PCI Security Standards Council®

Date: **September 2022**

Author: Best Practices for Container Orchestration Special Interest Group
PCI Security Standards Council

Information Supplement:

Guidance for Containers and Container Orchestration Tools

Document Changes

Date	Document Version	Description
September 2022	1.0	Initial release

Contents

1 Introduction

Organizations are increasingly adopting container technology to scale, secure, and rapidly deploy the applications on which they rely. These lightweight software components bundle an application, its dependencies, and its configuration in a single image, running in an isolated environment, allowing for highly distributed application infrastructures. When implemented and managed properly, containerized environments can enable swift deployment, increased scalability, portability, and security.

These advantages can be enhanced by the deployment of container orchestration tools that facilitate load-balancing, resource allocation, and security enforcement by automating the deployment, management, networking, and scaling of containers. However, container orchestration tools are not without security risk, and their use within a payment environment should be conducted with due consideration of applicable security best practices.

This document provides guidance for the secure use of containers and container orchestration tools in a payment environment. To contextualize container orchestration tool specific threats and best practices in a way that is meaningful to PCI stakeholders, this document presents best practice controls of common container use cases. Through this approach, this guidance will benefit merchants, service providers, and assessors in understanding how controls may be applied to securing various containerized environments.

The guidance in this document is structured in three parts:

1. A high-level description of containers and container orchestration tools.
2. A list of threats, and the best practice controls intended to address them, identified by common container orchestration use cases.
3. Use case descriptions and example threats to illustrate the application of specific best practices.

This document provides supplemental guidance which does not add, extend, replace, or supersede requirements in any PCI Security Standards Council (PCI SSC) standard. The PCI SSC is not responsible for enforcing compliance with any of its standards. Entities and third-party service providers should work with their acquirers and/or payment brands to understand any compliance validation and reporting responsibilities.

1.1 Intended Audience

The information in this document is intended for entities responsible for developing, deploying, managing, or assessing containerized environments, including:

- **Merchants and Service Providers** – Guidance on security considerations that apply to the use of container orchestration tools in containerized payment environments.
- **Assessors** – Guidance on security considerations to help assessors better understand security issues when assessing a payment environment that uses container orchestration tools.

1.2 Terminology

Some of the terms used in this document are defined in the *Payment Card Industry (PCI) Data Security Standard Glossary, Abbreviations, and Acronyms* (https://www.pcisecuritystandards.org/pci_security/glossary).

Term	Definition
(Auto-) Scaling	An (automatic) adjustment of the number of instances of running containers using the same definition, to address service demands and the availability of resources.
Cluster	A set of containers grouped together and running on nodes.
Container	A software package that includes all elements (application and dependencies) necessary to run on a container platform.
Container Engine	An application that generates an instance of a container from a container image.
Container Host	A physical or virtual device that hosts running container(s).
Container Image	A read-only template from which containers are created by the container engine. Also referred to as a Container Engine.
Container Orchestration	A process that automates the deployment, management, scheduling, and scaling of containers.
Container Orchestration Tool	A set of software tools that provide container orchestration functions. Sometimes referred to as a Container Orchestrator.
Container Runtime	An application that generates an instance of a container from a container image. Also referred to as a Container Engine.
Control Plane	A set of services within the network that perform traffic management functions, including security, routing, load balancing, and analysis.

Term	Definition
Image Registry	A collection of container images from which containers may be accessed by the container engine.
Master Node	A node that acts as a controller, acting as a front end to the cluster of one or more worker nodes, providing scheduling, scaling, implementing updates, and maintaining the state of the cluster.
Node	A physical or virtual machine that hosts a container and that may be defined as a worker node, manager, or master node.
OCI	The Open Container Initiative (OCI) is an open governance structure for creating open industry standards around container formats and runtimes.
Pod	A collection of one or more Kubernetes-coupled containers.
Registry Server	A file server storing container images.
Worker Node	A node that executes the container(s) and applications, often as clusters.
Workload	An application running on or managed by the container orchestration system. A workload can be a single component or several that work together.

2 Overview of Containers and Container Orchestration Tools

2.1 Containerization

Containerization allows the deployment of applications without concerns about on which specific machine(s) the application needs to run. Applications are packaged as “containers” that decouple them from any individual host, and this decoupling or abstraction is known as “containerization”.

While containerization has been available since Unix version 7’s chroot, implementation of containerization technologies has gained increasing popularity as the technology has developed. To understand how to secure containers and container orchestration tools, it is useful to understand where these technologies operate within the technology stack and compare them to other related technologies, such as hypervisor-based virtualization.

Traditional hardware-based infrastructures consist of individual, interconnected servers that contain their distinct CPU, memory, and storage, such as interconnected email servers, database servers, and web servers. Separate networking devices connect these systems in the working environment. This approach requires a large investment in equipment, physical space, and physical management of the separate devices and has limited ability to scale resources based on resource demand.

With the introduction of hypervisor-based virtualization, it became possible to make more efficient use of hardware investments by consolidating CPU, memory, and storage of multiple servers onto a single shared host. Further developments in network virtualization allowed for the abstraction of network resources through a decoupling of network services from networking hardware, to provide complete virtual networks through software-defined networking technologies.

Using container and container orchestration technologies is a natural evolution of the physical-to-virtual transition many IT organizations have already experienced. These technologies provide increased portability of applications, greater efficiency in deployment, and streamline the development of applications through the support of agile development methodologies. Pre-packaged containers exist for many popular software components, including web servers, database servers, application servers, network management tools, and system logging services.

2.1.1 What is a container?

A container is a unit of software that bundles an application and all its dependencies together and abstracts or separates it from the underlying operating systems, enabling containerized applications to run consistently across multiple platforms. Using containerized applications allows developers and system administrators to:

- Package applications with the confidence that they will operate in a production environment in the same way as they operate in the development environment.

- Both run the container application on different platforms and move containers between different platforms without making any changes to the container.
- Reduce system resource requirements by leveraging the container host's operating system, libraries, and other resources.
- Rapidly scale container instances to meet the production requirements.
- Quickly patch, update, and redeploy applications through centralized container image management.

Containers are often packaged as images that use a standards-based Open Container Initiative (OCI) archive format. Container images are commonly based on optimized versions of operating systems such as Linux, OS X, and Windows.

Container images become containers at runtime when run on a container engine. The container engine runs on the host platform and provides the basis of functionality for the container, including:

- Handling user input,
- Connecting to the orchestrator,
- Accessing the container image from the registry server, and
- Preparing the image prior to calling the container runtime.

The container engine is also responsible for isolating each container's compute, network, and storage resources from other containers and from the containerized host. Additionally, containerized platforms can isolate users, time, hostname, and other types of resources using namespace isolation (limiting the process view of resources) provided by the container host's operating system.

2.1.2 Basic Container Architectures

Each phase of a container's lifecycle might exist as a part of an entirely self-managed system or as part of a third-party-provided service. The containers may run on hardware, on virtual machines in an on-premises data center, or in a cloud or multi-cloud environment. Understanding these architectures is important in determining who owns, and is therefore responsible for, the security of each component in a container system. The following descriptions are not exhaustive of all container architectures but will provide some context in understanding how containers can be securely implemented and, by extension, container orchestration.

Self-Managed

Whether on-premises or in the cloud, a self-managed container environment is one in which the hosts, images, and running containers are controlled by an organization for its own use. This approach is especially common for the development and testing of software and containers themselves. Typically, in a self-managed environment, container hosts are provisioned like any other server, and the container

engine and orchestration software are installed for use by the system owners using or developing the containers.

Containers as a Service (CaaS)

CaaS solutions are similar to older virtualization-as-a-service solutions. Container hosts are provided as a managed service and can run images and containers which are internally developed, built, or specified by an organization. Underlying management of the hosts, including networking and storage, is usually owned by the service provider. Typically, it is up to the service customer in these cases to ensure that redundancy and adequate scaling for their workloads are provided. In some cases, CaaS providers may provide the options to run containers on hosts in different geographic regions and may provide different classes of container hosts, offering customers various combinations or quantities of resources like memory, CPU cores, network interfaces, storage media, etc.

Managed Container Services

Managed Container Services often include orchestration as part of the managed solution, called Orchestration-as-a-Service, or may be named after the specific orchestration platform provided. Typically, customers of Managed Container Services supply or specify the containers to run, often in the format needed by the orchestration platform. In turn, the service provider guarantees the availability and performance of the running containers, moving or scaling containers as needed.

Containers as a Platform

Some service providers take customer-provided container or application source code and automatically build, deploy, and orchestrate the containers. Often referred to as one form of “serverless” computing, these solutions handle all the underlying components of the infrastructure and container platform, allowing customers to focus on writing their unique application code.

2.1.3 Differences between containers and traditional hypervisors

Containerization technologies share some similarities with more “traditional” hypervisor-based virtualization technologies, as well as some key differences. For example:

- Both technologies abstract or separate system resources from the underlying hardware to provide more efficient use of resources across multiple workloads. Hypervisors abstract these resources to provide virtual machines that consist of a full, general-purpose operating system, whereas containerization software provides these resources to applications that run a minimalized operating system, relying on the underlying host operating system for basic services.
- Creating a new instance of a hypervisor-based virtual machine frequently may take minutes while the operating system is loaded from block storage. In comparison, initializing a new container to respond to increased demand is usually much faster, possibly taking only seconds.
- Because containers rely on the underlying container host operating system for basic services, there exists a tighter bond between the two. For example, Linux-based container hosts can run Linux-

based containers but cannot run Windows-based images. In contrast, hypervisors provide abstracted resources such that they are independent of the virtual machine's operating system.

