Container Security — From Image Analysis to Network Segmentation, Options Are Maturing

Published 28 August 2018 - ID G00366118 - 48 min read

Supporting Key Initiative is Cloud Security

Containers and microservices architectures require DevSecOps, a protection strategy different from traditional VMs with monolithic applications. Technical professionals tasked with securing containers must harden the CI/CD pipeline so that everything that ran through it can be considered secure.

Overview

Key Findings

- Leading container security products have controls that mitigate key risks in the CI/CD pipeline and integrate with well-known container platforms and architectures. The products provide similar controls, making it difficult for the client to choose between vendors.
- Security in the CI/CD pipeline needs to be end to end, with different constituency groups responsible for different steps, making it challenging for clients to select the appropriate staff to administer a diverse set of controls.
- Competition in the container security market is driving product prices down, making the acquisition of a container security product much easier for clients that can accept the performance impact and have container operations maturity.
- Some important controls must be built into the application architecture, such as resilience, message authorization or protection of sensitive data in message-oriented middleware such as Apache Kafka. You cannot buy and install these controls.

Recommendations

Technical professionals involved with application and data security or cloud security:

- Use secrets management and software component analysis as primary container protection strategies. Add Layer 7 network segmentation for operational containers that require defense in depth.
- Secure containers holistically through integrating controls at key steps in the CI/CD pipeline. Focusing solely on runtime controls — as you would for software installed on VMs — will leave you vulnerable at many ends.
- Select vendors that can integrate with the container offerings of leading cloud service providers, such as Amazon Web Services, Microsoft Azure and Google Cloud Platform.

Analysis

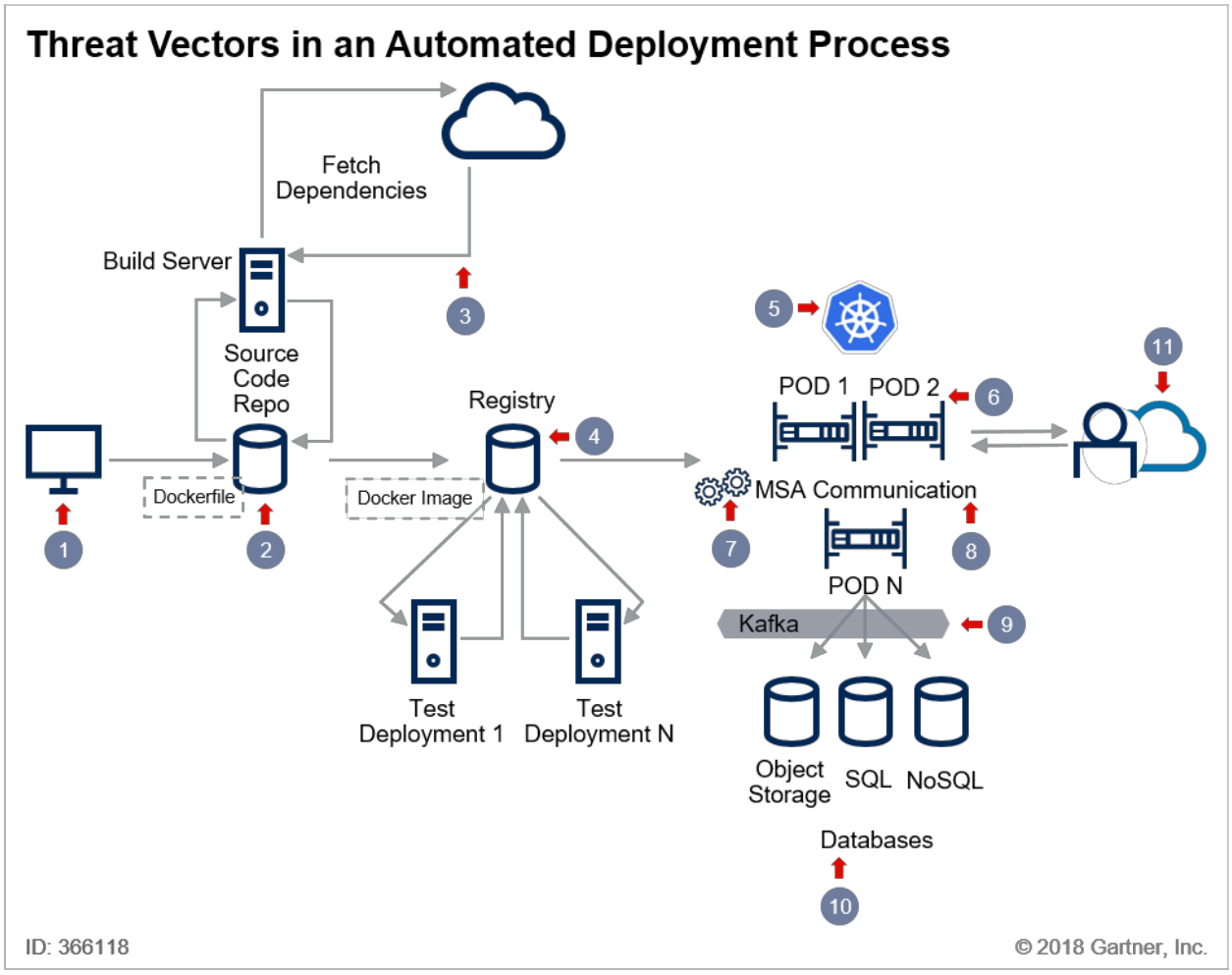
Client interest in container security has changed from focusing on the segregation capabilities of containers to the complex ecosystem that containers are now part of. Figure 1 illustrates a container ecosystem that embeds an automated continuous integration/continuous delivery (CI/CD) deployment pipeline. These deployment chains are composed of many elements running on different platforms from different vendors, raising concerns about end-to-end controls. While CI/CD deployment pipelines are instrumental for the adoption of containers, their security properties are not fully known.

This section starts with a list of threat vectors related to architectures that use containers to embed a CI/CD pipeline and microservices architectures (MSAs). With this in mind, we have derived a hierarchy of container protection needs. This document assumes intermediate knowledge of containers, the container ecosystem and the CI/CD pipeline.

CI/CD Pipeline Threat Vectors

Figure 1 illustrates the application life cycle as the code moves through the CI/CD pipeline into production. The circled numbers are stages or elements of the automated deployment pipeline that can potentially be vulnerable and need the attention of DevSecOps architects.

Figure 1. Threat Vectors in an Automated Deployment Process



Source: Gartner (August 2018)

It is evident that almost every hop in this pipeline has access to the code that is eventually deployed into production. Depending on the architecture, the CI/CD pipeline may consist of a mix of stages hosted on-premises and in the cloud. For example, the source code repository could be internal Git instances, or cloud-based GitHub or Bitbucket, and the image repository could be Docker Hub or Amazon Elastic Container Registry (ECR).

At the start of the automated pipeline is the developers' integrated development environment (IDE, Figure 1, Element 1). A developer's commit will trigger the build of a new image and (Element 2) that will eventually automatically[1] be pulled into production where it becomes part of the operational microservices architecture (MSA, Elements 6, 7 and 8).

The download of additional dependencies from external sources (Figure 1, Element 3) is frequently part of the build process, similar to optional test steps that can be added before production. In this case, the registry would first trigger the test machines to pull the image to be tested. The test machines would then use a callback URL to send the results back to the registry (Figure 1, Element 4), effectively greenlighting the deployment into production.

The above build process may include a number of unencrypted channels where attackers can easily get hold of secrets that are permanently stored in the application code by sniffing the network traffic. This can be done even without direct access to any of the elements of the pipeline.

The threat vectors listed below give an overview of potential security issues along the whole CI/CD pipeline and the infrastructure, such as the host-container relationship or the container orchestration platform used in larger environments. The threat vector numbers match the numbers in Figure 1.

It must be noted that Figure 1 illustrates a textbooklike example of a CI/CD. In practice, there are many more variables to account for. Many clients will still need a sign-off prior to production release. Binary repositories such as JFrog Artifactory or Sonatype Nexus can also function as container registries, but they typically hold other dependencies that code or container images may need in order to build. They can also proxy connections to external package managers and repos.

The developer may or may not build the container. The developer is certainly creating the application code, but someone else, like an infrastructure automation or DevOps engineer, may package it within a container for deployment. They may or may not use an IDE depending on whether they are script-writing experts. No matter what the deviations from our example architecture in your environment are, the threat vectors and control hierarchy we list will still apply to the parts you implement.

Threat Vector 1: Development System

Infrastructure in MSAs is increasingly developer-defined and coded on a developer laptop. It is not a bunch of switches, VLANs and cables any more, but a piece of code. For example, Docker Compose files or K8s manifests in YAML. While Docker Compose and K8s resources determine the application architecture, such as nature of services, exposed TCP ports, labels and environment variables,[2] developers are also in control of the infrastructure in, for example, Azure, AWS or Red Hat OpenShift. Infrastructure is programmatically defined in, for example, AWS CloudFormation templates or HashiCorp Terraform configuration files. The thought that the whole architecture might be in a GitHub repository that developers eager to showcase their skills share with the public is new to security teams. Introducing end-to-end traceability and accountability, such as controlling who can commit, thorough reviews of code merges, pull requests, etc., becomes important.

