How to Achieve Continuous Container Security

Build Security Into Your Container Process

Guide & Tool brought to you by NeuVector

NeuVector

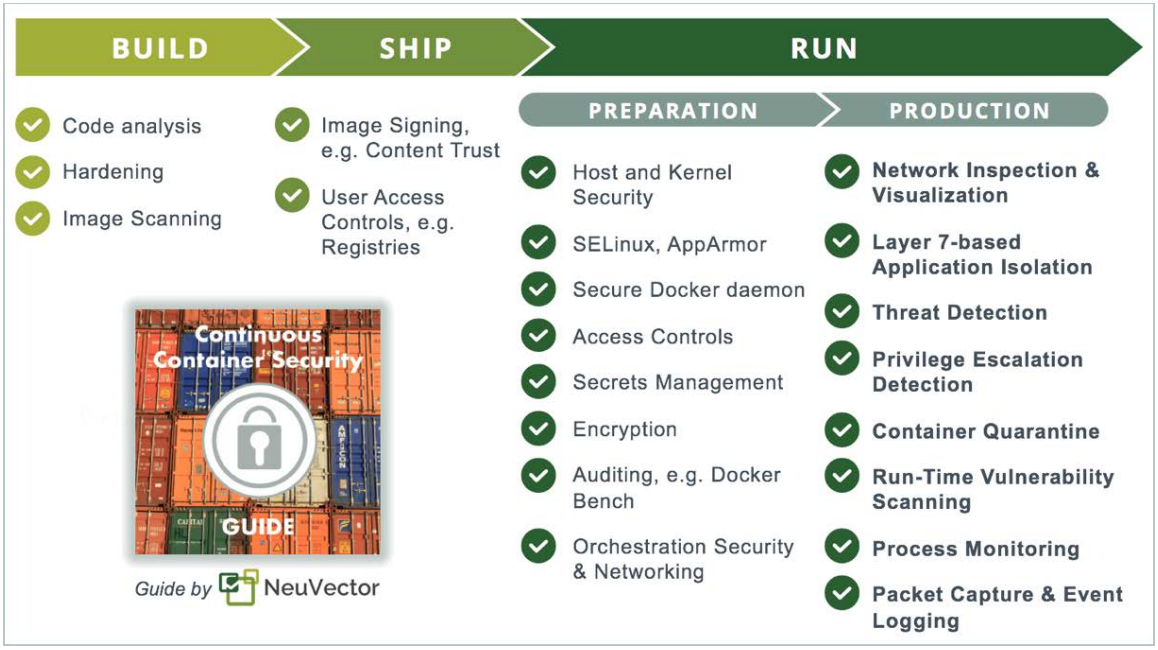
TABLE OF **CONTENTS**

BUILD, SHIP, AND RUN CONTAINERS **SECURELY**

There's been a lot of progress enabling companies to implement a container-based continuous integration and continuous delivery (CI/CD) pipeline. But when it's time for deployment into production, how can enterprises make sure continuous security is also built into the process?

Security should be built into the process from the beginning and cover all the Build, Ship, and Run phases in order to ensure continuous security. Like CI/CD, continuous security can be a fully automated process or a combination of automated and manual processes. In today's rapidly evolving container technology landscape it's more likely to require some manual steps in addition to automation.

Here's a convenient graphic that shows typical security requirements at each phase.



You can download the infographic here in the Graphics section.

Why Secure Containers?

All applications can be vulnerable to attacks and it is critical to reduce this risk. Important precautions can be taken during the Build and Ship stages to reduce the risk of a successful attack. During the Run phase, the entire stack from the host system and the Docker daemon to the application containers running on them must be secured and monitored.

Insecure containers are vulnerable to the same threats that non-container applications and systems face. With the increase in 'east-west,' or internal traffic, generated from containers and microservices, there are many more potential opportunities to compromise an application and the infrastructure.

