The Secure Software Container Pattern

MADIHA H. SYED, Florida Atlantic University
EDUARDO B. FERNANDEZ, Florida Atlantic University
PAULINA SILVA, Universidad Tecnica F. Santa Maria

Software containers have gained large popularity as portable lightweight virtualization components. They have enabled development approaches like DevOps by facilitating application deployment and distribution across computing environments. Containers offer operating-system-level virtualization where applications are executed in isolated environments sharing a host operating system (OS), binaries, and libraries with other containers. Containers offer low overhead and near native performance as compared to virtual machines (VMs). However, there are tradeoffs like flexibility and security when compared to VMs. This closer integration and interaction with their Host OS also results in an increased attack surface, host resources are exposed to the applications which can lead to security threats. The security of containers has improved since their inception but they still exhibit vulnerabilities that need to be addressed to enable their full adoption. We present a pattern for a Secure Software Container, which describes their security threats and possible defenses.

Categories and Subject Descriptors: **D.2.11 [Software Engineering]**: Software Architectures - Patterns

General Terms: Design

Additional Key Words and Phrases: software containers, security patterns, architecture patterns, security, virtualization, container ecosystem

ACM Reference Format:

Syed, M.H., Fernandez, E.B. and Silva, P. 2017. The Secure Software Container Pattern. HILLSIDE Proc. of Conf. on Pattern Lang. of Prog. PLoP (October 2017), 7 pages.

1. INTRODUCTION

A cloud-based computing system involves a variety of users and devices connected to it and provides multiple kinds of services. Typically, clouds provide three levels of service: Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Software as a Service (SaaS). Cloud computing has been using VMs as a virtualization solution but containers have gained significant popularity in recent times as a more portable and lightweight option. Software containers, although not new, have become very important to support convenient and low overhead applications. Container-based virtualization or operating-system-level virtualization runs applications in an execution environment where they share a host OS, binaries, and libraries with other containers, while isolating their execution environments. Containers facilitate DevOps processes by providing a means for application deployment and distribution across computing environments (Souza et al. 2015, White 2014).

In despite of their advantages, containers still have some shortcomings as compared to VMs when it comes to flexibility and security. We need to understand the security of containers in order to protect the systems which use them including applications and the host OS. We are studying containers as part of cloud ecosystems and in our previous work we have used architectural modeling and patterns to present a holistic and unified view of these systems (Fernandez et al. 2015, Fernandez et al. 2016a). An ecosystem is a collection of software systems, which are developed and co-evolve in the same environment. We have written patterns for VM Environment (VME) (Syed and Fernandez 2016), Software Containers (Syed and Fernandez 2015), and Container Manager (Syed and Fernandez 2017). Figure 1 is a pattern diagram (extended with UML notation) for a cloud ecosystem which shows the relationships between those patterns (Fernandez et al. 2016a). The Cloud Reference Architecture (RA) is the main pattern (hub) that defines this particular ecosystem. An RA is an abstract software architecture, based on one or more domains, without focusing on implementation aspects (Avgeriou 2003). Identified security threats can be controlled by adding security patterns to the Cloud RA, which results in the Cloud Security RA (SRA). The Container Environment

Author's address: Madiha H. Syed, email: msyed2014@fau.edu; Eduardo B. Fernandez, email: ed@cse.fau.edu; Dept. of Computer and Elect. Eng. and Computer Science Florida Atlantic University, Boca Raton, FL 33431, USA; Paulina Silva, Universidad Tecnica F. Santa Maria, Valparaiso, Chile

is part of the PaaS of the cloud. Detailed description of the remaining components can be found in (Fernandez et al. 2016b).

This paper is an addition of our ongoing work on cloud and container ecosystem modeling. We present here a pattern for the security of software containers, which extends our previous pattern on containers; we consider now the threats to containers and their defenses. However, it is important to mention that the security features presented here are more generic and form a comprehensive set, all of which may or may not be present in all container products. Our audience includes system architects and designers, software developers who are interested in building container solutions and system administrators who have to manage container environments. Containers were considered as important parts of cloud ecosystems but as additional related products have appeared they are becoming an ecosystem on their own.

In this paper we focus on the security of software containers not the whole container ecosystem. It will be covered in future work. In addition, it is vital to understand the difference between vulnerability and threat here. Threat exploits the system vulnerability which is weakness in the system. Vulnerability can arise because of the way system is implemented, used or configured.

