WHITE PAPER

SECURITY BEST PRACTICES FOR THE DIAMANTI BARE-METAL CONTAINER PLATFORM

DIAMANTI

TABLE OF CONTENTS

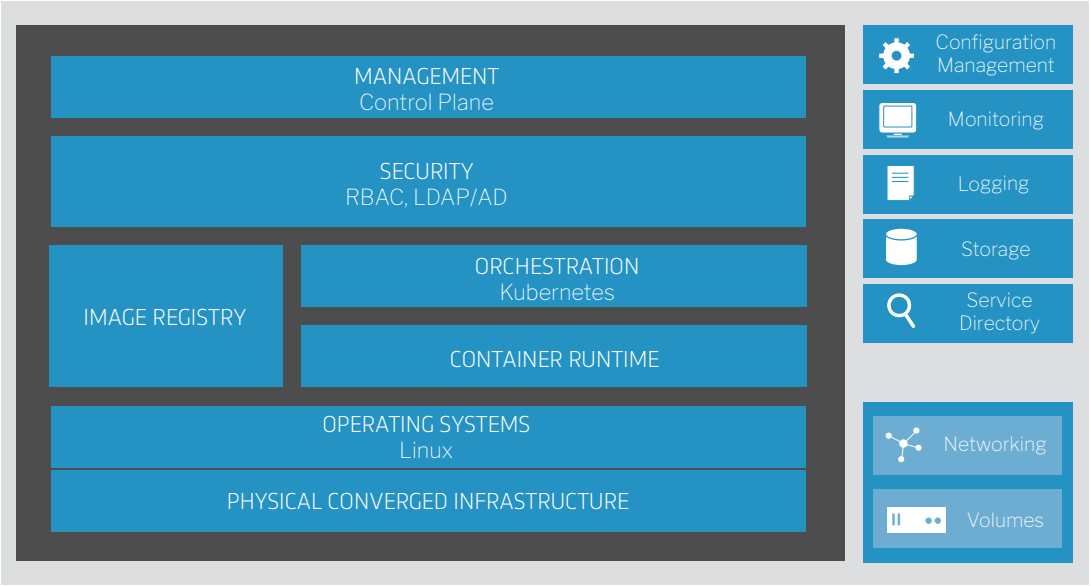
INTRODUCTION

Containers have significantly changed the way organizations develop and deploy software applications. Security considerations for containers are also dramatically different; the environments they comprise represent a different type of attack surface that requires new approaches and technologies to protect throughout the entire container lifecycle.

This guide provides a series of best practices for securing a container environment based on the Diamanti D10 bare-metal container stack running Docker and Kubernetes. It offers up an analysis of key security characteristics and requirements across each layer of the stack, and highlights the Diamanti product features, recommended practices, and third-party technologies that serve to protect containers from threats.

THE DIAMANTI BARE-METAL CONTAINER STACK

Diamanti's D10 bare-metal container appliance is the foundation for a full container stack supporting enterprise container environments from development to production.



The primary objective should be to minimize security risk at each stack layer, shrinking the overall attack surface as much as possible. This involves a variety of methods that apply to configuration, user access, OS and container image vulnerability management, and dedicated runtime protection for containers.

To form a D10 cluster, the Diamanti OS aggregates resources from individual member nodes and groups the nodes into a cluster. After a Diamanti cluster is formed, a virtual IP address (VIP) is assigned to the cluster, which provides access to a single point of management. The following Virtual IP network ports must be open to manage the cluster:

a. VIP:53
b. VIP:TCP:80
c. VIP:TCP:443
d. VIP:TCP:6443
e. Mgmt IPs:TCP:22
f. DNS:53

AUTHENTICATION AND AUTHORIZED ACCESS TO THE DIAMANTI D10 APPLIANCE

The logical starting point for securing the Diamanti bare-metal container stack is to set up authorized access to the D10 appliance for a restricted set of users and administrators, via Diamanti OS. The Diamanti D10 should already be installed in your data center, and connected to your existing network.

MANAGING CERTIFICATES

Diamanti OS has multiple components within its framework that use REST-based HTTPS communication leveraging encryption via TLS certificates. The Diamanti installer configures these certificates during creation of the cluster. The primary components that generate this traffic are as follows:

- Kubernetes API server and controllers
- etcd (distributed key value store for Kubernetes)
- nodes (node-to-node and node-to-Kubernetes API server, etc.)
- container image registry

CONFIGURING CUSTOM CERTIFICATES

You can configure custom serving certificates for the public host names of the API server and web console during initial installation or when re-deploying certificates. You can also use a custom certificate authority.

