National College of Ireland

A Framework of Docker Containers to Minimize Risk of ARP Protocol Poisoning Attack

MSc Research Project
Cloud Computing

Bhagirath Purushothama
x15047211

School of Computing
National College of Ireland

Supervisor: Manuel Tova-Izquierdo

National College of Ireland
Project Submission Sheet – 2015/2016
School of Computing

Student Name:	Bhagirath Purushothama
Student ID:	x15047211
Programme:	Cloud Computing
Year:	2016
Module:	MSc Research Project
Lecturer:	Manuel Tova-Izquierdo
Submission Due Date:	16/08/2017
Project Title:	A Framework of Docker Containers to Minimize Risk of ARP Protocol Poisoning Attack
Word Count:	5255

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are encouraged to use the Harvard Referencing Standard supplied by the Library. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action. Students may be required to undergo a viva (oral examination) if there is suspicion about the validity of their submitted work.

Signature:	
Date:	15th September 2017

PLEASE READ THE FOLLOWING INSTRUCTIONS:

1. Please attach a completed copy of this sheet to each project (including multiple copies).
2. **You must ensure that you retain a HARD COPY of ALL projects**, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer. Please do not bind projects or place in covers unless specifically requested.
3. Assignments that are submitted to the Programme Coordinator office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

A Framework of Docker Containers to Minimize Risk of ARP Protocol Poisoning Attack

Bhagirath Purushothama
x15047211
MSc Research Project in Cloud Computing

15th September 2017

Abstract

As Docker is slowly transforming the virtualization and deployment world, it is essential it does not have any security loopholes. But due to resource and kernel sharing with the host and no inbuilt authentication mechanism Docker greatly increases the surface area of the attack on the host machine. This paper addresses ARP poison attacks among docker containers, which is one among the many security issues in Docker. As there is no definitive way of creating a container, this paper suggests a way to effectively detect an ARP poisoned container by creating containers in a framework.

1 Introduction

In the world of datacenters and virtual machines where technology has enabled one to spin up an instance of a server or an entire machine up in minutes, we are always looking for more. More speed, more efficiency, more performance, etc. In this relentless search for more out of a virtualization software a new paradigm of virtualization was designed on share the resources and kernel of a host machine but can work and behave like a virtual machine and spin up deployable server environments in seconds. This is the Linux Container - LXC. LXC have been around for two decades now. But only in recent years they have been considered to be used as an alternative for commercially successful virtualization solutions. This was possible due to tools such as Docker, Mesos, rkt, Solaris Containers, CSDE(Commercially Supported Docker Engine) etc. These tools enabled the use of container as a deployable environment. Thus, effectively creating a virtualized environment within the realms of a host machine utilizing the resources available to the host without harming the host (well, almost!).

Docker, though having the provision for security features does not have any of it implemented ab-initio. One such security loophole is the possibility of ARP poisoning and ARP spoofing. To begin with Dockerfiles are, by default, build-able by any user and do not need any special permissions. Hence, a Docker image can be built from scratch just by copying a Dockerfile from a USB or a disk. Once the image is built, a user can push the image into the Dockerhub, All the while with no checks or security authentication of any sort. Due to this there is a chance that corrupt or poisoned images can find its way into a live production environment.

Docker containers are miniscule instances of an operating systems having only the bare minimum files required to run a process. Due to this, it essentially does not have any authentication and validation, hence, a container can be easily influenced and induced in a live environment to spoof on the activities of other containers on the machine. This is a major persisting issue in the current IT world (Combe et al.; 2016).

ARP is a basic protocol that works on the Data-link layer in the OSI model. ARP is not a reliable protocol meaning, it does not wait for a response for a respective request packet in a network. It just broadcasts the node information and captures the incoming node information of the other nodes in a network. This enables any intruder to easily create a pseudo node and spy on the network traffic. Hence, to overcome this among docker containers where the security is one of the highest concerns a framework of Docker containers has been proposed in this paper which can detect any ARP intrusion among many containers. More detailed description and discussion follows in further sections.

To begin with let us first understand what is Docker and how it works. Matthias and Kane (2015) is a guide to work with Docker and probably a good place to start understanding Docker. According to this book, more than understanding, what Docker is, we need to know what Docker is not. Docker is not a commercial virtualization solution like VMware, KVM, etc. (this can be confusing but, bear with me). Docker is not Cloud platform like Openstack, AWS, cloudStack, etc. Docker neither is configuration management tool like Puppet, chef, etc. But, Docker has the flexibility of the all of the above technologies. That is, it has the ability to provide a virtualization solution, it can form a cloud platform, it can also can be used as a configuration tool and deployment tool. In short Docker is all of it combined in one. Basically, Docker is just a process, but a powerful one, which shares the resources of the host machine.

2 Related Work

Before we dive right into the details of the project we need to research how many others have already worked on the similar topic as mine. Also, we need to understand and validate the existing problem. And if the problem exists then what is the possible solution for it and validate the solution as well. Without further ado lets dive right into the different sections of the literature review. [1]

This entire section is divided into four subsections. The subsection 2.1 describes and defines what Docker is and how is it changing the very ways in which applications are being developed in the current technological world. The subsection is intended on highlighting the positive aspects of Docker and other containerization software. Operating system virtualization being one of the simplistic and lightweight virtualization solutions, is the basis of Docker, Mesos and other LXC (Linux Containers).

The second subsection 2.2 deals with the Address Resolution Protocol. Various papers have been written on the ARP and how they affect the network security. ARP is a simple and non-reliable protocol which works at the Data link layer (layer 2) of the OSI model. ARP works on the principle of address resolution and reverse address resolution. This subsection intends on understanding how the ARP works and what are flaws which make the ARP an unreliable protocol.

After understanding the Docker and ARP, the 2.3 subsection deals with ARP poisoning or spoofing attack in Docker containers. This subsection highlights the how ARP

[1] Like this one: `http://www.ncirl.ie`

spoofing can affect the security of a system running Docker containers. And in what way is the system affected by the spoofed containers.

And finally, the subsection 2.4 highlights various solutions in which ARP spoofing and poisoning attacks can be contained in a network of Containers. This section highlights the solution in discussion in this paper and how effectively it dispels any threat of ARP poisoning attack.

