CONTAINER SECURITY

A BEST PRACTICES GUIDE

ALERT LOGIC®

Container popularity grew significantly in 2017 and it's currently positioned to continue its growth into 2019 and beyond. According to data from the 451 Research Group, the container virtualization market will quadruple its growth by 2021. The indicators for its growth are already here. There's already been a 40 percent increase in adoption of docker across all hosts according to a study by Datadog. There are millions of applications packaged as container images in repositories for download. And these numbers are growing daily.

As organizations rush to leverage the low overhead, power and security that comes with containerization, it's only logical that container-based attacks will grow in popularity as well. No matter how secure containers appear, we already know nothing is hack proof. We know it's a safe bet that attackers—and researchers trying to thwart these attackers–will continue to look for ways to attack the virtualization process.It is in our best interest to stay on top of best practices. Luckily, outside of the vulnerabilities that could affect software running inside the container, the biggest security issues that impact these virtual environments are absent the possibility of some unknown misconfiguration—memory corruption vulnerabilities.

To understand the implication of these low-level attacks, consider this recent example: A security researcher was able to break out of the container memory isolation (CVE-2017-5123) using a kernel vulnerability in the waitid() system call to modify the container capabilities and ultimately elevate privileges. Prior to that, the Dirty Cow vulnerability (CVE-2016-5195) allowed attackers to write to a read-only mounted file to elevate privileges. Both of these attacks leveraged memory corruption, ultimately giving the attacker the ability to hop from one container to the next. To limit your attack exposure, it's best to engage in best practices to prevent attackers from achieving the lower-level permissions they need to attain privileged access to begin with.

The goal of this workbook is to spare you some of the heavy lifting as you consider your organization's move to containers. It isn't an exhaustive look—but it should support your basic understanding of containers, considerations, trade-offs and differences of container types (namely docker and kubernetes). The workbook section at the end of this woorkbook is designed to help document security best practices and migration considerations.

KEY TAKEAWAYS

- **Containers are expected to grow exponentially over the next couple of years.**
- **There has already been a 40% increase in adoption of Docker across all hosts.**
- **Millions of applications have been already packaged as containers and that number grows daily.**
- **It's only logical to believe that container-based attacks will probably grow as well.**
- **As secure as containers are, nothing is hack proof.**
- **To limit your attack exposure, it's best to engage in best practices so attackers can't achieve the lower-level permissions they need to attain privileged access to begin with.**

WHAT ARE CONTAINERS ?

A container is a lightweight virtualized software image that is bundled with all the libraries and runtime tools it needs to run effectively. They are very easy to package and port to other platforms, which makes these small environments very useful and sought after for devops and general software development.

Containers utilize a lightweight process virtualization, encapsulating applications in their own contained memory space. This allows easy scalability and a faster deployment than the usual applications that share memory resources with other applications. Containers also leverage the host OS kernel, meaning they don't require a separate guest OS kernel to run. This is appealing because booting a whole virtual guest OS for development or general server environments can get costly when it comes to both memory consumption and disk space. Instead you just run the application from within a container image with control over all resources needed. This level of control and low overhead allows users to run multiple instances of the same software at the same time, taking microservice architecture to another level.