- Individual containers are at greater risk of impacting other containers on a single host than hypervisor-based virtual machines because of the availability and use of shared resources to containers, where the risk extends to impacting the container host itself.

The following diagram illustrates the architectural differences between traditional application deployments, hypervisor-based virtualized deployments, and containerized deployments.

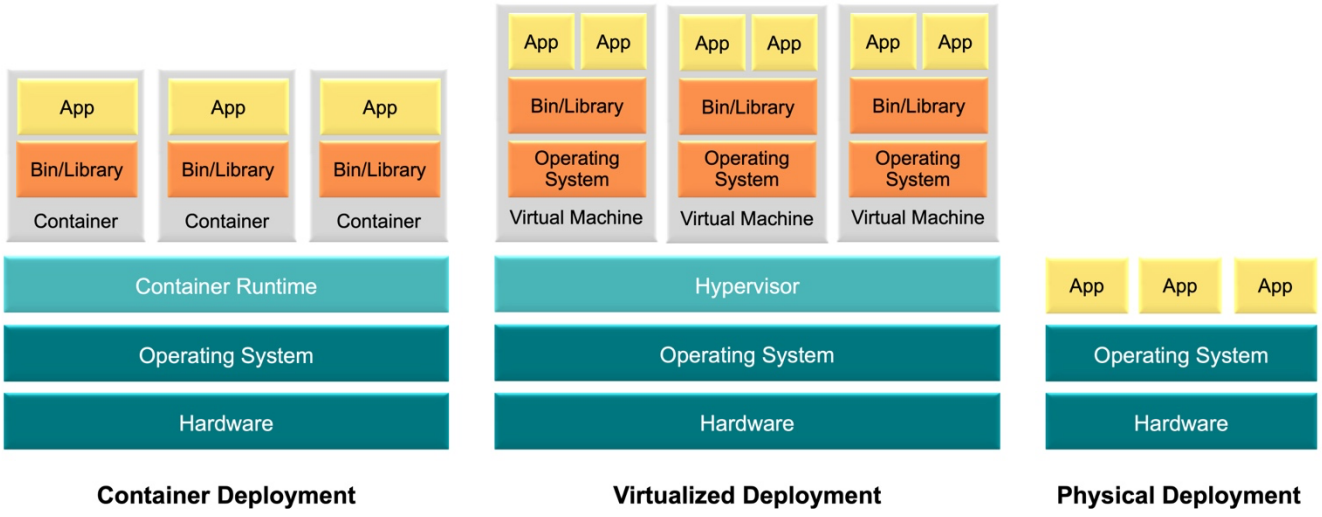

Figure 1: Architectural Differences between Traditional, Virtualized, and Containerized Deployment

2.1.4 Container Isolation

Application, communication, and resource isolation are fundamental security controls. It is important to understand how container runtimes provide isolation. The following lists some common approaches and their security properties, as provided in Figure 2, "Process, Sandbox, and Hypervisor-Based Isolation."

- **Process based** – (for example, Docker, ContainerD, CRI-O, Windows Server Container) This approach provides isolation using operating system features—for example, on Linux, cgroups and namespaces—and limits access using operating system security primitives. While providing a level of isolation, this approach is not a specifically designed security sandbox and is not suitable for higher risk applications.
- **Sandbox based** – (for example, gVisor) This approach builds a dedicated security sandbox on a host to isolate the contained process. It provides a reduced attack surface compared with process-based isolation; however, should a sandbox escape occur, access to a shared underlying host may still be possible.
- **Hypervisor based – (**for example, AWS Firecracker, KataContainers, Hyper-V) This approach runs the contained process in a dedicated virtual machine instance. A hypervisor will typically use

optimized operating systems to improve performance when used with containers. This approach has some similarities to traditional virtual-machine environments but uses container style workflows and a specialized hypervisor to allow for integration with container orchestration tooling.

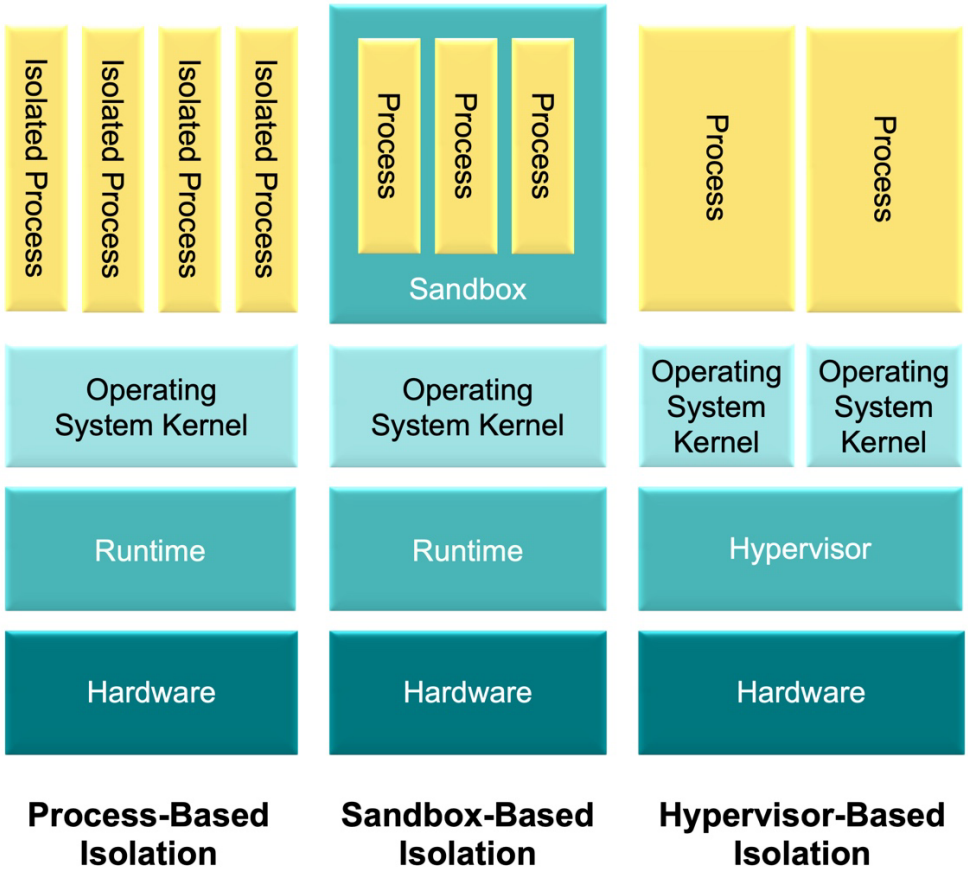

Figure 2: Process, Sandbox, and Hypervisor Based Isolation

2.2 Container Orchestration Tools

As the scale of containerized workloads grows beyond a small number of container hosts, managing the container environment becomes increasingly difficult. Container orchestration addresses this problem by bringing together several critical capabilities necessary to efficiently operate and manage a secure, container-based application delivery model at scale. These capabilities include:

- Providing a centrally managed repository of container images.
- Providing cluster management capabilities to pool container host nodes and distribute computer, storage, and network resources between the nodes.
- Assigning workload requests to container host nodes based on available resource capacity.

- Providing automation through:
  - "Infrastructure as code" configuration management, where the infrastructure is defined through definition files.
  - Use of a build/release pipeline tool to streamline the application release management process.
- Conducting instrumentation and monitoring of the performance, security, and overall health of the container hosts, containers, and container orchestration platform.
- Securing services by defining permitted interactions within and between services, and between containers and the container host.

2.2.1 Container Orchestration Tool Architecture

Container orchestration tool architecture is that of a containerized application providing automated services to other containers. As with all containers, a container orchestration tool server runs on a platform comprised of hardware, operating system, and container runtime. As with other containerized applications, the container orchestration tool can be implemented through various architecture models including Self-Managed, Container as a Service, Managed Container Services, or Container as a Platform.