2.1 Docker: An overview and its extent of usage

Lets begin with the big question. What are containers? According to Joy (2015) Isolating the processes to access and utilize the resources allocated only to the respective process environment without affecting the other processes - thats a simple definition of LXC or LINUX Containers. LXC are basically processes running independently utilizing the resources allocated inside the respective namespace. But the major advantage of containers is that they have their own network interface and can connect and interact with the other systems just as another system. Some of the advantages of the LXC are that they highly portable and lightweight, need minimal configuration and, the major advantage, they share the host system OS kernel and are not affected by the underlying host processes. This makes the containers a very ideal substitute for other virtualization rivals like hypervisors.

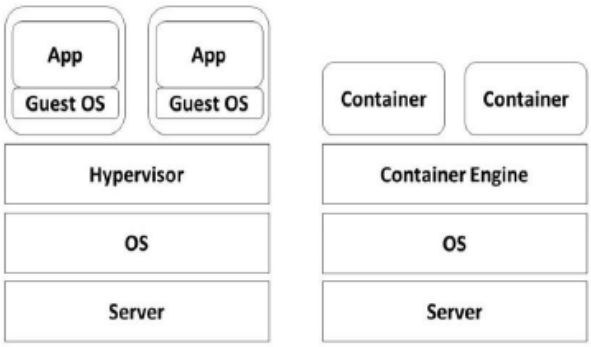

Figure 1: Virtual Machines vs Containers (Joy; 2015)

The figure 1 depicts the comparative depiction of a conventional virtual machine vs a container solution such as Docker. A typical VM makes use of hypervisor and isolates the hardware from the operating system. Whereas a containerization effectively isolates the host operating system from the containers processes. Hence, containers are also called operating system level virtualization. Containers mainly work with the help of namespace and control groups (cgroups for short). Namespaces takes care of the environment in which a process is running and the limits defined for the respective process. And cgroups manages the access and control of the process and helps isolates the process from each other. The figure 2 depicts the typical architecture of a Docker Container with respect to the namespace and cgroup.

Some popular containers software are Docker (leverages LXC), Mesos and Kubernetes (technically it is a tool to manage cluster of containers). And as popularity of the containers technology grows more, more applications are finding usefulness in the containerizing than sticking to the conventional virtual machines and hypervisors. In my study on

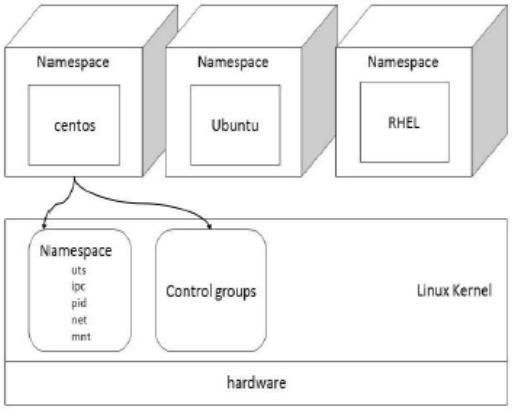

Figure 2: Namespace and ControlGroups (Joy; 2015)

Docker container found papers which were using Docker containers in their projects. A few of the papers are listed below.

Anderson (2015) points out the problems that Docker addresses with respect to a Virtual Machine. According to this paper Docker is an effective tool which enables a developer not to worry about changing and configuring the system environment and focus only on developing the application without messing up the environment. He points out that Docker is most helpful for the DevOps team as lesser configuration is needed and Docker helps isolate the process of coding and configuration. And a single system can be used as a host to more than one application without affecting the each other. This notion is concurred by the Choudhari and Mhatre (2015). In this paper, we can see why containers have an upper hand over the traditional solutions for virtual methods.

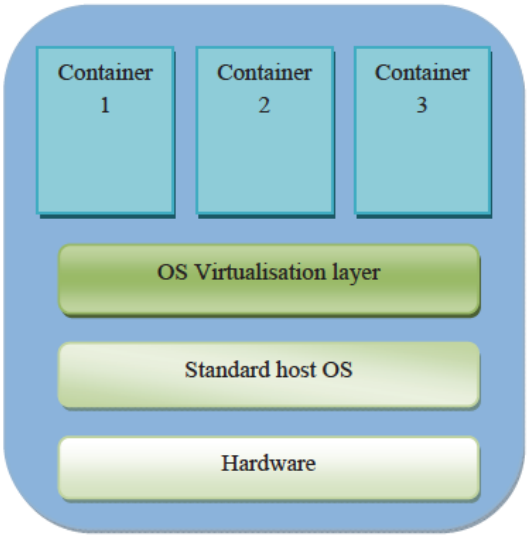

Figure 3: A Docker container architecture (Choudhari and Mhatre; 2015)

The figure 3 depicts the architecture of a Docker container. The usefulness of Docker can also be seen in the field of research and development. This is proven by Boettiger (2015).

The idea of the Boettiger (2015) is also confirmed by Di Tommaso et al. (2015) where Docker containers are used to set up the environment for research with Genomic

Pipelines. Typically, genomic pipelines are of experimental nature as often, third party software are used. Apart from this, they are constantly changing keeping pace with the technological changes. Hence, reproducing the particular version or a setting up of environment becomes increasingly difficult. Hence, Docker containers provide the solution by containerizing the particular version/set-up without affecting the other newer or older versions of the same. Knoth and Nst (2017) as well, speaks on similar lines where Docker containers are used to resolve the issue of reproducibility of environments for Geographic Object-Based Image Analysis (GEOBIA) which mainly use proprietary software but is now gradually migrating to Open-Source Software.

As the topic of the section suggests, Docker is finding its way into main stream technological solution. Usage of Docker in datacentre is one such example. This is supported by Guan et al. (2017;2016;). In this paper, the author convincingly portrays the effectiveness of usage of Docker in datacentre. The author(s) have proved by experimentation that the algorithm for resource allocation using Docker containers performs better than the conventional methods.

Another way of using Docker containers is by clustering and managing multiple containers and making them work as a single unit. The paper Sallou and Monjeaud (2015) deals with the one such utility where Docker containers are used to implement mechanism for executor plug-in and custom scheduler based on open source software for batch scheduling.

Map-Reduce is the most sought-after technology for crunching big data. Docker can be used as a good alternative even for the Map-Reduce as well. This is proved in Xavier et al. (2014). Here the author experimentally proves that Map-Reduce is more effective with containers as the CPU utilization and systems resources are used comparatively lesser when compared to the traditional VM software.

