A State-of-Art Review of Docker Container Security Issues and Solutions

Article · January 2017

CITATIONS
4

READS
3,293

1 author:

Jyoti Shetty
Rashtreeya Vidyalaya College of Engineering
26 PUBLICATIONS **130** CITATIONS

SEE PROFILE

American International Journal of Research in Science, Technology, Engineering & Mathematics

Available online at http://www.iasir.net

ISSN (Print): 2328-3491, ISSN (Online): 2328-3580, ISSN (CD-ROM): 2328-3629

AIJRSTEM is a refereed, indexed, peer-reviewed, multidisciplinary and open access journal published by International Association of Scientific Innovation and Research (IASIR), USA

(An Association Unifying the Sciences, Engineering, and Applied Research)

A State-of-Art Review of Docker Container Security Issues and Solutions

Sahana Upadhya, Jyoti Shetty, Raja Rajeshwari H S, Dr. G Shobha

Computer Science and Engineering Department

R V College of Engineering,

Bengaluru, Karnataka, India

Abstract: *Docker containers wrap up a piece of software in a complete filesystem that contains everything it needs to run i.e. code, runtime, system tools, system libraries – anything that can be installed on a server. It provides benefits of virtualization with improved performance as it avoids the overhead of hypervisor layer. But however the security issues have prevented Docker containers from being adopted in the production environment. This manuscript provides systematic study of Docker container security issues and ways to prevent it. The motive is to provide developers and researchers with state-of-art Docker container issues and developments.*

I. INTRODUCTION

Containers have been in the industry for a long time. But they have gained momentum only recently after the introduction of Docker in 2012. They provide an alternative way for virtualization which is slightly different from the traditional hypervisor based virtual machines. Containers are lightweight application processes that are running in isolation from other processes in the system. They share the host's kernel and therefore reduce their size in a way that is impossible to achieve through hypervisor based virtualization. This also makes them extremely fast, easily portable and superior in terms of distribution of the application.

However, this shared kernel is the single point of failure. A small security breach in the process running inside a container can jeopardize the host kernel. In fact, it has the power to bring down the entire kernel itself. This breakout from a container can even affect other containers running on the same system. For example, consider two containers communicating with each other- one running an apache server and the other running a database like MySQL. The container which has the database is holding a record of credit card details of all users. The job of the apache container is to retrieve the details of the credit card requested. But if this apache container goes haywire, it can do hazardous things like retrieving the credit card details of all the users. This example itself signifies the importance of security in containers.

A common misconception is that "containers actually contain". But containers are just another way of virtualization and do not actually safeguard the system from a rogue application since not all the resources the container has access to are namespaced. Docker security is all about limiting the chances of such attacks explained above and also reducing the attack surface of the host kernel. As Docker evolves, a multitude of container vulnerabilities are being addressed and fixed.

II. SECURITY ISSUES AND SOLUTIONS

According to a survey conducted by Forrester in January 2015 as to why enterprises weren't running containers in production, 53% of the enterprises reported that their biggest concern was security. The security issues that have been afflicting Docker containers are explained below [1].

A. Exploitation of the kernel

A process running inside a VM would find it relatively hard compared to a container to damage the host kernel. The reason behind this being that the process would first have to break through the VM kernel and then the hypervisor layer to be able to touch the host kernel. This two level protection is far superior compared to Docker containers that share the host kernel and can bring down the host kernel quite easily with even a very small vulnerability in the application container.

One of the solutions to this problem can be to deploy containers in conjunction with VMs. In this hybrid solution, an entire group of services to be isolated from each other can be placed inside containers and then grouped inside a virtual machine [2].

The recent release of Docker 1.10 introduced authorization plug-ins which restricts who can execute what based on granular policies that can be implemented by an organization or an individual. Sysadmins can use these plugins to configure user access policies for their infrastructure [3].
Some other ways we can prevent exploitation of the kernel are:

1. By not running containers with the - -privileged flag

The - -privileged flag gives all the capabilities to the container, and it also lifts all the limitations enforced by the device cgroup controller [4].

