Six Steps to Comprehensive Container Security
Containers accelerate the developer experience and tackle the classic problem, “It runs fine on my system”. When a team starts to build in containers, they can be confident their application will run smoothly in any environment—laptop, on-premises in the data center, or in the cloud.

Containers help developers deliver faster and more consistently, which better enables them to focus on what matters: Helping their customers succeed.

As with any technology, there are tradeoffs. With containers, those tradeoffs come via infrastructure complexity, which can have serious security consequences if not properly accounted for.

To understand these issues, let’s look at how containers accelerate the developer experience.
BUILDING FASTER

When you write a new application, you rarely write everything from scratch. Typically, you would start with a framework like React in JS, Django in Python, or Revel in Go. From there, you would pull in third-party libraries when required. After all, writing a home-grown input sanitization library is a waste of time given the fantastic open alternatives supported by the community.

That code itself may call out to standard libraries commonly made available by the operating system. Most languages take advantage of these libraries to improve performances or access resources on the host system.

This web of interdependencies, along with the reliance on libraries in other parts of the system that make up your application, leads to a dependency nightmare.

If the infrastructure team upgrades the underlying Linux® distribution, that could impact a shared library critical to your application. Maybe that upgrade to the operating system also included a new version of the Ruby YARV or JVM.

Containers address this challenge by allowing you to bundle all of your applications dependencies into the container itself.

This way, multiple versions of these critical tools can live side-by-side without causing conflict. You can think of a container as a large executable file. Simply deploy the container and run it. Voila, your app is up and live in production.

This is a huge win for developers, operations, and the entire business, as well as the key advantage of a containerized approach.

Now you can build by selecting a baseline container that meets your needs—like Ubuntu® or PostgreSQL®—and add your software and its dependencies. This creates a portable container that ensures your application will run in any container supported environment.
SECURITY AT SPEED

Given that containers simply package existing dependencies in a portable format, why would there be any different cybersecurity concerns? At its heart, cybersecurity is very simple.

The goal of cybersecurity is to ensure that whatever you build works as intended... and only as intended.

This means shifting your definition of “done” to include testing that ensures your code isn’t capable of being forced into producing an unexpected output. To make sure that the libraries your code depend on don’t include any unmitigated vulnerabilities, you need to ensure the infrastructure you deploy your containers to is stable and behaves as expected.

With this in mind, the process of securing containers is a continuous one. It should be integrated—like other tests and quality controls—into your development process, automated to remove the number of manual touch points, and extended into the maintenance and operation of the underlying infrastructure.
The main concerns are:
● The security of the container host
● Container network traffic
● The security of your application within the container
● Malicious behavior within your application
● Securing your container management stack
● The foundation layers of your application
● The integrity of the build pipeline

While addressing any one of these areas will help ensure that your applications are protected and work as you intend, it is only by addressing all of these container security areas that you can have full confidence. Your goal should be to accomplish that with as little additional operational burden as possible. Every tool you add to your stack generates additional complexity, which is often the enemy of security.

SECURING THE HOST

If you’re running the container host, you need to ensure that it is locked down, well maintained, and actively monitored. If you’re using a service like AWS Fargate®, Google App Engine™ platform, or Microsoft® Azure™ App Service, your cloud service provider takes care of this step for you.

Securing the host starts with selecting the operating system it runs. Whenever possible, you should use a distributed operating system that is optimized to run containers. Distributions like Container Linux (formerly CoreOS), RancherOS, Ubuntu Core, Google’s Container-Optimized OS, or Red Hat® Atomic are all great choices. These distributions have been tuned to remove any needless services in order to optimize performance and reduce the attack surface (potentially vulnerable services and features of the operating system).

If you’re using stock Linux distributions or Microsoft® Windows®, you’ll want to make sure that you disable or remove unnecessary services and harden the operating system in general.
Then, add a layer of security and monitoring tools to ensure that your host is running as you would expect. Tools like application control are very useful in this situation. A host operating system should run an extremely limited number of processes; you can ensure that this is the case with an execution whitelist enforced by application control.

The host is also a critical point in the container networking picture. The fact that all containers running on the host share resources, installing a network control like an intrusion prevention system (IPS) allows you to monitor and secure both north-south traffic (inbound and outbound from the host) and east-west traffic (between containers on the same host).

An IPS is a highly effective control against modern attacks. By examining each network package for malicious or malformed content, the IPS engine is able to prevent your application from ever seeing the attack. This helps you maintain stable operations.

**MONITORING TRAFFIC BETWEEN CONTAINERS**

Once your container is running in production, it will need to interact with other containers and resources. Traffic that goes to the internet will usually be protected by other security controls.

Web gateways and IPSs that are deployed at the network edge will provide an excellent level of filtering, which will help prevent direct attacks and attempts to setup a watering hole attack (compromising a service in order to attack its visitors).
The remaining challenge is to monitor and secure traffic between containers. This is network traffic that never leaves the organization and doesn’t cross the edge security controls.