AUTHORIZED ACCESS

CONTROLLING ACCESS USING LDAP AND ACTIVE DIRECTORY

Diamanti OS offers API access control via authentication and authorization, leveraging an external LDAP or Active Directory server. The following is an example of LDAP authentication:

```
$ dctl user auth-server create myauth --server ldap.example.com --port 389
--ldap-base 'dc=example,dc=com' --ldap-binddn 'cn=mybinddn'
--ldap-bindpassword 'mybindpassword' --ldap-filter '(sAMAccountName=%s)'

Name           : myauth
Server         : ldap.example.com
Port           : 389
LDAP Base: cd=example.com
Reachable: true

$ dctl user create myuser --group-list cluster-admin
NAME           BUILT-IN         LOCAL-AUTH       GROUP-LIST
myuser         false            false            cluster-admin
```

MANAGING USER GROUPS WITH RBAC

Diamanti OS offers role-based access control (RBAC) to regulate access to resources within the environment. As part of this system, Diamanti OS employs the following related constructs:

Users

People who can perform tasks within the environment

Roles

Collections of functions that users can perform. Within the Diamanti software, these functions are defined as methods that can act on objects within the system. There is a fixed set of predefined roles that cannot be changed

Groups

Collections of users that assume the same role within the system. User groups define the privileges that users are assigned in the system.

Diamanti OS offers built-in assignable roles and groups for the purpose of regulating common administrative tasks. Admins can also create custom groups for specific activities, and define users within the system.

OBJECT	NAME	DESCRIPTION
User	admin	Belongs to the cluster-admin group
Group	cluster-admin	Has edit and view privileges for Diamanti resources including containers, volume claims, networks, nodes, performance tiers, and volumes. Note that this group has full privileges in Kubernetes.
	container-admin	Has edit privileges for containers in the default namespace and view privileges for Diamanti resources including networks, nodes, performance tiers, and volumes.
	user-admin	Has view and edit privileges for users, group, and authentication servers.

Cluster administrators generally perform the following tasks:

- Manage the cluster configuration
- Monitor container deployments
- Monitor hosts, volumes, and networks
- Monitor and manage common infrastructure operations, including those of Diamanti D10 appliances

The **`dctl user role list`** command displays the list of user roles in the Diamanti OS CLI:

```
$ dctl user role list
NAME                    BUILT-IN         SCOPE
allcontainer-edit       true             Cluster
allcontainer-view       true             Cluster
container-edit          true             Resource
container-view          true             Resource
network-edit            true             Cluster
network-view            true             Cluster
node-edit               true             Cluster
node-view               true             Cluster
perftier-edit           true             Cluster
perftier-view           true             Cluster
requiered               true             Cluster
user-edit               true             Cluster
user-view               true             Cluster
volume-edit             true             Cluster
volume-view             true             Cluster
volumeclaim-edit        true             Resource
```

SECURING THE CONTAINER HOST OS

The Diamanti D10 features CentOS (a Linux distribution) as the host operating system for running application containers. For the purpose of protecting hosts against container breaches, it is highly recommended to keep the operating system up to date by running the latest version, and to employ some of Linux's most commonly-used built-in security modules: SELinux and seccomp.

SELINUX

SELinux (Secure Linux) is a Linux kernel security module that enable enhanced control and isolation for containers, allowing administrators to enforce mandatory access controls (MAC) for every user, application, process, and file.

SECCOMP

Seccomp (secure computing mode) used to constrain the system-level capabilities containers are allocated at runtime. The Docker runtime engine includes default seccomp profiles that drop system calls that are unsafe and typically unnecessary for container operation. Additionally, custom profiles can be created and passed to container runtimes to further limit their capabilities. At a minimum, organizations should ensure that containers are run with the default profiles provided by their runtime and should consider using additional profiles for high-risk applications.

Source: Application Container Security Guide (NIST Special Publication 800-190)

SECURE CONFIGURATION OF DOCKER AND KUBERNETES

With over 200 built-in checks available through CIS benchmarks for Docker and Kubernetes, organizations can make the container runtime engine and orchestrator more secure, and automatically enforce compliance policies across the container lifecycle. You can use built-in compliance templates for standards like PCI-DSS, GDPR, HIPAA, and NIST SP 800-190, or import your own custom SCAP policies.

