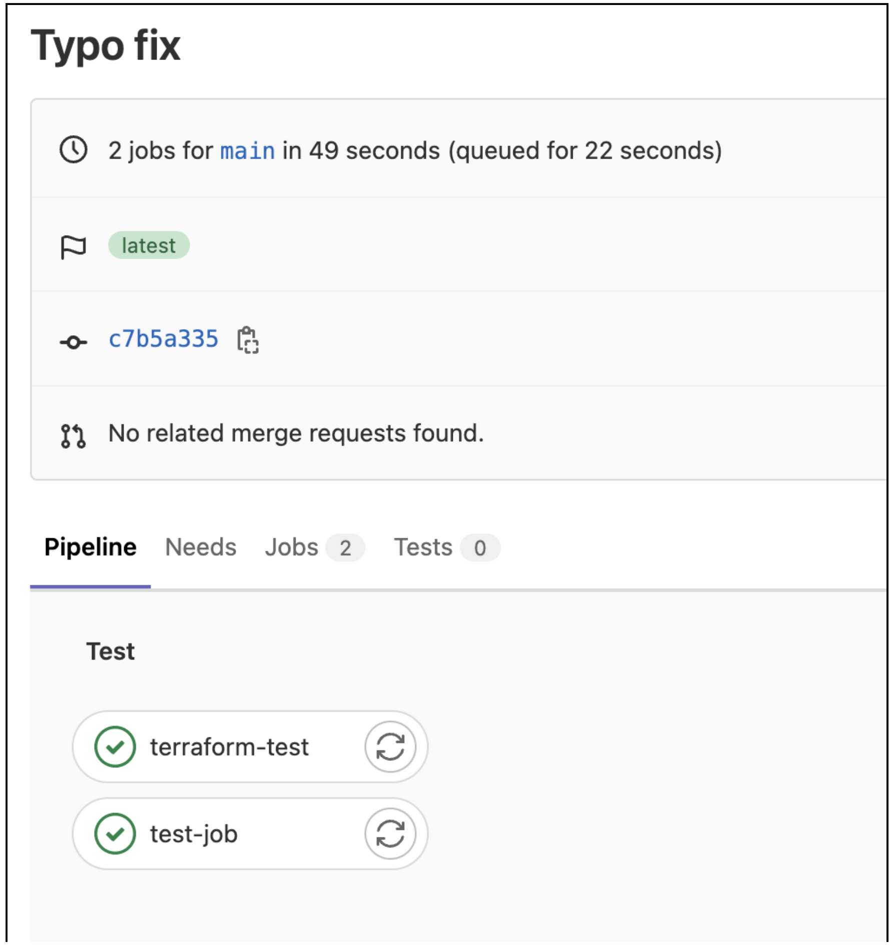
Quick Example - Architecture



Quick Example - Gitlab CI Snippet

```yaml
test-job:
  stage: test
  script:
  - echo "Hello, world! I'm testing Gitlab CI"
```

Quick Example - Gitlab CI Snippet



Typo fix

2 jobs for main in 49 seconds (queued for 22 seconds)

latest

c7b5a335

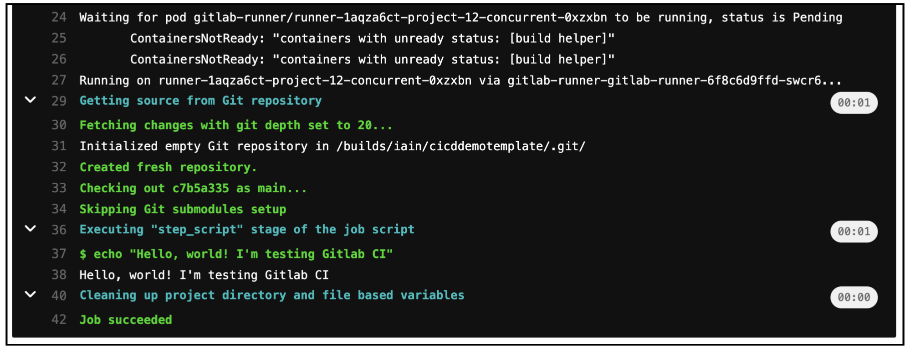
No related merge requests found.

