Mitigating Docker Security Issues

[1]Robail Yasrab

[1] School of Computer Science and Information Technology

University of Science and Technology of China, (USTC), Hefei China.

robail@mail.ustc.edu.cn

Abstract: It is very easy to run applications in Docker. Docker offers an ecosystem that offers a platform for application packaging, distributing and managing within containers. However, Docker platform is yet not matured. Presently, Docker is less secured as compare to virtual machines (VM) and most of the other cloud technologies. The key of reason of Docker's inadequate security protocols is; containers sharing of Linux kernel, which can lead to risk of privileged escalations. This research is going to outline some major security vulnerabilities at Docker and counter solutions to neutralize such attacks. There are variety of security attacks like insider and outsider. This research will outline both types of attacks and their mitigations strategies. Taking some precautionary measures can save from huge disasters. This research will also present Docker secure deployment guidelines. These guidelines will suggest different configurations to deploy Docker containers in a more secure way.

Keywords: Docker; LXC; Container; Virtual Machines (VMs); Linux kernel

I. INTRODUCTION

People have always been doubtful regarding the security of cloud technologies since their inception. People don't trust on cloud computing security aspects entirely as this is still emerging phenomena. In fact, the security vendors design a marketing response for clients by using traditional terms to market their products [1]. Though still a number of issues lurking around but new cloud security solutions are mitigating many security and privacy vulnerabilities.

Currently, cloud technologies are transforming traditional technology with new and more efficient practices [2,3]. One of such example is containers. Containers are considered as future of virtual machines (VM). Container-based virtualization is also known as operating system virtualization, which allows virtualization layer to run as an application within the operating system (OS) [4]. Containers appeared as micro virtual machines, more light weight and more efficient because there is just one operating system (OS) managing all hardware calls.

Currently container model is misrepresented. Apparently container technology looks secure, that it contains all dependences in one package. However it doesn't ensure its security. Container platforms are also vulnerable as other cloud platforms. There are number of threats which are threating from inside and outside [5].

Virtual environments offer a great deal of challenge to run services with great security, particularly in a multi-tenant cloud system [6]. It is already established that virtual machines (VMs) developed through hypervisor based virtualization methods are highly secured as compare to containers. The key reason behind this aspect is that VMs add an additional layer of isolation among the host and the applications.

VM based applications are only allowed to communicate with the VM kernel, not to the host kernel. So, an attacker has to bypass the VM kernel and the hypervisor before an attack can be made to the host kernel. On the other hand, containers model offers application to directly access and communicate with the host kernel as shown in Fig. 1. Such situation permits attacker to directly hit host kernel. This is one of the key aspects that raise the security concerns about the container technology as compared to VM platforms.

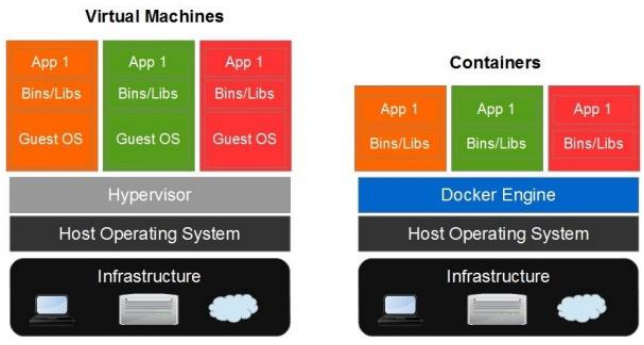

Fig.1. VMs vs. Container

Currently, Docker appeared as one of the top container-based virtualization platform. Docker containers also suffer from same security vulnerabilities. Docker demonstrates the idea of less friction; however security is just opposite to it.

The research is aimed to offer a deep insight of Docker and its analysis which leads to following research question. Is Docker offering a safe environment to run applications? The analysis will discover some of the key security vulnerabilities which are currently threating Docker. This research will also highlight the key mitigation policies to avoid such issues. This analysis will attempt to assess majority of security threats internally as well as externally.

The initial section of this paper is about the detailed overview of Docker platform. The second section will outline some of the major security issues and their mitigation strategies. The third section will outline recommendations for security deployment of containers at Docker platform and a new evolutionary hybrid architecture that is more secure and efficient. The last section is based on conclusion and recommendations.

