Securing Docker Containers from Denial of Service (DoS) Attacks

Jeeva Chelladhurai, Pethuru Raj Chelliah
IBM Global Cloud Center of Excellence
Bangalore, Karnataka, India - 560045
{jeeva.chelladhurai, pechelli}@in.ibm.com

Sathish Alampalayam Kumar
Coastal Carolina University
Conway, SC, USA - 29528
skumar@coastal.edu

Abstract: **The concept of containerization and virtualization is getting traction in the cloud based IT environments. Docker engine is popular implementation for simplifying and streamlining containerization technology. IT industry realizes numerous automation and acceleration features and facilities through the embracement of the Docker-sponsored containerization paradigm, which is an operating system (OS)-level and lightweight virtualization. However the security issues affect the widespread and confident usage of Docker platform. In this paper, we have discussed important security issues of the Docker containers as well as the related work that is being carried out in this area. Also we have proposed security algorithms and methods to address DoS attacks related issues in the Docker container technology. The preliminary experiments and testing of the security methods are promising.**

Keywords: ***Cloud Containers, Docker Technology, Security Issues, DoS Attacks, Micro Services Architecture***

I. Introduction

Docker Platform Threats And Vulnerabilities

Following are some of the critical security related vulnerabilities and threats for Docker Technology.
Control groups (Cgroups), the prominent capability of Linux kernel, helps ensure that each container gets its fair share of memory, CPU, disk I/O; and a single container cannot bring the system down by exhausting one of those resources. However they do not play a role in preventing one container from accessing or affecting the data and processes of another container [6]. But control groups play a vital role in fending off some Denial-of-Service (DoS) attacks.

ARP Spoofing and MAC Flooding Attacks: When Docker creates a new container, it also establishes a new virtual Ethernet interface with a unique name and then connects this interface to the bridge. The interface is also connected to the eth0 interface of the container, thus allowing the container to send packets to the bridge. The default connectivity model of Docker is vulnerable to ARP spoofing and Mac flooding attacks since the bridge forwards all of its incoming packets without any filtering.

Docker Daemon Attack Surface: Docker daemon sits in between Docker clients and the serving containers. The user does not directly interact with the daemon, instead interacts through the Docker client. This daemon currently requires root privileges and hence only trusted users should be allowed to control Docker daemon. Docker allows sharing a directory between the Docker host and a guest container. It allows doing so without limiting the access rights of the container. The issue here is that one can start a container where the/host directory will be the / directory on the Docker host and the container will be able to alter the host file system without any restriction. This has a strong security implication. Suppose there is a web server in a Docker host and it is possible for a malicious user to pass crafted parameters to create arbitrary containers in the web server host.

DoS attacks: VMs are claimed to be more secure than containers as they add an extra layer of isolation between the applications and the host. An application running inside a VM is only able to communicate with the VM kernel, not the host kernel. Consequently, in order for the application to escalate out of a VM, it must bypass the VM kernel and the hypervisor before it can attack the host kernel.

Containers can directly communicate with the host kernel, thus allowing an attacker to save a great amount of effort when breaking into the host system. This substantially raises a security concern over containers. When one or more containers got compromised, then there are enough ammunitions for subsequent attacks and risks. The attacker can perform various attacks such as denial of service (DoS) and privilege escalation.

Integrity of Docker Host Issue: The first question is about the integrity of Docker host. That is, how to ensure the trustworthiness of the Docker daemon and how to ensure Docker host getting booted with integrity. The second one is to verify the integrity of Docker containers. The Docker images

978-1-5090-2628-9/16 $31.00 © 2016 IEEE
DOI 10.1109/SCC.2016.123

are typically arriving from different untrusted sources. How to ensure whether the right image got launched? Further on, the security of Docker engine has to be guaranteed. Containers are talking to one another within the host as well as across hosts. Containers are interacting with their underlying host. If a container gets compromised, how to arrest the cascading and catastrophic effect is the pertinent and paramount question. The policy-based controlled connectivity, communication and collaboration are being insisted in order to reap the originally envisaged benefits of the containerization paradigm. There are questions raised on the aspects of the Docker management, identity authentication and authorization, container auditability, etc.

II. Existing Docker Platform Security Approaches

As discussed in Section 3, the unprecedented adoption of the Docker platform for building and deploying application-aware containers is beset with security problems. Docker inherently leverages the key Linux kernel security facilities, such as namespaces, Cgroups and Mandatory Access Control to guarantee an effective encapsulation and isolation of containers. Docker leverages Linux kernel namespaces to isolate users, processes, networks and devices, and Cgroups to limit resource consumption. The widely articulated mechanisms for ensuring a tighter security for Docker containers include process, file system, device, IPC, and network isolations. The other one is access restriction. There are best practices for the Docker world from security experts proposed in the literature. Considering the vitality of secured and risk-free containers, a number of policy-aware security solutions are being proposed. Best practices for strengthening container security are also documented and shared across.

Following are some of the related work that is carried out to address security for the Docker container.