The bottom line of this section being - Docker is effective, efficient, portable, ideal and faster than the traditional or conventional virtualization software such as HyperV, KVM, XEN, RHEV, etc. Though the traditional systems are robust and very secure in comparison to Docker, they come nowhere even close in terms of speed, efficiency and portability. And Docker has the potential to replace any of the leading virtualization solution.

2.2 ARP: An overview. What is spoofing?

We already know that Internet makes communication possible between two separate computer systems. In network, the Internet is divided into seven layers under the OSI model. And each layer speaks with the immediate layer above or below itself. These layer 'speak' with each other using protocols. Hence, each layer has its own set of protocol. The protocol in discussion here is Address Resolution Protocol in short ARP. ARP works in the Data-link layer of the OSI model and is an unreliable protocol. Meaning, after sending a packet of information as request on a Local Area Network, it does not wait for a response for that packet. The main purpose of ARP protocol is to collect and maintain a table of IP address and physical Media Access Control Address pair of every system in a network and store it. ARP does this in four simple steps; ARP Request, ARP response and RARP request and RARP response. This is done by broadcasting the IP address as well as the Physical address of the machine in a network. Once the packet of information is broadcast, the node does not wait for a response. If the node needs to route or send information to another machine in its network, it refers it's ARP table with the help of

RARP (short for Reverse-ARP) and finds out the position of a node in the network and forwards the respective packet to it. The Figure 4 depicts the typical structure of an ARP protocol.

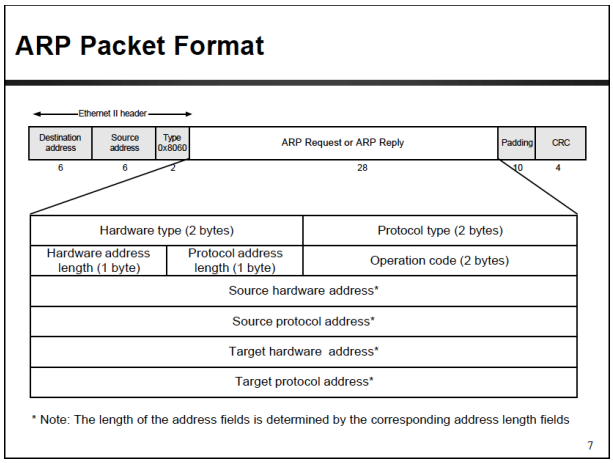

Figure 4: Structure of an ARP packet (*Address Resolution Protocol (ARP)*; n.d.)

As we already know the main functionality of ARP protocol is to map the physical MAC address in the physical layer to the IP address of the nodes in data-link layer. The physical MAC address of the nodes, a 48-bit value with 6-octets, is mapped to a IP address of the respective nodes, a 32-bit value with 4-octets. This can be referred in the ARP packet structure in Figure 4. Usually, the 32-bit value is associated to the LAN and the respective nodes in it. Meaning, each node has all the values associated to the nodes in that particular LAN. And ARP cache stores these value for faster retrieval. The most common use of ARP cache can be seen in default gateways and physical routers in the network.

That being the case, it becomes very easy for a hacker node to impersonate a pseudo node. Meaning, an impersonator node broadcasts an ARP packet with false information of another node (the node usually being the default gateway or a router). This enables all the traffic to redirect to the impersonated node and the Internet packet travel via the pseudo node. Rahman and Kamal (n.d.) clearly explains this with the help of a diagram as depicted in the Figure 5 below.

Moon et al. (2014) and (Zdrnja; 2009) have explained the issues in ARP in detail. According to the author(s) following are the major drawback of the ARP protocol.

- Statelessness: ARP and RARP are both unreliable protocol and work in a simplistic manner. Meaning, they do not have any states. The only broadcast or receive the broadcast packets of data.
- No Authentication: This is a major hurdle as any new entry in the network can easily gain access to the network details or the ARP cache of the LAN.
- ARP cache auto-update: The ARP cache table consisting of all the entries are updated without any verification based on the packets received.

The above drawbacks lead to the LAN being prone to various forms of intrusion and harm. Some of which have been listed below.

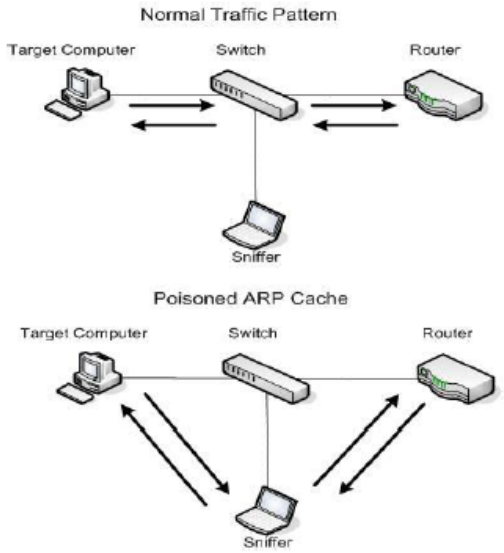

Figure 5: ARP cache poisoning attack (Rahman and Kamal; n.d.)

- Host block: Once an intruder is able to gain access into the LAN it is very easy for the attacker to redirect the packets isolating one or many nodes. This leads to the host being blocked to provide service for the requests.
- Host impersonation: Once, the user is able to isolate a particular system he can easily pose as a pseudo node without being detected. This leads to a host being impersonated by another impostor machine.
- Man-in-the-middle attack: Once, the user is able to impersonate the behaviour of a system. it becomes increasingly difficult for the network to detect the presence of an impostor as there is very little difference between the impostor node and actual real node. This makes it easy for the impostor to launch his own attack. On detection, the real node will be detected rather than the impostor.

Summary of the section is that ARP is a necessity but it is not smart enough to verify itself and can pose a serious threat to systems connected in a network. We cannot totally ignore the protocol as it works in the bottom layers of the OSI network model.