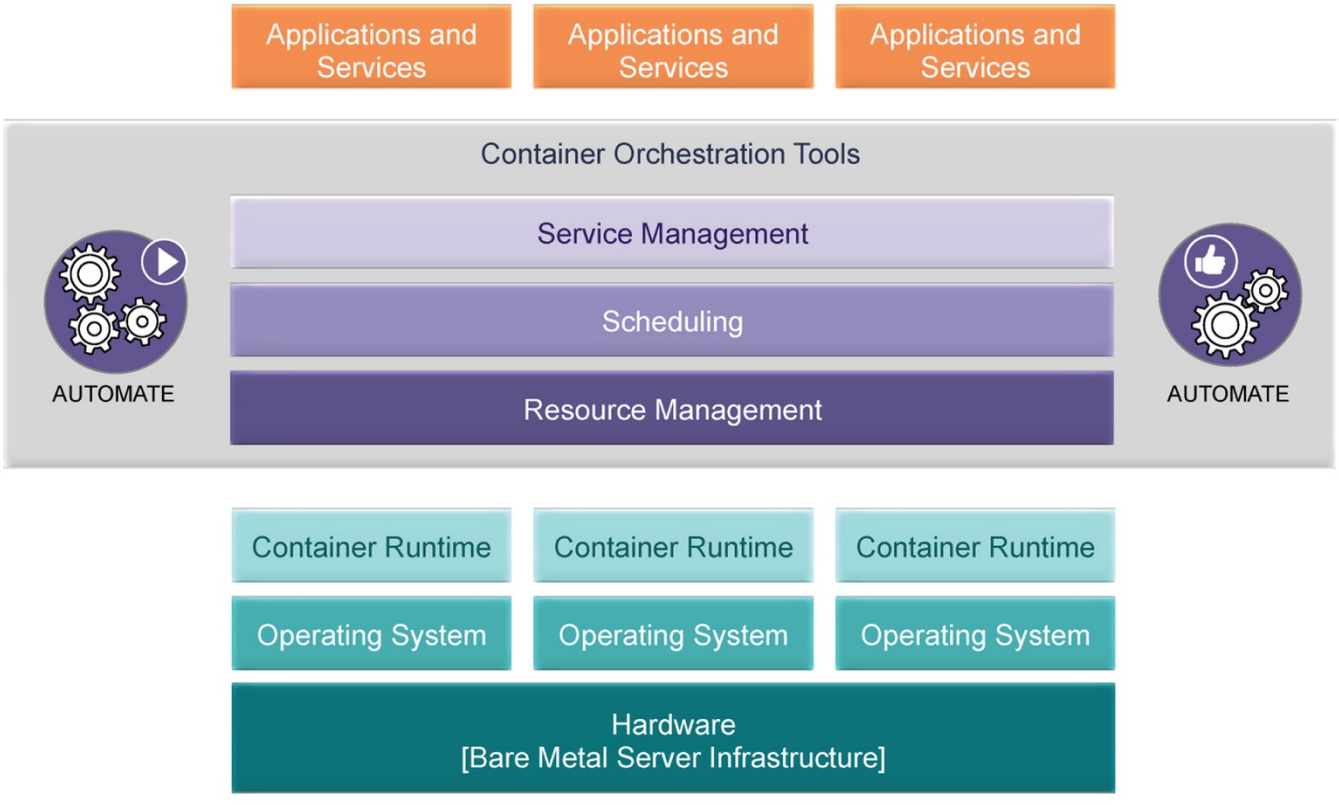

Figure 3: Container Orchestration Tool Architecture

Container orchestration tools generally support a hierarchical architecture consisting of controller or manager nodes and worker nodes.

Controller nodes (for example, Swarm manager in Docker)—provide managing services such as:

- Workload scheduling,
- Receiving API requests from administrators,
- Configuration and state storage,
- Administrative interfaces, and
- Controller management for groups of worker nodes.

Worker nodes are small components responsible for receiving and executing orders from the controller, as well as managing containers (for example, “kubelet” and “kube-proxy” in Kubernetes. These nodes are installed and configured on a host platform.

Image registries are private or public storage locations containing static container images which can be pulled by a worker node to be run as a container by the container engine.

The connection between these elements is illustrated in Figure 4, “High-Level Master Node-Worker Node Architecture,” which shows a single Master Node providing services to multiple Worker Nodes which receive the container images from the Image Registry.

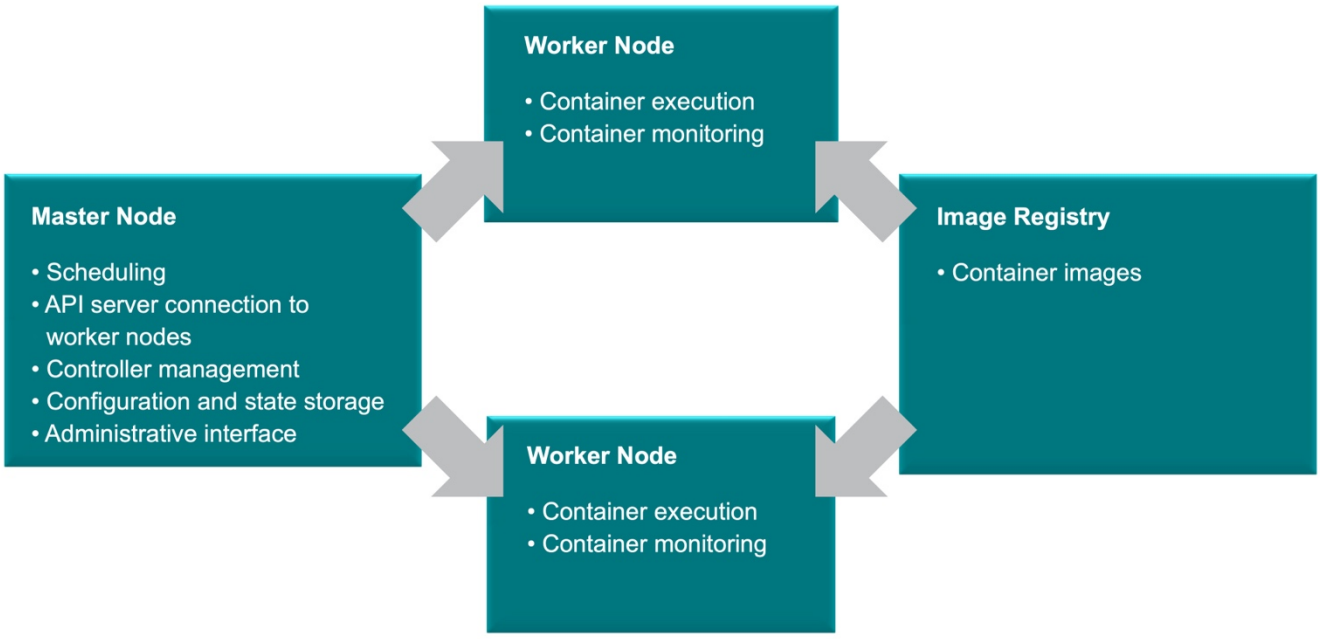

Figure 4: High-Level Master Node-Worker Node Architecture

2.2.2 Common Features of Container Orchestration Tools

While there are many available implementations of container orchestration tools, providing their own distinct approach to container management and often tailored to a vendor-specific managed environment, typically the solutions provide a set of common features and similar functionality. Some of these common features include:

Scaling

Based on demand, container orchestration tools can facilitate automatic scaling up or down by mounting or terminating container instances, resulting in increased resilience to additional loads. Scaling can occur at the application horizontally, by adapting the number of workload replicas, vertically, by adapting resource requirements for groups of workloads, such as pods, or at the container cluster itself, by changing the number of cluster nodes or containers. Scaling requirements are determined through monitoring of resource availability and performance, using built-in health checks or probe operations such as HTTP checks, which monitor the response to a service request.

Security

Controls that address security requirements are configured through the container orchestration tools features and functions, including:

- Approved and auditable software defined networking and segmentation
- Hardened images
- Simplified image updates and rollbacks
- Centralized logging
- Inventory management
- Infrastructure authentication (usually via x509 client certificates – a digital certificate that binds an identity to a public key using a digital signature)
- Vulnerability scanning of container images
- Prebuilt deployment templates and image validation
- Protection of code in images
- Removal of development tools in images
- Protection of sensitive data and proprietary software
- Improved security isolation between containers and OS
- Integration with secret managers

Functionality

Container orchestration tools are used to automate any of the following within the container environment, including applications and infrastructure:

- Scheduling
- Deployment
- Networking
- Resource management
- Capacity management (scaling)
- Health checks

Orchestration tools abstract and automate the activities and overhead required by system administrators to keep the containerized infrastructure running at scale, in good health, and with adequate capacity in response to demand based on predefined configuration parameters and triggers.

Operation

Container orchestration tools can be utilized in any environment where containers are used. They utilize a predefined container configuration file (typically, YAML or JSON) to automate the management of containerized applications/infrastructure. The configuration file specifies many properties of a containerized application including:

- Source – location of the container images, how to setup networking and connectivity
- Scheduling – conditions under which to launch a container or dispose of a container
- Resource management – compute, storage, etc.
- Service Management – availability, capacity, etc.
- Security – permissions, access control, auditing, communications, monitoring etc.
- Others – user-defined metadata

Defining these properties ensures that orchestration tools can consistently deploy pre-defined and standardized containerized applications across many environments and platforms without the need to re-design the applications.

External integrations

Container orchestration tools typically support integration with other tools to support other processes for:

- Continuous integration and delivery (CI/CD) integrations
- Image registry compatibility for image inventory management
- Secret management via secret servers, key management systems, and encryption servers/services

Some container orchestration tools go beyond managing containers to manage:

- Advanced networking and storage features
- Cloud load-balancers
- Virtual machines
- Other cloud services
- Via custom integrations—for example, Custom Resource Definitions (CRDs) in Kubernetes
- Custom logic—for example, operators in Kubernetes

2.2.3 Advantages and Disadvantages of Using Container Orchestration Tools

Common Advantages

The primary benefit of using container orchestration tools is the effective and efficient management of large-scale deployments of containers and microservices.

Specific benefits include:

- Security: Container orchestration tools make it easy to automatically scale security configurations such as permissions, access control, auditing, and monitoring. This provides a single configuration and seamless deployment across many workloads.
- Scalability: Application autoscaling is managed by the container orchestration's scheduler, adding or removing replicas based on service needs.
- Configuration consistency: Using predefined configuration files, container orchestration tools can automatically scale applications consistently (as specified in the configuration files) without errors/mistakes.
- Policy-driven configurations: By using policy tooling, such as Open Policy Agent (OPA), layered on top of orchestration tools, configurations can be constrained by policies which align with common control frameworks. Additionally, this tooling can be integrated with Open Security Controls Assessment Language (OSCAL).

Common Disadvantages

Using container orchestration tools can also introduce some disadvantages, including:

- Complexity: As with the implementation of any integrated application, orchestration tools can be complex to deploy and may result in the insecure implementation of the tool.
- Possible knowledge gaps: Compared to traditional infrastructure deployments, container orchestration is still a relatively new field, and as such, there may be a knowledge gap that could result in security failures.
- Misconfiguration of the tool: The use of container orchestration tools to automatically scale infrastructure based on pre-configured parameters may lead to more widespread security failures where the tools are misconfigured.