Pipeline Needs Jobs 2 Tests 0

Test

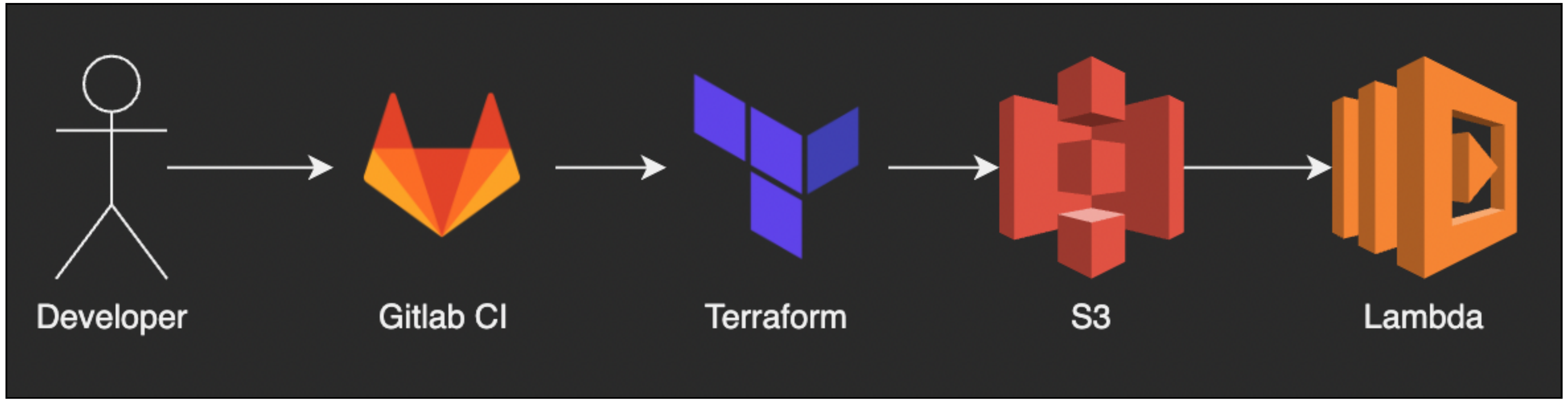
terraform-test

test-job



```
Waiting for pod gitlab-runner/runner-1aqza6ct-project-12-concurrent-0xzxbn to be running, status is Pending
       ContainersNotReady: "containers with unready status: [build helper]"
       ContainersNotReady: "containers with unready status: [build helper]"
Running on runner-1aqza6ct-project-12-concurrent-0xzxbn via gitlab-runner-gitlab-runner-6f8c6d9ffd-swcr6...
Getting source from Git repository                                                          00:01
Fetching changes with git depth set to 20...
Initialized empty Git repository in /builds/iain/cicddemotemplate/.git/
Created fresh repository.
Checking out c7b5a335 as main...
Skipping Git submodules setup
Executing "step_script" stage of the job script                                             00:01
$ echo "Hello, world! I'm testing Gitlab CI"
Hello, world! I'm testing Gitlab CI
Cleaning up project directory and file based variables                                      00:00
Job succeeded
```

Better Example - Deploy a Lambda



- Reminder to self - Do the deploy

Better Example - Deploy a Lambda

```yaml
terraform-deploy:
  stage: deploy
  rules:
  - if: $CI_PROJECT_NAME != "cicddemotemplate"
  script:
  - cd terraform
  - terraform init -backend-config="key=$TF_VAR_PROJECT_PREFIX/terraform.tfstate"
  - terraform apply -auto-approve
  image: git.test.lab:5050/iain/cicddemotemplate/terraform:latest
  only:
    changes:
    - terraform/*
    - demo_function/*
```

War Stories

Compromised Pipeline 1

- Access to an internal git repo, using representative developer credentials
- Codebase was an Apache Maven project
- External dependencies specified from dev-controlled URL
- Deployed through Jenkins runners

Compromised Pipeline 1

- Generated a Meterpreter payload
- Shell callback to attacker-controlled server
- Shell was limited to the build environment

Compromised Pipeline 1

- Recon phase: What's in the box?
- `cd ../../../../`
- Search for secrets
- Find an SSH key