Here are examples of threats to containers and the hosts they run on.

Container Threats

- *Application level DDoS and cross-site scripting attacks on public facing containers*
- *Compromised containers attempting to download additional malware, or scan internal systems for weaknesses or sensitive data*
- *Container breakout, allowing unauthorized access across containers, hosts, or data centers*
- *A container being forced to use up system resources in an attempt to slow or crash other containers*
- *Live patching of applications which introduces malicious processes;*
- *Use of unsecure applications to flood the network and affect other containers*

There have been many recent examples of application and system attacks which have seriously impacted businesses. These can apply to containerized versions (ie, Docker container versions) of these applications as well. Here are examples of attacks and exploits that could target running containers.

Container Attacks – Examples

- *The Dirty Cow exploit on the linux kernel allowing root privilege escalation*

on a host or container;

- *MongoDB and ElasticSearch ransomware attacks against vulnerable application containers*
- *Port scanning frequently seen on public cloud container instances, which can be the first step in the kill chain*
- *OpenSSL heap corruption caused by malformed key header and a crash caused by the presence of a specific extension*
- *Heap corruption and buffer overflow in Ruby and Python libraries allowing execution of malicious code*
- *SQL injection attacks that put hackers in control of a database container in order to steal data*
- *Vulnerabilities like the glibc stack-based buffer overflow, giving hackers control through the use of man-in-the-middle attacks*
- *Any new zero-day attack on a container that represents an on-going threat*

Securing a containerized environment and their application containers requires, like any public cloud or private data-center, a layered security strategy which builds in security at multiple layers of the stack.

It's helpful to look at the security requirements for each phase of the Build-Ship-Run container deployment process. Incorporating the appropriate security controls and automating testing throughout the process will help ensure compliance with internal and external requirements.

BUILD SECURELY

The first step to continuous security is to build secure applications. This requires that developers become knowledgeable - through application security training if necessary – about techniques for reducing the attack risk in application code.

Before moving on to integration and testing of an application, code analysis tools can also be run to report on potential vulnerabilities in the code.

Images should be hardened to reduce the attack surface by removing any

unnecessary libraries or packages.

Finally, vulnerability scanning should be performed on container images as a final step before the Ship phase. Vulnerability scanning in registries should occur frequently, as new vulnerabilities can be discovered which apply to existing images.

Automating Continuous Security: Automatically run code analysis tools and vulnerability scans on new builds. Automation can be applied in development, test, and registries, where appropriate.

SHIP SECURELY

It is critical when getting ready to deploy images that appropriate controls and verifications are put in place. Image signing, for example with Docker's Content Trust features, ensure that an image has not been tampered with, and that the author or publisher has been verified.

It's also critical to establish and enforce access controls for registries, orchestration tools, and other deployment tools/systems. These include the human identities for managing and building applications on the Docker infrastructure.

The Docker Bench for Security can help test for content trust and access control issues.

Automating Continuous Security: Images should be automatically signed and tagged for production use. Access controls for registry and deployment tools should integrate with directories such as Active Directory or LDAP directories. Run Docker Bench for Security tests as a standard, automated process.

Run Securely

Running containers securely in production requires both run-time **preparation** and **production** security steps.

Run-time Preparation

The run-time environment must be secured before running application containers live in production. This includes the host, kernel, Docker daemon, and any network or system services such as load balancers and name services.

Container management and orchestration tools can help to secure the environment by setting appropriate names, labels, and networking policies. They can also ensure that security containers always run on every host.

Make sure to audit security settings before and during run-time with tools such as the Docker Bench for Security and the Kubernetes 1.6 CIS Benchmark. This can help with compliance requirements for containers.

For an excellent, detailed technical guide for securing the OS, platforms, and systems, see the NCC Group report linked in the Additional Resources Appendix.