II. DOCKER OVERVIEW

Docker has quickly turned out as one of the leading projects for containerizing applications. Docker platform was initiated as an open source project that allows packing, shipping and running applications in lightweight containers. Docker containers offer unique capabilities of platform-agnostic and hardware-agnostic. These containers do not have any dependences regarding particular framework, language or packaging system. Docker containers can run within any technology based environment. This capability make these containers independent from particular stack or provider [7].

Solomon Hykes initiated Docker as an open-source internal project at dotCloud, which is a platform-as-a-service (PaaS) company. The initial version of open source Docker was released in March 2013. Docker was designed to utilize different interfaces to access virtualization characteristics of the Linux kernel as shown in Fig.2. Initially Docker was utilizing LXC as the default execution environment for its platform. However, on release of version 0.9 on March 13, 2014, Docker dropped LXC and introduced its own libcontainer library. Go programming language was used to write Docker libcontainer library [8].

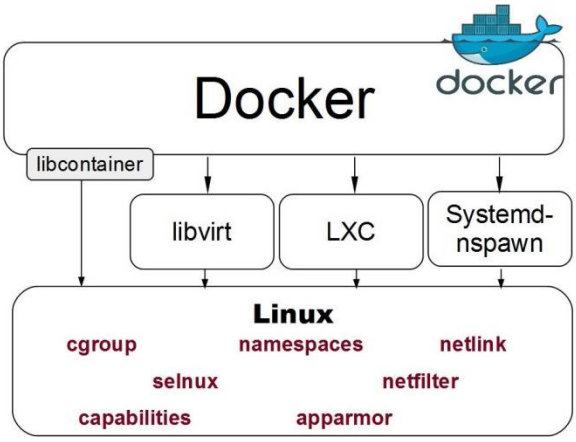

Fig.2. Docker uses virtualization features of the Linux kernel

Process of developing containers on Docker is based on number of steps. For developing container, we search for images in local Docker library. Docker offers images command lists locally to create a Docker container. However, if required image is not available then it can be downloaded from the central Docker repository. The next step is to build a container as well formulating necessary changes. The whole process is shown in Fig. 3.

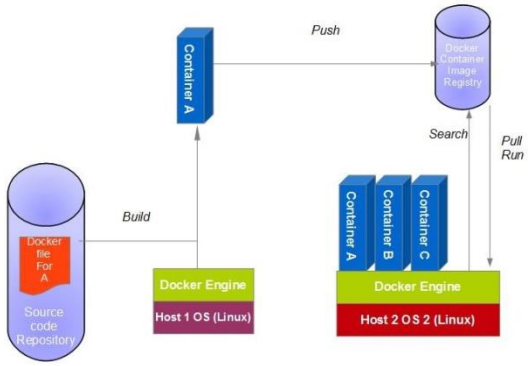

Fig.3. Developing a Docker container

Till now number of popular technology organizations worldwide showed their interest and dependence on Docker. Organizations like Red Hat, Google, IBM, Google, Amadeus IT Group and Cisco Systems have already contributed to Docker at great level. Red Hat is one of biggest contributor of Docker containers. Recently Red hat announced that now users can run Docker containers on Red Hat Enterprise Linux 7 (RHEL 7) and Red Hat Enterprise Linux Atomic (based on RHEL 7) systems [9].

On August 12, 2015, new version of Docker 1.8 was announced. This new version offers new capabilities of content trust, toolbox and updates to registry and

orchestration. This new version will also offer support for image signing, a new installer, and incremental improvements to Engine, Swarm, Compose, Machine and Registry [10]. Docker 1.8 version offers support for users of Mac OS X, Linux and Windows versions

III. SECURITY ANALYSIS

Current growing technology market demands enhanced virtualization technologies. Though a number of virtualization methods are not flexible enough to satisfy developer needs. Most of virtualization technology solutions offer considerable overhead that turn out to be a burden on the scalability of the infrastructure.

In such situations, Docker has evolved a light weight virtualization solution that minimize the overhead through containerization of applications and services [11]. Docker offers the same kernel as the host system to minimize the resources overhead, though this technique can expose containers to a number of security risks if not effectively configured [12].

Running services in multi-tenant cloud based virtual environment presents security as one of key challenges. VM allows an application to only communicate with the VM kernel, not with the host kernel. Conversely, containers based applications are able to directly communicate with the host kernel [13,14]. This is one of key weak point for containers security and privacy management. Docker is also based on similar container-based virtualization methodology, also having same security vulnerabilities.