Compromised Pipeline 1

- More recon
- nmap local subnets
- Find SSH servers

Compromised Pipeline 1

- SSH to Jenkins master node
- Dump all Jenkins variables
- Find Kubernetes kubeconfig file
- Compromise production Kubernetes cluster

Compromised Pipeline 2

- Red Team engagement
- Ended up with developer access
- Modified a pipeline to run "printenv"
- Service account credentials in the pipeline

Compromised Pipeline 3

- Internal infrastructure review
- Found a webapp with a SSRF vulnerability
- Read Kubernetes serviceaccounttoken

Brief Aside - K8s Auth

- By default, Kubernetes containers have authentication tokens in a predictable location
- These tokens can be used to authenticate to the apiserver
- Depending on RBAC, can get you various permissions

Aside to the aside - AWS EKS Auth

- AWS EKS uses a Kubernetes configmap called aws-auth
- Maps AWS Roles to Kubernetes roles
- AWS Roles don't need to be in the same AWS account

Compromised Pipeline 3

- SSRF granted access to edit configmaps
- Added AWS role from a different account
- Gained cluster admin over clusted

Compromised Pipeline 3

- Application containing SSRF was mid-build in a pipeline
- K8s cluster was providing compute
- We had now compromised the build pipeline, but not the source repo or prod environment
- `kubectl get pods` lists all env variables for pods
  - This includes git repository secrets
  - Found AWS IAM keys with access to ECR

Compromised Pipeline 3

- Used AWS keys to overwrite ECR image
- Production cluster used pull-based CI
- New image was launched with access to various secrets in production cluster
- Profit

Compromised Pipeline 4

- Developers were not permitted access to production environments
- Developers could make any changes they wanted in development
- Merge requests to main branch required approval
- Pipelines provided through CircleCI
- Pipeline configured through a .circleci.yml file
- Code used secrets as env variables, and used them based on the git branch being built

Compromised Pipeline 4

```
- name: Do Dev things
  image: registry.customer.com/terraform:v0.12
  environment:
    DEV_AWS_ACCESS_KEY_ID:
      from_secret: DEV_AWS_ACCESS_KEY_ID
    DEV_AWS_SECRET_ACCESS_KEY:
      from_secret: DEV_AWS_SECRET_ACCESS_KEY
  commands:
  - terraform apply
  when:
    branch:
    - feature/dev*
```

Compromised Pipeline 4

```
- name: Do Prod things
  image: registry.customer.com/terraform:v0.12
  environment:
    PROD_AWS_ACCESS_KEY_ID:
      from_secret: PROD_AWS_ACCESS_KEY_ID
    PROD_AWS_SECRET_ACCESS_KEY:
      from_secret: PROD_AWS_SECRET_ACCESS_KEY
  commands:
  - terraform apply
  when:
    branch:
    - main
```