This turns traditional security concepts upside down. In traditional security concepts, system administrators' endpoints were secured, locked-down PCs located in secured subnets, accessing the data center infrastructure through, for example, jump servers or out-of-band admin networks. Developers frequently enjoyed greater freedom and had local administrator rights on their laptops but were residing in an untrusted network (zone) dedicated to development and testing.

Today, it is exactly this formerly untrusted zone that holds the information to create or destroy your assets. Even if this is news to you, adversaries have known this for a while, and developers and development systems increasingly find themselves the targets of hackers trying to compromise the applications (or infrastructure) that developers ultimately control. For example, attackers attempt to compromise IDEs, weak credentials for repositories, and registries, to taint popular SDKs and code libraries or get hold of hard-coded secrets.

There is no single control that addresses all concerns. It is important to build authorization (who can commit to master) traceability, auditability and resilience into your pipeline. Some container security vendors provide static code analysis during the build phase, even if it is done on the developers' laptop. Furthermore, an easy-to-replicate and easy-to-recover MSA environment will show resilience in case of malicious or unintended deployments with adversarial impact.

Threat Vector 2: Git-Based Repository

The main threat vector here is account hijacking. A compromised account can lead to the introduction of malicious code into the repository, eventually leading to execution of malicious code that exfiltrates data or mines cryptocurrency on a large number of production containers. Accounts can be hijacked through successful phishing attacks.

The same compromises and effects can happen on the image registry (Element 4 in Figure 1). Image-signing techniques such as Docker Content Trust are only somewhat alleviating this attack vector. Image signing establishes a level of authenticity and trust between the party that signed the image in the registry and the end user. It does not automatically guarantee that the image was not compromised at an earlier stage (Elements 1, 2 or 3) outside the control of the organization signing the image.

Threat Vector 3: Retrieval of Dependencies

Dependencies and nested dependencies can pose a security risk, not only because they can possibly be back-doored (the depth of nesting can go quite deep, and there is almost always some vulnerability or outdated component in the mix). In practice, they are often outdated and vulnerable.

While some products will only focus dynamically linked dependencies, it is important that clients choose products that protect all dependencies, including statically linked binaries.

Threat Vector 4: Image Registry

Public image registries, such as Docker Hub, provide repositories of publicly curated container images. Some of the repositories are official, while others are unofficial, provided by third parties. Many container management systems include a private image repository where organizations can manage and curate their own images. The use of public images brings security risks. In 2015, Banyan found that more than 30% of images on the Docker Hub contain high-severity common vulnerabilities and exposure (CVE), and up to 70% contain high- or medium-severity vulnerabilities.[3] In June 2018, Docker had to remove 17 popular images that were back-doored from Docker Hub. The images had been pulled (downloaded) 5 million times after it had first been reported to Docker that the images may be back-doored.[4]

Threat Vector 5: Unsecured Orchestrator Platform

Excessive developer privileges in the orchestrator platform, such as K8s, pose a significant risk. A good example for the importance of applying best-practice security to K8s is the cryptojacking[5] hack of operational containers of the company Tesla that was possible because the K8s master nodes had not been password-protected.

Threat Vector 6: Host-Container Relationship

Properly configured container hosts (that is, the machine where the container engine is installed) and containers cannot provide 100% isolation for your applications in the same way that running them on separate hosts or virtual machines (VMs) would. They have innate security properties that make them vulnerable to kernel privilege escalation attacks. When containers became mainstream, attacks on host-container relationships and container-container relationships were top-of-mind for Gartner clients and vendors of security add-ons for containers. However, they have by now almost disappeared from client concerns and vendor product messaging.

Threat Vector 7: Rapid Rate of Change

Rapid-deployment paradigms always focus on the latest version and frequently do not remove dev tools from the latest image. They also do not remove previous, potentially vulnerable versions from the repositories, leading to a persistence of dev tools and outdated images and dependencies. In addition to code libraries and dev tools not keeping up with the speed of deployment — and allegedly short lifetime of services — the increased speed of deployment also poses new requirements for security controls.

The security control must present when a container is put into production but not delay its deployment. Controls that take five minutes to deploy and activate are impractical in a container that spins up within the (milli)second range and dies after three minutes. This may not be a prevalent concern for container-based MSAs, but it becomes very

evident in mixed MSAs that consist of containers and cloud functions, where the latter have a maximum lifetime of five minutes.

Threat Vector 8: MSA Communication and Network Segregation

The attack surface of the application is no longer a single-server IP address but is spread out over a significant number of containers. Every service instance is a unique network endpoint. Compared to monolithic applications, east-west traffic between containers is vastly increased in MSAs. More interservice east-west interactions (that are generally not mediated with API gateways) are required to make the application work.

For example, one north-south request from an application end user can result in hundreds of east-west transactions. Network access lists as they are part of K8s are a foundational control. Layer 7-based capabilities that also can provide you “laser cut” policies will be chosen based on the sensitivity and risk of the MSA.

Threat Vector 9: Interprocess Communication (IPC)

Message brokers such as RabbitMQ, 0MQ or Kafka are used as back-end “glue” of MSAs. The specific risks depend on the implemented message broker. For example, once access to the queue is permitted, no further authorization checks are performed (is the published data really safe with all subscribers?). Message-level confidence, such as message signing, is often not supported. Compensating controls are available but frequently not used.

Gartner recommends that security teams:

- Ask how trusted publishers and subscribers are identified
- Apply authentication and topic-level authorization
- Apply file-level POSIX permissions to control which users can access which data if the brokers (such as Kafka) store large amounts of operational data
- Evaluate if network communications need to be encrypted so that messages can be sent securely over untrusted networks

Threat Vector 10: Increased Number of Databases

Most services will need to persist data in some kind of database. To match the character of microservices, where ideally all elements are loosely coupled so that they can be modified without adverse impact on other services, developers strive to keep the data private to each service. For example, if the data is kept private, the corresponding microservice can consume it from the database that fits its

requirements best and can change its data model as required. All without negatively impacting any other components — in theory.

There are a few centralized ways to achieve this with traditional relational database management systems (RBDMSs), such as private tables, private schemas or private databases, but in practice data can be everywhere. For example, a Kafka cluster can hold many terabytes (TBs) of unsecured operational data, materialized views can be part of intentional persistence, code repositories or developer laptops. That is, locations where you do not have the traceability and accountability that you are used to from traditional database audit logs and the corresponding toolsets.

Threat Vector 11: Application Layer Attacks

No matter whether you have an n-tier-based application architecture (aka monolithic application) or a MSA, application layer attacks such as SQL injection or Cross Site Scripting (XSS) are still the same. For best practices on mitigating application layer attacks, see the Gartner research "Protecting Web Applications and APIs From Exploits and Abuse."

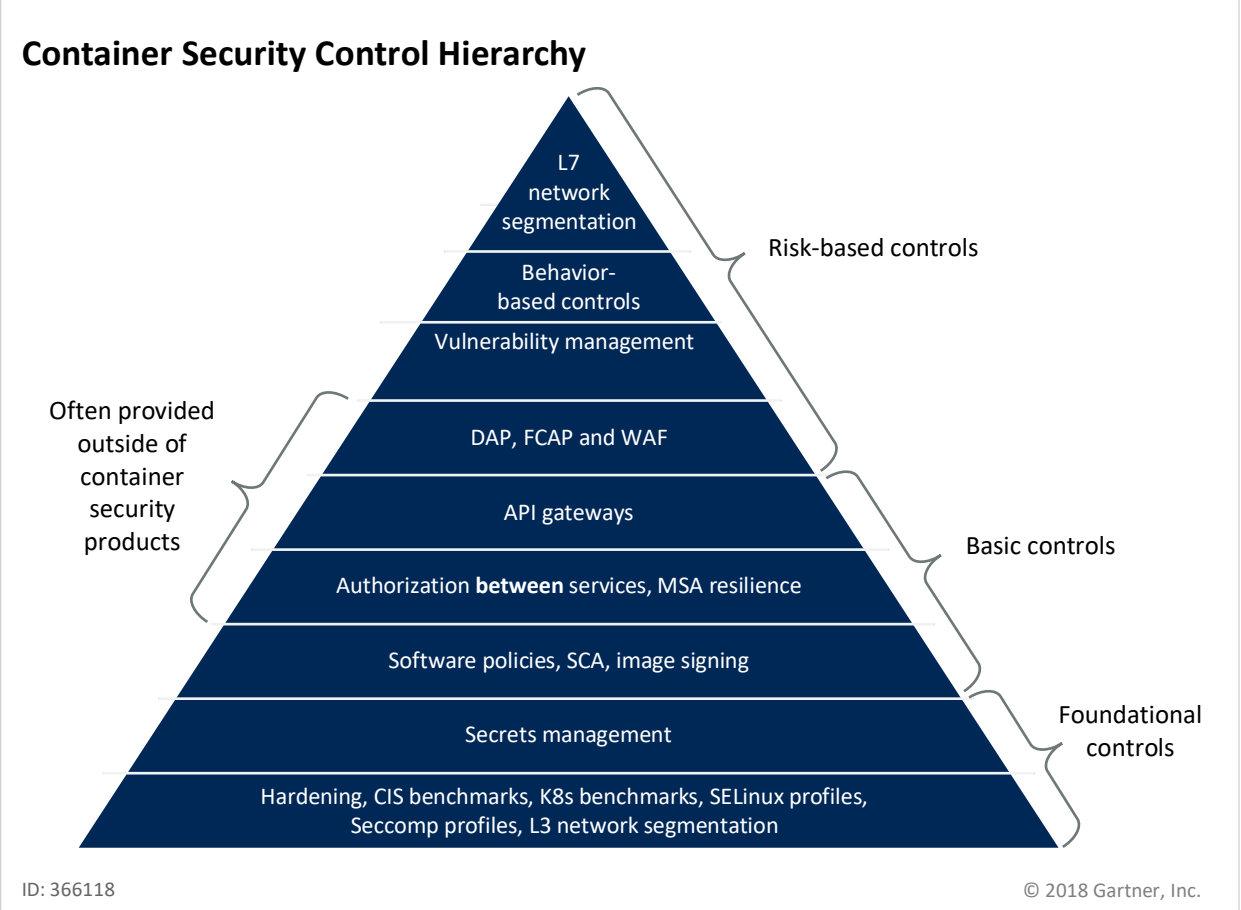
Collectively, these threat vectors create requirements that are significantly different from traditional virtual systems and monolithic applications. Information security leaders and architects must understand that in many cases new approaches are needed for securing containers and the CI/CD pipeline that they embed in.