2.2.4 When Containers and Container Orchestration Tools Should Not be Used

In addition to applying best practices whenever employing containers and container orchestration tools, it is important to consider situations or conditions that may lead to a decision not to use them in certain environments. While the use of containers and container orchestration tools can be beneficial in terms of cost, performance, manageability, and security, their use can introduce additional risk if applied under the wrong conditions. For example:

- Containers and container orchestration tools should not be used to resolve security, developer experience, or operational resilience deficiencies. Containers bring their own level of complexity and threats that need to be assessed. Consider whether the issues at hand are cultural or technical before moving away from tools that are working.
- Containers and container orchestration tools should not be used in environments that may have known or unknown platform compatibility issues, where the container may not have been tested on a particular platform—for example, a mainframe computer.
- Containers and container orchestration tools should not be used if there is a knowledge gap for individuals involved with the installation, operation, maintenance, internal functioning, and trouble-shooting of either the containers or the container orchestration tools. Invest time and money to learn how to use, scale, and operate containers before adopting this technology for running business-critical workloads.
- Container orchestration tools should not be used without a thorough understanding of their operation. It is important to treat the adoption of container orchestration tools as a paradigm shift, where several basic assumptions are either questioned or need to be reworked. Some examples include static vs. dynamic IP allocation, hypervisor vs. namespace-level isolation, repair vs. replacement of computer servers, and persistence of state in apps vs. immutability of containers.
- Container orchestration tools should not be used when application configuration is not centralized. Because of the ephemeral nature of containers, if an application exits inside a container all its configuration could be lost. Similarly, a manual workflow to fix configuration errors one at a time breaks down when multiple instances of containers share copies of the same configuration. This concern also applies in the case of password refreshes or cryptographic key and certificate renewals.
- Container orchestration tools should not be used for applications that do not need to run at a large scale or for which the traffic is predictable. Running a social network backend used by millions of users on container orchestration tooling may make more sense than running a static website with limited numbers of page requests.
- Container orchestration tools should not be used to avoid the maintenance and management of underlying infrastructure. Container orchestration tools help better manage the infrastructure by providing a higher-level abstraction, but the users of these tools are still responsible for managing and maintaining the underlying infrastructure for example, hardware or disk failures, server

operating system upgrades, network card failures, data corruption or loss, and routing failures are problems that will not be resolved solely with the adoption of container orchestration tools.

3 Use-case-based Container Orchestration Tools Threats and Best Practices

Employing container orchestration tools for developing, deploying, and managing containerized environments can provide increased convenience, reliability, and security where these tools are applied using industry best practices. Conversely, where such best practices are not applied, the use of containers and container orchestration tools can adversely impact the security of an environment, where exploitation of the tool leads to the exploitation of some or all of the containers. The correct application of industry best practices is critical for addressing plausible threats to the containers and container orchestration tools.

How and where containers and container orchestration tools are implemented may vary according to their purpose as well as the availability of both technical and physical resources. The implementation of some best practices is dependent on the applicability of the associated threats to a use case, and not all best practices apply to all use cases.

The most common use cases applicable in a payment environment include:

- A baseline case characterized by the generalized use of container orchestration tools. For the remainder of this document this use case is identified as “Baseline Case.”
- The use of container orchestration tools in a development and testing environment. For the remainder of this document this use case is identified as “Development and Management of Containerized Applications.”
- The use of containerized tools for services that transmit or process payment card account data. For the remainder of this document this use case is identified as “Containerized Services that Transmit or Process Account Data.”
- The use of containerized tools for services in a mixed scope environment. For the remainder of this document this use case is identified as “Containerization in a Mixed Scope Environment.”

Section 3.1, “Threats and Best Practices” presents a table of common threats to environments employing container orchestration tools applied to each use case, and possible best practices for addressing these threats.

Section 3.2, “Use Cases” provides a description of each use case, a sample scenario of a potential threat, and the best practices to employ in addressing the threat.

3.1 Threats and Best Practices

This section provides a list of common threats to containerized environments and possible security best practices to address each threat. Many of these best practices may be applicable outside of a containerized environment as provided in the best practices recommendations and may be required in some PCI SSC standards. Refer to applicable PCI SSC standards for more details.

The applicability of each best practice for a particular use case is identified under the heading "Applicable to Use Case."

		Applicable to Use Case			
Threat	**Best Practice**	Baseline Case	Development and Management of Containerized Applications	Containerized Services that Transmit or Process Account Data	Containerization in a Mixed Scope Environment
1. Authentication					
1.1 Unauthenticated access to APIs is provided by the container orchestration tool, allowing unauthorized modification of workloads.	**a.** All access to orchestration tools components and supporting services—for example, monitoring—from users or other services should be configured to require authentication and individual accountability.	**x**			
1.2 Generic administrator accounts are in place for container orchestration tool management. The use of these accounts would prevent the non-repudiation of individuals with administrator account access.	**a.** All user credentials used to authenticate to the orchestration should be tied to specific individuals. Generic credentials should not be used. When a default account is present and cannot be deleted, changing the default password to a strong unique password and then disabling the account will prevent a malicious individual from re-enabling the account and gaining access with the default password.	**x**			

Threat	Best Practice	Applicable to Use Case			
		Baseline Case	Development and Management of Containerized Applications	Containerized Services that Transmit or Process Account Data	Containerization in a Mixed Scope Environment
1.3 Credentials, such as client certificates, do not provide for revocation. Lost credentials present a risk of unauthorized access to cluster APIs.	**a.** All credentials used by the orchestration system should be revokable.	**X**			
1.4 Credentials used to access administrative accounts for either containers or container orchestration tools are stored insecurely, leading to unauthorized access to containers or sensitive data.	**a.** Authentication mechanisms used by the orchestration system should store credentials in a properly secured datastore.	**X**			
1.5 Availability of automatic credentials for any workloads running in the cluster. These credentials are susceptible to abuse, particularly if given excessive rights.	**a.** Credentials for the orchestration system should only be provided to services running in the cluster where explicitly required.	**X**			
	b. Service accounts should be configured for least privilege. The level of rights they will have is dependent on how the cluster RBAC is configured.	**X**			
1.6 Static credentials—i.e., passwords—used by administrators or service accounts are susceptible to credential stuffing, phishing, keystroke logging, local discovery, extortion, password spray, and brute force attacks.	**a.** Interactive users accessing container orchestration APIs should use multi-factor authentication (MFA).	**X**			
2. Authorization					
2.1 Excessive access rights to the container orchestration API could allow users to modify workloads without authorization.	**a**. Access granted to orchestration systems for users or services should be on a least privilege basis. Blanket administrative access should not be used.	**X**		**X**	

Threat	Best Practice	Applicable to Use Case			
		Baseline Case	Development and Management of Containerized Applications	Containerized Services that Transmit or Process Account Data	Containerization in a Mixed Scope Environment
2.2 Excessive access rights to the container orchestration tools may be provided through the use of hard-coded access groups.	**a**. All access granted to the orchestration tool should be capable of modification.	**X**		**X**	
	b. Access groups should not be hard-coded.	**X**			
2.3 Accounts may accumulate permissions without documented approvals.	**a.** Use manual and automated means to regularly audit implemented permissions.			**X**	
3. Workload Security					
3.1 Access to shared resources on the underlying host permits container breakouts to occur, compromising the security of shared resources.	**a.** Workloads running in the orchestration system should be configured to prevent access to the underlying cluster nodes by default. Where granted, any access to resources provided by the nodes should be provided on a least privilege basis, and the use of "privileged" mode containers should be specifically avoided.	**X**		**X**	
3.2 The use of non-specific versions of container images could facilitate a supply chain attack where a malicious version of the image is pushed to a registry by an attacker.	**a.** Workload definitions/manifests should target specific known versions of any container images. This should be done via a reliable mechanism checking the cryptographic signatures of images. If signatures are not available, message-digests should be used.	**X**	**X**		
3.3 Containers retrieved from untrusted sources may contain malware or exploitable vulnerabilities.	**a.** All container images running in the cluster should come from trusted sources.	**X**			

Threat	Best Practice	Applicable to Use Case			
		Baseline Case	Development and Management of Containerized Applications	Containerized Services that Transmit or Process Account Data	Containerization in a Mixed Scope Environment
4. Network Security					
4.1 Container technologies with container networks that do not support network segmentation or restriction allow unauthorized network access between containers.	**a**. Container orchestration tool networks should be configured on a default deny basis, with access explicitly required only for the operation of the applications being allowed.	**X**			**X**
4.2 Access from the container or other networks to the orchestration component and administrative APIs could allow privilege escalation attacks.	**a.** Access to orchestration system components and other administrative APIs should be restricted using an explicit allow-list of IP addresses.	**X**			**X**
4.3 Unencrypted traffic with management APIs is allowed as a default setting, allowing packet sniffing or spoofing attacks.	**a.** All traffic with orchestration system components APIs should be over encrypted connections, ensuring encryption key rotation meets PCI key and secret requirements.	**X**			**X**
5. PKI					
5.1 Inability of some container orchestration tool products to support revocation of certificates may lead to misuse of a stolen or lost certificate by attackers.	**a.** Where revocation of certificates is not supported, certificate-based authentication should not be used.	**X**			
	b. Rotate certificates as required by PCI or customer policies or if any containers are compromised.	**X**			