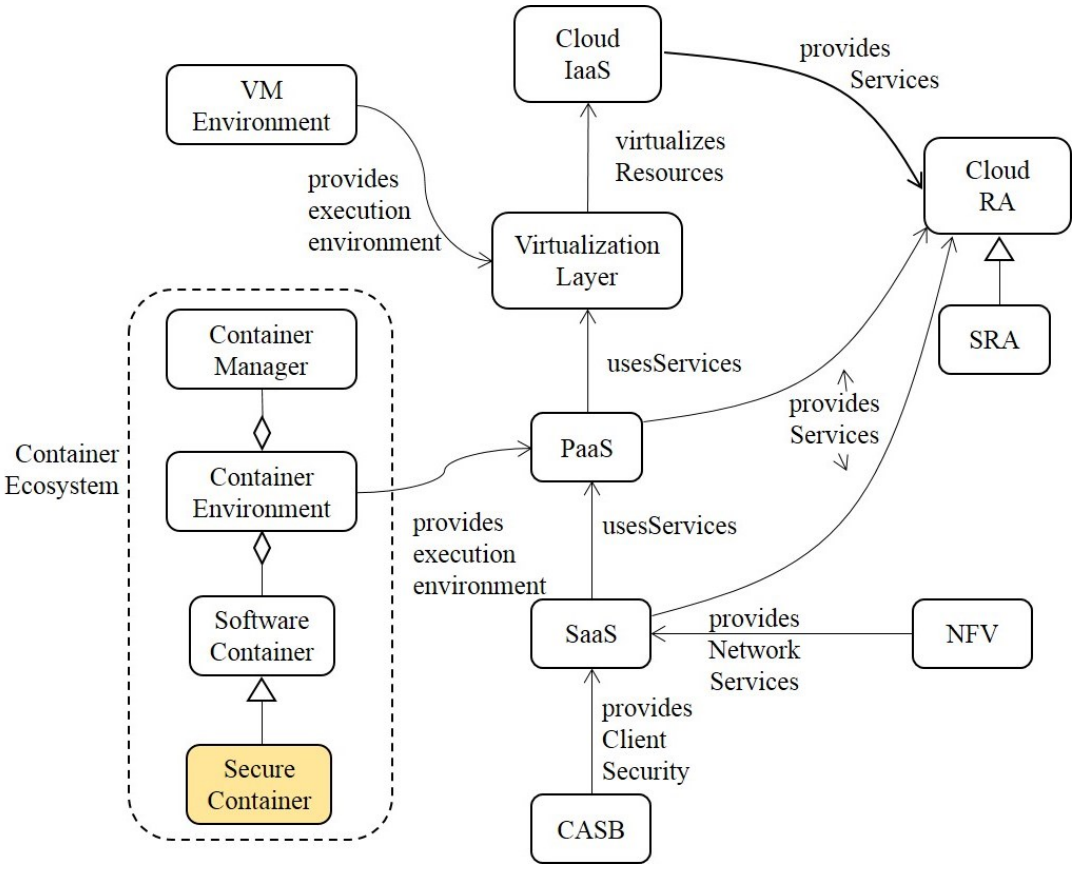

Fig.1. A partial cloud ecosystem

2. THE SECURE SOFTWARE CONTAINER PATTERN

2.1 Intent

To integrate security solutions as countermeasures to the threats of the existing software architecture of the containers. The countermeasures include mechanisms for execution environment isolation, access control to application data, and secure container image creation and storage.

2.2 Context

Software containers provide execution environments for applications sharing a host OS, binaries and libraries. These containers reside in multitenant/shared environments and are under control of the host OS which raises security concerns such as process interference or unauthorized data access. The container allows applications to share a Host OS that provides a set of Resources. Container images are templates that are used to launch container instances. A container image holds the necessary files, programs, and configurations required to execute them (Souppaya et al. 2017). The container images can be created and stored by anyone in public repositories. Different origins are possible for container images. In addition, host resources (e.g. Inter-Process Communication (IPC), filesystem, devices, and communication channels) are shared among these containers and exposed to the applications running in these containers. Figure 2 shows the class diagram of the Software Container pattern from our previous work (Syed and Fernandez 2015). The detailed explanation of components in shaded area can be found in the paper.

