Lars Larsson, PhD

Securing Kubernetes Container Platforms in 2021

NSA/CISA Kubernetes Hardening Guidance & Beyond

$ kubectl whoami

- Cloud computing since 2008
- PhD in Computer Science
- Many years of experience in software engineering and DevOps
- Senior Cloud Architect at Elastisys, makers of Compliant Kubernetes
- Follow on LinkedIn

Outline

- Threats
- NSA/CISA Kubernetes Hardening Guidance: summarized and explained
- Beyond the NSA/CISA recommendations

Uncomfortable question

What permissions have you given me in your Kubernetes cluster or cloud infrastructure if I manage to hack into your application?

Threats

- Supply chain risks
  - Hardware and software supplied by third parties
- Malicious threat actors
  - Outside threats, e.g., hackers or automated attacks
- Insider threats
  - Internal users with legitimate permissions to access systems
  - Can be intentionally malicious or coerced by outsiders
  - 2/3 of all insider incidents are due to negligence

NSA/CISA Kubernetes Hardening Guidance

What is it?

NSA/CISA Kubernetes Hardening Guidance

59-page tech report published publicly in August 2021

- Focused on Kubernetes itself
    - Detailed discussion of security-related configuration
    - Example code

- Mentions other security software in passing

- Focuses mainly on hardening, not security *as a process*

1. Scan containers and Pods for vulnerabilities or misconfigurations

- Container images are immutable
  - Insecure code stays in time capsule
- Component stability: infrequent updates → worse security
- Scan images for known vulnerabilities
  - Report: using an Admission Controller
  - My take: daily scan all images that are in active use

2. Run containers and Pods with the least privileges possible

- Containers run as “root” by default
- Container file systems: read-only until need arises
- Use most restrictive Pod Security Policies (<= v1.21) or Pod Security Standard (>= 1.22)
- Avoid handing Default Service Account to Pods
- My take: Also encode *your* policies for automatic enforcement via Open Policy Agent

3. Use network separation to control the amount of damage a compromise can cause.

- Default settings allow all Pods to network with all other Pods
  - Security is only as strong as the weakest point
- Network Policies are Kubernetes-aware firewall rules
  - Specify rules for IP blocks or Kubernetes objects
  - Allow only “backend” to connect to “database” -- nothing else

4. Use firewalls to limit unneeded network connectivity and encryption to protect confidentiality.

- Restrict access to Kubernetes core components
    - API server
    - etcd
    - Controller Managers

- Network traffic in Kubernetes clusters is unencrypted by default

- My take: Use a networking provider with transparent encryption, e.g., Calico with WireGuard support

5. Use strong authentication and authorization to limit user and administrator access as well as to limit the attack surface.

- Authentication on an opt-in basis
  - OpenID Connect and other options available
- Authorization on an opt-in basis
  - Role-Based Access Control (RBAC)
- My take:
  - Disable the perpetual admin token created during installation
  - OpenID Connect for user and group membership
  - Disable anonymous access
  - Enable RBAC
  - Restrict permissions as much as possible with RBAC

6. Use log auditing so that administrators can monitor activity and be alerted to potential malicious activity.

- All API calls can be logged for auditing purposes
  - Creates a huge amount of logs!
- Use an automated system for processing audit logs
- My take:
  - Falco can act as a simple Security Incident and Event Management (SIEM) system together with centralized logging, e.g., Elasticsearch

7. Periodically review all Kubernetes settings and use vulnerability scans to help ensure risks are appropriately accounted for and security patches are applied.

- Kubernetes has a new release ~3 times per year
  - N-3 security updates support (current and the two previous ones)
- Security features are typically opt-in, rather than *opt-out*
  - You need to opt-in as soon as possible
- Automated testing can help find insecure (default) settings

But: Are automated vulnerability tools sufficient?

- Kubescape
  - Relies on what it can determine via Kubernetes API calls
- kube-bench
  - Connects deeply into control plane host
    - Can inspect more than Kubescape
- Low-hanging fruit of vulnerability scanning
  - My take: you must do this to not scream “insecure cluster over here!”
- Limited in what they can investigate
  - Encryption at rest storage, firewall rules, security policies encoded in other systems than Kubernetes, underlying operating system, third-party software...

Beyond the NSA/CISA recommendations

1. Prevent misconfiguration, don't just check for it.

- 2/3 of all insider incidents are due to negligence

- RBAC is great, but limited in what it can express:
  - "Lars" allowed to "modify configuration" in the "production" environment
  - ...but is he allowed to make *any* configuration change he pleases?

- Open Policy Agent to the rescue (again!)
  - Library and other third-party rules as inspiration

2. Beware: any permission given to an application is also given to bad actors.

- Hacked applications have all the permissions that the application usually has
  - Third-party SaaS integrations
  - VPN-connected back-office locations
  - Databases

- Always restrict your app components as much as possible
  - Why would a REST API component ever get to do more than take in requests, process them, and send back responses?

3. Keep cloud resources, specifically, in mind, too.

Various Controllers and Operators in the community offer cloud integrations.

- How seriously do they take cloud security?
- Reject ones without configurable/restrictive permissions

4. Does your app unintentionally have permissions in your cloud?

- Beware of “instance profiles” that your cluster VMs may have ability to modify
  - DNS records,
  - autoscaling groups,
  - load balancers…

- ...because all applications can also get those permissions!
  - Just call the cloud’s metadata service and get a token with permissions
  - Applications are also “the VM” to the cloud

5. Regularly scan all your deployed container images, not just when they are new.

- Re-iteration of a point from before!
- To get up to date security scans, you just need to:
  - loop through all your Pods that are deployed,
  - determine which container images are in active use, and
  - scan those images.
- Do this daily!
- More secure than “scan on push” or “scan on deployment”

6. Regularly have your own staff security test your entire system.

- Building is hard, breaking is easy (and fun!)
- Your engineers have access to source code, hacker's don't
- Let your engineers try to break your application
  - Better if *they* find errors, than if hackers do!
- Foster a security-first mindset

7. Have a Disaster Recovery (DR) plan, and actually practice it.

- Disaster Recovery != “backups”
- Disaster could be “entire cloud region outage”...
  - ...or “we need to go back in time to five hours ago, before this attack started”
- How quickly can you destroy your entire infrastructure and get it back again?

8. Use an Intrusion Detection System (IDS) and a Security Information and Event Management (SIEM) system.

- Intrusion Detection System (IDS) verifies that applications behave “normally”

- Security Information and Event Management (SIEM) searches through logging systems to find and flag abnormal events
  - Could be false positives, but could also be indications of incidents!

- Falco is an IDS and can also be a simple SIEM

Summary

Kubernetes is neither safe by default, nor by itself.

- Restrict access (network, users, machines) and privileges
- Periodically use tools to assess current security practices
- Prevent misconfiguration, don't just check after the fact
- Cloud resources and permissions: be mindful!
- Security-conscious engineering culture
- Disaster Recovery applies also to security breaches

Uncomfortable question

What permissions have you given me in your Kubernetes cluster or cloud infrastructure if I manage to hack into your application?

Uncomfortable question: appropriately answered

No more than absolutely needed! And you will see that I am there, because you have automated systems that both limit what I can do, and raise an alert when I make the application behave in ways it doesn’t normally do.

Questions for me?

Do you have any questions for me?

Ask now or via lars.larsson@elastisys.com or linkedin.com/in/llarsson/