Automating Continuous Security: Use orchestration tools to ensure security containers always run and appropriate network and container settings are deployed. Make sure security tools are integrated with orchestration tools to ensure automated policy updates as configurations change and to enhance the visualization of the application and system containers. Finally, automate security testing with industry standard CIS benchmarks to ensure compliance with security requirements.

Production Security

Securing and monitoring containers at run-time, in production environments, is the most critical requirement for continuous security. Experienced security

professionals know that somehow, some way, a hacker will find a vulnerability and try to exploit it. There is usually a series of events, often called a 'kill chain' where the hacker moves from host or container to container to eventually find an exploit point. So there are often several opportunities to detect suspicious behavior.

Detecting suspicious activity by inspecting network behavior is the most convenient and effective monitoring point for run-time security. This is where detection is best as attacks spread and real-time response is required. Monitoring other run-time activities such as container and host processes can also provide visibility into suspicious activity.

Encryption can be an important layer of a run-time security strategy. Encryption can protect against stealing of secrets or sensitive data during transmission. But it can't protect against application attacks or other break outs from a container or host. Security architects should evaluate the trade-offs between performance, manageability, and security to determine which, if any connections should be encrypted.

Even if network connections are encrypted between hosts or containers, all communication should be monitored at the network layer to determine if unauthorized connections are being attempted

Automating Continuous Security: Ensure security tools are automatically deployed on each host and new container behavior is automatically learned and locked down, or programmatically added to the security policy. Do real-time monitoring and alerting from local consoles or from logs exported to SIEM systems such as Splunk.

See the Continuous Security Quick Reference Tool on the next page for a summary with detailed security requirements.

Continuous Security Quick Reference Tool

Here is a quick reference tool that summarizes the security controls that should be put in place to ensure continuous container security. Add additional requirements in the 'Other' rows, add additional rows, and verify the Plan for each requirement.

Phase	Requirement	Description	Plan
Build	**Code Analysis**	Analyze code for application specific vulnerabilities	
	Hardening	Remove unneeded libraries and packages; restrict functions	
	Image Scanning	Scan images for vulnerabilities at build; regularly in registries	
	Other		
Ship	**Image Signing, e.g. Content Trust**	Ensure trust with signing and author / publisher verification	
	User Access Controls, e.g. Registries	Restrict and monitor access to trusted registries and deployment tools	
	Other		
Run: Prep-ara-tion	**Host and Kernel Security**	Use SECCOMP, AppArmor, or SELinux or equivalent host security settings	
	Secure Docker daemon	Ensure authorized access to Docker commands and settings	
	Access Controls	Enable restricted access to system and Docker daemon	
	Secrets Management	Encrypt and protect secrets for applications and API access	
	Auditing, e.g. Docker Bench	Perform security audit using Docker CIS benchmark	
	Orchestration Security & Networking	Use orchestration features to restrict access, set network policies, and ensure security containers are running	
	Other		

Run: Pro-duct-ion	**Network Inspection & Visualization**	Inspect all container to container connections and build visualization for application stack behavior	
	Encryption	Use when necessary for connection encryption	
	Layer 7-based Application Isolation	Isolate and protect based on application protocols and container meta-data inspection, not simply L3/4 network settings; and block (if desired) without affecting containers	
	Threat Detection	Monitor applications for DDoS, DNS attacks and other network based application attacks	
	Host & Container Privilege Escalation Detection	Detect privilege escalations on hosts and containers to predict break outs and attacks	
	Container Quarantine	Quarantine suspicious containers, allowing for investigation	
	Run-Time Vulnerability Scanning	Ensure all running containers are scanned, and new ones get immediately scanned	
	Process Monitoring	Inspect container processes to determine suspicious activity	
	Packet Capture & Event Logging	Capture packets and event logs to enable forensics	
	Other		