CONTAINERS: A BRIEF HISTORY

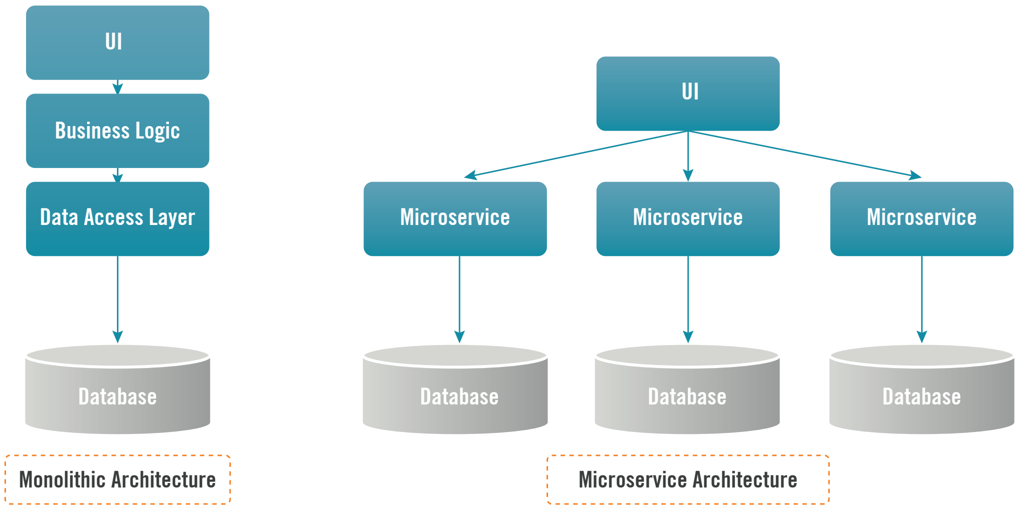
It's worth noting that the idea of containerization isn't exactly new. It was introduced in Unix in 1979 with a software protection known as chroot or by the more familiar term it was later given—the chroot jail/change root. A chroot jail is a modified environment that changes the root directory structure for the current running process and its children. This virtualization of root directories makes it significantly more difficult for intruders, malicious insiders and/or anyone else to cause harm to the operating system or web platform by accident or with intent. Chroot jails were eventually added to BSD in 1982 and their functionality continued to grow until 2003, when microservice providers began offering chrooted proxies, shells, and IRC servers as well as other types of software. In 2005 Sun released Solaris containers—which they called "chroot on steroids." Eventually in 2008 the Linux Container— fashionably named LxCO was released and then evolved into what we know today as the docker container.

KEY TAKEAWAYS

- A container is a lightweight virtualized software image that is bundled with the libraries and runtime tools it needs to run effectively with a low overhead.
- Containers are a virtualized software that leverage the host OS kernel meaning it doesn't require a guest OS, which can have a lot of overhead.
- Containers are very easy to package and port to other platforms, making them very useful for devops and general software development.
- The first type of this virtualization was actually introduced in Unix in 1979 with a software protection that is known as chroot or change root.
- Containers run independently in their own isolated memory space using the main OS kernel, meaning they don't require a separate guest OS kernel for functionality.
- The low overhead of containers allows users to securely run multiple instances of the same software at the same time.

WHY CONTAINERS? MONOLITHIC VS. MICROSERVICE ARCH

Many application platforms use what is known as a monolithic model. Essentially, you have single tiered software application with a couple of components, all dependent on each other to function effectively. Picture an online shopping cart application that has four main components: a web interface, products, a shopping cart, and a payment system. With a monolithic application all of these parts would access the same database and highly depend on each other for functionality—making each component a single point of failure. This can also make for a difficult situation when trying to scale, update software, or add a new component or feature enhancement to the existing platform.



The diagram above illustrates the differences between monolithic and microservice architecture

With microservices you can create a platform with four containerized applications, each compartmentalized with its own database, effectively operating almost fully independent of each other. The idea is if one component happens to go down, the rest of the platform will still function. Also, if you look at this platform from a security perspective the benefits are also in the favor of microservices. In a monolithic architecture if any component in your platform is breached, the attacker may be able to access all of your data or take your whole application offline, whereas within a microservice platform the isolation minimizes the damage. There are of course other benefits such as having control over the resources you wish to allow your containerized application; like how containers have their own file systems and isolated memory.

Currently, docker and kubernetes (lowercase d and k) are dominating the market as the two most popular container software, with Docker Swarm and Kubernetes (uppercase D and K) leading as the most popular container orchestration systems—go figure, right? There are many other containers like Microsoft containers, Apache Mesos, etc., but as far as these big two names and/or which orchestration software is best for you-that depends on your needs.

Docker Swarm is more focused on the management of smaller environments. Kubernetes—originally created by Google with the idea of handling enormous workloads of hundreds to thousands of containers, is much better with larger or mass deployments. Both have their individual strengths and weaknesses though so whichever one you choose is probably going to depend on your end goal. Each orchestration software is strong in a different area. And it's good to not only be aware of these differences before getting your feet wet, but to have a well thought out plan before moving forward. It will save you a lot of future headaches.

Below are some useful details for each platform:

	STRENGTHS
docker	1. Best for under 20 container deployments 2. Simple architecture 3. Very easy setup: only 2 commands and you're ready to go 4. Preferred for simple architecture 5. Auto load balancing features 6. Command line 7. Storage volumes can be shared between any containers in node
kubernetes	1. Best for 100+ deployments 2. Larger development community 3. Preferable with complex architecture 4. Complex, but stronger high availability features 5. Scales up by itself when traffic increases, scales down when traffic reduces. 6. Web based interface 7. Auto rollback on update failure

WHY CONTAINERIZE APPLICATIONS?