2. Set volumes to read-only

If there is no need to modify any files in attached volumes, then it should be made read-only to prevent malicious users from modifying it [4].

```
||/ Name                          Version                      Architecture Description
+++-=============================-============================-============-================================================================
ii  adduser                       3.113+nmu3ubuntu3            all          add and remove users and groups
ii  apt                           1.0.1ubuntu2.10              amd64        commandline package manager
ii  apt-utils                     1.0.1ubuntu2.10              amd64        package management related utility programs
ii  base-files                    7.2ubuntu5.3                 amd64        Debian base system miscellaneous files
ii  base-passwd                   3.5.33                       amd64        Debian base system master password and group files
ii  bash                          4.3-7ubuntu1.5               amd64        GNU Bourne Again SHell
ii  bsdutils                      1:2.20.1-5.1ubuntu20.7       amd64        Basic utilities from 4.4BSD-Lite
ii  busybox-initramfs             1:1.21.0-1ubuntu1            amd64        Standalone shell setup for initramfs
ii  bzip2                         1.0.6-5                      amd64        high-quality block-sorting file compressor - utilities
ii  console-setup                 1.70ubuntu8                  all          console font and keymap setup program
ii  coreutils                     8.21-1ubuntu5.3              amd64        GNU core utilities
ii  cpio                          2.11+dfsg-1ubuntu1.1         amd64        GNU cpio -- a program to manage archives of files
ii  cron                          3.0pl1-124ubuntu2            amd64        process scheduling daemon
ii  dash                          0.5.7-4ubuntu1               amd64        POSIX-compliant shell
ii  debconf                       1.5.51ubuntu2                all          Debian configuration management system
ii  debconf-i18n                  1.5.51ubuntu2                all          full internationalization support for debconf
ii  debianutils                   4.4                          amd64        Miscellaneous utilities specific to Debian
ii  dh-python                     1.20140128-1ubuntu8.2        all          Debian helper tools for packaging Python libraries and applications
ii  diffutils                     1:3.3-1                      amd64        File comparison utilities
ii  dmsetup                       2:1.02.77-6ubuntu2           amd64        Linux Kernel Device Mapper userspace library
ii  dpkg                          1.17.5ubuntu5.5              amd64        Debian package management system
```

```
sahana@sahana:~$ docker run -v $(pwd)/secrets:/secrets:ro debian touch /secrets/x
touch: cannot touch '/secrets/x': Read-only file system
```

3. Installing only the necessary packages in the container

In order to check what packages are installed in a container, run the following command [4]:

```
sahana@sahana:~$ docker exec dcff90deceae dpkg -l
```

Fig. 1. List of Installed packages in container

Where, dcff90deceae is the container ID. The result of this command is a list of installed packages as shown in Fig 1. In order to improve security in containers, download only the required packages.

4. By using secure computing (seccomp) profiles

Seccomp profiles were introduced in Docker 1.10 [5] to limit the system calls that can be made by the container. This feature adds another level of security as it ensures that the container can do only what it needs to do.

B. Denial Of Service Attacks

Docker containers run in such a way that each container believes that it is the only process running in the system. In fact, when we run the command **ps** inside the container, it will display only the processes running inside the container. However, if a container breaks out of this isolation, it can start consuming resources like CPU cycles, memory, user IDs (UID) etc. of the host system. It has the power to exercise control over other containers and deny access to parts of the system to the users.
Some of the ways in which we can prevent such attacks are:

1. Limiting the CPU shares for containers

All containers get an equal proportion of CPU cycles which is a weight of 1024 by default [6][7]. This proportion can be modified at run time by changing the container's CPU share weighting relative to the weighting of all other running containers as shown below.

2. Setting the container file system to read-only

If there is no need to modify files in a container, then make the filesystem read-only [4].

```
sahana@sahana:~$ docker run --read-only ubuntu:14.04 touch y
touch: cannot touch 'y': Read-only file system
```

3. Turning off Inter-container communication

By default, unrestricted network traffic is enabled between all the containers on the same host. Running containers with - -icc = false - -iptables in order enables only communication between containers that have been explicitly linked together [4].