Threat	Best Practice	Applicable to Use Case			
		Baseline Case	Development and Management of Containerized Applications	Containerized Services that Transmit or Process Account Data	Containerization in a Mixed Scope Environment
5.2 PKI and Certificate Authority services integrated within container orchestration tools may not provide sufficient security outside of the container orchestration tool environment, which could lead to exploitation of other services that attempt to use this chain of trust.	**a.** The certificates issued by orchestration tools should not be trusted outside of the container orchestrator environment, as the container orchestrator's Certificate Authority private key can have weaker protection than other enterprise PKI trust chains.	**x**			
6. Secrets Management					
6.1 Inappropriately stored secrets, including credentials, provided through the container orchestration tool, could be leaked to unauthorized users or attackers with some level of access to the environment.	**a.** All secrets needed for the operation of applications hosted on the orchestration platform should be held in encrypted dedicated secrets management systems.	**x**			
6.2 Secrets stored without version control could lead to an outage if a compromise occurs and there is a requirement to rotate them quickly.	**a.** Apply version control for secrets, so it is easy to refresh or revoke it in case of a compromise.	**x**			
7. Container Orchestration Tool Auditing					
7.1 Existing inventory management and logging solutions may not suffice due to the ephemeral nature of containers and container orchestration tools integration.	**a.** Access to the orchestration system API(s) should be audited and monitored for indications of unauthorized access. Audit logs should be securely stored on a centralized system.	**x**		**x**	

Threat	Best Practice	Applicable to Use Case			
		Baseline Case	Development and Management of Containerized Applications	Containerized Services that Transmit or Process Account Data	Containerization in a Mixed Scope Environment
8. Container Monitoring					
8.1 Local logging solutions will not allow for appropriate correlation of security events where containers are regularly destroyed.	**a.** Centralized logging of container activity should be implemented and allow for correlation of events across instances of the same container.	**X**	**X**		
8.2 Without appropriate detection facilities, the ephemeral nature of containers may allow attackers to execute attacks unnoticed.	**a.** Controls should be implemented to detect the adding and execution of new binaries and unauthorized modification of container files to running containers.	**X**		**X**	
9. Container Runtime Security					
9.1 The default security posture of Linux process-based containers provides a large attack surface using a shared Linux kernel. Without hardening, it may be susceptible to exploits that allow for container escape.	**a.** Where high-risk workloads are identified, consideration should be given to using either container runtimes that provide hypervisor-level isolation for the workload or dedicated security sandboxes.				**X**
9.2 Windows process-based containers do not provide a security barrier (per Microsoft's guidance) allowing for possible container break-out.	**a.** Where Windows containers are used to run application containers, Hyper-V isolation should be deployed in-line with Microsoft's security guidance.			**X**	**X**
10. Patching					
10.1 Outdated container orchestration tool components can be vulnerable to exploits that allow for the compromise of the installed cluster or workloads.	**a.** All container orchestration tools should be supported and receive regular security patches, either from the core project or back-ported by the orchestration system vendor.	**X**			

		Applicable to Use Case			
Threat	**Best Practice**	Baseline Case	Development and Management of Containerized Applications	Containerized Services that Transmit or Process Account Data	Containerization in a Mixed Scope Environment
10.2 Vulnerabilities present on container orchestration tool hosts (commonly Linux VMs) will allow for compromise of container orchestration tools and other components.	**a.** Host operating system of all the nodes that are part of a cluster controlled by a container orchestration tool should be patched and kept up to date. With the ability to reschedule workloads dynamically, each node can be patched one at a time, without a maintenance window.	x			
10.3 As container orchestration tools commonly run as containers in the clusters, any container with vulnerabilities may allow compromise of container orchestration tools.	**a.** All container images used for applications running in the cluster should be regularly scanned for vulnerabilities, patches should be regularly applied, and the patched images redeployed to the cluster.	x	x		
11. Resource Management					
11.1 A compromised container could disrupt the operation of applications due to excessive use of shared resources.	**a.** All workloads running via a container orchestration system should have defined resource limits to reduce the risk of "noisy neighbors" causing availability issues with workloads in the same cluster.				x
12. Container Image Building					
12.1 Container base images downloaded from untrusted sources, or which contain unnecessary packages, increase the risk of supply chain attacks.	**a.** Application container images should be built from trusted, up-to-date minimal base images.		x		
12.2 Base images downloaded from external container image registries can introduce malware, backdoors, and vulnerabilities.	**a.** A set of common base container images should be maintained in a container registry that is under the entity's control.		x		

Threat	Best Practice	Applicable to Use Case			
		Baseline Case	Development and Management of Containerized Applications	Containerized Services that Transmit or Process Account Data	Containerization in a Mixed Scope Environment
12.3 The default position of Linux containers, which is to run as root, could increase the risk of a container breakout.	**a.** Container images should be built to run as a standard (non-root) user.		**X**		
12.4 Application secrets—i.e., cloud API credentials—embedded in container images can facilitate unauthorized access.	**a.** Secrets should never be included in application images. Where secrets are required during the building of an image (for example to provide credentials for accessing source code—this process should leverage container builder techniques to ensure that the secret will not be present in the final image.		**X**		
13. Registry					
13.1 Unauthorized modification of an organization's container images could allow an attacker to place malicious software into the production container environment.	**a.** Access to container registries managed by the organization should be controlled.	**X**	**X**		
	b. Rights to modify or replace images should be limited to authorized individuals.	**X**	**X**		
13.2 A lack of segregation between production and non-production container registries may result in insecure images deployed to the production environment.	**a.** Consider using two registries, one for production or business-critical workloads and one for development/test purposes, to assist in preventing image sprawl and the opportunity for an unmaintained or vulnerable image being accidentally pulled into a production cluster.		**X**		
13.3 Vulnerabilities can be present in base images, regardless of the source of the images, via misconfiguration and other methods.	**a.** If available, registries should regularly scan images and prevent vulnerable images from being deployed to container runtime environments.	**X**	**X**		

Threat	Best Practice	Applicable to Use Case: Baseline Case	Applicable to Use Case: Development and Management of Containerized Applications	Applicable to Use Case: Containerized Services that Transmit or Process Account Data	Applicable to Use Case: Containerization in a Mixed Scope Environment
13.4 Known good images can be maliciously or inadvertently substituted or modified and deployed to container runtime environments.	**a.** Registries should be configured to integrate with the image build processes such that only signed images from authorized build pipelines are available for deployment to container runtime environments.	**X**	**X**	**X**	
14. Version Management					
14.1 Without proper control and versioning of container orchestration configuration files, it may be possible for an attacker to make an unauthorized modification to an environment's setup.	**a.** Version control should be used to manage all non-secret configuration files.			**X**	
	b. Related objects should be grouped into a single file.			**X**	
	c. Labels should be used to semantically identify objects.			**X**	
15. Configuration Management					
15.1 Container orchestration tools may be misconfigured and introduce security vulnerabilities.	**a.** All configurations and container images should be tested in a production-like environment prior to deployment.			**X**	**X**

Threat	Best Practice	Applicable to Use Case: Baseline Case	Applicable to Use Case: Development and Management of Containerized Applications	Applicable to Use Case: Containerized Services that Transmit or Process Account Data	Applicable to Use Case: Containerization in a Mixed Scope Environment
	b. Configuration standards that address all known security vulnerabilities and are consistent with industry-accepted hardening standards and vendor security guidance should be developed for all system components, including container orchestration tools. i. Address all known security vulnerabilities. ii. Be consistent with industry-accepted system hardening standards or vendor hardening recommendations. iii. Be updated as new vulnerability issues are identified.	x			
16. Segmentation					
16.1 Unless an orchestration system is specifically designed for secure multi-tenancy, a shared mixed-security environment may allow attackers to move from a low-security to a high-security environment.	**a.** Where practical, higher security components should be placed on dedicated clusters. Where this is not possible, care should be taken to ensure complete segregation between workloads of different security levels.				x
16.2 Placing critical systems on the same nodes as general application containers may allow attackers to disrupt the security of the cluster through the use of shared resources on the container cluster node.	**a.** Critical systems should run on dedicated nodes in any container orchestration cluster.				x

		Applicable to Use Case			
Threat	**Best Practice**	Baseline Case	Development and Management of Containerized Applications	Containerized Services that Transmit or Process Account Data	Containerization in a Mixed Scope Environment
16.3 Placing workloads with different security requirements on the same cluster nodes may allow attackers to gain unauthorized access to high security environments via breakout to the underlying node.	**a.** Split cluster node pools should be enforced such that a cluster user of the low-security applications cannot schedule workloads to the high-security nodes.				**x**
16.4 Modification of shared cluster resources by users with access to individual applications could result in unauthorized access to sensitive shared resources.	**a.** Workloads and users who manage individual applications running under the orchestration system should not have the rights to modify shared cluster resources, or any resources used by another application.				**x**

3.2 Example Use Cases

The following example use cases illustrate some possible threat scenarios and the application of best practices to address the threats. Each example case provides:

- A description of the use case
- A graphic representation of a possible implementation
- A description of a threat, and
- The corresponding best practices are taken from Section 3.1.

These example best practices are not an all-inclusive list. It is possible that different best practices could be applied to address the described threat scenario.

3.2.1 Baseline Use Case

3.2.1.1 Description

A common baseline use case includes the use of a container orchestration system to deploy and manage the lifecycle of production workloads in a payment environment. In such cases, multiple users may have access to deploy workloads to their respective namespaces and several applications are run on a group of underlying cluster nodes.