01 STEP 1 **PACKAGE**

- Installation and configuration of applications is complicated and time consuming, but doing it once is more efficient.

02 STEP 2 **SHARE**

- Easily share applications between architecture, deployment, security, and operations teams. Quickly experiment with new applications.

03 STEP 3 **DEPLOY**

- Deploy new and existing applications in seconds. All of the heavy lifting was done during the build.

BEST PRACTICES

We've discussed some details of container-based virtualization—and to some extent, container orchestration software. You've done all of your due diligence and decided which virtualization platform you will use. Now that you're a virtual environment guru (I'm kidding), lets discuss best practices related to security and the overall health of your container deployments.

PERMISSIONS – As with any software, we want to run our container process using the lowest privileges possible. Luckily, docker and kubernetes subprocesses should not run with root privileges out of the box; however, you should be mindful of any container-based actions you make using the root account.

IDS/LOG MONITORING, AUTOMATION AND ACTION PLANS – You should always keep an eye on what is going on in your environment and have predetermined action plans on what to do should there be a service interruption. IDS provides us with a holistic view of network traffic between containers and alerts us based on bad or malicious traffic. Logs provide us with forensic data we can use to get a picture of what is going on at the system level. Using both of the protections gives you an edge by informing you what is going on both between containers and at the container level. This is not only important from a security perspective, but also a network and general software perspective. (Note: For continuous logging on docker you may have to configure the default logging driver to write logs to your desired location (/var/log/, /var/log/docker/).)

REGULAR BACKUPS – This goes hand and hand with the previous best practice.. Always create backups at important time intervals, such as before updates or any major development changes. Also, use regular automatic updates for disaster recovery

TRUSTED SOFTWARE ONLY – You should only pull images from well known, trusted repositories. It may be tempting to (after reading a good article on a random blog or receiving a link) pull an image from an unknown repository. **Don't**. If you can't find the images you want in trusted repositories there's probably a reason for this.

LIMIT SYSTEM RESOURCES – We discussed earlier about how containerization utilizes process isolation and is a lightweight alternative to traditional software. Using container orchestration frameworks like Docker Swarm and Kubernetes you can limit memory allocation, which can help reduce DOS attacks and general resource hogging.

A HEALTHY HOST IS A HAPPY HOST – Focusing on your container health is great, but don't forget to keep your main host up-to-date and healthy with periodic restarts.

PORTABILITY IS KEY – Make sure your approach operates across multiple platforms so you can securely manage containers across platforms, in hybrid enviorments, and on-premises.

THINK BIG PICTURE SECURITY – Whether you're using containerization for development or running production servers for ecommerce, outline your goals and security posture before you make any moves. This way, nothing is overlooked. In large production or dev environments it's easy to overlook or simply forget about the smaller parts.

JOIN A COMMUNITY FORUM – docker, AWS, Azure, kubernetes, etc. all have either their own support forums or there are other independent forums built around these. Find a popular community and join the conversation.

WORKBOOK

GETTING STARTED WITH CONTAINERS: A CONSIDERATIONS GUIDE

Containers are ideal for adaptable, virtual environments where control and consistency are essential. Built for speed and ease of deployment, these deploy once, use everywhere solutions offer significant advantages over traditional environments. This workbook is intended to help you work through some of the considerations that might impact your deployment choice: Fill out the boxes below to begin planning your container effort.

WHY DO YOU WANT TO GET STARTED WITH CONTAINERS? WHAT DO YOU WANT TO ACCOMPLISH?

(RANK YOUR RESPONSES 1-5, ONE BEING THE MOST IMPORTANT TO YOUR EFFORT, FIVE BEING THE LEAST)

Easy set-up	
Simple architecture	
Orchestration	
Other	

ORCHESTRATION COMPARISON: DOCKER SWARM V KUBERNETES

	Docker SWARM	Kubernetes	Rate how important this feature is to your organization's container effort
REST API	●	●	
CLI	●	●	
WebUI	○	◒	
Topology Deployment Orchestrator	◒	●	
"Management Node" Failover	●	●	
"Compute Node" Failover	◒	●	
Containers Replica Failover	○	●	
Cluster Flapping Prevention	○	●	
Container Placement Management	●	●	
Dynamic DNS Service	○	●	

KEY: ○ Not Covered ◒ Partially Covered ● Covered

ORCHESTRATION COMPARISON: DOCKER SWARM V KUBERNETES CONT.

