SYNOPSYS®

WHITE PAPER

# Maturing Container Security in Your Organization, A Synopsys Blueprint

By David Benas

# Introduction

While no longer considered wholly greenfield, some organizations are adopting containers for the first time. Unfortunately, though, the baseline security expected in traditional network or cloud environments is often overlooked in containers. Whether containers are new to your organization or you're just now coming to terms with the complexity of securing them, this whitepaper discusses a model by which to pursue the ideal state: demonstrably robust and secure containerization across your organization. You can use it as a blueprint for either starting your container security journey or maturing it from where it currently exists.

# The Container Security Life Cycle

Containerization encompasses many application and network architectures, but the catchall term used to describe them can abstract away the complexities of managing them at scale. Containers must come from somewhere, and that somewhere is the development life cycle. Using the NIST 800-93 model of categorization, we can analyze container security programs in the various phases of the container life cycle and see how robust their readiness for adoption is.

## Initiation

Initiation refers to a container security program's back-end features and is sometimes referred to as its "meta tasks." This very important area underscores the rest of the container life cycle's design, implementation, and operation phases, meaning it underlays the foundation of the organizational approach.

To understand its importance, we can further break down this category into the following subcategories:

- *Education*. How educated are your teams on containers and the implications of container security? Is knowledge centralized around a few individuals and teams? As containerization is still a fairly greenfield activity, organizations commonly fall into the pitfall of keeping it too centralized, with limited security knowledge often a symptom of limited implementation.
- *Training*. Distinct from education, training refers to the actual knowledge transfer by which all teams that interact, even tangentially, with containers experience the relevant security aspects. As determined by their level of exposure to containers, teams should expect to receive security training on a regular cadence or while they're onboarded. Training and education interplay as a very important baseline before and during container deployment to ensure that security is not siloed.
- *Process and policy creation*. Before beginning the journey of design and deployment, your organization will want to develop a baseline layer of policy and processes around container security. You can significantly reduce a lot of friction if the baseline is put into play before deployment.

## Planning and Design

In an ideal "shift left" security model, information security teams step up into the architectural phases of development. The benefits are clear: flaws identified earlier in the life cycle are simultaneously cheaper and easier to fix than rearchitecting an already built and deployed application.

Container security offers no exception to this rule, in fact, early planning and design are just as—if not more—important here. Containerized environments rely heavily on the containerized aspects of their security features, so not truly understanding them can lead to unwanted consequences later in the application's life cycle.

With these multiple moving pieces to consider, planning and design can be extremely complex and difficult to implement properly in organizations opting for containerization. A useful baseline for this phase consists of the following elements:

- *Security architecture*. Important elements of a traditional security architecture also apply to the container environment, from having available security reference architectures to ensuring that commonsense controls are in place, including perimeter security, properly designed Identity & Access Management (IAM), adequate encryption and data protection controls, network security (both in and out of the container), proper incident response logging and reporting, and the defense-in-depth controls required to establish traceability.
- *Container security expertise*. Unfortunately, container security expertise is often noticeably lacking in the design phases of containerized architectures. This seems counterintuitive, but if your organization has a centralized knowledge approach, the lack of presence in this area is likely a direct consequence. Container security experts should be present in some way during the design phase. This doesn't necessarily mean that a security-trained employee must sit in on all design conversations and panels, but resources in their stead should be available.

- *Architecture review*. As with anything else in the software security world, you need to "trust but verify," and the planning and design phase is no exception. In fact, it is as necessary here as it is during the deployment phase. Architecture risk analyses identify the assets an application should protect, the controls that protect those assets, and the threats attempting to violate those controls. By performing an architecture review, it's easier to see if missing controls or as-of-yet unmitigated threats are present. This also becomes a useful exercise in the reverse—identifying threats your organization may not necessarily need to consider in an architecture—thus saving effort by accepting risk in an informed manner.