How to Mitigate the Threat Vectors

Gartner recommends that organizations apply a risk-based security approach for their container security architecture, MSA architecture or CI/CD pipeline. Some MSAs will be less critical, for example, because they do not handle sensitive data or they are not exposed to untrusted parties and require fewer controls. Others with extremely business-critical functions and data are likely to use more controls. Others may be partially using PaaS services, such as AWS Fargate, where more of the security responsibilities are with the cloud service provider (CSP) and cannot be influenced by the client (summarized in Table 2).

With this in mind, we have derived a hierarchy of container protection needs (illustrated in Figure 2) to help enterprises prioritize their security investments and to help evaluate vendors with capabilities in this market.

Figure 2. Container Security Control Hierarchy



Source: Gartner (August 2018)

Figure 2 illustrates our recommended prioritization of security controls for containers that embed a CI/CD pipeline. Controls toward the bottom of the pyramid are more critical (foundational). Those toward the top are less important. Controls are split into three categories:

1. Foundational controls: These are the foundation that you always need. These controls are widely accessible. For example, host-hardening scripts based on the CIS benchmarks are available on GitHub. These can be part of the container host OS or lightweight Linux distributions that are designed to make hosting containers as simple and efficient as possible, such as Red Hat Enterprise Linux Atomic Host or CoreOS. Secrets management is available, for example, in Docker Enterprise Edition (EE), as part of K8s or AWS (the offering is called AWS Secrets Manager) or as a dedicated product, such as HashiCorp Vault. Not all implementations have the same level of security. For example, K8s versions lower than 1.7 will store all your secrets in a clear-text file.
2. Basic controls: These are security controls and architectures that have become best practices for containers and MSAs in operation.

3. Risk-based controls: These are specialized controls that are configured based on the outcome of a structured risk assessment. For example, vulnerability management and configuration management are more important for containers running persistent monolithic components, such as MongoDB. They often consist of heavyweight Linux distribution images with an attack surface that is much bigger than the attack surface of containers running well-described microservices. Another example is mutable infrastructure as opposed to immutable, even in infrastructure automation. Although somewhat of an antipattern, many clients have achieved immutable infrastructure, yet they find it easier to SSH into an instance and fix it as opposed to building a new image and redeploying.

Depending on the specific risk profile of the application and the legal/regulatory requirements of the data and the geography, enterprises may weight their evaluations differently. Depending on the consumption model, some of the capabilities shown may be supplied by a CSP, or another SaaS-based tool, such as Host/K8s hardening or SCA. Thus, they may not be heavily weighted for some enterprises. Finally, containers that are used as VM replacements to run monolithic applications are a different use case, where some of the basic controls are less applicable.

Foundational Controls

Start with solid foundational controls that are at the bottom of the pyramid in Figure 2.

Host Hardening

Since their initial release in the year 2015, the Center of Internet Security (CIS) security benchmarks for Docker have become the de facto best practice for hardening the Linux host configuration and the configuration of Docker hosts. Since that release, Docker has provided and published a Dockerfile on GitHub that builds a privileged container that compares the actual security configuration of a Docker host against the majority of the CIS benchmark tests.

If you are deploying third-party security add-ons, the Docker benchmark may report false positives because it is not aware of the presence of proprietary controls.

The Docker benchmark is certainly very useful, but it is not a panacea. Although some tests are very specific and actionable, others stay descriptive and lack detail. For example, the presence of SELinux or AppArmor is required, but the benchmark gives no actionable guidance as to how these should be configured.

Since 2017, CIS benchmarks are also available for K8s. Aqua Security[7] and NeuVector[8] as well as private maintainers have published scripts on GitHub that can be used to run and automate the CIS security checks for K8s.

Mandatory Access Controls

To maximize the security of containers, it may also be necessary to set up mandatory access control mechanisms. The MAC mechanism is a generalized safety net that holds even if something goes wrong at the kernel level, such as vulnerabilities that would allow an application or attacker to escape its container. Restrictions configured with MACs remain in place even if a user or process gains root privileges outside the container. The MAC mechanism thereby helps to ensure that the host OS or other containers cannot be compromised, even when exposed to kernel privilege vulnerabilities or containerized applications that alter their identity with the setuid() system call. The following kernel security mechanisms can be used to implement MAC on Linux operating systems:

- Security-Enhanced Linux (SELinux) can be used to restrict the actions that installed software can take, including the ability to make system calls. SELinux attaches security labels to processes and resources, and ensures that processes only interact with other processes and resources that have the same label. This approach effectively prevents applications from compromising an entire system, whether or not they are running in containers.
- AppArmor provides similar capabilities as SELinux, but it defines application profiles based on file system paths, rather than processes and resources.

Kernel security modules such as SELinux and AppArmor have an affinity with particular Linux distributions.

MAC mechanisms can be part of the underlying container host OS, such as enterprise Linux distributions, container-optimized Linux distributions, Docker EE, or security add-ons such as Aqua Security or Twistlock. MACs are very difficult to configure, and most clients will need to limit themselves to the preinstalled MAC policies for container hosts that are part of the host OS. Third-party add-ons have shifted from actively configuring and tuning MAC policies for the clients to merely supporting the standard policies that are present anyway, by not interfering with them.

Secure Computing Mode Profiles

All Linux containers on the same container host use the same kernel and interact with it through system calls. While in environments that use hypervisor-based separation (that is, VMs), it is only indirectly possible for a process in a guest VM to contact the hypervisor. In the Linux OS and in Linux containers, every system call is a direct interaction with the kernel — the very same kernel that all segregation features depend on. System calls are a significant attack surface, where nothing must go wrong.

Secure computing mode (seccomp) profiles limit the system calls that a containerized process can make. They shrink the attack surface for the privilege escalation vulnerabilities of the kernel. By default, the Docker EE seccomp disables 53 out of 313 system calls, thus exporting broad and powerful interfaces to containerized processes

and giving application programmers great freedom, but at the cost of a significant attack surface. Consequently, to date, seccomp is only moderately protective for the kernel.

Docker's vision is to expand this to container-native seccomp that will only allow system calls that are required by the trusted process. A starting point for the autogeneration of seccomp is, for example, DockerSlim. For monolithic applications deployed in containers, Twistlock has curated a library of profiles that define the minimum set of syscalls required for common containerized apps such as MongoDB, MySQL, Redis, NGINX and others. The Twistlock container security product matches the application in the container with the relevant syscall profile. Aqua Security creates dynamic seccomp profiles based on machine-learned use of syscalls in a running container.

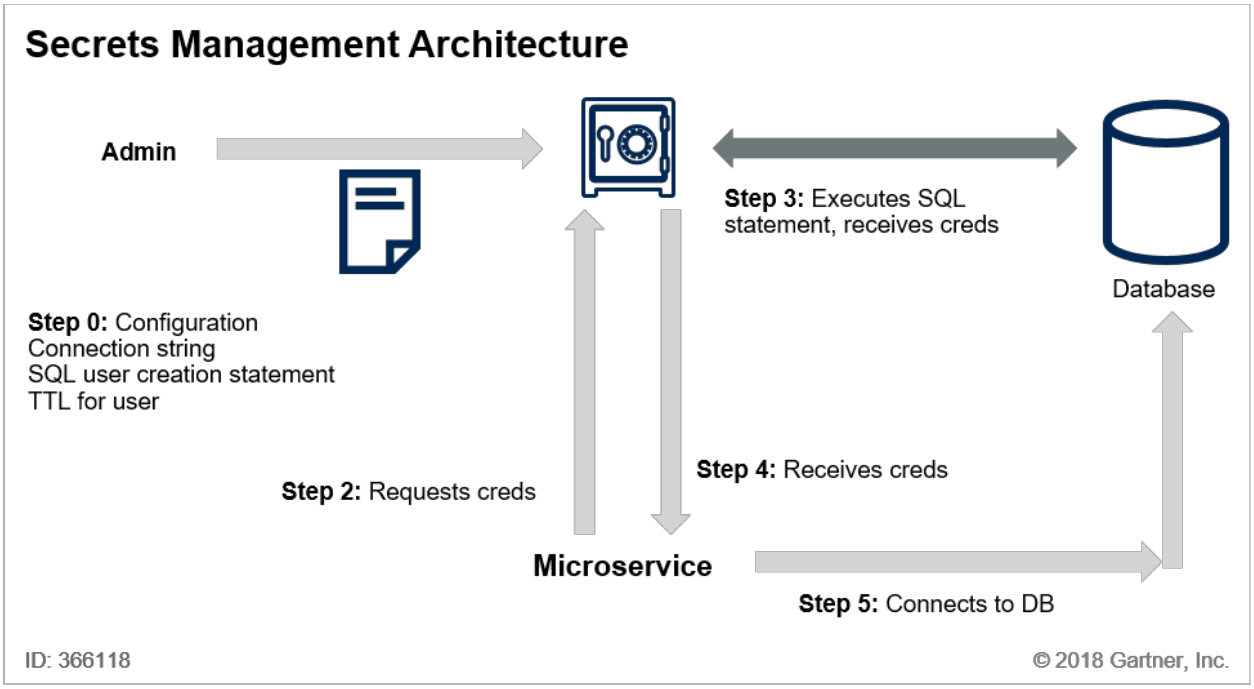
Vendors generally state that the quality and tightness of machine learning (ML)-based controls exceed that of controls that are based on prebuild rules or profiles. Gartner observes that the quality of ML strongly depends on the training data that was used by the vendor and cannot always be predicted. Gartner recommends clients that intend to base their controls on ML to do a proof of concept in their own environment.

