Hewlett Packard Enterprise

HPE SW Container Security POV

Are containers secure?

Version: 0.3
Date: 11th June, 2017
Author: Erez Yaary, HPE SW Chief Technologist

Contributions from: Elena Kravchenko, Security Expert

Table of Content

Introduction	2
Constructs of the Solution	3
Shift Left	3
Automation	3
Prevention	3
Secure the Container Lifecycle	4
Build	4
Ship	5
Deploy	6
Run	7
Summary	9

Introduction

Linux, and soon Windows containers, are rapidly adopted by organizations as developers seek ways to
push innovation faster into production environments and in customers hands. Containers allow to preserve the application environment all the way into production, maintaining consistency across the development and operations pipeline.

Containers are not inherently un-secure, but they are being deployed in an un-secure manner by developers, with little or no involvement from security teams and little guidance from security architects.

There are several security considerations when deploying containers that security architects must acknowledge and address:

- Containers use a shared OS kernel. Thus, a compromise of the host OS kernel by a rogue container could lead to a loss of separation and the result is complete access to all running containers on the host as well as potentially other hosts on the network.
- Containers are often not scanned for vulnerability before hitting production systems, so like any software, most successful attacks on containers will have a root cause that can be attributed to missing patches and misconfiguration. Unauthorized images are being deployed and executed in production environment.
- Insecure configurations in testing and production environment increasing the attack surface size.
- Containers are rare built from scratch and are based on existing images and open-source software (OSS) and often includes known vulnerabilities that developers were unaware of, or ignored.
- Because containers use a shared OS kernel, existing host-based security controls won't (without modification) understand the context of the containers running on top of the OS and won't be able to apply differential security policy.
- Most enterprises will lose inter-container traffic visibility and existing external network security controls for fire-walling and intrusion prevention system won't work within the container environment, which can lead to network attacks via insecure containers having exposed and non hardened network
- Developers are the main driving force behind container creation, and with DevOps style workflows gain more responsibilities due to shift left momentum, this might increase the risk of loss of separation of duties between development, operations and security. Lack of accountability as containers pass several hands en route from development and into production

The use of containers is commonly associated with rapid DevOps-style workflows as a way to streamline service delivery from development into production (and back again) with a high degree of consistency throughout the life cycle. The use of containers doesn't require DevOps, nor does the use of DevOps require containers, but the two approaches are highly complementary. In addition, container security best practices can be mapped into the DevOps workflow (CI/CD) to deliver DevSecOps.

Constructs of the Solution

Shift Left

Developers are gaining more and more control over activities previously handled by other teams. With QA functions folded under development and some operational tasks with DevOps teams, now that the container image is being built by development, additional responsibilities will fall with development.

The container image, built by developers, is running the same way whether it is hosted in testing or production environments, hence it must be self contained and appropriate for them all. It must be configured to operate in the production environment, container relevant security and hardening policies, proper tuned base operating system that can connect with production resources etc.

Container security must be pushed left along with that and start as early as when the code is written and compiled. Attempting to address security concerns en route into production will generate a lot of waste as fixing findings will mean stopping the innovation pipeline and shifting back the container to development.

Automation

Containers are artifacts of modern DevOps practices and enable micro-services as the new de-facto application architecture. With containerized micro-services, different development teams can increase pace of innovation ten-fold by setting up parallel development pipelines per each micro-service.

As those micro-services hit production, they are deployed and managed by orchestration software, working based on policies to maintain business continuity by making container placement decisions far better than any human can. This effectively removes humans from operating containerized production environments, focusing on setting up policies and monitoring exception versus active management.

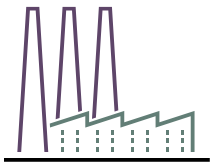
As the development pipeline and operations systems are software enabled and solely operated, container security must fit and be plugged in. This means automation and API enabled, policy operated and ability to assess and take actions by itself.

Prevention

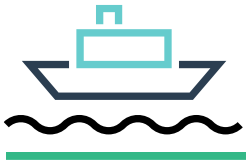
As the modern environment becomes software controlled and automated, pausing for assessing and evaluating before taking actions is not an option. Standard practice is to promote innovation to production whenever its available and with micro-services, that could be several times a week and even per day.

Container security should focus on maintaining security posture by prevention as early on as possible, and it every step along the way with appropriate capabilities that will allow workflows and pipelines to progress while preventing potential issues to materialize per their security profile.