4. Setting the amount of memory a container can use

Another feature of Docker is that of limiting the amount memory a container can use. By setting the maximum amount of memory a container can use, we can prevent it from starving out other container's resources [8]. This can be achieved by using –m and then specifying the memory size in terms of bytes or by adding a suffix k, m or g. In the following example, the memory of the container is limited to 512 MB.

```
sahana@sahana:~$ docker run -d --cpu-shares 512 ubuntu:14.04
```

C. Container Breakouts

Up until recently, Docker didn't have username space for containers. This was a major security issue. If a process breaks out of a container, since it is not user namespaced, the process would have all the privileges it had in the container on the host too. If the process had root privilege (UID 0) inside the container, it would have root privileges in the host system. This creates a possibility of privilege escalation attack i.e., when one user gains system rights of another user. Although the possibility of a breakout is very small, it should not be considered insignificant.

With the release of Docker 1.10, namespacing is the most exciting security feature introduced. Docker now separates the processes in the container from the processes of the host. Each process now has its own set of user and group IDs. These UIDs are then mapped to other UIDs of the host system. For example, if the process in the container is the root with UID 0, then it may be mapped to a UID of 3000 on the host. This therefore prevents the process running in container from gaining root privileges on the host.

D. Poisoned Images

Docker hub, which is the official registry for Docker containers, has over a million images to download from. There is no guarantee that the image is from a trusted source. An attacker might put in some malicious code along with the image which can help him gain access to the system and the data within. There is also a possibility that the images might be out dated and many contain vulnerabilities.

There have been many efforts to address this issue and with the most recent release of Docker 1.10, Docker has added a new security feature in which image IDs reflect the content inside the image.

Docker Cloud, which is a hosted service that provides a Registry with build and testing facilities for Dockerized application images can scan images in private repositories to verify that they are free from known security vulnerabilities or exposures and report the results of the scan for each image tag. This security scanner, however, is free for private repository subscribers until August 1st, 2016 [9].

The Docker Notary project is working on secure signing and verification of Docker images. As an extension to this, Docker has also introduced hardware signing of container images. It is an open source system for certifying the validity of the sources of Docker images pushed to public repositories and encrypting the contents of these images [10][11].

E. Compromising secrets

It is common to create containers with applications that may need access to databases and other services on the host system. Most of these services may require API keys and database passwords. If an attacker gains access to these secrets, then he can also gain access to these services. Therefore, these passwords and keys must be kept secure.

1. **We can prevent this by not using environment variables to share secrets.**
   Environment variables are easily leaked when debugging and exposed in too many places including child processes and linked containers. Instead of using environment variables, using volumes to pass secrets in a file is safer [4].
2. **Setting container file systems to read-only** and running containers without the - -privileged flag which were mentioned earlier can also help in increasing the security [4].

III. CONCLUSION

Docker containers, during its early days, were plagued with security issues. These security issues have prevented Docker containers from being adopted in the production environment. The major security issues are the Docker daemon attack surface vulnerabilities, malicious images in Docker hub, denial of service attacks and container break outs. One of the major issues was the absence of user namespaces. But with the release of Docker 1.0, this has been included. Along with this, seccomp profiles have also been introduced to limit the number of system calls that can be made by a container. Most of these security issues can be prevented by taking some precautions such as the ones explained in the report. As Docker is evolving, these issues are being addressed and fixed rigorously.

REFERENCES

[1] Adrian Mouat, " Docker Security- Using Docker containers safely", O'Reilly Media, August 2015
[2] Thibault Bronchain, Is Docker secured?, https://www.airpair.com/hyper/posts/hyper-is-docker-secured.
[3] Jessie Frazelle, Docker Engine 1.10 Security Improvements, February 2016, https://blog.docker.com/2016/02/docker-engine-1-10-security/
[4] Docker Security Cheat Sheet, Container Solutions, 2015, https://container-solutions.com/content/uploads/2015/06/15.06.15_DockerCheatSheet_A2.pdf
[5] Seccomp profile, https://github.com/docker/docker/blob/master/docs/security/seccomp.md
[6] https://docs.docker.com/engine/reference/run/
[7] Marek Goldmann, Resource management in Docker, https://goldmann.pl/blog/2014/09/11/resource-management-in-docker/#_example_managing_the_cpu_shares_of_a_container
[8] Marek Goldmann, Resource management in Docker, https://goldmann.pl/blog/2014/09/11/resource-management-in-docker/#_memory
[9] Docker Cloud, https://cloud.docker.com/
[10] Notary, https://github.com/docker/notary
[11] Susan Hall, "Docker Container Security: Signed, Sealed and Delivered from Vulnerabilities", November 2015, https://thenewstack.io/3-new-security-features-docker/