To analyze the security vulnerabilities of Docker, we have to assess two sides of system, 'outsider' and 'insider'. Security attacks can be happened from outside as well as inside of the system. Outsider attacks try to get access to container and damage the data and exploit resources. While insider attacks carried out by a malicious user present inside, through getting access to the Docker commands. The basic objective of both kinds of attacker is same, to damage and exploit container intellectual resources.

For example, a simple command (given below) can do huge damage if your system is not properly configured. The below given command is executed through dummy flag:

```
docker run --dontpastethis --privileged -v
/usr:/usr busybox rm -rf /usr
```

The above stated command deletes all of host's /usr folder. So, we can assess that how much a system can be damaged if security precautions are not taken properly.

The objective of this analysis is to find out most vulnerable security issues (insider/outsider) and suggest some precautions and mitigation strategies to counter such attacks.

Insider attacks can be managed through some simple configurations. Running Docker Daemon with –selinux flag will prevent containers from doing damage to the host system through developing additional security layer. Further details of different insider attacks are outlined in coming sections.

Let's consider security of the container in comparison to the security of a ship. The ship acts like a dock. It can build very strong and secure container, however if the ship is insecure, it doesn't matter how strong the container is? Similarly, a great deal of care should be taken to avoid outsider attacks. Through removing capabilities of containers, a number of outsider security threats can be avoided. Container capabilities are the partition of root into 32 different categories. By default many of these capabilities are disabled in Docker containers. For instance, by default, Docker container' IPTables rules cannot be manipulated. For disabling all of these capabilities, the command is given below:

```
docker run -ti --cap-drop ALL debian /bin/bash
```

IV. DEFENDING AGAINST SECURITY ATTACKS

Security is now one of key aspects for cloud application. Docker also suffers from number of security attacks. This section will outline different insider and outsider security attacks and their mitigation policies.

1. Kernel Exploits

Kernel mange and deal all container operations and processes. In case of a kernel-level exploit, the applications running inside containers are at the verge of compromising and exploit. All containers share the same kernel architecture [5]. In this situation, if some contained application is hijacked and obtain some privileged rights of kernel, then such condition leads to compromise all running containers as well as host platform. Likewise, there is no possibility that two containers utilize different versions of the same kernel

module.

Docker is currently taking security very seriously and trying to offer more solid and effective solutions to tackle and deal major security issues. To deal with kernel level exploits Docker specifically recommends following precautions to implement a secure and safe cloud environment.

It is recommended that AppArmor or SELinux should be executed while running Docker Engine.

Docker states that mutually-trusted containers should be mapped together in groups at separate machines. Untrusted applications should not be running with root privileges. Docker Engine has also started support for user namespaces which will offer an additional layer of security for containers.

To avert kernel exploits, container file system must be set to read-only. Through turning off inter-container communication, such attacks can be avoided. Avoiding unnecessary package installations in the container is also a good way to keep such dangers away.

2. Denial of Service (Dos) Attacks

Denial-of-Service or DoS is one of the most well-known attacks on network resources. In such attacks a process or a group of processes try to consume the entire resources of the system, thus breaking down or disrupting normal processing or operations.

In containers based processing architecture all containers share kernel resources. DoS condition happens when one container exploits access to a resource. In such conditions, it will starve out all other containers.

To encounter such attacks, there is a need for OS-level virtualization solution that should fulfill requirements like: file system isolation, process isolation, IPC isolation, device isolation, network isolation and controlling resource allocation [5].

So, through controlling resource allocation to each container, such attacks can be prevented. Docker implements Cgroups as a key tool to deal with such issues. Cgroups control and manage the resource limits, e.g. CPU time, memory space, and disk I/O that any Docker container can use. They ensure that every container gets its fair share of the resources and avoiding any container monopolizes all resources. Moreover, Cgroups allow Docker to control and configure resource allocation constraints for every container. For instance, one such constraint is controlling the CPUs availability to a specific container [5].

3. Container Breakouts

In such attacks an attacker breakout a container, then he/she be able to get access to the host and other containers. After getting access, the attacker will be able to access files outside the container.

open_by_handle_at() function allows process to access files on a mounted filesystem through file_handle structure. file_handle structure utilize inode numbers to distinguish files. To call this function needs CAP_DAC_READ_SEARCH capability. A superuser inside a container is having this capability by default. This allows an attacker to bypass simfs constraints and access the entire files on a primary filesystem comprising other VE's residing on the similar filesystem.