## Implementation

With the initiation and planning/design phases complete, the next step is to implement what was designed. Top-tier software security programs perform baseline implementation with the following elements:

- *Security tooling*. Containers are a relatively young technology, so unsurprisingly, the available tooling is immature relative to what organizations may expect. While this situation is progressively improving as major players offer more advanced functionality, it is important to not lose sight of the technology-agnostic function of security tooling—that is, that such tooling simplifies and automates tedious processes and performs tasks better than humans can. For containers, this often means image scanning, but this should not be the limit of their contribution to your organization. A tooling baseline should be established across ***all*** technology stacks, no matter where deployments occur. However, tooling is only as effective as the results are ingested—gaps can easily open up if close attention is not paid to where the results go.
- *Coding standards*. Organizations that have robust traditional security programs may find it valuable to expand their coding standards to include infrastructure-as-code and container configuration files into their workflows. Coding standards should include some level of security code review as a prerequisite for deployment, especially container-centric code, which can occasionally fall between the cracks if tooling or existing expertise don't inherently cover it.
- *Testing*. Penetration tests of containerized applications and architectures can't fall into the same boxes as "other" penetration tests. All too often, organizations will perform a penetration test of a web application hosted on an orchestration platform and do a configuration review of the cloud platform (or do a network penetration test on the internal network, if on-prem), then check the compliance box. However, this approach is ineffective because it leaves out too many necessary contextual elements of dynamic container security. A more reasonable approach is to ensure that each step is container-centric at its core—that is, for a web application penetration test, ensure that a simultaneous assessor is testing the insides of the containers, confirming that external influences on the web application do not have unintended consequences on the orchestrated or containerized platform's internal state.
- *Usage and support*. From a security perspective, usage and support can be considered as similar buckets—when containers are used, some elements inherently support their usage from a security perspective. But does the organization have the necessary security support for its goals? This can be considered, in part, an operational question, but it should also form the prerequisites of the champions providing security knowledge about the container technologies in play.

## Operation

The operation of containers encompasses the day-to-day tasks related to ensuring their continued functionality, their orchestrators, and everything else running within them. Therefore, the operational elements of running containers, which include maintaining their environment and disposing of them once retired, are as important as design and implementation to the organization's container security model.

In accordance with NIST 800-93, we can split the operations phase into the following elements:

- *Operation and maintenance*. A key element related to operations is the visibility available to the teams responsible for them. Logging and monitoring need to be in place, but adequate access policy and training should also be available to teams so that they can use them. Simply having the checkboxes in place does little for organizational security, established processes will help your threat response and other security-tangential issues that may arise. Another important element is keeping up with the so-called hydration of containers, that is, ensuring that images, code, and even the platform itself are up to date and patched. Famously, one of the foundational elements for implementing containerization is the ease with which containers can be updated and patched. Of course, this doesn't mean there is no complexity involved, but having a grounded policy and multiple parties working together will help achieve the benefits of containerization faster, with little to no security friction.
- *Disposition*. The handling of containerization throughout its life cycle is often referred to as "dispositioning," but here we categorize dispositioning as the provisioning and deprovisioning of containers. Your organization should have a set process for when containers or environments as a whole are terminated, along with timelines and proper communication channels among the security teams. Accumulation of outdated, unused, and unmaintained containers inherently results in significantly larger attack surfaces that can and have been breached. The important elements for decommissioning containers are that the responsibilities for doing so are well defined and that someone ultimately ensures that they have been removed as planned.

# Benchmarking Container Security

In the prior section, we discussed the underpinnings of a successful container security program in an organization by analyzing what qualifies as a secure, robust life cycle. A huge gap was (intentionally) left in that analysis: the security of containers themselves. Based on years of industry experience and derived best practices, we have put together the overarching categories and capabilities within those categories that encompass what "container security" consists of. We touch on the broader categories into which these capabilities fall under, rather than every element that composes them.

Imagine this model as a pyramid of sorts, where the lowest level holds the foundational elements of host security, then moving up a row, we have platform security elements, and finally, the elements of the container and orchestrator itself at the top. Relevant to all these elements, we underlay security monitoring and threat analytics, which are important in all layers. Of course, in a world where managed cloud providers remain dominant in application architectures and deployments, some of the bottom levels of the pyramid can be safely said to be out of scope for certain applications. However, the shared responsibility of security is just as important, so it's important to understand each step taken to secure your workload.

## Foundational Elements of Host Security

Infrastructure security encompasses the underlying infrastructure as well as the robustness of the physical and virtual resources that support the running of containers, both metaphorically and literally. Containers ultimately run on physical systems, so the security of a system's hosting environment directly impacts the security of the containerized environment that resides on it.