Once a cybercriminal gains a foothold on a system, they quickly look to move laterally. Their goal is to gain as large of a foothold as possible, in order to maximize their return. After getting past the controls at the edge, attackers take advantage of the fact that most organization don’t add sufficient internal controls.

The 2018 Verizon DBIR report—a gold standard in the industry—puts it succinctly, “Lateral movement and other post-compromise activities often reel in other systems that are available for infection and obscuration.”

You can counteract this threat by ensuring all network traffic from your containers passes through an IPS. This changes how you deploy the security control. Instead of implementing a small number of large IPS engines, you would implement the system at the host level, which allows for all traffic to be effectively monitored without significantly impacting performance.

SECURING THE APPLICATION IN THE CONTAINER

It’s easy to think that once your application is built into a container and deployed to production that it won’t change. However, once your container is running in production it is constantly changing. It’s processing data for your application, generating log files, caching files, and the list goes on.

Security controls can help ensure that these are ordinary activities and not malicious. The real-time anti-malware controls running on the content in the container are critical to success

1 https://enterprise.verizon.com/resources/reports/dbir/
An IPS plays a role here as well—in a usage pattern called virtual patching. This is where a known vulnerability is mitigated at the network layer. If a vulnerability is exposed remotely, the IPS engine can detect attempts to exploit it and drop packets to protect your application.

This buys you the time needed to address the root cause in the next version of that container instead of pushing out an emergency fix. This powerful workflow will help reduce the operational burden of responding to a security vulnerability.

**MONITORING YOUR APPLICATION**

All of the controls discussed so far work outside of your application. This has some advantages—like ease of deployment—but also comes with some challenges. Some aspects of your application are only visible from within the application code.

For this reason, when you’re writing your code, you often employ a logging strategy or surface various application metrics. These both provide a way to monitor the health of your application by creating an inside view.

The security of your application deserves the same.

When deploying your application into a container, a runtime application self-protection (RASP) security control can help—adding another layer to your defenses. These controls run within your application code and often intercept or hook key calls within your code.

Besides security features like Structured Query Language (SQL) monitoring, dependencies checking and remediation, URL verification, and other controls, RASP can also solve one of the biggest challenges in security: Root cause identification.

By being positioned within the application code, these controls can help connect the dots between a security issue and the line of code that created it. That level of awareness is difficult to compete with and creates a huge boost in your security posture.
SECURING YOUR CONTAINER MANAGEMENT STACK

From a security perspective, the management stack helping to coordinate your containers is often overlooked. Any organization that is serious about its container deployment will inevitably end up with two critical pieces of infrastructure to help manage the process; a privacy container registry and Kubernetes® (to help orchestrate container deployment).

A container registry is a centralized location to search for, store, and distribute containers. Registries simplify sharing containers and help teams build on each other’s work.

You want to make sure that this foundation is strong. To ensure that each container meets your development and security baselines, you need an automated scanner. Scanning each container for known vulnerabilities, malware, and any exposed secrets, before it is made available in the registry, helps to reduce issues downstream.

Additionally, you’ll want to make sure that the registry itself is well protected. It should be run on a hardened system or a very reputable cloud service. Even in the service scenario, you need to understand the shared responsibility model and implement a strong role-based approach to accessing the registry.

On the orchestration side, Kubernetes is a complex piece of software. But once it’s running and deployed within your environment, it offers a significant number of advantages that help ensure your teams get the most out of your environment.

The Kubernetes team has excellent guidance on how to secure your own deployment, but this is another area where a managed service has major advantages over running this critical piece of infrastructure yourself.

From a security viewpoint, Kubernetes also provides the ability to implement a number of operational and security controls. Implementing Pod (cluster level resources) and network security policies allow you to enforce various options to meet your risk tolerance.
The combination of a container registry and Kubernetes allows you to automatically enforce a set of quality and security standards for your containers before, and as they are deployed into your environment. The bonus? When optimized correctly, it won’t slow down your development workflow.

**BUILDING YOUR APPLICATION ON A SECURE FOUNDATION**

When starting an application, you rarely start from scratch. It’s common practice to start with an existing container, such as a low level container like Ubuntu or CentOS, something higher up the stack like node or nginx, or a custom foundation for your organization. Regardless of which container you chose to build on, the point is, you’re building on top of something that may or may not pose a risk to your data.

The same applies when you deploy a container from your preferred registry. You need a workflow in place to ensure that the containers you used as building blocks are reliable and secure against common threats. The best workflow for this is container image scanning. This class of tools will scan the contents of a container, looking for issues before they are used as a building block for your application, as well as a final set of checks before a container is deployed to production.

A good scanner will look for malware, known vulnerabilities, and any secrets or confidential information—like application programming interface (API) keys and tokens—so that you can mitigate risks before you continue to develop your application. This critical step will help ensure that you are building on a secure foundation.