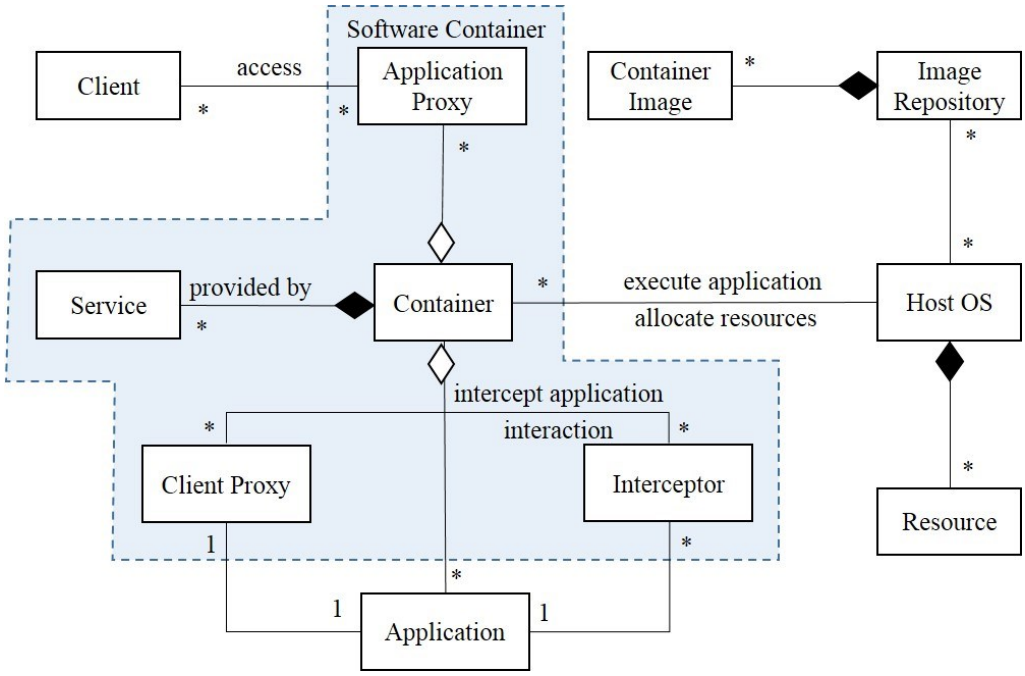

Fig. 2. Class Diagram for Software Container Pattern

2.3 Problem

How do we provide an environment for secure execution of containers from different sources when we have a number of containers sharing resources on a single host? Sharing and variety of origins can lead to many security threats.

The following are the **forces** for this pattern, which in this case correspond to their threats or vulnerabilities:

- *Image poisoning:* Container images can be created by anyone, which can be used by attackers to insert malicious images in the repository. This can result in a variety of attacks for the users of those images.
- *Image vulnerabilities:* Images can have vulnerabilities which can be exploited by attackers.
- *Cross container interference:* Compromised containers can try to observe or interfere with the execution of other containers sharing the same host, resulting in confidentiality or integrity threats.
- *Denial of Service:* An application can use up all the resources of the host, starving other containers sharing the same OS.
- *Interference with inter-process communication:* Compromised containers can intercept and interfere with the IPC of other containers to get unauthorized access to information or disrupt the communication.
- *Unauthorized file access:* Compromised containers can try to read or modify the files of the host or of other containers.

- *Unauthorized host device access:* Compromised containers can damage the host system as they have access to the device nodes for physical memory, storage or terminal. Device nodes are files used by kernel in order to access the hardware.
- *Network-based attacks:* Attackers can use compromised containers for eavesdropping on or manipulating network traffic of the Host or other containers.
- *Privilege escalation:* Containers can gain excessive privileges which gives them full access to the host and other processes which in turn can lead to attacks such as Denial of Service (DoS) and data leakage (Combe 2016).
- *Malicious containers:* Containers can be malicious if they are created from poisoned container images. These containers can be used to attack the host as well as other containers sharing the same environment.