While many deployments are done in a variety of flavors, the hosting OS's security and controls are important for the same reasons that infrastructure security is so important—that is, if the OS itself is compromised, there is nothing that can be done to protect the workloads running in it. Using best practices such as effective IAM, OS security controls, and secure deployments while operating under an assumed compromise model predicate the foundational security elements of containerized architectures.

## Platform Security

An important piece of securing a container platform is understanding security controls as they relate to the platform itself, not just the infrastructure hosting the containers or the applications hosted in them.

Defending the various components of orchestrators and container platforms is a key objective of the penetration testing, configuration, and design review of an organization's containers. The platform to which applications or services are being deployed directly impacts their security and can often be a source of confusion for security teams—who's responsible for a particular layer, and what can or should be done to defend it?

Additionally, the runtime itself, which is ultimately responsible for a container's isolation and execution aspects, should be inspected to ensure that controls are implemented in the expected way and whether those controls are indeed as effective as designed.

Finally, management interfaces need to be robust. As these are often shortcuts into an environment, they will be treated as such by attackers as well and hold an important part of the overall threat model.

## Security of the Orchestrator and Containers

The following concepts should be a fundamental part of your organization's overall containerization security posture:

- *Image security*. While it should be treated as important despite its smaller impact, image security does not on its own define a container's security program. Verifying image integrity is a seemingly straightforward security goal that can often be the cause for significant security issues. Tooling can help ease this friction, but tooling alongside centralized policies and defined responsibilities for each element can be the difference between ineffective and slow vs. robust and agile.
- *Runtime security*. Often a gap, securing runtime for containers should be treated no differently than securing the servers running on a traditional network. Endpoint protection in the container world looks quite different and often involves the overhead of agents running in sidecars, which increases the problem's complexity. However, ensuring that threat detection and behavioral elements are in play allows for a defense-in-depth approach to container security. Even if the architecture has flaws, the code has bugs, or the configuration is incorrect, being able to flag and detect potentially malicious state-changing behavior within containers allows for these issues to be significantly more limited than they would be otherwise.

- *Network security*. Traditional network security is often as robust as the tooling available, with design notes and lessons learned growing steadily over the decades. Many of those lessons do not make their way into the containerized realm, but a secure network should include the network deployed in an orchestrated or containerized environment, alongside how containers or orchestrators may interact with elements outside their scope. Service meshes are common in the container realm, but they are not the magic fix-all for networking and should be analyzed as any other network. Network policies and tooling in this space are still fairly young, but implementing a baseline allows for later implementation of more mature solutions with significantly less tension and without compromising security down the line.
- *Incident response and reporting*. A tale seemingly encountered time and again by security experts is that "traditional" organizational incident response teams do not have the access or the expertise required to adequately detect and respond to threats in containerized environments. Some organizations would agree with this as it has been a problem for lifted-and-shifted applications to the cloud in recent years as well. Containers are not unique in this regard but should be treated as equally important. "Fire drill" exercises with IR and forensics teams should include containerized environments, in fact, these teams may find containers to be their preferred environment to work within as containers are the perfect medium to apply forensics and IR techniques. By design, containers are self-contained, they have an expected input, output, and function, and their boundaries allow for a straightforward analysis of what they're actually doing and the potential delta with what they should be doing.

## Conclusion

Implementing a robust and secure container security program in your organization is no easy task. However, armed with an understanding of the key features a mature container security program should have, along with an understanding of what elements encompass "secure containers," this otherwise monumental task can be broken down into component parts. Ideally, this whitepaper has presented you with an understanding of those parts.

One key takeaway is that no organization starting this journey should be expected to do so alone—it's best to seek guidance from trained experts in container security who have proven experience in helping organizations implement each of these tasks. A trusted team can help you take the next steps, from understanding where you currently stand on the container security continuum to helping you adapt container security for your own organization.

SYNOPSYS®

# The Synopsys difference

Synopsys provides integrated solutions that transform the way you build and deliver software, accelerating innovation while addressing business risk. With Synopsys, your developers can secure code as fast as they write it. Your development and DevSecOps teams can automate testing within development pipelines without compromising velocity. And your security teams can proactively manage risk and focus remediation efforts on what matters most to your organization. Our unmatched expertise helps you plan and execute any security initiative. Only Synopsys offers everything you need to build trust in your software.

For more information, go to www.synopsys.com/software.