2.3 Security issue and ARP Spoofing in Docker Containers

As all seemingly perfect systems, even Docker has a catch. LXC Container is a Free Open-Source Software (FOSS). Meaning every LINUX experienced person can not only easily understand and work, but also develop his/her own techniques and software applications using this tool. Meaning, there are only guidelines which needs to be kept in mind when working with Docker, and no prescribed manner in which to standardize the usage of Docker. This leads to varied and distinct methodologies for Docker usage. All these leads to one logical conclusion, if there are more than one ways of working on it there is also more than one way of breaking it. Standardizing the usage of Docker reduces the security loops in the software and narrows the surface for security breach in the system. In this section, we will be discussing the possible security drawbacks of Docker tool.

The fundamental question any developer would ask is, why do we need Docker when there are alternatives available, if not better, which are robust and provide high security?

Why would technological enthusiasts want to make use of Docker? Combe et al. (2016) answers exactly that. The author(s) has researched that though very efficient and effective than its other virtualization peers, Docker comes with its fair share of security holes. The following are the ones mentioned in the paper.

Manu et al. (2016b) aims at bringing out the underdeveloped side of Docker. The author(s) of this paper have clearly mentioned as Docker daemon works along with and is connected to the host kernel, it gives a direct access to the host machine itself. This leads to attack surface widened attack surface every time a new container or image is loaded onto the Docker host. The conclusion of this paper clearly states that Docker technology is in its infancy yet and the security patch up that needs to be implemented is possible bigger than the technology itself.

According to Manu et al. (2016a) the security aspects of Docker containers is literary unchartered territory. The LXC and containerization has been around for two decades now but Docker combines use of container technology and containers configuration in a single bundle. But, the problem being, the tool needs to be flexible and must provide as much configuration management as possible without tarnishing the ease of use. So, technically speaking, there are no security features inbuilt like other tool. But, they can be configured to implement one. This paper is addressing one such problem of security problems in Docker containers with respect to ARP spoofing and ARP poisoning attack.

During, the course of my dissertation, I got in touch with the official Docker channel. Grattafiori (2016) is white paper provided by them to understand the security features and hardening the Docker containers. According to the paper, Docker has enough security feature but it does not implement them by default. Hence, to overcome the issue of ARP poisoning, we need to use of a customized bridge which can be programmed. This is discussed in detail in further sections.

As stated earlier, the problem of intrusion and poison attack is common among a system. As Docker containers are treated as individual systems in a network they are not immune to these attacks. With regards to Docker containers security very limited papers are available and thus more research is necessary in this topic.

2.4 Existing solution and proposed solution

There is more than one way to overcome ARP poisoning attacks. The most common ways are to monitor the ARP cache of the machines in a network, checking for a flood of gratuitous packets. Paper such as Rahman and Kamal (n.d.), Lv and Li (2011), Kavan et al. (2014) and Wu et al. (2016)propose new and exhaustive methods to detect and nullify ARP poisoning attacks. Not only systems, ARP attacks can greatly influence the websites and their cross scripts also. (Zdrnja; 2009) and Zhang et al. (2012) discuss this in detail. Cross scripts are JavaScript that are captured in a network and replaced with malicious scripts or viruses that run by themselves on a computer harming or bugging the system for a long period. Kim and Huh (2011) proposed ways of detecting the intrusion over a domain. Detecting a phishing attack over an entire domain is not as effective as detecting over a network. The paper has proposed a phishing detection technique over an entire DNS by examining the network performance characteristics. The author(s) has examined four classification algorithms: -

- Linear Discriminant Analysis
- Naive Bayesian

- K-Nearest Neighbour
- Support vector machine

Among these the most effective one for detecting the phishing in a domain is K-nearest with 99.4% detection rate.

Similar issue of intrusion and poisoning can also be seen in Peer-to-Peer Network Protocol(P2P). P2P supports scaling and is a very robust protocol but it does not detect any intrusion by default. Ismail et al. (2017) discusses this issue and proposes a technique to detect a poisoned node. The paper proposes an algorithm to detect and prevent intrusion in P2P protocol.

Cha et al. (2017) is a unique paper where DoS attacks and eavesdropping of packets over VoIP and mVoIP can be detected and stopped by using Docker as a tool. Though not completely relevant this paper brings out the idea that Docker is actively being used in cloud based technologies and is susceptible to attacks over the network as a result of its raw implementable nature.

As the ARP is somewhat basic and (can be called primitive) protocol, it might very much need upgrading. (Nam et al.; 2010) suggests a new improved ARP to detect and overcome intrusion detection in a network. Due to non-authenticity and unreliable nature of ARP system in a network are vulnerable to attack. This can be overcome by improving the ARP to detect and prevent Man-In-The-Middle(MITM) attacks. The mechanism proposed in this method is based on a voting where each node rates the node entries in its ARP cache and is always self-aware of the adjacent node info. This reduces the MITM attacks greatly. Also, Min Su et al. (2014) discusses a new and improved ARP protocol (on similar lines of (Nam et al.; 2010)) which works on the principle of routing trace. This improved version of ARP is backward compatible and can be deployed instead of the existing one incrementally. The paper Trabelsi and Shuaib (2008) also proposes an improved Man-In-The-Middle attack detection scheme for switched networks.

Lastly, Chen et al. (2016) discusses the security for an entire cloud system network as a whole to secure the critical cloud infrastructure. As we analyse, it can be observed that a little research has been done on detecting and preventing the poising attacks in Docker. I've tried to implement a framework of containers that can be used to detect and prevent the ARP poisoning and intrusion.

The gist of this subsection being, there are solutions to prevent an ARP poisoning and ARP spoofing attacks in network. But in Docker there is no such software or mechanism. Hence, we will be discussing the framework of containers to detect an ARP poisoned container.

3 Methodology

To begin with the project was started with an assumption that a framework of containers all use individual network connection from the host network bridge. And initially all the containers had to be connected with each other to create a framework of docker container. So by initial assumption, Our framework of containers will look as it is depicted in the Figure 6.