2.4 Solution

Apply appropriate security patterns to avoid or mitigate the identified threats (see consequences) in order to secure the execution environment of containers. These security patterns include authentication, authorization, access control and secure image repository, which can be implemented by various components of the container architecture. In addition, container systems also use encryption for safe communication and logging for anomaly detection and forensics (Souppaya 2017). We describe below how the container structure supports and is integrated with the security services (Figure 3).

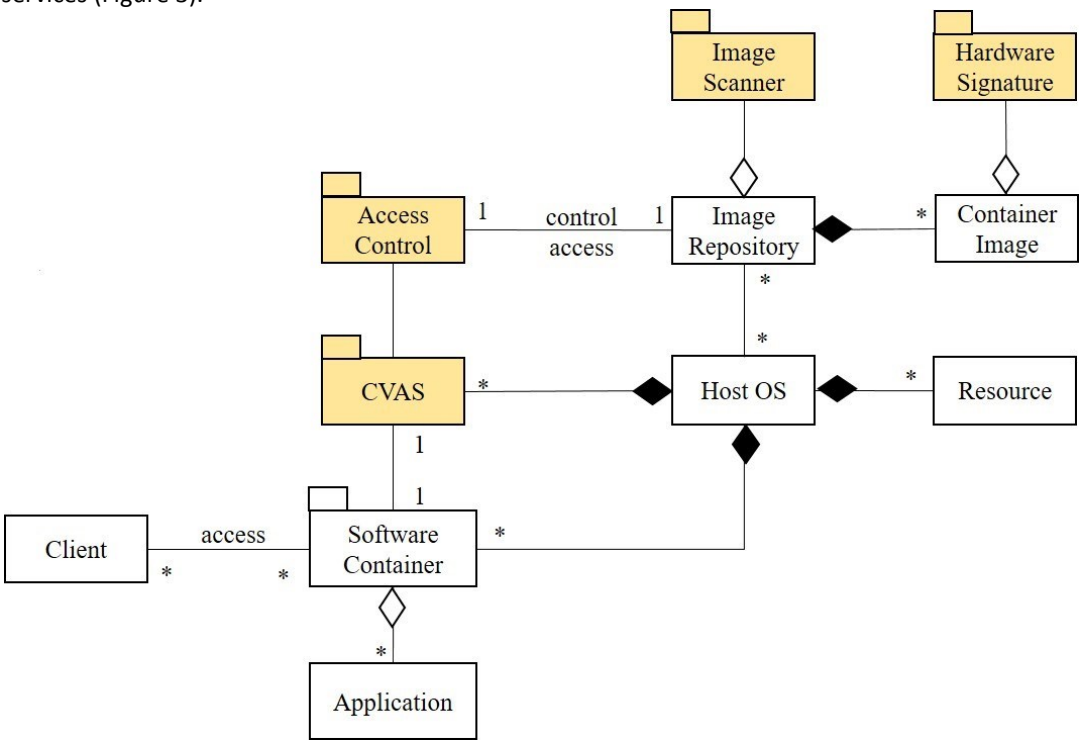

Fig.3. Class diagram of the Secure Container

Structure

Figure 3 shows the class diagram for the Secure Container. This diagram is based on the Software Container pattern (Syed and Fernandez 2015) where security patterns have been added (highlighted). A **Software Container** provides an execution environment for a set of **Applications** sharing a **Host OS,** which provides **Resources** for their execution. The **Image Repository** stores **Container images,** which are authenticated by the use of **Hardware Signatures.** The Image repository uses an **Image Filter** for filtering image vulnerabilities. **Access Control** is used to enforce rights in a Controlled Virtual Address Spaces (**CVAS or sandbox),** and to control access to the Image Repository.

2.5 Implementation

Most of the features discussed in this pattern are based on security features of the Linux Kernel like namespaces and control groups that can be used to limit resource usage. In addition, the host OS enforces the access constraints

imposed on containers. Images are authenticated by hardware signatures. An Image scanner is used to scan and filter all images in the repository for vulnerabilities.

As to provide access control, each container's privileges are separated from root access on the host. Default configurations for the authorization rules in container platforms are rather secure. For example, in Dockers, each container is assigned separate namespace and cgroup. These rules, however can be configured to provide more capabilities to containers thus weakening the security (Combe 2016).