3.2.1.2 Graphic Representation of the Use Case

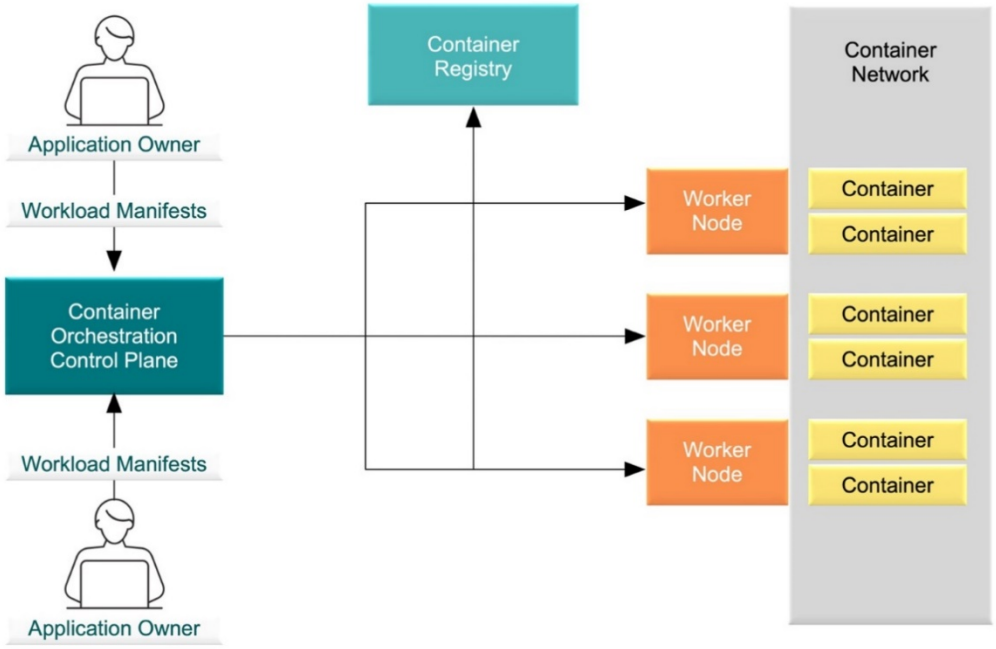

Figure 5: Baseline Use Case

3.2.1.3 Example Threat Scenario

A common configuration for cloud-based container orchestration systems exposes the API server directly on the Internet. If anonymous access is allowed to those API servers, or if attackers can make use of an unpatched security vulnerability in the software, they can compromise not only the applications running in the cluster, but also any credentials stored by those applications for use in other parts of the environment. As container orchestration APIs effectively allow for remote command execution on cluster nodes, an attacker who gains access to the orchestration system API can often gain privileged access to those cluster nodes.

This threat scenario has been exploited in the real-world on several occasions, and it is a well-known attack path.

Example implementation of selected best practices:

Best Practice	Result of Best Practice to Address the Security Threat
1.1.a All access to orchestration tools components and supporting services—for example, monitoring—from users or other services should be configured to require authentication and individual accountability.	Applying this best practice reduces the risk of compromise of the orchestration system or API server by an unauthorized individual using either anonymous access or authorized access without accountability.
4.2.a Access to orchestration system components and other administrative APIs should be restricted using an explicit allow-list of IP addresses.	Restricting access to the APIs to a limited set of known IP addresses reduces the attack surface of the system and prevents trivial enumeration of valid systems which could then be attacked.
7.1.a Access to the orchestration system API(s) should be audited and monitored for indications of unauthorized access. Audit logs should be securely stored on a centralized system.	Detection of unusual access activity through monitoring and logging of system API access provides an opportunity to both address an ongoing attack and to provide evidence required for a forensic investigation.
10.1.a All container orchestration tools should be supported and receive regular security patches, either from the core project or back-ported by the orchestration system vendor.	Ensuring that orchestration system components are receiving regular security updates will reduce the risk of an attacker gaining network level access to the API server and exploiting a vulnerability to compromise the service.

3.2.2 Development and Management of Containerized Applications

3.2.2.1 Description

Creating and managing a container-based workflow for application development and deployment involves several steps, including the initial creation of the container images to be used by the application, the flow of the images as artifacts through the companies CI/CD pipeline, secure storage of the images in a container registry, and their ongoing management and updating.

Phases of the deployment process include:

- Initial development targets application deployment using a container based on a common base image. The container image is used by Continuous Integration processes in the SDLC.
- The container image is placed into a container registry during testing and deployment.
- The container image is deployed into a production environment to be managed by a container orchestration system.

3.2.2.2 Graphic Representation of the Use Case

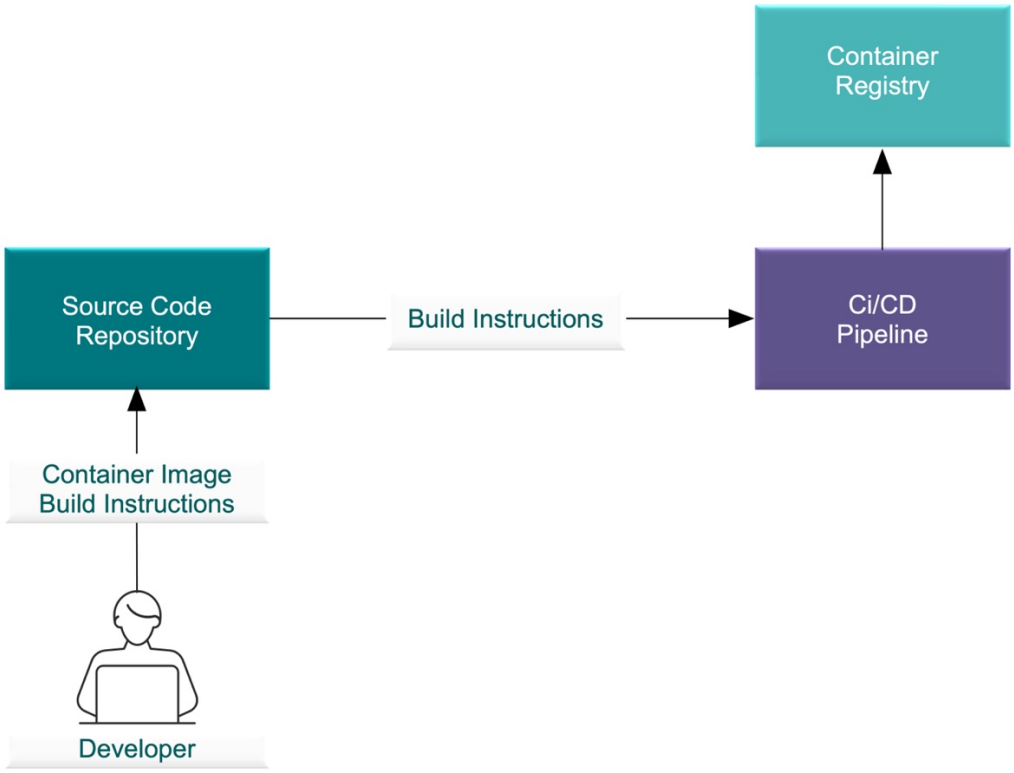

Figure 6: Container Build Process

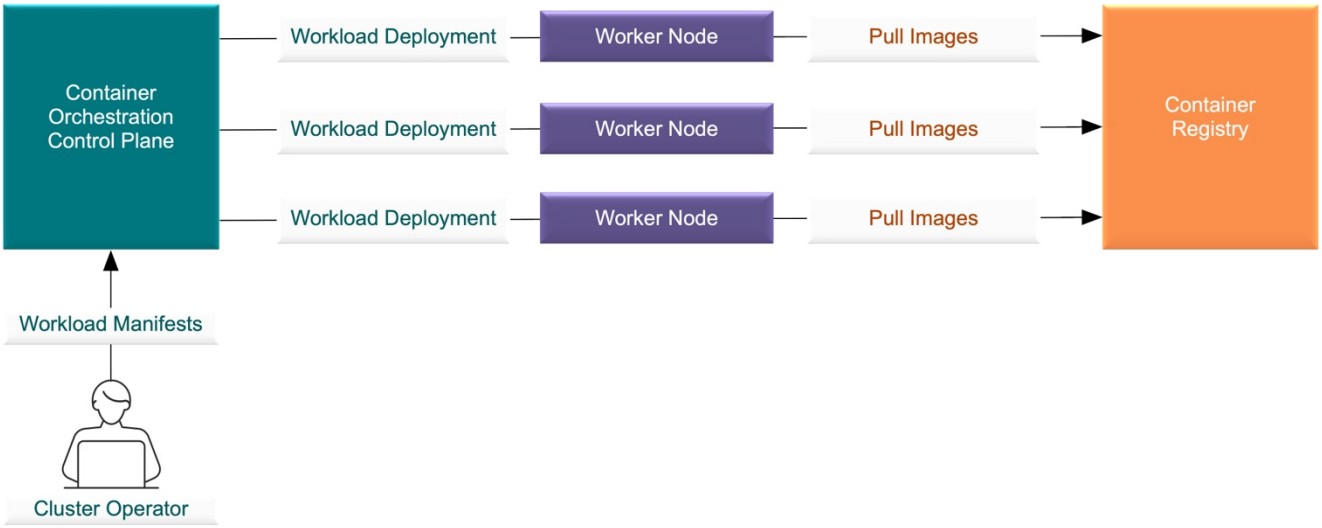

Figure 7: Container Use Process

3.2.2.3 Example Threat Scenario

When building container images, a common requirement is to use secrets—for example, credentials or API keys—to access private data stores to retrieve information. If those secrets are embedded in the resulting container images, attackers can extract the secrets and gain unauthorized access to resources including source code repositories, CI/CD systems, or even container orchestration APIs.