Sockets and API: A number of changes have been effected in the latest releases of the Docker platform. The REST API endpoint used by the Docker CLI to communicate with the Docker daemon got changed recently. That is, a Unix socket is being used in place of TCP socket bound on 127.0.0.1. It is also possible to expose the REST API over HTTP by ensuring that it will be reachable only from a trusted network or VPN or protected with stunnel and client SSL certificates. For further security, we can use HTTPS and certificates.

Security Hardening: Docker containers are quite secure if we are careful enough of running workloads inside the containers in the non-privileged mode. Security holes and threats are being constantly unearthed and appropriate solutions are being developed in order to boost the users' confidence on containers. We can add an extra layer of safety by enabling AppArmor, SELinux, Seccomp, GRSEC or other hardening solutions. Newer security approaches and frameworks need to be formed and firmed up at different levels. Rocket is a recently-created and simple alternative container format. Simpler systems like Rocket are often easier to analyze and thus potentially easier to secure. Apcera's Hybrid Cloud Operating System (HCOS) is another interesting way for enhanced security for containers. Twistlock is an end-to-end security solution that addresses the number one obstacle to adoption of containers. Developers get a customizable and friendly solution they can use to apply quality gates prior to pushing the containers into production. Operation teams get a centralized location from which they can manage the security aspects of all Dockerized applications.

Mandatory Access Control (MAC): There are enhancement being made towards the utmost security of Docker containers. They argue that 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, Docker takes advantage of the policy defined in the scope of the sVirt project, 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 (file and directory). The policy defines the permitted actions among types and the kernel enforces these rules strictly. A label with a reduced set of privileges is assigned by Docker to all the processes that are executed 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 (MCS): The label assigned to a subject or an object can be further subdivided with multiple categories to create different instances of the same type. An access request is accepted if it is allowed by TE first and the subject and the object are in the same category. However it is advantageous if different containers are assigned different categories to have a clear separation from each other even with same type. At this point of time, 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 limiting factor. 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. Therefore specializing the type per container would go a long way in tightening the security of Docker containers.

Secure Containers with Security-Enhanced Linux (SELinux): SELinux is an implementation of a mandatory access control (MAC) mechanism, multi-level security (MLS), and multi-category security (MCS) in the Linux kernel. The sVirt project builds upon SELinux and integrates with Libvirt to provide a MAC framework for virtual machines / containers. This architecture provides a secure separation for containers as it prevents root processes within the container from interfering with other processes running outside this container. The containers created with Docker are automatically assigned with a SELinux context specified in the SELinux policy [7].

Docker Security Policies: 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 here too, the user is in charge of defining a suitable extension to the policy. Solutions such as introducing specific SELinux types for different containerized processes in a transparent way for the user are being proposed. Building and shipping Docker images with a SELinux policy module is the promising option for realizing the required security levels. The module will be installed in the host system and defines the types that will be associated with the processes in the image [1]. These modules are named DockerPolicyModules (DPM). 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 there are multiple processes running in the same image, it is still 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. Thus incorporating policy modules with Dockerfile, the faster mechanism for building Docker images, the access restriction is imposed on Docker containers.

The Security Best Practices: In addition to these security methods, following are some of the techniques proposed in the literature for ensuring foolproof security for Docker images, containers and hosts [10]. There are a number of knobs to be tweaked and twiddled to further increase the isolation and to circumvent the limitations imposed by Docker. These include:

- Optimized Images - By reducing the number of binaries and services running in containers, it is possible to decrement the attack surface.
- Using read-only filesystems – File systems are an important source for hackers and attackers. By having read-only file systems, it is possible to stop any kind of uploading malicious scripts, defacing HTML files and overwriting sensitive data.
- Limiting kernel calls – This is another interesting and informed option. That is, by leveraging the SELinux capabilities, it is possible to lock down the system calls that a container can make.
- Policy-enablement and enforcement – Typically Docker images are being realized through the Dockerfile and as described above, additional security-centric policies can be incorporated in the Dockerfile as security modules to enact enhanced security for Docker containers.
- Restricting networking – That is, by allowing only linked and trusted containers to interact with one another. This way, you can stop hackers to probe and use containers to proceed with their evil intentions. This can be achieved by using the --icc=false and --iptables flags when starting the Docker daemon.
- Limiting memory and CPU – As illustrated in our solution, you can limit the amount of CPU and memory allocated to a container.

III. Proposed Security Approach And Experimentation For DoS Attacks In Docker Platform

As we are aware, DoS has been a serious problem in the IT world by making productive IT systems and resources unusable through a cunningly coordinated efforts of bulldozing the systems with unwanted stuffs and requests. Containers are also not left alone and hence there is an insistence for sophisticated solutions for circumventing DoS attacks [8]. In this section, we have formulated a pragmatic method to address DoS attacks and implemented the method to demonstrate its capability in eliminating DoS attacks. Our solution methodology for Docker Container DoS attacks is illustrated in the following Figure 3.

