DockerPolicyModules: Mandatory Access Control for Docker Containers

Enrico Bacis, Simone Mutti, Steven Capelli, Stefano Paraboschi
DIGIP — Università degli Studi di Bergamo, Italy
{enrico.bacis, simone.mutti, steven.capelli, parabosc} @ unibg.it

***Abstract*—The wide adoption of Docker and the ability to retrieve images from different sources impose strict security constraints. Docker leverages Linux kernel security facilities, such as *namespaces*, *cgroups* and *Mandatory Access Control*, to guarantee an effective isolation of containers. In order to increase Docker security and flexibility, we propose an extension to the *Dockerfile* format to let image maintainers ship a specific *SELinux* policy for the processes that run in a Docker image, enhancing the security of containers.**

I. Introduction

The idea of Linux containerization (i.e., operating-system-level virtualization) has been around for some time (e.g., LXC, OpenVZ), but it saw a sudden surge in popularity with the advent of Docker in 2013 [1]. Docker adopts a simple *Dockerfile* format that defines the actions needed to generate a Docker image, which is then used to instantiate containers. The image can be built upon other images, available in online repositories. This facilitates the deployment of lightweight containers to run software in isolation. More and more Platform-as-a-Service providers are considering the use of Docker in order to reduce the resource overhead imposed by traditional virtualization.

Containerization introduces new security challenges. In fact, as opposed to classical virtualization, Docker does not need separated operating systems, but it uses the services made available by the Linux kernel in order to isolate the containers. The major threat is represented by compromised or malicious guests attacking other containers that are running on the same system using local exploits. The security and isolation of the containers is correctly perceived as the most critical point for container security.

II. Docker Security

Docker leverages Linux kernel security features such as *kernel namespaces* to isolate users, processes, networks and devices, and *cgroups* to limit resource consumption. When dealing with containers, the kernel Discretionary Access Control (DAC) is usually considered insufficient, due to the flexibility it gives to the subjects and the limited control it provides on the security policy. With Mandatory Access Control (MAC), subjects cannot bypass the system security policy. SELinux is one of the most widespread implementations of MAC. In systems that use SELinux (e.g., RHEL, Centos, Fedora), Docker takes advantage of the policy defined in the scope of the sVirt project [2], which aimed at defining SELinux policies for different virtualization systems. In SELinux it is possible to separate processes in two ways:

Type Enforcement (*TE*): a *label* containing a type is associated with every subject (process) and system object (e.g. file, directory). The policy defines the permitted actions among types, and the kernel enforces these rules. A label with a reduced set of privileges is assigned by Docker to all the processes that are run in containers. *TE* is used to protect the Docker engine and the host from the containers, which can come from untrusted sources;

Multi-Category Security[1] **(*MCS*)**: the label assigned to a subject or an object, can be further specialized with one or more categories, in order to create different *instances* of the same type. An access request is accepted if it is allowed by TE and the subject and the object are in the same category. Different containers are assigned different categories, thus they are separated from each other even if they have the same type.

Currently all the containers run with the same SELinux type, *svirt_lxc_net_t*, as defined in the policy configuration file *lxc_contexts*. Running all the containers with the same type is a serious limitation. In fact, we have to grant *svirt_lxc_net_t* the upper bound of the privileges that a container could ever need. For example, since different applications operate on different network ports, *svirt_lxc_net_t* is allowed to listen to and communicate over all the network ports [3]. Specializing the type per container (or even per process) would permit to tighten the security of Docker containers.

Docker already offers the user the ability to start the processes in a container with a different SELinux type, through the *–security-opt* parameter. However, in this case the user is in charge of defining a suitable extension to the policy. Recently, an SELinux policy for the *Apache httpd* container has been proposed by Daniel Walsh [3]. When the policy is installed, the container can be run with the specific type using:

```
docker run -d --security-opt type:
    docker_apache_t httpd
```

Although it is possible to start containerized processes with specific SELinux types, there are still limits to the applicability of this concept. It is reasonable to expect that many users will either be unfamiliar with the SELinux syntax and semantics, or do not know how to compile and install a policy module.

III. proposal

We propose a solution able to introduce specific SELinux types for different containerized processes in a transparent

[1] Docker also integrates the SELinux Multi-Level Security (*MLS*), but it will not be discussed here since it is not relevant in our proposal.