According to Docker's website, container breakout issue and vulnerability was only present until Docker version 0.11. According to the new details this vulnerability was fixed in Docker 0.12, which was ultimately turned out to be Docker 1.0.

To mitigate such kind of security vulnerabilities there is a need to set container file system to read-only. Running containers with the privileged flag can be dangerous and can cause such kind of security attacks. Setting containers volumes to read-only is an effective way for discouraging container breakout.

4. Poisoned Images

Container Images may be injected through some virus or trojan infected software. Problem of poisoned images also happens if someone is running outdated, known-vulnerable software versions.

According to Docker, a downloaded image is "verified" by system. This verification is solely based on the presence of a signed manifest. Though Docker never authenticates downloaded image checksum from the manifest. In such situations, an attacker could transmit any image together with a signed manifest. Such kind of security issues can lead to numerous serious vulnerabilities [15].

In Docker, images are downloaded from an HTTPS server. These images pass through an insecure streaming processing pipeline inside Docker daemon: [decompress] -> [tarsum] -> [unpack].

Inside Docker, this pipeline is effective, however, extremely unprotected. Therefore, there is a need that unauthenticated input steam should not be assessed before confirming its digital signature.

Docker users need to be alert that the code used for downloading images is shockingly insecure. In this scenario, users should only download authenticated and trusted images. Another better option to manage such security issues is to block index.docker.io locally. Through this, a user will be able to download and authenticate images manually before importing to Docker platform through Docker load.

5. Compromised Secrets

Compromising business or personal secrets are the big security dangers in container based technology. In case of theft of API keys and database passwords; the overall system can be compromised. Docker allows user to run multiple containers at the same time. In case of security breach, overall services and operations can be disrupted. So a lot more is needed to be done to protect database passwords and API keys. Such details must be kept secret to avoid any possible security breach.

To further protect Docker from such attacks, there is a need to set container file system to read-only option. For sharing secrets, utilizing environment variables is not a good option. Running containers without privileged flags will also be considered a great help to avoid compromising security attacks.

6. Man-in-the-Middle (MitM)

In such attacks a malicious actor inserts himself/herself into a communication among two legitimate parties. Such attacker monitor, alter or steal valuable information which transmits among two parties.

To avoid such attacks in containers, network isolation is the most significant aspects to prevent such network-based attacks. There is a need to configure containers in such a way that they are incapable to manipulate or eavesdrop on the network traffic of the host or other containers [5].

In this scenario, OpenVPN (open virtual private network) offers a best way to implement virtual private networks (VPNs) by means of TLS (Transport Layer Security) encryption. OpenVPN defends the network traffic from man-in-the-middle attacks and eavesdropping.

Docker offers an easy way to encapsulate the OpenVPN server. In this way OpenVPN server process and configuration of data can be managed more easily at Docker' platforms. Image of the Docker OpenVPN is prebuilt. It contains everything that is necessary to run the server in a well-balanced and persistent environment. Docker includes scripts those considerably automate the standard use case, however if desired, it also offers full manual configuration. A Docker volume container is utilized to hold the EasyRSA PKI and also configuration certificate data.

7. ARP spoofing

ARP (Address Resolution Protocol) spoofing is a kind of security attack in which an attacker sends fake ARP messages over a LAN (local area network). In such attacks, attacker is able to link his/her MAC address with the IP address of a legitimate system on the network. As the attacker’s MAC address is linked to legitimate IP address, the attacker will start getting each and every bit of data from that specific IP address.

Initially, Docker developers paid less attention to the fact, that the so-called ARP is employed to map IPv4 to Ethernet hardware (MAC) addresses, which can also be utilized by the virtual bridge to distribute the Ethernet frames to right container. As ARP packets are not filtered so there is no security mechanism available in ARP itself. Therefore containers can certainly imitate other containers or even the host. Such situations can present an ARP spoofing or ARP cache poisoning attack scenario. The NDP (Neighbor Discovery Protocol) in IPv6 is used in the similar way.

If an attacker gains access to one of the containers, through compromising the container’s security, the attacker can obtain, manipulate or redirect any information of the bridge. This information can be any traffic running among containers and the outside world. In such situations, attacker might sniff any secret details (passwords) sent between web application and database containers. Moreover, attacker will also be capable to inject malicious payload into network

connections.

To mitigate such security attacks, there are number of ways; one of the most powerful ways to save container, is to run the container without NET_RAW capability. In this way, programs inside container will not be able to create PF_PACKET sockets. Without PF_PACKET sockets, ARP spoofing attack cannot be performed. This method has few drawbacks.