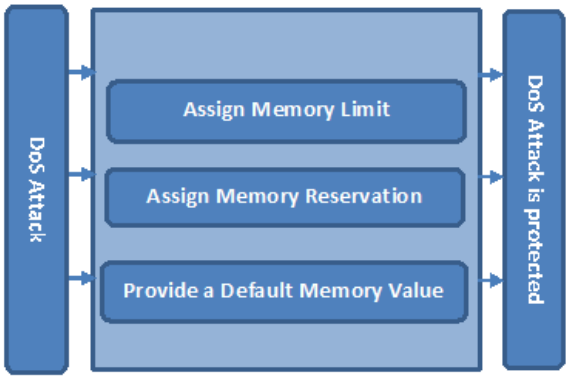

Figure 3. Proposed Solution Methodology for Docker Container DoS attacks

Experimentation

We performed the experimentation by implementing the threat models, invoking the DoS protection implementation and testing the DoS security method using the test cases indicated as follows.

Test Case 1: Assigning memory limit in the docker run command.

In this test case, the memory limit for each container via the docker command using the option '-m'. For instance, the docker command is as shown below:

```
docker run -it -m=512mb ubuntu:lastest /bin/bash
```

When this command is executed, the Docker daemon runs to create a container of size 512 MB built on Ubuntu image. In this case, even when a complete memory usage program is run in the container, it just uses up the memory of 512 MB and hence system cannot be brought down.

Test Case 2: Assigning memory reservation in the default file. When the memory is not assigned in the docker command, the Docker daemon checks the default file in the source code. If there is memory size mentioned, then the container is built of that size. The file is present in the path /etc/ default/docker.

Test Case 3: Memory limit is not set - When both of the above both the cases are false, we have to provide a default value so that any DOS attack won't crash the system. For this test case we limit the memory resource by dividing the total memory, so that a minimum number of containers are allowed. Following listings indicate the threat models that were implemented to conduct the experimentation.

```
int *p;
while(1) {
  p=(int*)malloc(4);
  if (!p) break;
```

Listing 1 – Memory use up code to cause DoS Attack

```
double start, end;
  double runTime;
  start = omp_get_wtime();
  int num = 1,primes = 0;
  int limit = 1000000;
#pragma omp parallel for schedule(dynamic) reduction(+ : primes)
  for (num = 1; num <= limit; num++) {
    int i = 2;
    while(i <= num) {
      if(num % i == 0)
        break;
      i++;      }
     if(i == num)
      primes++;   }
  end = omp_get_wtime();
```

Listing 2 – CPU use up code to cause DoS attack

Test results demonstrate that with such kinds of subtle changes, the security and safety of Docker containers can be substantially improved for the DoS attacks. Further, access control, authentication, and authorization solutions can be hardened in order to make them relevant for the container world.

IV. Conclusion

The aspect of containerization is gaining a lot of attention especially with the surging popularity of Docker engine for simplifying and streamlining containerization. A number of automation and acceleration features and facilities are being realized in IT environments through the sagacious embracement of the Docker platform. However the security comes in the way of widespread and confident usage of the containerization and hence unbreakable and impenetrable security solutions are being solicited. In this paper, we have discussed security issues and how they can be tackled through various methods. In addition, we have proposed pragmatic solution for the DoS attacks. The preliminary experimental results are promising. Further innovations on the existing solutions is needed to fortify the security requirements for containerized workloads and environments.

References

[1] P. Rad et al., "Secure image processing inside cloud file sharing environment using lightweight containers"in 2015 IEEE International Conference on Imaging Systems and Techniques (IST), 2015, pp. 1-6.

[2] R.Zhang, M. Li and D. Hildebrand, "Finding the Big Data Sweet Spot: Towards Automatically Recommending Configurations for Hadoop Clusters on Docker Containers" in 2015 IEEE International Conference on Cloud Engineering (IC2E) Year: 2015, pp. 365 - 368

[3] J. Rey et al., "Efficient Prototyping of Fault Tolerant Map-Reduce Applications with Docker-Hadoop", in 2015 IEEE International Conference on Cloud Engineering (IC2E), pp. 369 – 376

[4] Di Liu and Libin Zhao, "The research and implementation of cloud computing platform based on docker", in 11th Intl Computer Conference on Wavelet Active Media Technology and Information Processing (ICCWAMTIP), 2014, pp. 475 – 478

[5] Dirk Merkel,"Docker:lightweight Linux containers for consistent development and deployment", Linux Journal , Vol. 2014, No. 239

[6] C. Priebe et al., "CloudSafetyNet: Detecting Data Leakage between Cloud Tenants", in the proceedings of the 6th edition of the ACM Workshop on Cloud Computing Security, November 2014, pp.117 - 128

[7] M. Garrett, "Container Security with SELinux and CoreOS", September 29, 2015 [Online] accessed Nov 2015 from https://coreos.com/blog/container-security-selinux-coreos/ on Nov, 2015

[8] Steven J. Vaughan-Nichols,"For containers, security is problem #1", ITworld, May 2015 [Online] accessed Nov 2015 from http://www.itworld.com/article/2920349/security/for-containers-security-is-problem-1.html

[9] E. Bacis et al., "DockerPolicyModules: Mandatory Access Control for Docker Containers" in Proceedings of 2015 IEEE Conference Communications and Network Security (CNS), 2015

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