Scanning extends further than just making sure that foundation is strong. When properly implemented, scanning becomes a natural part of your coding process. It’s a fully automated process that can quickly and easily identify any issues made as you develop your application and its containers. Catching any issues like vulnerabilities, malware, or exposed secrets earlier in the development process makes them easier to address and saves you time.
ENSURING THE INTEGRITY OF THE BUILD PIPELINE

In addition to each of the specific aspects of your container environment, the build pipeline is quickly becoming a target for attackers. Attackers have started to shift their attacks towards earlier stages of your continuous integration/continuous delivery (CI/CD) pipeline. If an attacker successfully compromises your build server, code repository, or developer workstations, they can reside in your environment for significantly longer. This is because most secure programs don’t actively monitor these critical resources.

The first step is to ensure that these systems have a strong set of security controls and are kept up to date. Your organization’s code is often one of its most valuable assets, you need to protect it.

The second step is to implement a strong access control strategy throughout the pipeline. This starts at your code repository and branching strategy, extending all the way to the container repository. You need to ensure that you implement the principle of least privilege (only providing as much access as needed to accomplish the required tasks) and audit that access regularly.

These two steps will help to ensure the integrity of your build pipeline. It’s a critical tool in your application development process, treat it as such.
PUTTING IT ALL TOGETHER

Container security is challenging, as it covers so many aspects of the development process and supporting infrastructure. The overall strategy can be simply put as “secure outside in”. Starting with a strong foundation in the infrastructure and eventually ending with a solid, high-quality code base.

As a sequence of steps, “secure outside in” can be implemented as the following:

1. Secure the container host
   1. Select a container-focused operating system to host your containers. This will help reduce your overall attack surface by removing services that aren’t required to host your container workloads.
   2. Add your monitoring tools so you are aware of the health of the hosts.
   3. Use a strong set of security controls, like those available in Trend Micro™ Deep Security™, in order to secure the container host systems. These systems will run all of your container workloads, ensuring their integrity.
      - You can avoid this step by using a managed container service from a reputable cloud service provider. In this scenario, they secure the host on your behalf and you simply run your containers.

2. Secure the networking environment
   1. Traffic moving north-south, to and from the internet, should leverage controls like an IPS and web filtering in order to stop attacks and filter malicious content.
   2. An IPS should also be deployed to monitor inter-container traffic. After attackers gain a foothold in a network, they quickly move laterally to expand their reach. Monitoring internal traffic is a critical aspect of a defense in-depth strategy.

3. Secure your management stack
   1. Ensure that your container registry is properly secured and monitored.
   2. Lock down your Kubernetes installation and take advantage of features like Pod and network policies to enforce your security and development standards.
   3. Leverage a tool like Trend Micro™ Deep Security™ Smart Check to scan and validate the configuration of each container as it is added to the container registry. This will make sure that only containers meeting your development and security standards can be deployed.
4. Build on a secure foundation
1. Make sure to review and watch for communications from the project teams, regarding any dependencies used in your applications. When they patch their software, you’ll need to integrate those changes in order to reduce the risk to your application.
2. Use a container image scanner, like Deep Security Smart Check, to verify that your containers don’t have any malware, known vulnerabilities, exposed secrets, as well as sweep for custom indicators of compromise (IoCs). This allows you to mitigate any risk before developing further or deploying to production.

5. Secure your build pipeline
1. Strong endpoint controls, found in Trend Micro Apex One™, on developer workstations help prevent malware, visits to malicious websites, and a host of other security challenges.
2. A thorough and consistent access control scheme is a must. Ensuring that only authorized users can access code repositories, integrate branches, and trigger builds that get pushed to production is a critical step to safeguarding the integrity of your pipeline.
3. Remember that the servers running these tools also need to be secured. Using a tool like Deep Security will provide a set of strong security controls with minimal overhead, to help meet your security goals.

6. Secure your application
1. Your code should follow all of your best practice in order to increase its quality. Most security vulnerabilities are a result of simple mistakes or poor design choices. Focusing on code quality always pays security dividends.
2. Use runtime self-protection controls to help connect the dots between security vulnerabilities and issues in specific lines of code. This helps close the gap during root cause analysis and leads to better overall security outcomes.
WHAT’S NEXT?

There is no silver bullet when it comes to container security. Despite the simplicity of the container itself, the underlying infrastructure can grow to be quite complex. That complexity needs to be understood, monitored, and secured like the critical asset it is.

Containers offer a huge advantage to developers by solving the never-ending challenge of effectively packaging an application.

There is no doubt your organization can benefit from adopting this technology. But as with any new technology adoption, a strong security plan is a must. Focusing on an “outside in” strategy will help to create a step-by-step plan to automate the security of your containers and build pipeline that creates them.
Deep Security and Deep Security Smart Check can help you implement this strategy without adding to your operational overhead. Backed by rich APIs, these tools integrate into your existing stack without forcing you to change how your team works.

This holistic approach also ensures that you’re addressing the needs of the rest of the teams within your organization. The security is no longer left on the outside looking in or showing up at the last minute with demands to change your workflow. Building trusted security controls and automated processes from the start addresses their concerns and makes it easier to bridge the gap between teams.

The Deep Security platform is advanced, automated security at its best. Learn more about these specific solutions at https://trendmicro.com/containers.