Where, the master container is the parent container and all other containers are laid out in the form a binary tree. And each container is allowed to have one parent and two children only. All the containers are logically connected to communicate only in the flow

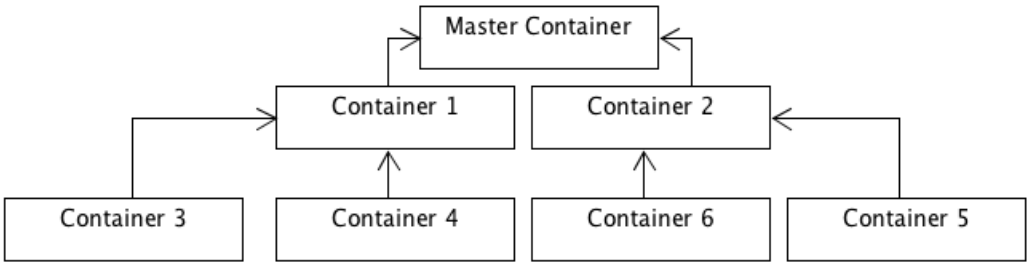

Figure 6: Framework of Containers

of the binary tree with the help of iptables. And also, each container acts as a monitor to it's immediate child node and keeps track of the packet data being transmitted to it's child. If one among the tow nodes of are collecting more packets than the other, then parent node is suspicious about the child node and informs the master node to isolate the particular container from the framework.

There are two major issues with the above approach. First, the assumption on that docker container directly connect with the ethernet was false (Grattafiori; 2016). Docker has a dedicated bridge docker0. Second, even if we find a way to connect the containers in the form of a binary tree, the proposed framework would be ineffective as the size of the tree increases the number of levels on the tree will increase and transmission of data packet will be slower.

According to Docker white paper Grattafiori (2016) and the official website Docker (2017) it was clear that docker creates its own default bridge 'docker0' which acts as a connection center for all the containers hence, all the containers communicate via the 'docker0' bridge. As all the networking is through a single bridge it is better to use the existing bridge to keep track of the packet movements. The below Figure shows a bridge connecting three Docker containers to the Ethernet via a proxy server. The proxy server, the default bridge and the NAT connection of all the three are a part of Docker daemon and are installed together with the Docker-machine.

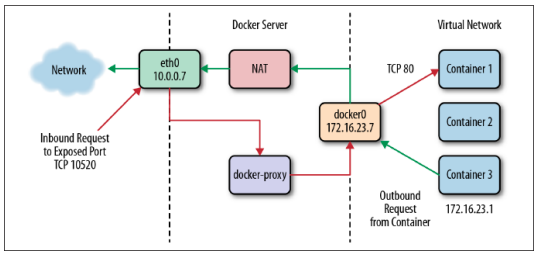

Figure 7: A Docker bridge connecting containers to Ethernet

Following the above pattern I came up with steps to create a custom framework making use a custom bridge and subnet masks.

1. Create a custom bridge for Docker

2. Create a custom subnet mask under the new bridge

3. Create containers with static IP address and assign them under the new custom subnet mask

4 Implementation

The following are commands and code to create a custom bridge in Docker and assign static IP addresses to every container while connecting to the bridge.

The first step is to create a custom bridge in Docker and add that bridge to be used by another container.

Before we begin, we need to stop the currently running Docker daemon either manually or by stopping it in background.

Once the Docker daemon is stopped, configure and create a new bridge as follows

```
sudo brctl addbr custom_bridge_0

sudo ip addr add 172.132.1.1/24 dev custom_bridge_0

sudo ip link set dev custom_bridge_0 up
```

Once configured, confirm the configuration settings

```
ip addr show custom_bridge0
```

It should output the following

```
bridge0: <BROADCAST,MULTICAST> mtu 1500 qdisc noop state UP group default
    link/ether 66:38:d0:0d:76:18 brd ff:ff:ff:ff:ff:ff
    inet 172.132.1.1/24 scope global custom_bridge_0
       valid_lft forever preferred_lft forever
```

We are good to go with the new bridge. Configure Docker to use the new bridge instead of the default one. Open the file /etc/docker and edit the file daemon.json to have the following entry.

```
{
‘‘bridge’’: ‘‘custom_bridge_0’’
}
```

Restart the Docker daemon with the following command

```
sudo docker service restart
```

And once Docker is restarted, we can delete the now unused bridge ‘docker0’

```
sudo ip link set dev docker0 down

sudo brctl delbr docker0

sudo iptable -t nat -F POSTROUTING
```

We should be able to use the custom bridge for docker. We can verify this by

Once we have a custom bridge, the next step is to assign static IP address to every container that we create. This is one of the approaches. Another and a more secure approach is that we can create a custom network with user defined subnet mask and assign the containers to it.

Once we have the custom network we can create containers with static IP and assign it to the new network.

```
dvm@dvm-machine:~$ sudo brctl show
bridge name     bridge id               STP enabled     interfaces
br-ec6e2bda8301         8000.0242d54a233c       no
custom_bridge_0         8000.000000000000       no
```

Figure 8: A custom bridge

```
dvm@dvm-machine:~$ docker network create --subnet=172.18.0.0/16 custom_network
49e5e6ecbc71bd74ba24933f18d12f80dd75fafe94ad827b46e22df020792ad7
dvm@dvm-machine:~$ docker network ls
NETWORK ID          NAME                DRIVER              SCOPE
e5e45bbb0ef7        bridge              bridge              local
49e5e6ecbc71        custom_network      bridge              local
d95ff5a167c9        host                host                local
f75366ca8233        none                null                local
```

Figure 9: A custom network

```
dvm@dvm-machine:~$ docker run --net custom_network --ip 172.18.0.5 -it alpine
/ # ifconfig
eth0      Link encap:Ethernet  HWaddr 02:42:AC:12:00:05
          inet addr:172.18.0.5  Bcast:0.0.0.0  Mask:255.255.0.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:16 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:2070 (2.0 KiB)  TX bytes:0 (0.0 B)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)
```

Figure 10: A container with static IP

5 Evaluation

In this section, we will be testing and evaluating the framework by manually creating containers and testing them against ARP attacks. Two scenarios are considered

1. Containers outside the framework
2. Containers inside the framework

In both the cases we will be introducing a rogue container to capture details of another container. The subsequent behaviour of the system would be recorded and the results would be displayed.