Another more suitable way to protect Docker container from such attacks; is to utilize "ebtables" to filter out Ethernet frames. In this way, ARP packets with wrong sender protocol or hardware address (ARP spoofing) can be caught and detected [16]. It also allows filtering out the incorrect source addresses (MAC spoofing). In this situation, attacker has no chance to perform ARP spoofing attack.

V. PROPOSED SOLUTION

1. Access Control Policy Modules

We have proposed a solution that is a simple and more reliable. This is based on access control methodology to ensure appropriate access management. It is based on specific SELinux types of containerized procedures in a more clear way for the client. In this method image maintainers ship the SELinux policy module together with their images to the host platform. Here the module will be placed on the host system and outlines the types that will be linked with the processes in the Docker image. SELinux modules and aforementioned Policy Modules (PM) have to fulfill the properties to not to pose a threat to the host system [17]. The policy modules for an image will be stated in the Docker file and placed in the image metadata at their build-time. For running containerized processes along with some SELinux types, the image-maintainer is able to label the binaries in the image by particular types, and write a type transition procedure. Thus, the process is allocated to the SELinux type stated in the rule, when the binary is executed. It is possible to run different SELinux labels, even if multiple processes have been running at the similar the same image. When a Docker container holds different images, all the policy modules for the images that comprise the container will be installed. It also offers SELinux types intended for processes in the parent images.

The Docker Hub Registry has to ensure that policy modules must not alter the system policy as well as can only have an influence on processes and resources linked with the DPM itself. Fig. 4 shows that two Docker containers (apache, and mysql) are running, using explicit SELinux types stated in the policy modules inserted in the Docker images.

It also needs to ensure that new types stated in policy modules have to always operate inside the boundaries defined through the svirt_lxc_net_t type. A policy model offers the flexibility of outlining numerous types with diverse privileges. Consequently, a container can switch to the least privilege domain required to obtain the present task.

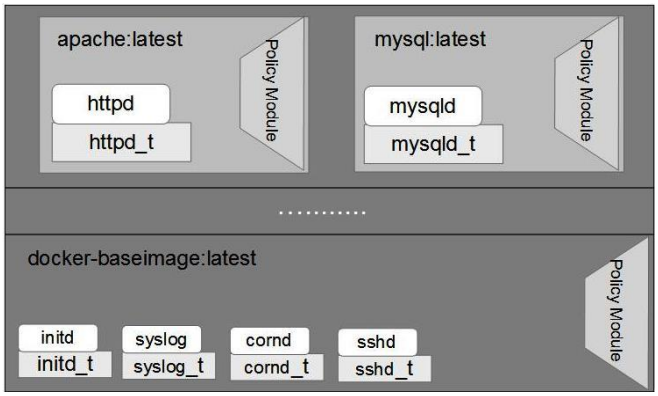

Fig.4. Two Docker containers running processes in, using SELinux types based Policy Modules

2. Secure Deployment Guidelines

Taking precautionary security measures can save us from huge vulnerabilities. The same case is with the Docker container security. By taking number of precautionary measures for Docker container security, more secure and reliable applications could be developed. This section will outline some secure deployment guidelines for Docker platform.

2.1. Docker Images

As discussed earlier that poisoned images are one of the key security issues at Docker. However, from Docker 1.3 and onward versions offer cryptographic signatures support. Through this approach, a user will be able to discover real origin and integrity of official repository images. This capability will reduce the dangers of poisoned images and also reduces the chances of possible security threats. It is highly recommended that all images should be downloaded from authenticated source and support cryptographic signatures.

2.2. Network Namespaces

Running Docker on a TCP port can cause a serious security hazard for containers. Such approach permits anyone to get access to specific port to obtain access to container. This leads to get root access on the host or may be to Docker group. Therefore, it is critically required to ensure that communications are adequately encrypted using SSL while offering access to the daemon over TCP. This approach will prevent unauthorized parties from interacting with it.

For more enhanced security management, kernel firewall iptables rules can be implemented to docker0. For example, the source IP range can be restricted for a Docker container [18]. This will prevent container from talking with the outside world. The following iptables filter is used to prevent such access.

```
iptables -t filter -A FORWARD -s
<source_ip_range> -j REJECT --reject-with
icmp-admin-prohibited
```

2.3. Logging & Auditing

Logging and auditing offers an additional layer of security for Docker security management. In this way, a user can monitor the traffic to ensure that no suspicious activities are being performed.