Process interference is prevented by using PID namespaces which are hierarchical and isolate container process ID number space from that of the host. In this way, each container gets limited visibility to the processes in the same container or in its nested namespaces. CVAS system namespaces (mount namespaces) are used to isolate associated filesystem for each container. We can limit container access of device nodes by using the cgroup feature "device whitelist controller" which doesn't let the containers create new device nodes as well. Access to IPC resources is limited for processes running in a container using IPC namespaces so they can only use a subset of these resources for communication, which prevents them from causing interference with other containers or host.

Network namespaces are used to create independent networking stacks for each container to provide network isolation. Each container has its own network interface which allows it to communicate like an autonomous host. Some vulnerabilities still exist, for example, all containers are connected via Virtual Ethernet Bridge. This bridge forwards all incoming packets without filtering them. Consequently, containers are exposed to ARP poisoning and MAC flooding attacks (Bui 2015). Cgroups are used to limit resource usage of every container on the shared system to prevent DoS attack on other processes. They ensure fair distribution of available system resources like CPU, memory, etc. between containers by allowing proper configurations. Linux capabilities and kernel security modules (LSM) are used to restrict privileges for containers. For security reasons, a lot of capabilities are by default disabled for containers. The Linux kernel includes LSMs like SELinux, AppArmor and Seccomp to further enhance its security by providing Mandatory Access Control in addition to Linux Discretionary Access Control. Additional rules regarding system objects and processes can be defined in the form of policies.

2.6 Known Uses

Dockers (Docker 2017), LXC and Rocket all are examples of Linux containers and use cgroups, namespaces and capabilities for isolation and access control (Combe 2016). Dockers also provide hardware signing for container image using YubiKey4 which offers 2 factor authentication. Release number and known vulnerabilities are detected by scanning container images in the image repository (Babcock 2015).

In addition to these, we also have Microsoft containers (Microsoft 2016) and Google containers (Google 2017).

2.7 Consequences

The pattern provides a comprehensive set of security features which may not be available in all containers, including the following defenses to the identified threats (see also Implementation for more specific details):

- *Image poisoning:* Hardware signatures are used to authenticate images. The Image repository may also use authentication and authorization to control access (Fernandez et al. 2013).
- *Image vulnerabilities:* The Image Repository can be scanned and filtered for known vulnerabilities.
- *Cross container process interference:* This interference can be prevented by the CVAS.
- *File unauthorized access:* Host and container files can be protected by using the CVAS.
- *Unauthorized host device access:* Container access to host devices can also be controlled by the CVAS.
- *Interference with inter-process communication:* IPC resources can also be protected by the CVAS preventing containers from causing interference for other containers and host.
- *Network-based attacks:* They can be controlled using standard cryptographic methods and providing independent networking stack and network interface for each container.
- *Denial of service:* Resource usage can be controlled by setting limits in the CVAS.
- *Privilege escalation:* The CVAS can prevent the misuse of rights.
- *Malicious containers:* Authentication and scanning of container images makes it safer to trust image sources and consequently resulting containers. In addition, even if container is compromised, access control configurations can be used to protect the system.

2.8 See also (related patterns)

- Software Container (Syed and Fernandez 2015). A Software Container provides an execution environment for applications sharing a host OS, binaries, and libraries with other containers. Containers have less execution overhead but are less flexible than VMs.
- Controlled Virtual Address Space (Sandbox) (Fernandez 2013). How to control access by processes to specific areas of their virtual address space (VAS) according to a set of predefined access rights? Divide the VAS into segments that correspond to logical units in the programs. Use special words (descriptors) to represent access rights for these segments.
- Secure Virtual Machine Image Repository (Fernandez et al. 2013). Provides authorization and authentication to prevent creation of unauthorized images.
- Digital Signature with Hashing allows a principal to prove that a message was originated from it (Fernandez 2013). It also provides message integrity by indicating whether a message was altered during transmission. Hardware signatures are a special case of digital signatures.

ACKNOWLEDGEMENTS

We thank our shepherd, Sumit Kalra, for his careful and insightful comments that have significantly helped to improve this paper. We also appreciate the beneficial feedback we received from PLoP 2017 writer's workshop participants.

REFERENCES

P. Avgeriou. 2003. Describing, instantiating and evaluating a reference architecture: A case study. *J. Enterprise Architect* (June 2003). 24.