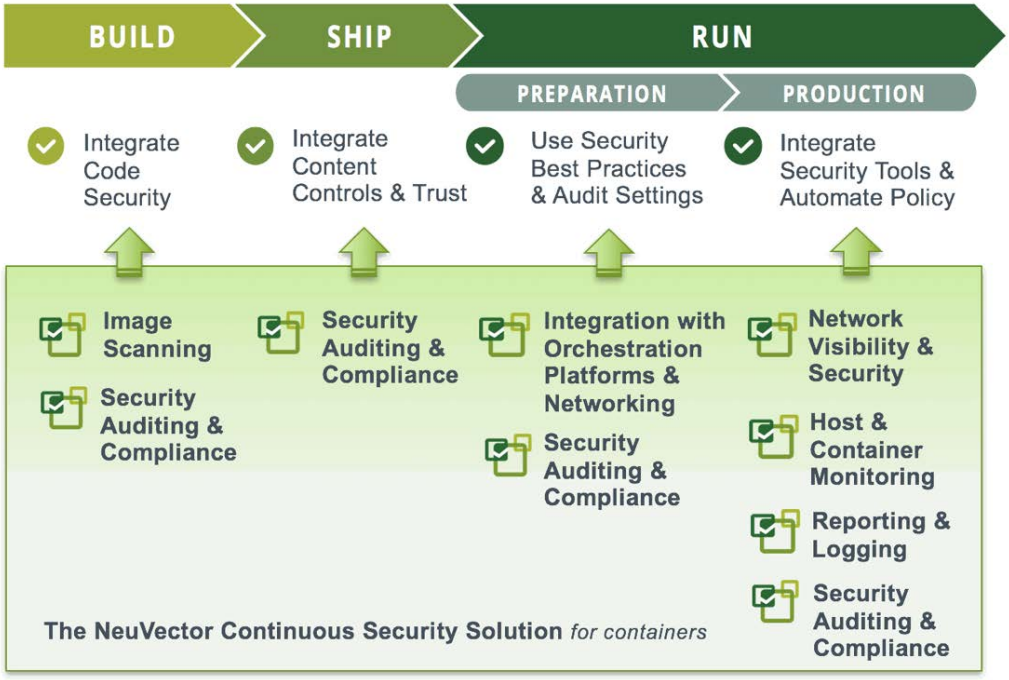
You can download an Excel based version of the tool here, complete with NeuVector security container capabilities mapped to the requirements.

NeuVector Continuous Security

There are four integration points for building in security into the container deployment process.

1. Build - Integrate Code Security
2. Ship - Integrate Content Controls & Trust
3. Run/Preparation - Use Security Best Practices & Audit Settings
4. Run/Production - Integrate Security Tools & Automate Policy

NeuVector provides a leading solution which can be integrated to any workflow designed to ensure continuous container security for CI/CD processes and help with container compliance requirements.



NeuVector helps automate the continuous security process by:

- Automatically auditing security settings for hosts and containers
- Incorporating behavioral learning to create and update security policies
- Integrating with orchestration tools and networking plug-ins
- Providing a CLI & REST API for integration with other tools & systems.

Find out more at http://neuvector.com.

Additional Resources

The following additional resources are recommended to research deeper technical advice for securing containers. Existing resources focus mainly on securing the environment and integration process before run-time. There aren't many resources for securing and preventing container attacks during run-time because this is a new area where companies like NeuVector are pioneering run-time security solutions.

- ✓ ***Understanding and Hardening Linux Containers*****. By the NCC Group, Author: *Aaron Grattafiori***
- ✓ ***Docker Security*****. By Docker Inc.**
- ✓ ***Docker security. Using containers safely in production*****. BY ADRIAN MOUAT**
- ✓ ***Securing Your Deployment Pipeline With Docker.*** **SLIDESHARE BY MAXIMILIAN SCHÖFMANN – CONTAINER SOLUTIONS**
- ✓ ***Container Security Guide*****.** Red Hat Enterprise Linux Atomic Host.
- ✓ ***Introduction to Container Security*****. By Docker Inc.**