Secrets Management

Applications frequently need to work with secrets. Secrets, such as database credentials, API keys, SSL keys or encryption keys, can be hard-coded (persistent) in the source code that is pushed through the CI/CD pipeline and can be compromised in many ways. For example, the secrets can be read by internal attacker with access to the repository, to the registry, to the unencrypted network traffic or by external attackers compromising the corresponding microservice. As a workaround, secrets are frequently stored in configuration files or environment variables — measures that make it only marginally more difficult to compromise the secrets and even contribute to secret sprawl.

Security experts and developers alike understand the risks, and there is broad consensus on the need for secrets management (see Figure 3). Centralized secrets management is very powerful, but it frequently needs application changes.

Figure 3. Secrets Management Architecture



Source: Gartner (August 2018)

Figure 3 illustrates the use of a secrets management product (here symbolized with a vault) for microservices. The first step is to configure the secrets manager with the credentials and policies, such as the lifetime and the connection string that the application will use. In the application, the credentials (aka the secrets) are replaced with a connection string. A vendor-provided library is required[9] to call the secrets management product at runtime and to dynamically retrieve the secrets (frequently, Shamir's secret sharing algorithm is used) before they are needed, for example, to log into a database.

Secrets management is available from AWS (AWS Secrets Manager), HashiCorp (HashiCorp Vault), K8s or several container security products. The breadth and security of the solutions are not equal. For example, AWS Secrets Manager and HashiCorp Vault store secrets in encrypted form while K8s stores secrets in clear text. Although secrets management products can be used to store encryption keys, they are in no way comparable to key management servers such as AWS Key Management Service (KMS), Azure Key Vault or Safenet Luna. Key management servers can be configured to back secrets management products. For more information on secrets management, see "Approaches for Securing Application Development Environments and Artifacts."

Basic Controls

Above this foundational level of controls, the following controls should be basic controls implemented as best-practice protection for containers and MSAs in operation.

Software Composition Analysis

Many successful attacks will be rooted in vulnerable libraries or code components. An effective way to secure containers is to scan their contents for security issues prior to release into production.

This limits the risks of:

- Malicious software being deployed in an operational container. Depending on the intention of the attacker, malicious software could do all sorts of things (for example, data exfiltration).
- Attackers being successful because of known vulnerabilities that exist in the software that is deployed in operational containers.
- Legal risk to the organization from some open-source licensing models.

Containers should be scanned as early as possible in the development process. Clients can use software composition analysis (SCA) at different stages of the CI/CD pipeline. For example, they can start with simple testing integrated directly into the IDE (for example, available from Aqua Security) and add automated predeployment checks to their Git-type repository.

Docker EE and third-party security products have added SCA at different stages of the build phase of the container life cycle:

- Analysis of Dockerfiles in a Git-based repository
- Software composition analysis of Docker images in the image registry

Analysis of Dockerfiles in a Git-Based Repository

Frequently, the Dockerfile will be located in a Git-based repository. Docker image builds are automated. Whenever a change to the Dockerfile in the Git repository is detected, a new Docker image is built and published to the configured Docker registry. Images can be searched for by name in the registry and “pulled” (downloaded) as base images or run, thereby operating much like a source-code management system.

Through Dockerfile scans in the repository, security products can find vulnerable software and library versions before the build process is triggered and assess the configuration of the container for compliance with best-practice security. If any of the

tests fail, a rebuild of the Dockerfile will be initiated. Focusing on the Dockerfile alleviates the need for using a supported registry. Gartner observes that, depending on the severity of the vulnerability found, clients are allowing grace periods during that container images with known vulnerabilities can still be built and deployed. This allows vulnerable containers to exist in production but mitigates the problem (an availability risk) that containers cannot be built and deployed during the time window where a vulnerability is known but no update or fix has been issued.

Software Composition Analysis of Container Images in the Image Registry

The SCA functions provided by almost all container security vendors are generally shallower than established SCA and open-source software governance (OSSG) products from vendors such as Flexera, JFrog, Sonatype, Snyk, Synopsys (Black Duck or Protecode) and WhiteSource. Docker-based SCA can be quickly adopted by developers and DevOps teams without predetermining the choice of source code repository, thus making the hurdles for adoption low. Notably, some of these vendors support image scanning also in Amazon ECR, either with a permanent license (for example, Synopsys [Black Duck]) or as a SaaS (for example, Aqua Security).

SCA is not designed to be a full-scale static application security testing (SAST) tool that does deeper code analysis, such as Fortify, Checkmarx CxSAST, IBM AppScan Source or Micro Focus Fortify Static Code Analyzer. If the goal is to get rid of as many zero-day vulnerabilities in your homegrown code as possible before deployment, then there is no way around a traditional SAST tool. However, you may have to do some integration yourself (or have it done by professional services) so that you can use the SAST tool with containers.

Currently, clients running Docker Trusted Registry or the full Docker EE platform have access to Docker SCA. Clients who run local private Docker registries that use the Amazon EC2 Container Registry (ECR), CoreOS (Quay.io) or the Google container registry cannot at this time benefit from Docker SCA. Third-party Docker security products frequently make their Docker SCA available to container registries that implement the Docker registry API. There are also “light” free offerings, including Aqua Security’s MicroScanner, Anchore and CoreOS Clair.

Software Composition Analysis as Part of the CI/CD Platform

Some container security products provide plug-ins for CI/CD servers, such as Bamboo, Jenkins or TeamCity, to shift Docker SCA even further left in the SDLC. The plug-ins make it possible to display vulnerability information directly in the console of the CI/CD server. By making this vulnerability data available to developers early in the life cycle, clients can seamlessly deal with security vulnerabilities and other security issue information within CI/CD and also fix security issues more cheaply and with less risk. It must be noted that the integration with deployment servers is not complicated

and can usually be done with a few lines of script running before every deployment. If you can develop and handle microservices, you can handle this.

Authorization Between Microservices and MSA Resilience

Authorization between services is a growing concern. Think of scenarios such as who can communicate directly with whom, who administers topics in the message-oriented middleware (MOM), who can consume your data and who approves access. In this field, you move quickly into service meshes and mesh application and service architectures (MASA) with, for example, Istio/Envoy, Linkerd, NGINX or Conduit, which are out of scope of this document.

Resilience is a separate concern. In resilient MSAs, it should be quick and easy to recognize and roll back unauthorized deploys without larger impact on the application. Designing resilience into your MSA is important but equally out of scope of this document.

API Gateways

Each microservices architecture exposes a set of endpoints potentially increasing the attack surface. API gateways can be used in this case to mediate the communication with APIs. API gateways are essentially proxy servers that act as a single entry point into the system and control the traffic directed toward the application. API gateways mediate access requests by invoking multiple (micro)services and aggregating the results. API gateways can also have additional capabilities to bolster security, such as authentication, monitoring, load balancing, caching, request shaping and management, and static response handling.

Security and authentication controls can be implemented at the individual service level, and the services concept in Kubernetes can provide pod access to other pods or external applications,[10] but these approaches increase complexity. An API gateway or microgateway enforces standard security across microservices by mediating API traffic and enforcing defined access control policies, often defined via API management tools. The API gateway need not be a traditional monolith like some large appliance residing at the perimeter of a data center. It can be software-based and even cloud-hosted, mediating north-south traffic at the edge (i.e., an outer gateway). It can also mediate east-west traffic within a network (i.e., an inner gateway). In the case of microservices and MSA, microgateways are the recommendation to mediate interservice communication due to their high performance and lightweight design.

For successful execution, an API gateway or microgateways must be properly developed and often paired with API management. Alternately, fully managed API gateways for container environments, available for example in AWS, can be used. Amazon API Gateway, a fully managed service, can be used to create API for applications running on Amazon EC2, Amazon ECS or AWS Elastic Beanstalk.[11]Amazon

API Gateway can be used to access AWS administration and security tools, such as AWS IAM and Amazon Cognito; verify signed API calls; verify incoming bearer tokens; and remove authorization concerns. Other API gateway offerings include Axway API management system for containers,[12] and Red Hat 3Scale API gateway.[13] For in-depth information about API gateways, see "Selecting the Right API Gateway to Protect Your APIs and Microservices."