Following command can be used to access log files outside the container from the host:

```
docker run -v /dev/log:/dev/log
<container_name> /bin/sh
```

Log files can also be accessed by using the built-in Docker command:

```
docker logs ... (-f to follow log output)
```

For permanent storage into a tarball, log files can be exported using following command:

```
docker export ...
```

2.4. SELinux / AppArmor

Docker offers Linux kernel security modules like AppArmor and Security-Enhanced Linux (SELinux). These Linux based Linux kernel security modules can be configured through access control security policies. Through configuring these security modules, the users can implement mandatory access controls (MAC) for limiting set of system resources or privileges.

Through configuring SELinux, Docker will have an additional layer of security through permission checking policy MAC. SELinux manage everything through labels. In Docker system, every process, file/directory and system object has a label. These labels are being used by system administrator to write rules to manage access between system objects and processes.

Similar like SELinux, AppArmor is another MAC based security enhancement model to Linux. AppArmor offers control access to individual programs. Through this model administrator is able to load the security profile into every individual program to restrict and manage the capabilities of the program.

These features are available in Docker version 1.3 and onwards. Docker offers an interface for loading AppArmor pre-defined profile while launching a new container on AppArmor supported systems [19].

To load SELinux or AppArmor security policies are using label confinement; intended for container. It can be configured using the newly added --security-opt argument in Docker as shown below:

```
--security-opt="label:user:USER" : Set the label
user for the container
--security-opt="label:role:ROLE" : Set the label
role for the container
--security-opt="label:type:TYPE" : Set the label
type for the container
--security-opt="label:level:LEVEL" : Set the
label level for the container
--security-opt="apparmor:PROFILE" : Set the
apparmor profile to be applied to the container
Example: docker run
--security-opt=label:level:s0:c100,c200 -i -t
centos bash
```

2.5. Daemon Privileges

It is recommended not to use the --privileged command because --privileged command will permit the container to access the all devices on host as well as it would provide the container with explicit LSM (i.e AppArmor or SELinux)

configuration. LSM configuration would give the similar level of control as host processes [18].

Through avoiding --privileged command could help to diminish the security risks and host compromises. A legitimate user should have the ability to launch the daemon by using -u option. It can reduce the privileges which are enforced inside the container. For example:

```
docker run -u <username> -it <container_name>
/bin/bash
```

2.6. cgroups

The cgroups, or control groups, offer a way for accounting as well as limiting the resources for every container. So cgroups offered great deal of capability to avoid the Denial of Service (DoS) attacks through restricting system resource exhaustion [20].

CPU usage:

```
docker run -it --rm --cpuset=0,1 -c 2 ...
```

Memory usage:

```
docker run -it --rm -m 128m ...
```

Storage usage:

```
docker -d --storage-opt dm.basesize=5G
```

2.7. SUID/GUID binaries

Buffer overflow security attacks can be serious for containers. To avoid such attacks, SUID and SGID binaries should be prohibited. This can be achieved by decreasing the capabilities offered to containers by specific command line arguments.

```
docker run -it --rm --cap-drop SETUID
--cap-drop SETGID ...
```

Another way is to mount filesystem with the nosuid attribute. By applying this command, a user can avoid SUID resultant buffer overflow security attacks.

2.8. Devices control group (/dev/*)

Device isolation is also one of the key ways to avoid number of security vulnerabilities. By default containers includes all permissions. To avoid such issues there is a need that devices should be mount by means of the “--device" option which is built-in and don’t use “-v” with the “--privileged" argument [21].

A set of strict permissions can be utilized for device by means of third set of options; “rwm” to override read, write, and mknod permissions respectively. For example, sound card read-only permission can be set by following command:

```
docker run --device=/dev/snd:/dev/snd:r ...
```

2.9. Services and Applications

If a Docker container security is compromised; while there are number of sensitive services running, such situations can lead to huge disaster. So, to avoid such situation, consider isolating sensitive services. Through running sensitive services (SSH service) on bastion host or in a VM we can add an additional security layer into our system. Also untrusted applications should be avoided to run with root privileges within containers.

2.10. Linux Kernel

Sometimes out-dated kernels are more likely to be exposed to publicly disclosed vulnerabilities. Therefore, it is so important that kernel is up-to-date with updated utility offered by the system (e.g. apt-get, yum, etc). There is a great deal of effective security against memory corruption bugs, through using kernel with GRSEC or PAX.