	Docker SWARM	Kubernetes	Rate how important this feature is to your organization's container effort
Container Auto Scaling	○	◒	
Load-Balancing Service	○	●	
Multi-Networks Per Container	●	○	
App Networks	●	○	
Distributed Storage Volume	●	●	
Secret Management	○	●	
Multi-Tenancy and Resource Isolation	○	◒	
Tags Selection	◒	●	
Authentication Providers	○	●	
ACL Management	○	◒	

WHICH DEPARTMENTS SHOULD BE INVOLVED? (LIST OUT DEPARTMENTS)

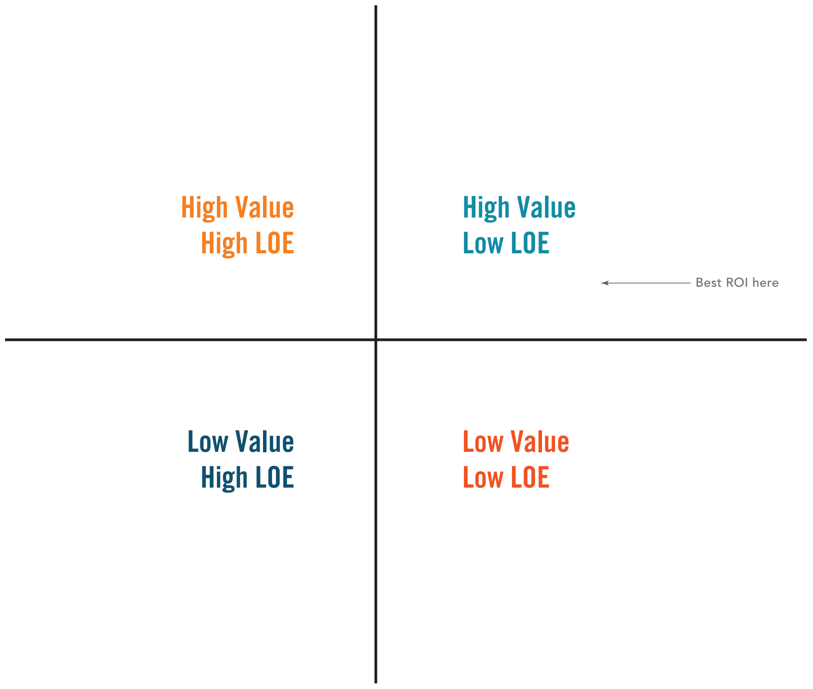
WHO WILL BE IMPACTED? (LIST OUT DEPARTMENTS)

WHO HAS THE MOST TO GAIN FROM MIGRATION? (LIST OUT DEPARTMENTS)

CREATE YOUR PROBLEM STATEMENTS:

EXAMPLE: OUR CURRENT OPERATIONAL PROCESSES ARE NOT ABLE TO KEEP UP WITH THE PACE OF DEVELOPMENT. WE SEEK TO ADDRESS THIS WITH CONTAINERS. TO ACHIEVE THIS, THE FOLLOWING GOALS SHOULD BE KEPT TOP OF MIND (LIST CONSIDERATIONS).

Using this chart prioritize your efforts



HAVE YOU:

	YES	NO
Organized your migration effort by application type?		
Designed for security: apply principle of least priviledge for your containers?		
Monitored traffic going to and from containers?		
Monitored traffic from containers to the base host?		
Documented container roll-back procedures in case of an event?		
Incorporated continuous scanning and monitoring?		
Integrated your container effort into other security procedures?		
Other considerations:		

SOURCES & RESOURCES:

https://451research.com/blog/1657-featured-insight
https://www.datadoghq.com/docker-adoption/
https://www.docker.com/what-container
https://aws.amazon.com/what-are-containers/
https://github.com/scumjr/dirtycow-vdso
https://techcrunch.com/2017/12/18/as-kubernetes-surged-in-popularity-in-2017-it-created-a-vibrant-ecosystem/
https://opensourceforu.com/2017/12/the-current-popularity-and-the-future-of-docker/
https://blog.aquasec.com/a-brief-history-of-containers-from-1970s-chroot-to-docker-2016
https://vexxhost.com/blog/kubernetes-vs-docker-swarm/
https://docs.docker.com/engine/
https://www.contino.io/insights/beyond-docker-other-types-of-containers
https://www.twistlock.com/2017/12/27/escaping-docker-container-using-waitid-cve-2017-5123/
https://blog.aquasec.com/dirty-cow-vulnerability-impact-on-containers
https://github.com/scumjr/dirtycow-vdso