An evolution of MSA is the concept of service mesh, which aims to provide such functions as service discovery, load balancing and traffic management, mutual Transport Layer Security (TLS) authentication, and improved visibility in microservices environments. While microgateways can still be relevant technologies in service mesh where they may be instantiated as sidecar proxies, they exist as part of a more complex architecture that is typically composed of a control plane and a data plane. Some examples of enabling technologies for service mesh include Conduit, Envoy, Istio, Linkerd and NGINX. Service mesh is still an evolving space and beyond the scope of this research, but Gartner covers it in other research, including "Assessing Microservices for Agile Application Architecture and Delivery" and "Selecting a Cloud Platform for DevOps Delivery With Microservices architecture."

Risk-Based Controls

Risk-based controls are specialized controls that are configured based on the outcome of a structured risk assessment. For example, you may need to choose to implement network segmentation for your MSA if you assess that the network policies that you use to configure the K8s resources are not granular enough. Or you may require a defense-in-depth approach to network security that includes Layer 7 (application layer) analysis.

Data-Centric Controls: DAP and FCAP

Well-architected MSAs consist of decoupled microservices where data is kept private to each service, effectively siloing the data in, for example, dedicated databases, tables or materialized views. MOM such as RabbitMQ[14] and Kafka[15] are popular underlays to exchange data between otherwise decoupled services. While traditional message queues remove data from the queue once it has been consumed, Kafka is designed to retain data until the configured retention period has been reached. Developers frequently set this to "forever."

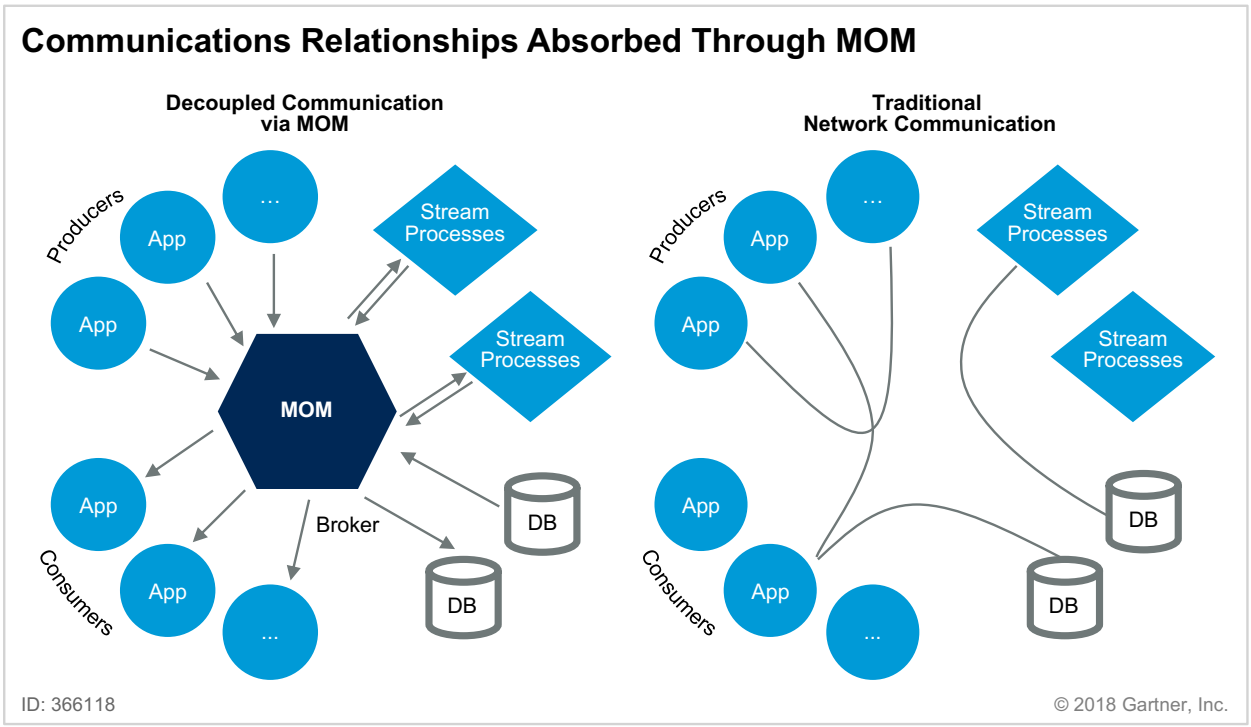
This results in the following issues that security experts involved with data security need to prepare for:

- Data sprawl across many silos that must be secured and monitored.
- Cloaking of communication relationships by absorbing them into MOM (illustrated in Figure 4). Data producers (aka publishers) do not send their data directly to the

destination service, but put it onto a distributed broker (such as MOM) where it waits until a consumer retrieves it. With standard networking tools, it is not possible to prohibit rogue participants or to track and audit communication relations.

- Large amounts of data being stored in persistent MOM, such as Kafka, where it is unclear how to audit those.
- Shift of access control and authorization from systems and services (also databases are services) to MOM.

Figure 4. Communications Relationships Absorbed Through MOM



Source: Gartner (August 2018)

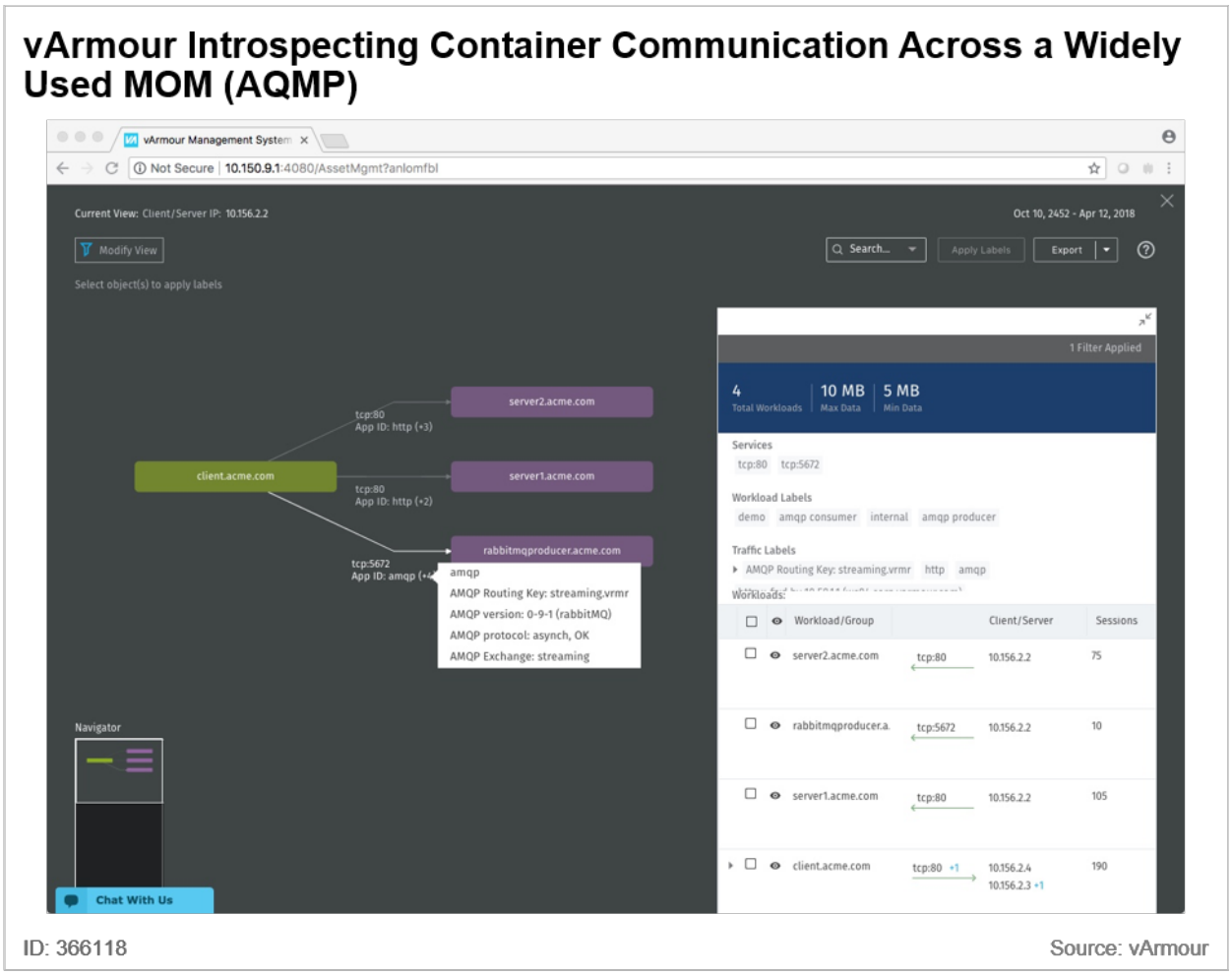
Database audit and protection (DAP) products can be used to implement centralized auditing of decoupled heterogeneous databases. DAP product suites, such as IBM Security Guardium, have visibility on all the transactions against many database types, but cannot introspect MOM that retains large amounts of data.

Leading DAP products use artificial intelligence (AI) to identify unauthorized or unusual behavior. Some products can also block access to certain commands or operations. Disregarding the security aspects, using DAP for siloed data stores in MSAs is a good way to enforce consistency across the data silos. However, DAP product suites are not meant to be a replacement of the native access controls.