Charles Babcock. 2015. Docker tightens security over container vulnerabilities. Retrieved December 1, 2015 from http://www.informationweek.com/cloud/platform-as-a-service/docker- tightens-security- over-container-vulnerabilities/d/d-id/1323178.

Jan Bosch. 2009. From software product lines to software ecosystems. In *Proceedings of the 13th International Software Product Line Conference* (SPLC '09). Carnegie Mellon University, Pittsburgh, PA, USA, 111-119.

T. Bui. 2015. Analysis of Docker security. Aalto University, arXiv:1501.02967v1, 2015. Retrieved February 5, 2017 from https://arxiv.org/abs/1501.02967v1

Frank Buschmann, Regine Meunier, Hans Rohnert, Peter Sommerlad, Michael Stal. 1996. Pattern-Oriented Software Architecture, Vol. 1: A System of Patterns. J. Wiley & Sons, Inc.

Theo Combe, Antony Martin and Roberto Di Pietro. 2016. To Docker or not to Docker: A security perspective. *IEEE Cloud Computing*, Sept./Oct. 2016, 54-62.

Docker. 2016. Introduction to Container Security: Understanding the isolation properties of Docker. Retrieved May 25. 2017 from https://www.docker.com/sites/default/files/WP_IntrotoContainerSecurity_08.19.2016.pdf

Eduardo B. Fernandez. 2013. Security patterns in practice: Designing Secure Architectures Using Software Patterns. J. Wiley & Sons, Inc.

Eduardo B. Fernandez, Raul Monge, and Keiko Hashizume. 2013. Two patterns for cloud computing: secure virtual machine image repository and cloud policy management point. In *Proceedings of the 20th conference on pattern languages of programs* (PLoP 2013), Monticello, IL

Eduardo B.Fernandez, Raul Monge, and Keiko Hashizume. 2016a. Building a security reference architecture for cloud systems. *J. Requirements Engineering*. DOI: 10.1007/s00766-014-0218-7, June 2016, Volume 21, Issue 2, pp 225-249

Eduardo B. Fernandez, N. Yoshioka and H. Washizaki. 2015. Patterns for Security and Privacy in Cloud Ecosystems. In *Proceedings of the 2nd International Workshop on Evolving Security and Privacy Requirements Engineering* (ESPRE 2015), IEEE (August 2015), 13-18. DOI:10.1109/ESPRE.2015.7330162

Eduardo B. Fernandez, N. Yoshioka, H. Washizaki and M. H. Syed. 2016b. Modeling and security in cloud ecosystems. *J. Future Internet* 2016, 8(2), 13; doi:10.3390/fi8020013 (Special Issue Security in Cloud Computing and Big Data)

Google. 2017. Containers at Google. Retrieved November 15, 2017 from https://cloud.google.com/containers/

Microsoft. 2016. Windows Containers. Retrieved November 15, 2017 from https://docs.microsoft.com/en-us/virtualization/windowscontainers/about/

M. Souppaya, J. Morello and K. Scarfone. 2017. Application Container Security Guide. Retrieved September 25. 2017 from https://doi.org/10.6028/NIST.SP.800-190

TB Sousa, F Correia, HS Ferreira, 2015. Patterns for software orchestration on the clouds. In *Proceedings of the 22nd Conference on Pattern Languages of Programs* 2015, Pittsburgh, PA, October 24-26, 2015

Madiha H. Syed and Eduardo B. Fernandez. 2015. The Software Container pattern. In *Proceedings of the 22nd Conference on Pattern Languages of Programs* 2015, Pittsburgh, PA, October 24-26, 2015

Madiha H. Syed and Eduardo B. Fernandez. 2016. A pattern for a virtual machine environment. In *Proceedings of the* 23rd *Pattern Languages of Programs Conference* (PLoP 2016), Monticello, IL, October 24-26, 2016

Madiha H. Syed and Eduardo B. Fernandez. 2017. The Container Manager Pattern. In *ACM Proceedings of European Conference on Pattern Languages of Programs*. EuroPLoP (July 2017). DOI: https://doi.org/10.1145/3147704.3147735

L. White. 2014. Containers for Development. (September 2014). Retrieved November 30, 2015 from http://www.drdobbs.com/architecture-and-design/containers-for-development/240168801?pgno=1.

Received May 2017; revised September 2017; accepted February 2018