5.1 Containers outside the framework

Creating 5 containers outside the framework for the purpose checking for ARP spoofing

```
docker run --name dynamic_container_1 -d -it alpine
```

```
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS              PORTS               NAMES
c686ed4c02c1        alpine              "/bin/sh"           11 seconds ago      Up 10 seconds                           dynamic_container_7
42f34b3df6fb        alpine              "/bin/sh"           16 seconds ago      Up 15 seconds                           dynamic_container_6
cbf73edaa90b        alpine              "/bin/sh"           33 seconds ago      Up 32 seconds                           dynamic_container_5
5e90d57042b9        alpine              "/bin/sh"           38 seconds ago      Up 37 seconds                           dynamic_container_4
9ca8b958e3a7        alpine              "/bin/sh"           43 seconds ago      Up 42 seconds                           dynamic_container_3
0f37b117258e        alpine              "/bin/sh"           47 seconds ago      Up 46 seconds                           dynamic_container_2
94248ec40fcf        alpine              "/bin/sh"           54 seconds ago      Up 53 seconds                           dynamic_container_1
```

Figure 11: List of containers outside the framework


```
"Containers": {
    "3b15bb4d77125622b4824aa9786517078fc27d66884f1f31a248acfaad05be43": {
        "Name": "dynamic_container_5",
        "EndpointID": "406eda123e7172ee13b92c44fa7f7195ac32a477d40d9013c372f3843c93da27",
        "MacAddress": "02:42:ac:84:01:06",
        "IPv4Address": "172.132.1.6/24",
        "IPv6Address": ""
    },
    "6f2975528aabbc41fb880bcdfa04b1197c0da8e12723f85dc44076e746e71225": {
        "Name": "dynamic_container_2",
        "EndpointID": "1e9b3102acef39b43602b44b1b01c2f875042bf8a981d3914aae636da5069792",
        "MacAddress": "02:42:ac:84:01:03",
        "IPv4Address": "172.132.1.3/24",
        "IPv6Address": ""
    },
    "74087f4946af2d5345e9584d580a653ef03594bb66f1b24b8435e12f8adede59": {
        "Name": "dynamic_container_4",
        "EndpointID": "837cf0480a625bce0acba0db2c70f9ce48884125da7328f63351c52b31918e58",
        "MacAddress": "02:42:ac:84:01:05",
        "IPv4Address": "172.132.1.5/24",
        "IPv6Address": ""
    },
    "b143de8716855b4b6643a563c9a5774acb64502b18d85b3aae797deb1c3f834f": {
        "Name": "dynamic_container_1",
        "EndpointID": "3fde18ef9431475b252eaac5a35284be2de2f41143c6077782086aa6080ccc81",
        "MacAddress": "02:42:ac:84:01:02",
        "IPv4Address": "172.132.1.2/24",
        "IPv6Address": ""
    },
    "e02eb3a02ad3854a0fa0c164b9a5fe5e4c55d256270c3b782f470b3262ec7a42": {
        "Name": "dynamic_container_3",
        "EndpointID": "ae1911252b75b72f9e78d62e96c7bf2b20d5372eff06880cb44833b83af8eace",
        "MacAddress": "02:42:ac:84:01:04",
        "IPv4Address": "172.132.1.4/24",
        "IPv6Address": ""
    }
},
```


Figure 12: Container details which are out of Framework

After creating 5 such containers (Figure 11)
On inspecting the containers (Figure 12)
If we stop any one of the containers from the framework, and add a new container the default bridge assigns an already used IP address to the new container


```
"fd6ffbec1e016132f415f7df3eba786cbc41816d6b36097986bb3d73c0db127c": {
    "Name": "dynamic_container_6",
    "EndpointID": "1a9a76cab2814eb362b564be9ae568f9c60431fbdf78f5fc31afbe3999202937",
    "MacAddress": "02:42:ac:84:01:06",
    "IPv4Address": "172.132.1.6/24",
    "IPv6Address": ""
}
```


Figure 13: Container details of the new dynamic container created

This means even though the container is different it is assigned the same IP address and the MAC address and instantly recognized in the ARP table. This can lead a container being induced into the bridge by default with arising any suspicion.

5.2 Containers in the framework

Creating 5 containers inside the framework for the purpose checking for ARP spoofing

```
docker run --name static_container_1 --net custom_network
--ip 173.130.1.2 -d -it alpine
```

After creating 5 such containers,

```
64d454c2e1d6   alpine   "/bin/sh"   About an hour ago   Up About an hour   static_container_4
7a5d94c572f2   alpine   "/bin/sh"   About an hour ago   Up About an hour   static_container_3
647a9c038df5   alpine   "/bin/sh"   About an hour ago   Up About an hour   static_container_2
9b2b5f760e2b   alpine   "/bin/sh"   About an hour ago   Up About an hour   static_container_1
```

Figure 14: List of containers inside the framework

On the other hand, all the containers in the framework have fixed IP address and once a container is removed its IP address is not used again and the subsequent container being added will have a new IP address.


```
"Containers": {
    "647a9c038df51b3304e842812b63ad4cac6f137d990d1b59efff19a0337c491e": {
        "Name": "static_container_2",
        "EndpointID": "43f68e4c70c0db900dc88f785a7182f557583ac34c1ee8d2814b31c855f24f01",
        "MacAddress": "02:42:ad:82:01:03",
        "IPv4Address": "173.130.1.3/25",
        "IPv6Address": ""
    },
    "64d454c2e1d6101166a879edd2628659b1d86a47a3e97d2716992bbaa665b73d": {
        "Name": "static_container_4",
        "EndpointID": "9f1530fcd78acfdb02d1a08d5317350650cc266f95b158771838fe524853c6ea",
        "MacAddress": "02:42:ad:82:01:05",
        "IPv4Address": "173.130.1.5/25",
        "IPv6Address": ""
    },
    "7a5d94c572f2635ec7a5eb3a9321f27bd31cc76455b41a502a41c81e855e2868": {
        "Name": "static_container_3",
        "EndpointID": "393a663c6f1578f29ceb4a961e405cccb80460b7bcdd4ee9ba16cd7a0bdb8f70",
        "MacAddress": "02:42:ad:82:01:04",
        "IPv4Address": "173.130.1.4/25",
        "IPv6Address": ""
    },
    "9b2b5f760e2bd07e89e2018e3c677646cef5ebc311e44f1e6984f7eaff1b5f5f": {
        "Name": "static_container_1",
        "EndpointID": "a9a627d7197c253ad7add9cb186b17ef36844f7713abbc05632d0e278cb83adf",
        "MacAddress": "02:42:ad:82:01:02",
        "IPv4Address": "173.130.1.2/25",
        "IPv6Address": ""
    },
    "e79448deb1fd6b2888b92d04f7479b66b82ba5d7e37e287788352dfb89af373e": {
        "Name": "static_container_5",
        "EndpointID": "fe14237084977cef090b24eaa1bfc7db3bff833861a78b0d340f2d2f6d402a0a",
        "MacAddress": "02:42:ad:82:01:06",
        "IPv4Address": "173.130.1.6/25",
        "IPv6Address": ""
    }
},
```