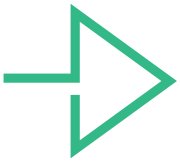
Secure the Container Lifecycle



Build



Ship



Deploy

Run

Build

Secure Images

Oftentimes container images are built from base images that provide operating system components and application middleware, such as node.js or Tomcat and then developers extend those with their own code that form application fabric of micro-services.

In other cases application middleware, such as Kafka or Vertica, are used as is without modification.

In sourcing base images off public repositories, such as Docker Hub, there is little understanding developers of security risk they introduce by leveraging untested and unverified base images.

Images should be scanned for vulnerabilities in order to detect whether they contain Common Vulnerabilities and Exposure (CVE) within them, thus increasing the attack surface of the final container image.

Proper image scanning should include few levels:

- Image scan for third-party known vulnerabilities derived from the base images and/or open source libraries used to build the image
- Static scan of configuration and deployment scripts to identify misconfiguration issues earlier and dynamic infrastructure/hardening scanning for deployed image.

Additional scanning aim to validate the container image behaves correctly, leveraging best practices such as single process per container etc.

A security score is created during scanning time and then used across the lifecycle based on implemented policies.

Sign and Register Trusted Images

When observing a container image, how can one know if it was scanned and tested for security? This might be important to know as it influences further checkpoints downstream, like move to production.

Upon successful image scanning and security score creation, the container image could be re-signed, including security score and tests results, indicating it had been tested and achieved a specific security posture grade.

Observe Application Behavior

As developers compose their workloads and applications from many containerized micro-services, the network becomes the application fabric instead of a single 3-tier bloated architecture. The network dynamically binds all micro-services in concert to drive business transactions.

If before all logic is bounded in compilation time, now micro-services are loosely bounded and form connections with other services as needed and in runtime.

What constitutes a normal versus abnormal behavior, caused by rogue or malicious code hidden in one or more containers? Threat modeling is much harder with an application that is decomposed into many micro-services, that serve more than one application.

Observing a micro-serviced architecture during development and integration time allows to learn of expected normal behavior, aid in building a threat model, later used in production to detect abnormal behavior for quarantine purposes.

Information Sharing Between Dev, Sec and Ops

In today's fast paced container DevOps, pace of innovation has accelerated to the point where Ops and Sec have a real challenge keeping up with the frequent container images that need to be pushed to production environments.

More work than before is automated, leading to machines making decisions based on human-set policies, such as how to scale and based on what conditions, or which images to deploy.

Relying on human interaction will not scale with the technology, hence a need to code more knowledge in the automated systems and processes to allow for automation systems to maintain security posture based on security policies.

Coding security information in container images and container security solutions drives a better security posture as container images that do not meet policies are blocked from progressing through the lifecycle.

Ship

Approve What you Know

As container images get promoted from one image repository to another, whether internally or externally operated, so does the risk of accepting unknown and vulnerable images increases.

Having multiple container image registries that span the container lifecycle and operated by different teams increases risk of security policies misconfiguration.

Following corporate policies for what constitutes acceptable image layer constructs, from the base operating system to application layers will ensure contained security posture and reduce the risk of penetration.

Container security systems need to validate container images as they pass through container image registries and upon non-compliant cases, block and quarantine the relevant image(s).

An additional enforcement should be performed for an every promotion of container to new level (e.g. from dev to testing or from testing to production) to ensure any configuration added on the previous stage for easier debugging/monitoring/benchmarking purposes is not mistakenly promoted together with the container itself.

Risk Score Approval

Security policies for containers and image layers are vast and many, which makes it hard to set a human makneagabile security policy that is consistent and easy maintained.

Codifying security policies in a way that yield Risk Score for every container image in every checkpoint allows to standardize container lifecycle and allow to set a minimum security threshold in every important gate or checkpoint that will control container lifecycle if minimum level is not met.

Risk Score also aids in bringing together Dev, Sec and Ops on common grounds as a single unified Risk Score could be easily understand by different stakeholders and allow consistent behavior across multiple teams and professions.

Deploy

Automated Deployment

Developers are creating micro-services in container format at an ever increasing pace. Not only is it difficult to manage DevOps pipelines, in production environments humans are giving way to machines in the form of schedulers and orchestrators that automate deployment of containerized micro-services.