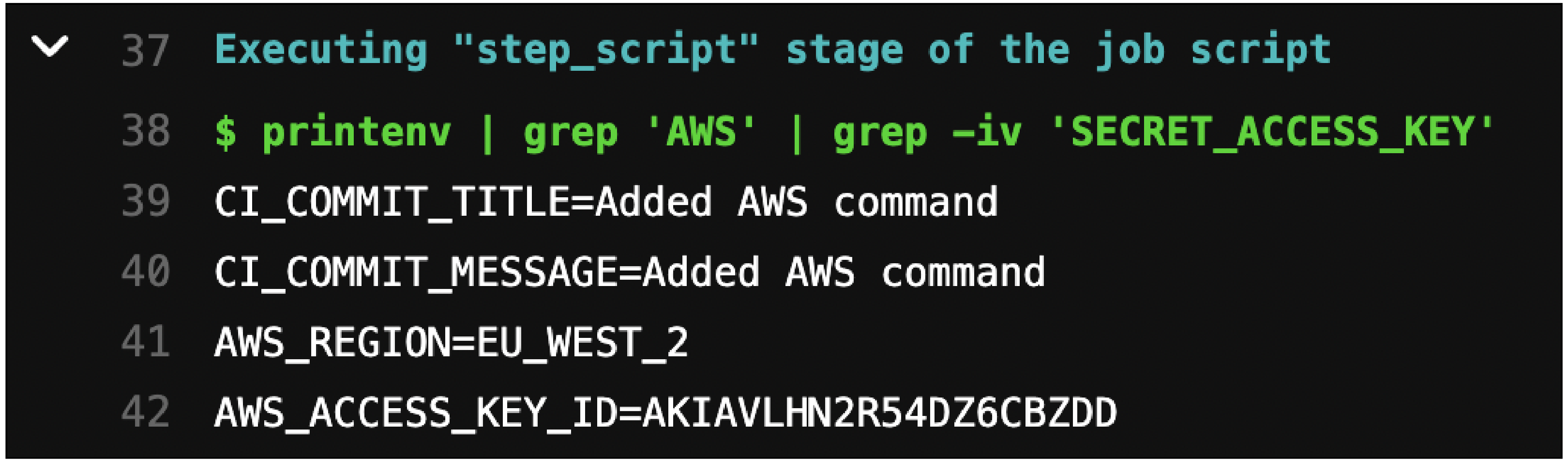
Compromised Pipeline 4

- Developers can change pipeline config file on non-main branches
- Pipeline runs automatically on any branch
- All secrets are available to all pipelines

Compromised Pipeline 4

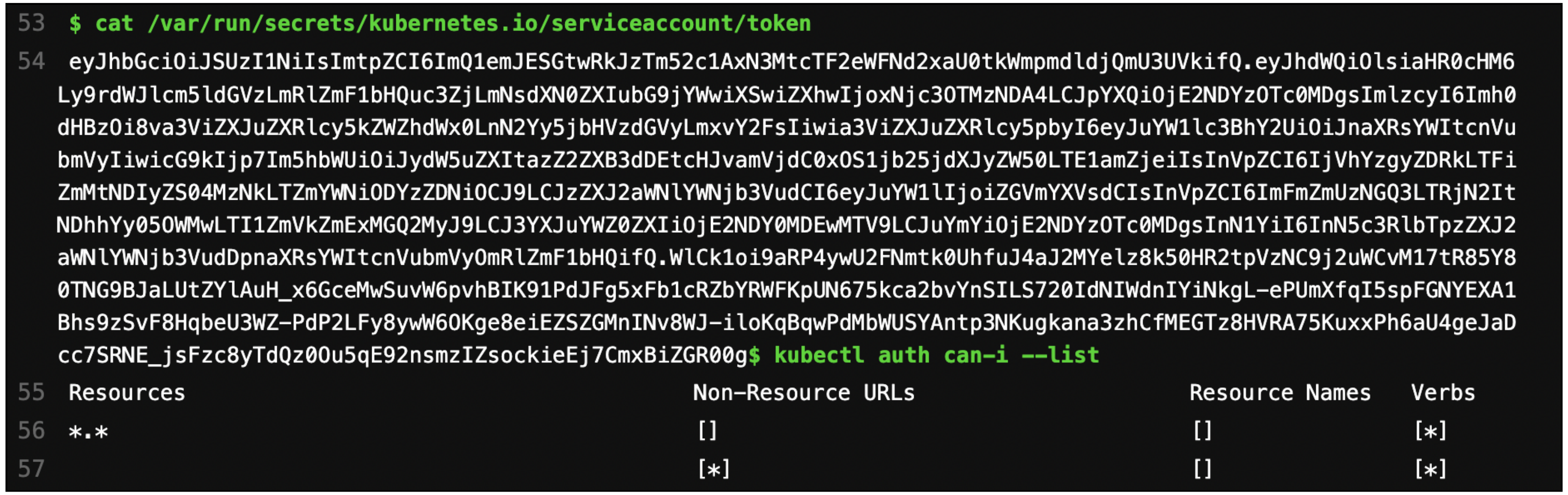
```
- name: Do Hacky things
  image: registry.customer.com/terraform:v0.12
  environment:
    PROD_AWS_ACCESS_KEY_ID:
      from_secret: PROD_AWS_ACCESS_KEY_ID
    PROD_AWS_SECRET_ACCESS_KEY:
      from_secret: PROD_AWS_SECRET_ACCESS_KEY
  commands:
  - printenv
  when:
    branch:
    - *
```