Figure 15: Container details which are inside the Framework

While stopping and adding a new container subsequently an IP address is configured along with it. This prevents the reuse of IP address


```
"a47cfdb979d35414470914ae21fc831d842999949457bb0295d7d9280f20d67a": {
    "Name": "static_container_6",
    "EndpointID": "fa46cefd5614df8c42ac8b0cbbb82eb7867af597a18f0a56553302000d459d93",
    "MacAddress": "02:42:ad:82:01:07",
    "IPv4Address": "173.130.1.7/25",
    "IPv6Address": ""
}
```


Figure 16: Container details of the new static container created

As the result shows that though the MAC address is the same but the IP address are different. This leads to a new entry in the ARP cache table which authenticates that the new container is not a copy of another container and is not a potential ARP poison threat.

5.3 Discussion

From the above experiment, it was found that duplicate ARP cache records can be prevented over a network of containers when the containers are in the framework. Or to be more precise, the containers outside the framework have more chance of being the possible poisoned container. Establishing the proper framework of containers enables the containers to have a unique entry in the ARP cache records rather than taking the place of a pre-existing container. Hence, it can be said that the possibility of poisoned containers finding its way into the production environment is considerably reduced, in turn making the container environment more secure for the applications.

6 Conclusion and Future Work

The framework of containers in discussion here resolves the problem of ARP poisoning attack among docker containers. The framework disables duplicate entry in ARP cache preventing any threat in the form of ARP poisoning by creating a static entry in ARP table even before an intruder can spoof the identity of another container. This makes the entire network of containers secure and reliable. A secure network additionally needs lesser security implementations on the hosting machines which will reduce the cost of maintaining and setting the server environment.

The future work could include the automation of creating custom bridge along with static IP address for containers in a single command in Docker by contributing to the source code to add an option of private and secure bridge creation along with static IP address for the containers. Another possible enhancement could be to integrate an ARP cache monitor daemon to constantly monitor the Docker network bridge to detect a duplicate entry in the ARP cache and instantly isolate the respective container from the network.

References

Address Resolution Protocol (ARP) (n.d.). Accessed: 2017-03-01.
URL: *www.cs.virginia.edu/čs458/slides/module06-arpV2.pdf*

Anderson, C. (2015). Docker [software engineering], *IEEE Software* **32**(3): 102–c3.

Boettiger, C. (2015). An introduction to docker for reproducible research, *ACM SIGOPS Operating Systems Review* **49**(1): 71–79.

Cha, B., Kim, J., Moon, H. and Pan, S. (2017). Global experimental verification of docker-based secured mvoip to protect against eavesdropping and dos attacks., *EURASIP Journal on Wireless Communications and Networking* **2017**(1): 1 – 14.
URL: *http://search.ebscohost.com/login.aspx?direct=truedb=eoahAN=41653973site=pfi-live*

Chen, Z., Xu, G., Mahalingam, V., Ge, L., Nguyen, J., Yu, W. and Lu, C. (2016). A cloud computing based network monitoring and threat detection system for critical infrastructures, *Big Data Research* **3**: 10 – 23. Special Issue on Big Data from Networking Perspective.
URL: *http://www.sciencedirect.com/science/article/pii/S2214579615000520*

Choudhari, S. and Mhatre, M. (2015). Docker : run, ship and building applications, *International Journal of Latest Trends in Engineering and Technology* **6**(2).

Combe, T., Martin, A. and Pietro, R. D. (2016). To docker or not to docker: A security perspective, *IEEE Cloud Computing* **3**(5): 54–62.

Di Tommaso, P., Palumbo, E., Chatzou, M., Prieto, P., Heuer, M. L. and Notredame, C. (2015). The impact of docker containers on the performance of genomic pipelines, *PeerJ* **3**: e1273.

Docker, I. (2017). Docker reference architecture: Securing docker ee and security best practices - docker, inc., `https://success.docker.com/Architecture/Docker_Reference_Architecture%3A_Securing_Docker_EE_and_Security_Best_Practices`. Accessed: 2017-08-14.

Grattafiori, A. (2016). Ncc group whitepaper understanding and hardening linux containers.
URL: *https://www.nccgroup.trust/globalassets/our-research/us/whitepapers/2016/april/ncc_group_understanding_hardening_linux_containers-1-1.pdf*

Guan, X., Wan, X., Choi, B.-Y., Song, S. and Zhu, J. (2017;2016;). Application oriented dynamic resource allocation for data centers using docker containers, *IEEE Communications Letters* **21**(3): 504–507.

Ismail, H., Germanus, D. and Suri, N. (2017). P2p routing table poisoning: A quorum-based sanitizing approach, *Computers Security* **65**: 283–299.

Joy, A. M. (2015). Performance comparison between linux containers and virtual machines, *2015 International Conference on Advances in Computer Engineering and Applications*, pp. 342–346.

Kavan, D., kodov, K. and Klma, M. (2014). Network-based intrusion prevention system prototype with multi-detection: A position paper, pp. 1–9.

Kim, H. and Huh, J. H. (2011). Detecting dns-poisoning-based phishing attacks from their network performance characteristics, *Electronics Letters* **47**(11): 1.

Knoth, C. and Nst, D. (2017). Reproducibility and practical adoption of geobia with open-source software in docker containers., *Remote Sensing* **9**(3): 1 – 24.
URL: *http://ezproxy.ncirl.ie/login?url=http://search.ebscohost.com/login.aspx?direct=trueAuthType=ip,cookie,uiddb=a9hAN=122025585site=eds-livescope=site*

Lv, H. and Li, H. (2011). Research on intrusion recognition and tracing under attack and defense confront environment, *2011 International Conference on Information Management, Innovation Management and Industrial Engineering*, Vol. 3, pp. 209–214.

Manu, A. R., Patel, J. K., Akhtar, S., Agrawal, V. K. and Murthy, K. N. B. S. (2016a). Docker container security via heuristics-based multilateral security-conceptual and pragmatic study, *2016 International Conference on Circuit, Power and Computing Technologies (ICCPCT)*, pp. 1–14.