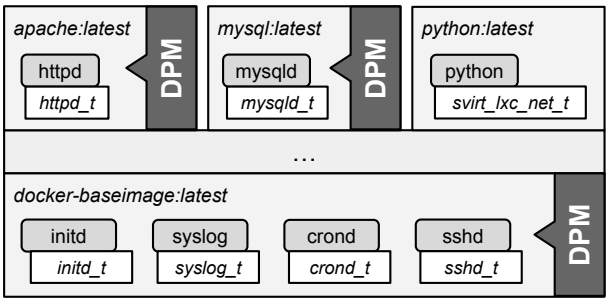

Fig. 1. Processes running in three Docker containers (*apache*, *mysql* and *python*), using specific SELinux types defined in the DockerPolicyModules embedded in the images.

way for the user. This is based on Docker allowing image maintainers to ship an SELinux policy module together with their images. The module will be installed in the host system and defines the types that will be associated with the processes in the image. These modules are named *DockerPolicyModules* (*DPM*) and are SELinux modules that must also satisfy the properties defined in the following, in order not to represent a threat for the host system. The DPM for an image will be specified in the Dockerfile and embedded in the image metadata at build-time. In order to run containerized processes with specific SELinux types, the image maintainer can label the binaries in the image with specific types, and write a type transition rule. In this way, when the binary is executed, the process is assigned the SELinux type defined in the rule. Even if we have multiple processes running in the same image (e.g., the widely adopted *docker-baseimage* runs *init*, *syslog*, *cron* and *ssh*), it is possible to execute them with different SELinux labels. When a Docker container consists of different images, all the DPMs for the images that compose the container will be installed. This makes available also the SELinux types for processes in the parent images. Figure 1 represents different Docker containers with custom types provided by their DPM.

Recently we proposed a similar approach for Android third-party apps and we studied the security implications involved [4], [5]. The Docker scenario appears to fit quite well with this proposal, with the additional advantage that we expect image developers (as opposed to image users and app developers) to be more familiar with the role and impact of MAC policies.

In order to avoid the possible threats that can emerge from letting Docker images install SELinux modules in the host system, we need to analyze the cases that derive from the combination of the system policy and a DPM. Due to the fact that each SELinux rule has a source (σ) and a target (τ) type, and they can be defined either in the system policy or in the DPM, we have four possible scenarios, described in Table I.

A: the DPM must not change the system policy and can only have an impact on processes and resources associated with the DPM itself. Since containers can not be trusted a priori, it is imperative that the provided DPM does not have an impact on privileges where both σ and τ are system types;

B and **C**: new types defined in a DPM must always operate within the boundaries defined by the *svirt_lxc_net_t* type. The SELinux *typebounds* rule is used to confine the types, imposing an upper bound to the privileges that a type defined in a DPM can request. If a DPM defines a type not typebounded by *svirt_lxc_net_t*, or a privilege not compliant with the *typebounds* rule, it is considered invalid;

TABLE I. VALIDATION OF THE AVC RULES IN A DPM.

	$\tau \in BASE$		$\tau \in DPM$	
$\sigma \in BASE$	(A)	**INVALID** *(threat for the system)*	(B)	***OK / INVALID*** *based on* typebounds
$\sigma \in DPM$	(C)	***OK / INVALID*** *based on* typebounds	(D)	**OK**

D: a DPM provides the flexibility of defining multiple types with different privileges so that the container, according to the functionality in use, may switch to the one that represents the "least privilege" domain needed to accomplish the current task. This permits to limit the abuse that may derive from the exploitation of internal vulnerabilities and to tighten the overall container security.

The Docker Hub Registry must ensure that the DPM of any uploaded image satisfies the requirements expressed in Table I. Any image with a DPM not compliant with the above rules will be rejected. To protect the client from a compromised Docker Hub, a pre-processing phase will be added to Docker download and update routines in order to verify, before installing it, that the DPM does not represent a threat to the system.

IV. CONCLUSIONS

SELinux is a sound security solution and its support for policy modules has already proved to be a significant enhancement in several services. The adaptation to Docker of the support for policy modules will allow the specification of SELinux domains for the different images, leading to an increase of security in Docker. We do not assume that all the image maintainers will include a DockerPolicyModule in their images, but the ones who are aware of the benefit that SELinux provides will certainly appreciate the proposed extension.

V. ACKNOWLEDGEMENTS

This work was partially supported by a Google Research Award (winter 2014), by the Italian Ministry of Research within the PRIN project "GenData 2020" and by the EC within the 7FP and H2020 program, respectively, under projects PoSecCo (257129) and EscudoCloud (644579).

REFERENCES

[1] D. Merkel, "Docker: lightweight linux containers for consistent development and deployment," *Linux Journal*, vol. 2014, no. 239, p. 2, 2014.

[2] J. Morris, "sVirt: Hardening linux virtualization with mandatory access control," in *Linux. conf. au Conference*, 2009.

[3] D. J. Walsh, "Tuning Docker with the newest security enhancements," 2015. [Online]. Available: http://opensource.com/business/15/3/docker-security-tuning

[4] S. Mutti, E. Bacis, and S. Paraboschi, "Policy Specialization to Support Domain Isolation," in *SafeConfig 2015: Automated Decision Making for Active Cyber Defense (SafeConfig15-ACD)*, Oct. 2015.

[5] E. Bacis, S. Mutti, and S. Paraboschi, "AppPolicyModules: Mandatory Access Control for Third-Party Apps," in *Proceedings of the 10th ACM Symposium on Information, Computer and Communications Security*. ACM, 2015, pp. 309–320.