2.11. User Namespaces

Currently, user namespaces are not directly supported by Docker. However, they can be used by Docker’ containers on supported kernels, through applying the clone syscall, or using the ‘unshare’ feature. UID mapping is presently supported through the LXC driver, however; not in the native libcontainer library.

User namespaces feature would permit the Docker daemon to execute as an unprivileged user on the host [22]. However, this Docker daemon will appear as executing like root inside containers.

2.12. libseccomp (and seccomp-bpf extension)

Syscall processes are not significant to system operation. It should be restricted in order to avoid abuse or misuse inside a compromised container. To restrict Linux kernel’s syscall procedures, libseccomp library is used. This feature is presently under-development. This features available in LXC

driver but not in libcontainer.

Docker daemon can be restarted to use the LXC driver by using following command:

```
docker -d -e lxc
```

2.13. Full Virtualization

Escalation from the container to the host can be so dangerous. It can happen if kernel vulnerability is exploited inside the Docker image. To prevent such issues, use full virtualization solutions that contain Docker, for example KVM. Docker offers capability to nest Docker images to offer KVM virtualization layer. (Docker-in-Docker utility)

2.14. Security Audits

Security audits are one of the key ways to protect system from major security risks. Host system and containers should be audited regularly to assess and identify any vulnerabilities and misconfigurations if any. These vulnerabilities and mis-configurations could become dangerous for our systems and may compromise intellectual resources [23].

2.15. Multi-tenancy Environments

Containers should run on dedicated hosts. It is very important for the security of containers. It becomes more important when user is dealing with some sensitive operations.

It is recommended because of the shared nature of Docker containers' kernel. Therefore, multi-tenancy environments of Docker containers' kernel can offer secure separation of duty. So, it is highly suggested that containers should be running on dedicated hosts [24].

More secure environment can also be achieved through reducing the inter-container communications to a very small level. It can be achieved by setting the Docker daemon to utilize --icc=false. Also it is useful to specify -link with Docker run when required.

2.16. Docker Content Trust

Content Trust is a new feature that can offer an additional layer of security for containers. It is open-source software that is able to offer legitimacy of container images. This feature is added in Docker version 1.8.0. It will allow Docker users to conform the legitimacy of container images (available at any public Docker Hub) before downloading Docker images [25].

The basic idea behind this feature is to secure the Docker platform and offer assurance to users that they won't be deploying anything possibly hazardous atop their technology infrastructure.

VI. RESULTS AND DISCUSSION

The proposed guidelines offer a great deal of capability to protect against any attacks from malicious client or network. Adopting the aforementioned methodologies and guidelines will ensure a great deal of capability to protect systems against possible breaches. The proposed SELinux types based Policy Modules for access control will ensure better security and improved access management of possible insider and outsider attacks. Application of these policy modules for different images will offer a comprehensive firewall.

To perform a network defensive drill we have to know the difference between the efficiency among real world attacks and the simulation. Therefore, to test the proposed technique, DDoS program executed on the Docker container. Cyber-attacks program will produce a huge number of subroutines those are sending requests to attack the targets. Docker container equipped with aforementioned tools and capabilities defended better as compared to naive Docker containers. Table 1. shows the experiment aspects where we run Docker version 1.2 on a native OS, with policy modules and specified guidelines. It can be seen that source to base presence of policy modules can ensure higher protection and security.

Table 1. Policy Module Application

	Target to Base	**Target to Policy Module**
Source to Base	Threat Present	Partial Threats
Source to Policy Module	Partial Threats	No Threats

In March 2015, Docker Inc. published a research, which offered a hybrid solution (Combining Containers and Virtual Machines) to enhance the security and isolation [26]. Regular Virtual machines cannot be scaled down up to the level of running a single application service. VMs are able to support the rich set of applications. However, this approach can present some conflicts among collaborating

micro-services. On the other hand, running one micro-service per VM is so costly from resource point of view. To resolve these issues, Docker containers can be deployed in conjunction with VMs. This hybrid infrastructure will be based on combination of containers and VMs. It will allow running complete group of services in an isolated way and also it's grouping inside of a virtual machine, as shown in Fig.5.