For more information on DAP, see [“When to Use Database Audit and Protection to Enhance Database Security and Compliance.”]

vArmour is a microsegmentation product that takes a topic-centric approach to visualizing communications in a publish-subscribe network. It integrates this view into its application services fabric, which also extends to containers and the rest of the data center, effectively keeping track of which services can communicate with other services (see Figure 5). For more information about network-based controls, see the Network Segmentation for Containers section below.

Figure 5. vArmour Introspecting Container Communication Across a Widely Used MOM (AQMP)



Source: vArmour

The controls in your MOM strongly depend on the chosen middleware and are frequently not part of the open-source distribution. Also, the controls that are part of the distribution are frequently immature, difficult to manage and rarely used in

operation. Gartner recommends that clients keep track of which services can communicate with each other - either direct or through MOM - and where your data is.

Vulnerability Assessment

Vulnerability assessment of running containers is important if your containers:

- Have longer lifetime, such as databases where preserving state is crucial and that cannot be replaced at will. During the lifetime of this container, new vulnerabilities for any of its component may become known, and you must have ways to identify vulnerable containers and components, even at runtime.
- Can experience configuration drift. Ideally, containers should be immutable, but many containers do not have textbook properties and become more different as time goes on, due to manual, ad hoc changes. Containers could be compromised by, for example, arbitrary code injection into legitimate containers.
- Are used as VM replacements to provision traditional monolithic applications. According to recent studies, the majority of containers published on Docker Hub fall into this category. The same requirements for vulnerability management and system patching/updates apply to this type of container as to traditional systems.

  Leading container security products can identify vulnerabilities of containers that have been put into operation based, for example, on the bill of materials that has been identified during the SCA process. However, they do not scan containers in runtime, and allow image-specific policies that prevent changes (adding executables, adding files, etc.) to a container that drifts from its originating image.

  Containers that are used as VM replacement need to be managed, scanned and updated like VMs. You will generally not find the required tools in container security products.

Behavior-Based Controls

If an attacker has bypassed all the preventative controls and compromised a workload, how would you know? Leading container security products provide behavioral monitoring, baselining and anomaly detection for defense-in-depth against targeted attacks, leveraging rule- and ML-based approaches to detect malicious activity evidenced by patterns of anomalous behavior in a variety of categories.

For example, Twistlock Runtime Defense uses four distinct sensors — file system, network, processes and system calls — to create a unique model for every image across the environment through a combination of static analysis and behavioral ML. Twistlock will alert or block any anomalous behavior outside of its active runtime models. Twistlock Incident Explorer analyzes audit events and assembles them into chains so that you can more quickly identify and address unfolding attacks. Incident

Explorer identifies incidents, such as backdoor admin accounts, backdoor SSH accounts, brute force, cryptominers, data exfiltration, hijacked processes, lateral movement, port scanning, service violations and weak settings. Similar capabilities are available from Aqua Security, NeuVector, StackRox and Sysdig.

It is, however, very difficult for clients to judge what they will get and how good an ML base's capability really is. Gartner advises clients that need these capabilities to do a proof of concept in an operational environment. To generate more transparency and insight as to what clients will get with Falco, Sysdig has taken a different approach and open-sourced its rule engine and the core behavioral rules that can impact all systems. This allows the customers and community to introspect, validate and improve the core policies for everyone.

Network Segmentation for Containers

Gartner observes that the K8s network model as the network model currently prevailing in operational container deployments. This section focuses on the K8s network model (illustrated in Figure 6).

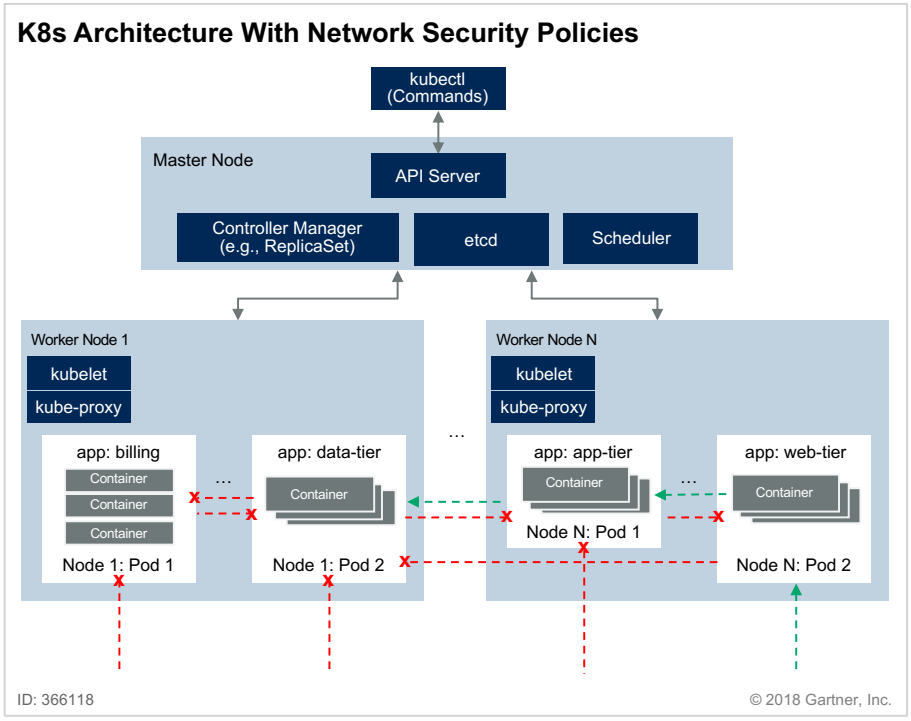
MSAs and containers are an area where the requirement for network segmentation is frequently questioned. Developers frequently assume that they can achieve "good enough" container segmentation by using K8s network security policies[16] together with containers that embed one microservice that does one thing only. In K8s, developers aggregate containers that are closely related into so-called "pods." Without K8s network security policies, all containers can communicate with each other without network address translation (NAT) using the etcd to discover the counterparts that they need to talk to.

K8s network security policies enforce access controls on the ingress levels of every pod. Interpod traffic between the containers in the pod and egress traffic from containers leaving the pod cannot be controlled. Traffic sources that can be matched in the K8s network security policy are applications or name spaces — it must be noted that it is fully decoupled from network addressing. Commercial network security add-ons for containers use label-based security through, for example, importing the required labels from K8s.

Figure 6 shows examples of K8s network security policies:

- DENY all traffic to application "billing"
- Limit traffic to application "data tier"
- ALLOW traffic from external clients' application "web tier"
- Implement traditional tier-based security zoning

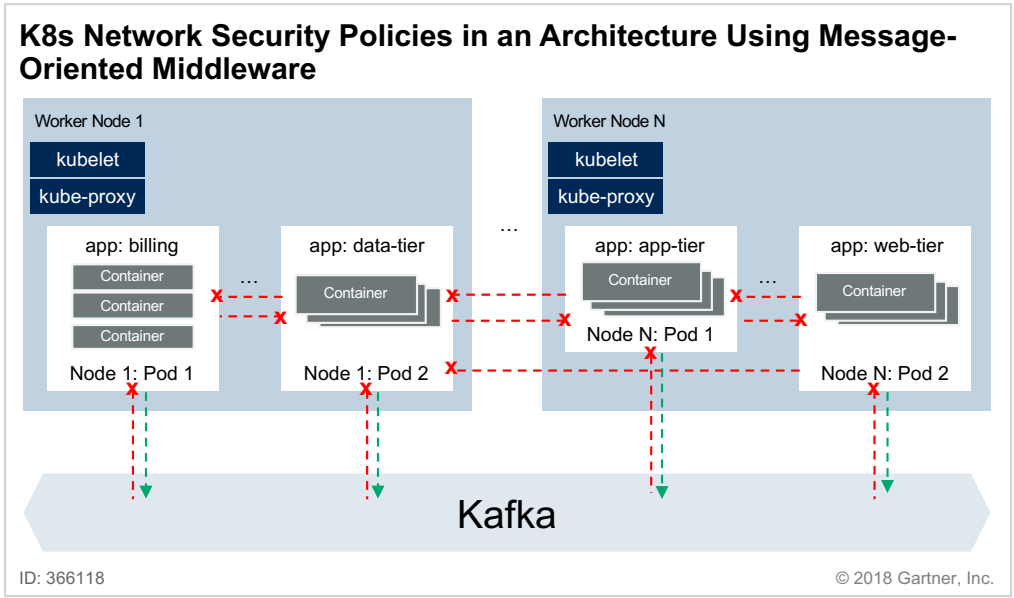
Figure 6. K8s Architecture With Network Security Policies



Source: Gartner (August 2018)

Figure 7 illustrates the use of K8s network security policies in an environment that uses MOM for communication between microservices. Pods have no minimum size and can be as small as a single container, thus enabling the isolation of high-risk microservices.

Figure 7. K8s Network Security Policies in an Architecture Using Message-Oriented Middleware



Source: Gartner (August 2018)

Developers frequently challenge the need for additional network security controls beyond K8s network security policies and anticipate that additional network security will slow them down — effectively annihilating the sense of having containers. If labels and rule sets are configured properly, adding additional network security to containers does not impact speed or agility of the workflow. Depending on the enabled features, network security products for containers will add approximately 10% additional load on the CPU of every worker node.