Manu, A. R., Patel, J. K., Akhtar, S., Agrawal, V. K. and Murthy, K. N. B. S. (2016b). A study, analysis and deep dive on cloud paas security in terms of docker container security, *2016 International Conference on Circuit, Power and Computing Technologies (ICCPCT)*, pp. 1–13.

Matthias, K. and Kane, S. P. (2015). *Docker: Up & Running: Shipping Reliable Containers in Production*, " O'Reilly Media, Inc.".

Min Su, S., Jae Dong, L., Young-Sik, J., Hwa-Young, J. and Jong Hyuk, P. (2014). Ds-arp: A new detection scheme for arp spoofing attacks based on routing trace for ubiquitous environments., *The Scientific World Journal, Vol 2014 (2014)* .

Moon, D., Lee, J. D., Jeong, Y.-S. and Park, J. H. (2014). Rtnss: a routing trace-based network security system for preventing arp spoofing attacks, *The Journal of Supercomputing* .

Nam, S. Y., Kim, D. and Kim, J. (2010). Enhanced arp: preventing arp poisoning-based man-in-the-middle attacks, *IEEE Communications Letters* **14**(2): 187–189.

Rahman, M. F. A. and Kamal, P. (n.d.). Holistic approach to arp poisoning and countermeasures by using practical examples and paradigm.

Sallou, O. and Monjeaud, C. (2015). Go-docker: A batch scheduling system with docker containers, *2015 IEEE International Conference on Cluster Computing*, pp. 514–515.

Trabelsi, Z. and Shuaib, K. (2008). A novel man-in-the-middle intrusion detection scheme for switched lans, *International Journal of Computers Applications* **30**(3): 234–243. Copyright - Copyright ACTA Press 2008; Document feature - Equations; Tables; Graphs; ; Last updated - 2016-08-28.

Wu, H., Dang, X., Wang, L. and He, L. (2016). Information fusion-based method for distributed domain name system cache poisoning attack detection and identification, *IET Information Security* **10**(1): 37–44.

Xavier, M. G., Neves, M. V. and Rose, C. A. F. D. (2014). A performance comparison of container-based virtualization systems for mapreduce clusters, *2014 22nd Euromicro International Conference on Parallel, Distributed, and Network-Based Processing*, pp. 299–306.

Zdrnja, B. (2009). Malicious javascript insertion through arp poisoning attacks, *IEEE Security Privacy* **7**(3): 72–74.

Zhang, J., Yang, C., Xu, Z. and Gu, G. (2012). *PoisonAmplifier: A Guided Approach of Discovering Compromised Websites through Reversing Search Poisoning Attacks*, Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 230–253.
URL: *https://doi.org/10.1007/978-3-642-33338-5*$_1$2

A Procedure and Output

```
sudo brctl addbr custom_bridge_0

sudo ip addr add 172.132.1.1/24 dev custom_bridge_0

sudo ip link set dev custom_bridge_0 up

ip addr show custom_bridge0
```

Open the file /etc/docker and edit the file daemon.json to have the following entry.

```
{
 ‘‘bridge’’: ‘‘custom_bridge_0’’
}

sudo docker service restart
```

And once Docker is restarted, we can delete the now unused bridge ‘docker0’

```
sudo ip link set dev docker0 down

sudo brctl delbr docker0

sudo iptable -t nat -F POSTROUTING
```

We should be able to use the custom bridge for docker. We can verify this by

```
dvm@dvm-machine:~$ sudo brctl show
bridge name     bridge id               STP enabled     interfaces
br-ec6e2bda8301         8000.0242d54a233c       no
custom_bridge_0         8000.000000000000       no
```

Figure 1: A custom bridge

```
dvm@dvm-machine:~$ docker network create --subnet=172.18.0.0/16 custom_network
49e5e6ecbc71bd74ba24933f18d12f80dd75fafe94ad827b46e22df020792ad7
dvm@dvm-machine:~$ docker network ls
NETWORK ID          NAME                DRIVER              SCOPE
e5e45bbb0ef7        bridge              bridge              local
49e5e6ecbc71        custom_network      bridge              local
d95ff5a167c9        host                host                local
f75366ca8233        none                null                local
```

Figure 2: A custom network

Building containers based on alpine base image

```
#Base Image

FROM alpine

CMD apt-get update
```

Build and run as many containers as you want by creating a docker-compose file

```
version: '2'

services:
  app1:
    image: alpine
    networks:
      custom_network:
        ipv4_address: 172.16.238.10
  app2:
    image: alpine
    networks:
      custom_network:
        ipv4_address: 172.16.238.11
  app3:
    image: alpine
    networks:
      custom_network:
        ipv4_address: 172.16.238.12
  app4:
    image: alpine
    networks:
      custom_network:
        ipv4_address: 172.16.238.13
  app5:
    image: alpine
    networks:
      custom_network:
        ipv4_address: 172.16.238.14

networks:
  custom_network:
    driver: bridge
    driver_opts:
      com.docker.network.enable_ipv6: "true"
      com.docker.network.bridge.enable_ip_masquerade: "false"
    ipam:
      driver: default
      config:
      - subnet: 172.16.238.0/24
        gateway: 172.16.238.1
      - subnet: 2001:3984:3989::/64
        gateway: 2001:3984:3989::1
```

Then,

```
docker-compose up
```

```
dvm@Docker:~/sandbox/thesis/alpine_image$ docker attach static_container_2
/ # ifconfig
eth0      Link encap:Ethernet  HWaddr 02:42:AD:82:01:03
          inet addr:173.130.1.3  Bcast:0.0.0.0  Mask:255.255.255.128
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:33 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:3863 (3.7 KiB)  TX bytes:0 (0.0 B)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

/ #
```

Figure 3: A container with static IP

```
64d454c2e1d6   alpine   "/bin/sh"   About an hour ago   Up About an hour   static_container_4
7a5d94c572f2   alpine   "/bin/sh"   About an hour ago   Up About an hour   static_container_3
647a9c038df5   alpine   "/bin/sh"   About an hour ago   Up About an hour   static_container_2
9b2b5f760e2b   alpine   "/bin/sh"   About an hour ago   Up About an hour   static_container_1
```

Figure 4: List of containers inside the framework