Example implementation of selected best practices:

Best Practice	Result of Best Practice to Address the Security Threat
6.1.a All secrets needed for the operation of applications hosted on the orchestration platform should be held in encrypted dedicated secrets management systems.	Where secrets are required for running containers, a dedicated secrets management system is employed to ensure that secrets are securely encrypted and made available to only the containers which require them. These systems can determine which containers require access to a specific secret and then inject those secrets into the running container as a mounted file.
12.4.a Secrets should not be included in application images. Where secrets are required during the building of an image (for example to provide credentials for accessing source code), this process should leverage container builder techniques to ensure that the secret will not be present in the final image.	If an attacker can access source code repositories, CI/CD systems, or the container API, proper management of secrets—for example, not being included in application images, including binary files—prevents these secrets from being used to access additional resources. Ensuring that secrets are not embedded in images can be achieved by using techniques such as multi-stage builds. Here separation between source code compilation and the final container image is achieved by having multiple build processes, and only copying compiled application programs and necessary configuration files to the final stage.

3.2.3 Use of Containerized Services that Process/Transmit Payment Card Account Data

3.2.3.1 Description

Implementation of containerized services to process and transmit payment card account data while conducting payment transactions, including capturing, authorizing, settlement, and chargeback. This use case applies to:

- **Example 1 – See Figure 8 below:** Any container workload that receives payment card account data as an input or provides this data as an output to both a process and the container host system providing the container runtime environment:
  - Application services performing business logic
  - Combined application and web presentation tier platforms
- **Example 2 – See Figure 9 below:** Container workloads and infrastructure responsible for connecting other workloads where CHD is present:
  - Container host servers that provide container runtime services
  - Container and container orchestration infrastructure providing network services such as basic IP packet routing or network proxy services
  - Application and/or network load-balancing containers such as HAProxy
  - Containers responsible for providing information security services

3.2.3.2 Graphic Representation of the Use Case

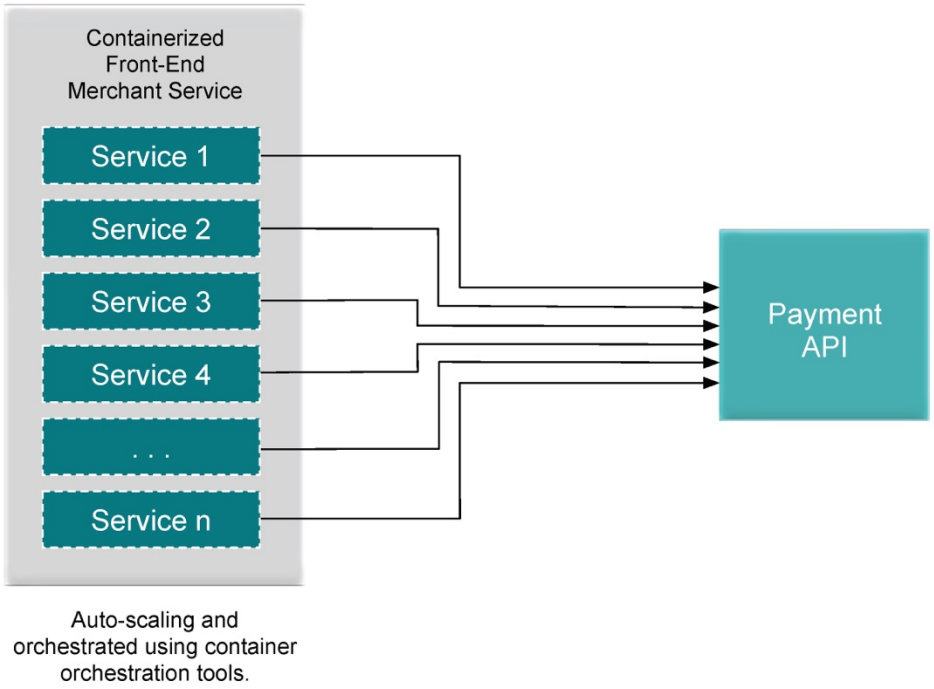

Figure 8: Example 1 – Multiple Containerized Services Accessing a Payment API

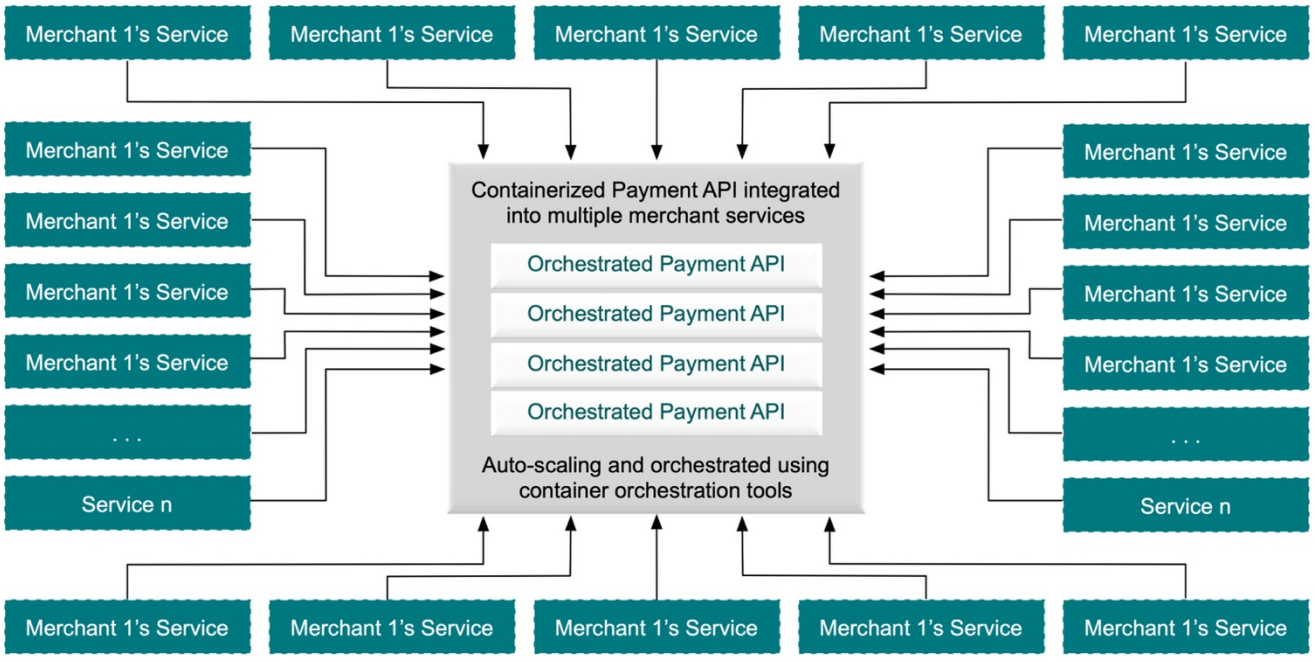

Figure 9: Example 2 – Multiple Merchant Services Integrated into an Orchestrated, Autoscaling, Containerized Payment API

3.2.3.3 Example Threat Scenario

When developing sensitive applications that could impact the security of payment card account data and that operate in a shared container cluster, attacks can result from excessive permissions to the container orchestration APIs. A user with valid credentials may be able to escalate their rights to interact with sensitive applications and, in doing so, gain unauthorized access to payment card account data, either by directly executing commands in the container (via the orchestration API) or by gaining access to shared cluster nodes hosting multiple workloads. With access to the shared cluster node, it may be possible to access payment card account data from any workload scheduled to that system.

Example implementation of selected best practices:

Best Practice	Result of Best Practice to Address the Security Threat
2.1.a Access granted to orchestration systems for users or services should be on a least privilege basis. Blanket administrative access should not be used.	The inappropriate use of administrator access, providing unnecessary rights to a user or service, provides an attacker with additional resources with which to mount the attack. Rights to interact with running containers and to schedule new containers to the cluster should be carefully controlled to reduce the risk of unauthorized access to these workloads. The availability of the minimum privileges required to perform the required tasks reduces the opportunity for an attacker to leverage provided privileges to inappropriately access sensitive information, including cardholder data stored, processed, or transmitted by the container services.
3.1.a Workloads running in the orchestration system should be configured to prevent access to the underlying cluster nodes by default. Where granted, any access to resources provided by the nodes should be provided on a least privilege basis, and the use of “privileged” mode containers should be specifically avoided.	If an attacker can access source code repositories, CI/CD systems, or the container API, proper management of secrets—for example, not being included in application images, including binary files—prevents these secrets from being used to access additional resources. Ensuring that secrets are not embedded in images can be achieved by using techniques such as multi-stage builds. Here separation between source code compilation and the final container image is achieved by having multiple build processes, and by only copying compiled application programs and necessary configuration files to the final stage.
8.2.a Controls should be implemented to detect both the adding and execution of new binaries and any unauthorized modification of container files to running containers.	Patterns of access which are unexpected—for example starting a shell in a running container—can be detected and alerts sent to security teams to trigger an investigation.

3.2.4 Use of Containerization for a Mix of Services with Different Security Levels

3.2.4.1 Description

Container orchestration systems provide the option to have different workloads running in a single cluster. It would be technically possible to run containerized applications requiring different security levels in the same cluster.