Whether to implement additional network security controls that are similar to microsegmentation used for traditional VMs for your containers is a risk-based decision. Products such as Cilium (open source), NeuVector, vArmour or VMware NSX-T Data Center generally include the following features that go above and beyond K8s network security policies:

1. Logging of all connections (denied or allowed traffic in ingress and egress direction).
2. Deep inspection of network connections for embedded threats and attacks (for VMware NSX-T Data Center this is a roadmap item).
3. Segmentation based on Layer 7, not just IP address and port. This is very similar to state-of-the-art network enterprise firewalls (also a VMware NSX-T Data Center roadmap item).
4. Detection of data exfiltration using tunneling protocols like DNS/ICMP. In K8s network security policies, there is no possibility of controlling outbound traffic in pods. Therefore, developers are frequently surprised about connections attempted by open-source apps that they didn't know were going to happen (such as license verification, DNS, updates, etc.).
5. Packet capture (only NeuVector and VMware NSX-T Data Center; sysdig can provide a system capture "syscap" instead).
6. Network visualization — depending on the product also into MOM — to verify application communications.

Clients also need to consider that containers are not alone in the data center and may also interact with traditional VMs and physical servers. VMware NSX-T can be used to protect VMs and containers alike. vArmour can be used to protect containers, VMs and bare-metal servers. Table 1 lists selected container security products with network features that are on par with state-of-the-art microsegmentation like you would use to secure VMs. For in-depth information about microsegmentation, see "Comparing Products for Microsegmentation in Virtualized Data Centers."

Table 1: Selected Network Security Products for Containers

	Architecture	Policy Computation	L7 Protocol support	Network Visualization	Packet Capture	Traffic Recording (Forensics)	Scope
Cilium	CNI plug-in	No	Few (HTTP, protobuffs)	Unclear	No	No	Containers
NeuVector	Proprietary	Yes	Comprehensive	Yes	Yes	Yes	Containers
vArmour	NFV	Yes	Comprehensive	Yes	No	No	Containers, bare-metal servers, VMs
VMware NSX-T Data Center	Hypervisor API, CNI plug-in	Roadmap	Roadmap	Yes	Yes	Yes	Containers, bare-metal servers, VMs

Source: Gartner (August 2018)

Architectural Considerations

When evaluating container security solutions, there are several key architectural considerations that vary among products. Table 2 is a shared responsibility matrix showing what responsibilities are with the client and what responsibilities are with the CSP — depending on the chosen architecture. All items that are managed by the CSP or even abstracted away from the client must be assumed secure and do not require third-party security. The content of Table 2 is also explained below.

Container platforms: Includes Docker EE and Red Hat OpenShift Container Platform. Container platforms are installed on-premises or on infrastructure as a service (IaaS). Clients manage workload, deployment server (such as Jenkins), physical and virtual servers, container engine, orchestrator (such as K8s), the underlay network and the container-

specific software-defined network (SDN). Support for third-party security add-ons is generally very good. Some platforms are closed systems with proprietary security extensions, for example, Apcera.

Container as a service (CaaS): Includes Amazon Elastic Container Service (ECS) and Azure Container Service (ACS). CaaS requires clients to provision a cluster of cloud-based virtual systems (IaaS) that run an image with the container engine. The image is provided by the CSP and upgraded on demand. The containers are managed using CSP proprietary APIs. Clients manage the workload, deployment servers and the SDN. Generally, clients can install security agents on CaaS, but they may need to be redeployed after scheduled updates to the cluster nodes made by the CSP. Since the cluster images are managed by the CSP, it will not make sense any more to evaluate the cluster images for their compliance with, for example, CIS benchmarks.

K8s CaaS: Includes Amazon Elastic Container Service for Kubernetes (EKS), Azure Kubernetes Service (AKS), Google Kubernetes Engine (GKE) and IBM Cloud Kubernetes Service. This is a CaaS environment orchestrated with a managed K8s platform in place of the CSP-provided APIs used in CaaS. The possibilities to install third-party security for K8s CaaS are similar to plain vanilla CaaS. The situation around the supported SDN plug-ins is currently unclear.

Function platform as a service (fPaaS, aka serverless computing): Includes AWS Fargate, AWS Lambda (functions) and Azure Functions. The provisioning and management of that server infrastructure is fundamentally abstracted from the consumer of the services. The client only deploys and orchestrates the workloads. In this environment, it is not possible to install agents or preferred SDNs. Security products frequently use workarounds to at least provide some of their security features to fPaaS containers. For example, they provide a binary that clients can add to every container image, thus controlling containers deployed on fPaaS from within.

Table 2: Shared Responsibility Model for Container Deployment Architectures

	Container Platforms	CaaS	K8s CaaS	fPaaS
Workload	Client	Client	Client	Client
Deployment server	Client	Client	Client	Client
Servers	Client	CSP	CSP	CSP
Node image	Client	CSP	CSP	CSP
Orchestrator	NA (proprietary)	Client	Client	NA
Underlay network	Client	CSP	CSP	CSP
SDN	Client	Client	Client or CSP	CSP

Source: Gartner (August 2018)

Figures 8 and 9 show the container capabilities of representative vendors (not an exhaustive list).

Figure 8. Representative Commercial Products Offering Container Security Capabilities (Part 1)

	Pervasive Container Security							Container Network Security		
	Aqua Security	Docker EE	StackRox	Sysdig Secure	Tenable	Trend Micro	Twistlock	NeuVector	vArmour	VMware NSX-T Data Center
FOUNDATIONAL CONTROLS										
HARDENING										
CIS Docker benchmarks	Y	Y	Y	Y	Y		Y	Y		
Implementation of CIS benchmarks for K8s	Y		Y	Y	Y		Y	Y		
KERNEL VULNERABILITY PROTECTION										
Enforce SELinux default policies	Y	Y		Y	Detection		Y	Detection		
Tune/configure SELinux		Y		Y						
Enforce seccomp profiles	Y	Y		Y	Detection		Y	Detection		
Tune/configure seccomp profiles	Y		Modelling sup.	Y			Y			
SECRETS MANGEMENT										
Product proprietary secrets store	Y	Y	R							
Leveraging HashiCorp Vault as secrets store	Y		R		Y		Y	R		
Leveraging AWS Secrets Manager or AWS KMS as secrets store	Y		R				Y			
Leveraging Azure Key Vault as secrets store	Y		R				Y			
Secrets management with K8s		Y	Y		Y		Y	Y		
BASIC CONTROLS										
IMAGE AND SOFTWARE COMPONENT SECURITY										
Software composition analysis for vulnerable software	Y	Y	Y	Y	Y	Y	Y	Y		
Software composition analysis for unwanted license types (e.g., accept only BSD-licensed libs)	Y		Y							
Software composition analysis finds secrets in code in Git-type repo or container image	Y			Y			Y			
Image scanning during build (on developer laptop)	Y						Y	Y		
Verify container image signature	Y	Y	Y	Y	Y		Y	R		
Maintain list of trusted signatures or packages	Y				Y	Y	Y			
API integration or plug-ins for deployment servers such as Jenkins or Bamboo	Y		Y	Y	Y	Y	Y	Y		
CONTAINER COMMUNICATION SEGMENTATION & IP-BASED FEATURES										
Layer 3 firewall	Y		Y	Detection		Y	Y	Y	Y	Y

Source: Gartner (August 2018)

Figure 9. Representative Commercial Products Offering Container Security Capabilities (Part 2)

	Pervasive Container Security							Container Network Security		
	Aqua Security	Docker EE	StackRox	Sysdig Secure	Tenable	Trend Micro	Twistlock	NeuVector	vArmour	VMware NSX-T Data Center
MSA ARCHITECTURE SECURITY										
MSA (message) authorization										
MSA resilience										
API gateway										
DATA SECURITY										
Visibility on communication relations across MOM									Y	
DAP, FCAP										
RISK-BASED CONTROLS										
APPLICATION RUNTIME SECURITY										
Container malware scanning	Y		Y		Y	Y	Y	R		
Anomaly detection (ML-based)	Y		Y	Y			Y	Y	Y	
Anomaly detection (preconfigured application runtime profiles)	Y		Y	Y	Y	Y	Y	Y		
Identify new container vulnerabilities at runtime (configuration drift)	Y		Y	Y	Y	Y	Y	Y		
Identify container vulnerabilities at runtime via the image/in the repo	Y		Y	Y	Y	Y	Y	Y	Correlation	
WAF rules						Y	Y			
Incident response	Y		Y	Y			Y	Y		
Prevent file access of container	Y		Y	Detection			Y	Detection		
CONTAINER COMMUNICATION SEGMENTATION & IP-BASED FEATURES										
Prevent network access of container	Y		Y	Detection			Y	Y	Y	Y
Network packet capture				Syscap				Y		Y
Layer 7 firewall								Y	Y	R
Security zoning and segmentation of container traffic based on container identity (non-IP)	Y		Y				Y	Y	Y	Y
IP-based threat intelligence	Y					Y	Y	Y		

Source: Gartner (August 2018)

Strengths

- There is plenty of choice in container security tools. Container security products and options are maturing. Leading products have features that mitigate key risks in the CI/CD pipeline.
- Security tools integrate well with important container platforms and architectures. Clients that start out with container platforms on-premises often very quickly move on to cloud-based offerings, making it important that products do not lock in to a certain architecture.
- Competition in the container security market is driving product prices way down, making the acquisition of a container security product an easy decision for clients that can accept the performance impact and have container operations maturity.