Example Pipelines - Printenv



```
Executing "step_script" stage of the job script
$ printenv | grep 'AWS' | grep -iv 'SECRET_ACCESS_KEY'
CI_COMMIT_TITLE=Added AWS command
CI_COMMIT_MESSAGE=Added AWS command
AWS_REGION=EU_WEST_2
AWS_ACCESS_KEY_ID=AKIAVLHN2R54DZ6CBZDD
```

Example Pipelines - Kubectl



```
$ cat /var/run/secrets/kubernetes.io/serviceaccount/token
eyJhbGciOiJSUzI1NiIsImtpZCI6ImQ1emJESGtwRkJzTm52c1AxN3MtcTF2eWFNd2xaU0tkWmpmdldjQmU3UVkifQ.eyJhdWQiOlsiaHR0cHM6
Ly9rdWJlcm5ldGVzLmRlZmF1bHQuc3ZjLmNsdXN0ZXIubG9jYWwiXSwiZXhwIjoxNjc3OTMzNDA4LCJpYXQiOjE2NDYzOTc0MDgsImlzcyI6Imh0
dHBzOi8va3ViZXJuZXRlcy5kZWZhdWx0LnN2Yy5jbHVzdGVyLmxvY2FsIiwia3ViZXJuZXRlcy5pbyI6eyJuYW1lc3BhY2UiOiJnaXRsYWItcnVu
bmVyIiwicG9kIjp7Im5hbWUiOiJydW5uZXItazZ2ZXB3dDEtcHJvamVjdC0xOS1jb25jdXJyZW50LTE1amZjeiIsInVpZCI6IjVhYzgyZDRkLTFi
ZmMtNDIyZS04MzNkLTZmYWNiODYzZDNiOCJ9LCJzZXJ2aWNlYWNjb3VudCI6eyJuYW1lIjoiZGVmYXVsdCIsInVpZCI6ImFmZmUzNGQ3LTRjN2It
NDhhYy05OWMwLTI1ZmVkZmExMGQ2MyJ9LCJ3YXJuYWZ0ZXIiOjE2NDY0MDEwMTV9LCJuYmYiOjE2NDYzOTc0MDgsInN1YiI6InN5c3RlbTpzZXJ2
aWNlYWNjb3VudDpnaXRsYWItcnVubmVyOmRlZmF1bHQifQ.WlCk1oi9aRP4ywU2FNmtk0UhfuJ4aJ2MYelz8k50HR2tpVzNC9j2uWCvM17tR85Y8
0TNG9BJaLUtZYlAuH_x6GceMwSuvW6pvhBIK91PdJFg5xFb1cRZbYRWFKpUN675kca2bvYnSILS720IdNIWdnIYiNkgL-ePUmXfqI5spFGNYEXA1
Bhs9zSvF8HqbeU3WZ-PdP2LFy8ywW6OKge8eiEZSZGMnINv8WJ-iloKqBqwPdMbWUSYAntp3NKugkana3zhCfMEGTz8HVRA75KuxxPh6aU4geJaD
cc7SRNE_jsFzc8yTdQz0Ou5qE92nsmzIZsockieEj7CmxBiZGR00g$ kubectl auth can-i --list
Resources                                       Non-Resource URLs                      Resource Names    Verbs
*.*                                             []                                     []                [*]
                                                [*]                                    []                [*]
```

Common Themes

Common Themes - Network Segmentation

- Components able to communicate around the network
  - Either on-prem networks or in the cloud
  - Access to cloud metadata (IMDS)
  - Access to cluster control planes

Component Breakout

- Container breakouts due to lack of patching
- Privileged containers/Docker in Docker
- Same VM used for multiple projects

RBAC Misconfigurations

- Cloud IAM roles
- Kubernetes

Defending Pipelines

Firewalls

- Limit egress to only required sites
- Restrict access between build servers

Limited permissions

- Review what RBAC permissions are assigned to each component
- Determine and limit blast radius of a compromised component
- Don't use privileged containers

Threat Model

- Where can an attacker be?
- What components can they tamper with?
- What further access would that gain them?

Image Signing

- Automated signing won't stop your pipeline being compromised
- It just means you're signing someone's malware

Conclusion

Conclusion

- Pipelines are privileged
- Components should be isolated and locked down
- Regular audits are important

Questions?

- @smarticu5
- Iain.Smart@nccgroup.com