ISOLATING APPLICATION CONTAINERS AND INFRASTRUCTURE RESOURCES

To further strengthen the overall security posture of your container stack, the isolation properties of containers should extend to the network and storage resources that containerized applications consume.

NETWORK ISOLATION AND SEGMENTATION

ISOLATION VIA NAMESPACES

The Linux kernel's network namespaces feature enables isolation of container pod networks. Each namespace will have its own private network stack, which includes a set of IP addresses, its own routing table, socket listing, connection tracking table, firewall, and other network-related resources.

This feature is commonly used to isolate developer, test and production environments from one another within a cluster.

Each application running in its own namespace can have its own RBAC policies.

NETWORK SEGMENTATION

Via Diamanti OS, you can segment network traffic on a single cluster to enable a multi-tenant cluster that isolates users, teams, applications, and environments. This is achieved by Diamanti's ability to configure VLANs using Layer-2 technology. Also, Kubernetes has built-in support for namespaces, so applications can be assigned their own name spaces.

SECURING CONTAINER STORAGE

Diamanti D10's persistent NVMe storage is provisioned and managed in a way that is makes it secure.

STORAGE AND SELINUX

The Diamanti D10 appliance leverages SELinux capabilities to secure the root of the mounted volume for non-privileged pods, making the mounted volume owned by and only visible to the container with which it is associated.

PERSISTENT VOLUMES AND PERSISTENT VOLUME CLAIM

The Diamanti software supports Persistent Volumes (PV) and Persistent Volume Claim (PVC). A Persistent Volume is storage that has been provisioned by an administrator and has a lifecycle independent of any individual pod that uses the PV.

A Persistent Volume Claim (PVC) is a request for storage by a user. PVCs consume PV resources, with a claim requesting a specific size and access mode. Volumes persist across container failures and restarts. PVCs can be controlled because PVC objects are included in namespaces in Kubernetes. As such, they can be subject to any RBAC rules defined by the user.

HARDENING CONTAINERS

PRIVILEGE RESTRICTION

As a critical first step in reducing the container attack surface, containers should be configured to run as non-root user processes. Dropping the default privilege level or creating containers with the least amount of privileges possible is highly recommended.

IMAGE VULNERABILITY ASSESSMENT

Another important practice in hardening containers is to scan container images for known vulnerabilities, typically documented in the Common Vulnerabilities and Exposures (CVE) database, which is currently maintained by Mitre Corporation.

Vulnerability scanning is a mainstay of container build/deploy phase security practices. A variety of container image vulnerability scanners are available today from Docker (Security Scanning), RedHat, Qualys, Tenable, Aqua Security, Twistlock, CoreOS (Clair), and Layered Insight.

In addition to offering image vulnerability assessment, several third-party build/deploy security tools can also prevent the deployment of vulnerable images across the environment with alerting and enforcement policies covering the entire CI/CD process as containers are built, shipped, and run on the Diamanti D10 appliance.

SECURING CONTAINERS DURING RUNTIME

The ephemeral and distributed nature of container environments presents security challenges that can't be solved with traditional enterprise security solutions. Detecting and responding to threats requires the ability to instrument and monitor container environments, surface Indicators of Compromise (IOCs) against a backdrop of normal application container behavior, and take policy-driven action.

There are a variety of container-native security solutions compatible with Docker and Kubernetes, and those should deploy and run on a Diamanti container stack. Popular solutions are available from vendors such as Twistlock, Aqua Security, Neuvector, StackRox, and Layered Insight.

SUMMARY AND KEY RECOMMENDATIONS

Securing a container environment is a significant challenge. Containers represent an entirely new type of attack surface, and reducing threat exposure requires that IT and security teams address multiple aspects at each layer of the container stack. Below are some general recommendations:

1. Only group containers with the same purpose, sensitivity, and threat posture on a single host OS kernel to allow for additional defense in depth
2. Operate according to the principle of 'least privilege' when configuring containers
3. Establish a secure configuration baseline for the Docker CE engine using CIS benchmarks
4. Leverage container-specific vulnerability management tools and processes for container images to harden them against known vulnerabilities
5. Use a dedicated solution for threat prevention, detection, and response that protects containers at runtime

ADDITIONAL REFERENCE MATERIAL

- NIST provides a thorough Application Container Security Guide, which offers up a wealth of detailed practices for securing containers
- CIS benchmarks for Docker CE