Orchestrators can make better placement and scaling decisions than humans, and so can consistently implement security policies.

In order to maintain a well managed security posture of an acceptable and consistent level, it is advised to connect container security software to deployment systems in order to adhere to security policies set by humans in a consistent manner.

Based on Infrastructure as Code principle all code written to support the automated deployment tasks should be scanned and validated for security flaws on the same level as an application code.

Secure Infrastructure

Deploying a clean and non-vulnerable container on a host that is less secure and hardened than a teenager's laptop will still create a security risk. The container compute infrastructure also needs the relevant hardening best practices to be implemented, otherwise there is little protection against rogue containers.

Scanning a containerized infrastructure, looking for deviations from hardening best practices in the form of CIS Benchmark or corporate hardening policies, alerting the administrators and providing a security score for the containerized infrastructure.

Audit Users and Machines

A complete audit trail allows to investigate what had led to a security incident so that remediation actions could take place and new policies implemented.

It is required to know who did what and which container scheduler deployed what image to a physical or virtual host, or why was a container image blocked from deploying or accessing a given resource.

Container security system is required to connect with human and machine operated systems in order to create an accurate audit trail of everything happening on the container infrastructure as well as log its own action and activities.

Run

Secrets Management

In a highly secure system, secrets such as passwords and security tokens are an essential ingredient of the security system, often stored and managed in a dedicated secrets solution. This holds the keys to environment resources, required when an access is needed to deploy a container image on a host or access a network based resource.

Storing secrets in the container or environment variables exposes them to anyone with access to the container.

For obvious reasons, container systems require secrets to be used when performing operations, such as container deployment, hence the container security system often needs to access the secrets systems for injecting the right secret in container commands for deploy and run.

Integration with leading secrets systems like HashiCorp's Vault, allows only specific users and containers to have access to specific secrets, or scanning of container images to validate there are embedded secrets that will need to migrate over to the central secrets solution.

Prevent Running

There are cases where container images are deployed regardless of specific policies that should have blocked them from getting on a host. There can even be an attempt to run them. This could pose a severe security breach that not only could jeopardize the host, it could allow an attack on the entire accessible network segment.

A container security solution would actively block unauthorized container images from running, effectively

Isolation from the host

Containers running on a host might gain access to its resources, such as compute, storage and network. Moreover, as containers usually contain micro-services, by nature they should be limited to a specific set of tasks and generally do very little.

Once a rogue container gains access to host resources, especially the network, it could grab additional resources off the network to further penetrate other hosts and systems.

Running containers should be restricted to access and consume only specific and approved host resources, thus limiting its impact on the hosting machine as well as the network.

Container-Specific Network Zones

Standard network security tools often protect the network from unauthorized access originating from one host to the other. Traditionally, when hosts usually denoted application components, that was enough to protect application services from rogue access and tampering.

Now that a single host serves multiple containers, each one interacting with neighboring containers on the same host and/or network based services, relying on network security measures is not enough anymore as whatever happens within the host is invisible to those solutions.

Container security solution will need to get closer to where the action happens, in the host, by means of containerized agent that carefully monitors all host network activities, flowing in and out of containers running on the host as well as observing how containers interact with one another on the network across hosts.

Once a network interaction profile is built, any out of bounds activity is blocked from occurring as active enforcement of network traffic is in place.

The proper network segmentation should be planned in advance based on threat modeling performed at the build stage, enforced and validated to ensure the compromised container can affect as limited segments as possible and can be easily terminated/replaced without significant effect on entire system.

Enforce Application Profile

The application fabric, built from numerous micro-services, each one having one or more instances for resilience and scale purposes, is an ever changing landscape of containers being spawn up or down by orchestration solutions.

Keeping tabs of what's the right application deployment is becoming increasing hard, so does maintaining the right, or expected, application topology versus observing for unexpected behavior.

Unexpected behavior could be caused by software defects that escaped into production environments in the lighter case or malicious code that penetrated through third party open source libraries.

The normal, or expected, application topology and behavior could be captured in the development phase, later used to compare the actual behavior in production environments versus the profiled one.

Additionally, the expected application profile could also be captured in production environments by allowing the application to run for some time, then capture what constitutes normal network behavior and save it as the expected application behavior profile.

Summary

Securing the container lifecycle requires careful attention while delivering specific capabilities to different lifecycle steps as outlined in this article.