3.2.4.2 Graphic Representation of the Use Case

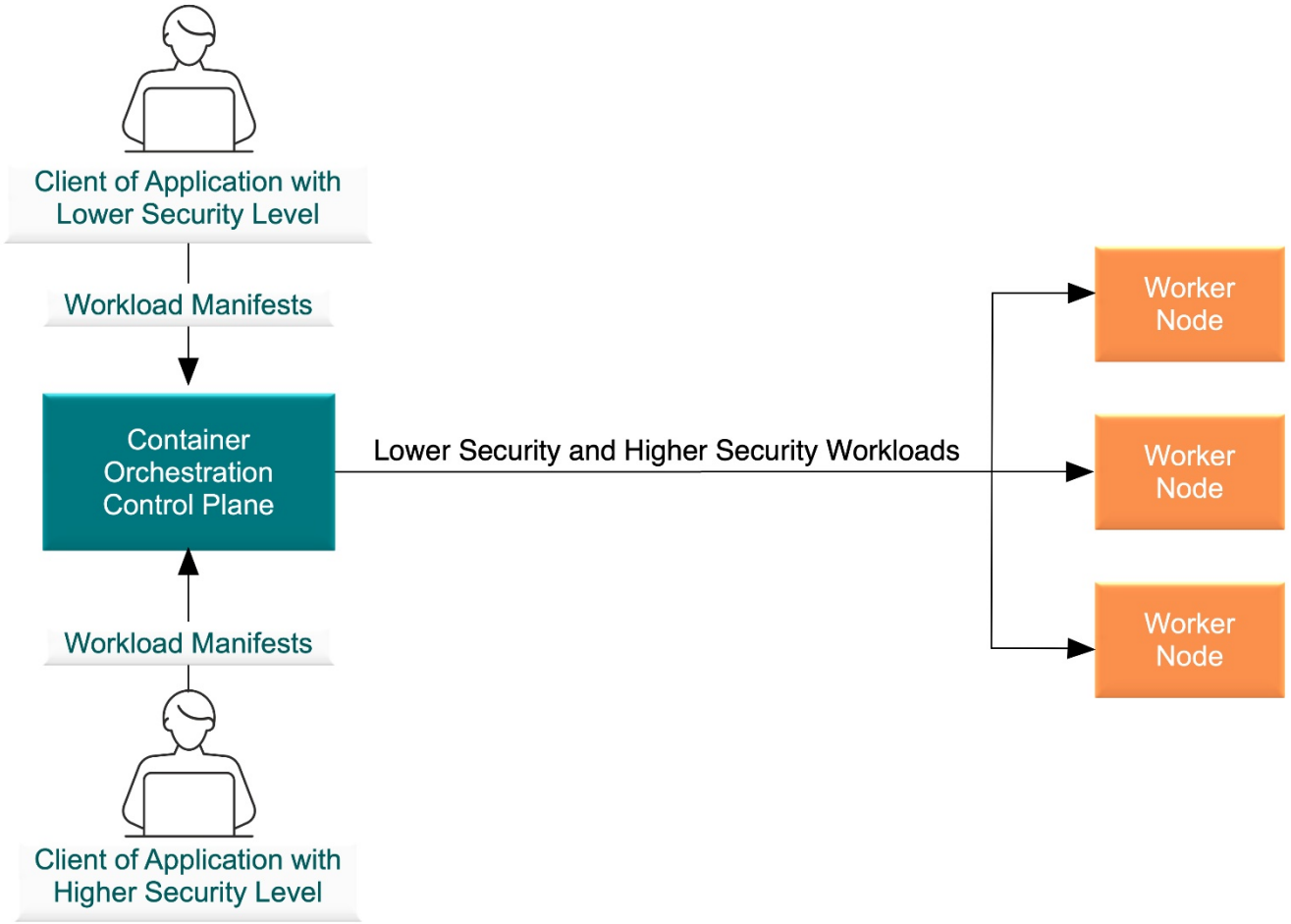

Figure 10: Multi-tenant Cluster hosting a mix of workloads requiring different security levels

3.2.4.3 Example Threat Scenario

In a multi-tenant cluster, a user with access to a single application might gain unauthorized access to other applications through a lack of workload isolation. Such isolation can prevent vulnerabilities in one workload from impacting others where workloads may share computing, networking, or other container orchestration tool resources. Many container orchestration tools provide a default flat local

network for all containers, making it easy for an attacker to target those services. The orchestrator may also offer service discovery features that make it easy for attackers to find which applications to target. Additionally, attackers may try to break out to underlying cluster nodes. There have been several container breakout vulnerabilities found which could facilitate this attack so that an unpatched container runtime may be exploitable.

Example implementation of selected best practices:

Best Practice	Result of Best Practice to Address the Security Threat
4.1.a Container orchestration tool networks should be configured on a default deny basis, with only access explicitly required for the operation of the applications being allowed.	Deploying a default deny network policy on a multi-tenant cluster is a key control to reduce the risk of network attacks across the container network. Workloads should only be able to communicate with white-listed services both inside the container network and externally to reduce the risk of compromise.
4.2.a Access to orchestration system components and other administrative APIs should be restricted using an explicit allow-list of IP addresses.	Access to orchestration system components and other administrative APIs should be restricted using an explicit allow-list of IP addresses.
16.1.a Where practical, higher security components should be placed on dedicated clusters. Where this is not possible, care should be taken to ensure complete segregation between workloads of different security levels.	Though not always practical or possible, the placement of containerized applications with different security levels on different dedicated clusters restricts the susceptibility of attack from a lower security or untrusted application.
16.3.a Split cluster node pools should be enforced such that a cluster user of the low-security applications cannot schedule workloads to the high-security nodes.	Where applications of different security levels are deployed to a single cluster, dedicated node pools should be provided for each environment and administrative controls put in place to prevent inappropriate deployments to a given node pool. This reduces the impact of the attacker breaking out to the underlying node.

Appendix A: Other PCI SSC Reference Documents

The following resources are also available from the Document Library on the PCI Security Standards website: https://www.pcisecuritystandards.org/document_library:

- PCI DSS
- *Information Supplement: PCI SSC Cloud Computing Guidelines*
- *Information Supplement: PCI DSS Virtualization Guidelines*

Additionally, common terms in the payment card industry used within this document are listed in the *Payment Card Industry (PCI) Data Security Standard Glossary, Abbreviations and Acronyms*: https://www.pcisecuritystandards.org/pci_security/glossary

Appendix B: Other Non-PCI SSC Reference Documents

- CIS Benchmarks: https://media.defense.gov/2021/Aug/03/2002820425/-1/-1/1/CTR_KUBERNETES%20HARDENING%20GUIDANCE.PDF
- CSA: https://cloudsecurityalliance.org/artifacts/best-practices-for-implementing-a-secure-application-container-architecture/
- Docker: https://docs.docker.com/engine/security/
- Kubernetes: https://kubernetes.io/docs/concepts/security/
- NIST: https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-190.pdf

Acknowledgments

PCI SSC would like to acknowledge the contribution of the Best Practices for Container Orchestration Special Interest Group (SIG) in the preparation of this document. The Best Practices for Container Orchestration SIG consists of representatives from the following organizations:

1stSecureIT LLC (dba GM SECTEC)
Acumera, Inc.
Agio, LLC
Ally Financial Inc.
Amazon
Aqua Security
Armor Defense Inc.
AT&T Consulting services
AWS Security Assurance Services LLC
AXENIC LIMITED
Banco PAN S.A.
Bank of New Zealand
Barclaycard
BDO USA, LLC
BSI Cybersecurity and Information Resilience Ireland Limited, dba BSI Group
Cadence Assurance, an affiliate of The Cadence Group
Center for Internet Security
Certus Cybersecurity Solutions, LLC
Coalfire Systems
Computer Services Inc
ControlGap
Crowe Horwath LLP
CVS Caremark
Fiserv Solutions Inc.
Foregenix
Fujitsu Services Ltd
Global Payments Direct Inc.
PagSeguro Internet S/A
Paladion Networks Private LTD
Panacea InfoSec (P) Ltd.
Pine Labs Pvt. Ltd
Privity Systems Inc.
Protiviti
Qualys
Resources Connection LLC
Right Time Limited
Risk Associates Europe Ltd.
RSM US LLP (formerly McGladrey LLP)
Schellman & Company, LLC
Schwarz IT KG
Secure Logic pty Ltd
SecureTrust, Inc.
Securisea
Security Compass
Sentor Managed Security Services
SERVADUS
Shopify
SIX Payment Services Ltd
Southwest Airlines
Square
SRC Security Research & Consulting GmbH
Stripe, Inc.
Synchrony Financial
Sysxnet Limited DBA Sysnet Global Solutions
TELUS Security Solutions
Thales

Information Exchange Inc.
IQ Information Quality
JP Morgan Chase
Kirkpatrick Price, Inc. dba Raven Eye
Lattimore, Black Morgan and Cain, PC
Microsoft
Nationwide Mutual Insurance Company
NCC Services
Online Enterprises DBA Online Business Systems
Oracle Corporation
Thales Transport & Security (Hong Kong) Ltd.
TRG
U.S. Bancorp
usd AG
Verifone
Verizon
Virtual Inc.
VMware, INC.
Wells Fargo

About the PCI Security Standards Council

The PCI Security Standards Council is an open global forum that is responsible for the development, management, education, and awareness of the PCI Data Security Standard (PCI DSS) and other standards that increase payment data security. Created in 2006 by the founding payment card brands American Express, Discover Financial Services, JCB International, Mastercard, and Visa Inc., the Council has more than 700 Participating Organizations representing merchants, banks, processors, and vendors worldwide. To learn more about playing a part in securing payment card data globally, please visit: pcisecuritystandards.org.