Weaknesses

- Many container security products provide similar controls, making it confusing for the client to choose between products. Gartner observes that in the absence of meaningful differentiators, clients choose product price as the decision-making criterion and go for the cheapest offer.
- Clients struggle to identify who should configure container security and maintain it in day-to-day operations. Container security controls are pervasive throughout the CI/CD pipeline, making it hard for clients to find appropriate staff to configure and maintain a diverse set of controls. Gartner observes that security professionals frequently experience a steep learning curve when acquiring knowledge about, for example, MSAs, SCA and MOM. This limits them to the foundational controls until they become acquainted with the new concepts.
- Gartner clients find that container security products frequently have a 15% CPU performance impact on the worker nodes. This is more than what many clients are willing to accept. Vendors prefer not to talk about this, instead seeking options to let clients limit CPU impact by, for example, disabling CPU intensive features that the client may not use.

Guidance

Make Security Pervasive in the CD/CD Pipeline

Security controls should be added to every step of the CI/CD pipeline, starting with the developers' IDE through production. Leading container security products offer controls mitigating key risks in the CI/CD pipeline. Depending on the overall maturity of your technology platform, processes and policies, you may need to implement these controls gradually. When the required controls are implemented, you will eventually need to change your processes from getting security approval on a per-release basis to considering all releases that have been pushed through the fortified CI/CD pipeline secure enough.

Use Secrets Management and Software Composition Analysis as Your Primary Container Protection Strategies

Secrets are information you must keep confidential. Gartner recommends that clients use secrets management products to provide secure, just-in-time access to credentials without losing control over their usage.

If you are currently using SCA for other code artefacts, such as Flexera, JFrog, Sonatype, Snyk, Synopsys (Black Duck or Protecode) and WhiteSource, continue what you have been doing and expand the scope to include container images. If you are new to SCA, take the SCA functionalities that you get anyway from most of the container security products (see also Figure 6). Drawing full value from SCA won't be a quick fix. Rather, it will be a larger project. For more information, see "Approaches for Securing Application Development Environments and Artifacts."

Add Layer 7 Network Segmentation for Operational Containers That Require Defense in Depth

For network isolation of production containers, either standardize your containers and K8s access lists to communicate via MOM, thereby making network separation no concern, or implement Layer 7 network security products such as NeuVector and vArmour. Layer 7 network security for containers is still a rarity since most products will include Layer 3 filtering that arguably does only a little more than the K8s access lists. Risk assessments must answer the question: Does the sensitivity of your MSA and its data make defense in depth such as Layer 7 network security a prerequisite?

Require Vendors to Integrate With the Container Offerings of Leading Cloud Service Providers

Container security products need to be able to secure container platforms, CaaS, K8s CaaS and fPaaS. Visibility and security of client-managed layers in cloud-based container services (listed in Table 2) are emerging critical requirements that must be provided by the implemented security layer. Clients will find that vendors are good at checking boxes confirming that they support everything in all environments. Gartner recommends clients check what are roadmap items versus mature, reliable features for operationalized containers. If something is too easy and too good to be true, it frequently is an ambitious roadmap item.

The Details

The vendors listed in this document do not imply an exhaustive list. This section is intended to provide an overview of container security and container network security products and vendors.

Figures 8 and 9 above show representative commercial products and how they compare across the controls of the container security hierarchy (Figure 2) that is the basis of this document.[17]Commercial products in this table are categorized as generalized container security products or network security products. NeuVector has offerings in both categories but is listed as network security product.

Figures 10 and 11 show what controls of the container security hierarchy can be implemented using open-source software or free software. To match the control requirements with open-source software requires the installation of more than one product that you must manage. The breadth and depth of functionality of open-source and free software in this space are impressive.

Figure 10. Representative Open-Source and Free Products Offering Container Security (Part 1)

Representative Open-Source and Free Products Offering Container Security

	Open-Source or Free Solutions
FOUNDATIONAL CONTROLS	
HARDENING	
CIS Docker benchmarks	Aqua Security, Docker Bench
KERNEL VULNERABILITY PROTECTION	
Enforce SELinux default policies	Falco, K8s
Tune/configure SELinux	Falco
Enforce seccomp profiles	Falco, K8s
Tune/configure seccomp profiles	Falco
SECRETS MANGEMENT	
Product proprietary secrets store	Conjur, HashiCorp, K8s
BASIC CONTROLS	
IMAGE AND SOFTWARE COMPONENT SECURITY	
Software component analysis for vulnerable software	Anchore, OpenSCAP
Software component analysis for unwanted license types (e.g., accept only BSD-licensed libs)	
Software component analysis find secrets in code in Git-type repo or container image	Anchore, Falco
Image scanning during build (e.g., on developer laptop)	Aqua Security, Dagda
Verify container image signature	Docker Notary
Maintain list of trusted signatures or packages (e.g., accept all containers signed by Oracle)	
Plug-ins for deployment servers such as Jenkins or Bamboo (pls. list deployment servers)	Anchore
CONTAINER COMMUNICATION SEGMENTATION & IP-BASED FEATURES	
L3 Firewall	Falco (detection), K8s
MSA ARCHITECTURE SECURITY	
MSA (message) authorization	Aporeto Trireme
MSA resilience	
API gateway	
DATA SECURITY	
Visibility on communication relations across MOM	
DAP, FCAP	

ID: 366118

Source: Gartner (August 2018)

Figure 11. Representative Open-Source and Free Products Offering Container Security (Part 2)

	Open-Source or Free Solutions
RISK-BASED CONTROLS	
APPLICATION RUNTIME SECURITY	
Container malware scanning	
Anomaly detection (ML-based)	Falco
Anomaly detection (preconfigured application runtime profiles)	Falco
Identify new container vulnerabilities at runtime (configuration drift)	Chef InSpec, Falco
Identify container vulnerabilities at runtime via the image/in the repo	Falco
WAF rules	
Incident response	
Prevent file access of container (TBD)	Falco (detection)
CONTAINER COMMUNICATION SEGMENTATION & IP-BASED FEATURES	
Prevent network access of container	K8s
Network traffic dump in/egress of containers	
Layer 7 firewall	Cilium
Security zoning and segmentation of container traffic based on container identity (non-IP)	Cilium, Falco, K8s, Aporeto Trireme
IPS features	
IP-based threat intelligence	

Source: Gartner (August 2018)

Representative Generalized Container Security Products

The following vendors offer solutions generalized container security products:

Aqua Security

NeuVector

StackRox

Sysdig

Tenable

Trend Micro

Twistlock

VMware (AppDefense)

Representative Container Network Security Products

The following vendors offer container network security products that have Layer 7 defenses:

NeuVector

vArmour

VMware NSX-T Data Center

Open-Source Projects to Secure Containers

The following open-source projects exist to secure containers:

Anchore (software composition analysis)

Aporeto Trireme (microsegmentation and mutual authentication of workloads)

Aqua Security Microscanner (software composition analysis)

Aqua Security Kube-Bench (CIS Kubernetes benchmark testing)

Aqua Security Docker-Bench (CIS Docker benchmark testing)

Chef InSpec (audit configurations of instantiated containers)

CoreOS Cilium (network communication mediation)

CyberArk Conjur (open-source secrets management)

Dagda (combines OWASP dependency check and retire.js for build time analysis and Sysdig Falco for runtime analysis)

Docker Notary (image signing)

Grafeas and Kritis (container metadata organization and governance/restrict instantiation in K8s)

HashiCorp Vault (open-source secrets management)

NeuVector Kubernetes CIS Benchmark (CIS Kubernetes benchmark testing)

OpenSCAP (software composition analysis)

Project Falco, Draios/Falco (behavioral monitoring of containers)

Sysdig (container security platform)

Evidence

[1] For detailed information on how to set up automation for your CI/CD pipeline, read the Docker documentation "Configure Automated Builds on Docker Hub," Docker.

[2] Note that sometimes environment variables are used to pass secrets. Every process in the container will be able to read the secrets. Your secrets may become part of your log files.

[3] "Over 30% of Official Images in Docker Hub Contain High Priority Security Vulnerabilities," Banyan.

[4] "Backdoored Images Downloaded 5 Million Times Finally Removed From Docker Hub," Ars Technica.

[5] Cryptojacking is when your systems are used (by malware) to mine cryptocurrency without you knowing.

[6] "CIS Benchmarks," CIS.

[7] "aquasecurity/kube-bench," GitHub.

[8] "neuvector/kubernetes-cis-benchmark," GitHub.

[9] Or as part of instance bootstrapping, ability to call a web API hosted by the secrets management service.

[10] "From Monolith to Microservice Architecture on Kubernetes, Part 1—The API Gateway," Medium.

[11] "Amazon API Gateway FAQs," AWS.

[12] "API Management in Docker Containers," Axway.

[13] "3Scale Deploys API Gateway as a Docker Image," Container Journal.

[14] "(RabbitMQ) Home Page," RabbitMQ.

[15] "Apache Kafka Home Page," Apache Kafka.

[16] Recipes can be found on "ahmetb/kubernetes-network-policy-recipes," GitHub.

[17] That list/table is based on the Gartner container security hierarchy. For a complete or exhaustive list, Gartner recommends clients to use published vendor documentation and data sheets.