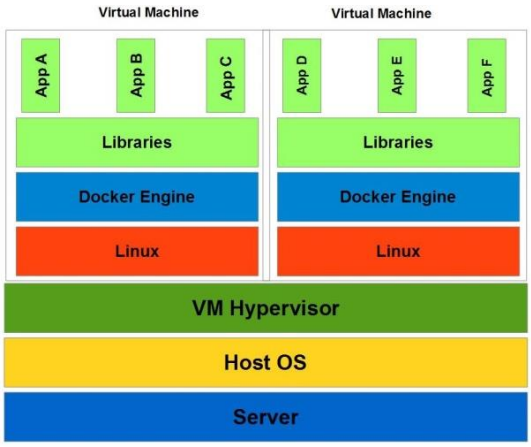

Fig.5. Combining Containers and Virtual Machines

One of the key features of this approach is the enhanced security by introducing two layers, VMs and containers, to the distributed applications. The other feature of this technique is to utilize resources in a better way. Moreover, it increases the density of containers; whereas decreases the number of VMs necessary for the defined isolation and security objectives.

VII. CONCLUSION

Container applications are getting popular. Docker, LXC, Rocket or other projects are getting momentum in container application field. This technology is here to stay. As the technology and related processes are getting mature, they will address many of the risks, few of them have been outlined above. Docker Containers are offering a lightweight and efficient way to package application with all their dependencies. However, some security issues hindering their wide spread adoption. This research has addressed and outlined some potential security issues and vulnerabilities, and offer mitigation strategies to manage these issues. This research also outlines some security deployment strategies to deploy applications at Docker in a more security and safe way. These all guidelines and precautionary measures can offer a more secure and reliable container platform for future application development. Currently, Docker has offered its 1.8 version with new updates and fixes. Now Docker is offering quite secure container based application development platform.

Docker suggests that user can ensure security of processes through running inside the containers as non-privileged users (i.e., non-root). It is also recommended that by enabling SELinux, AppArmor, GRSEC, or hardening solution, we can add additional layer of security for our applications. Through configuring the right security policy and following the secure deployment guidelines we can ensure the greater security of Docker containers.

ACKNOWLEDGMENT

This research is supported by CAS-TWAS and University of Science and Technology, China. We pay special gratitude to our colleagues who provided insight and expertise that greatly assisted this research work.

REFERENCES

1. Moral-García, S., Moral-Rubio, S., Fernández, E.B., Fernández-Medina, E.: "Enterprise security pattern: A model-driven architecture instance". Computer Standards & Interfaces **36**(4), 748-758 (2014).
2. Kalloniatis, C., Mouratidis, H., Vassilis, M., Islam, S., Gritzalis, S., Kavakli, E.: "Towards the design of secure and privacy-oriented information systems in the cloud: Identifying the major concepts". Computer Standards & Interfaces **36**(4), 759-775 (2014).
3. Blanco, C., Rosado, D.G., Sánchez, L.E., Jürjens, J.: "Security in information systems: Advances and new challenges". Computer Standards & Interfaces **4**(36), 687-688 (2014).
4. Bernstein, D.: Containers and cloud: "From lxc to docker to kubernetes". IEEE Cloud Computing (3), 81-84 (2014).
5. Bui, T.: "Analysis of Docker Security". Aalto University T-110.5291 Seminar on Network Security (2014).
6. Li, S.-H., Yen, D.C., Chen, S.-C., Chen, P.S., Lu, W.-H.,Cho, C.-C.: "Effects of virtualization on information security". Computer Standards & Interfaces **42**, 1-8 (2015).
7. Turnbull, J.: "The Docker Book". ames Turnbull; 1.8.0

edition, (2014)
8. Github: “Docker: the container engine”. https://github.com/docker/docker (2015). Accessed 08 25 2015
9. RedHat: “Get Started with Docker Formatted Container Images on Red Hat Systems”. https://access.redhat.com/articles/881893 (2015). Accessed 07 09 2015
10. Docker: “Announcing docker 1.8: content trust, toolbox, and updates to registry and orchestration”. http://blog.docker.com/2015/08/docker-1-8-content-trust-toolbox-registry-orchestration/ (2015). Accessed 09 02 2015
11. Boettiger, C.: “An introduction to Docker for reproducible research, with examples from the R environment”. arXiv preprint arXiv:1410.0846 (2014).
12. Merkel, D.: “Docker: lightweight Linux containers for consistent development and deployment”. Linux J. 2014(239), 2 (2014).
13. Gomes, J., Pina, J., Borges, G., Martins, J., Dias, N., Gomes, H., Manuel, C.: “Exploring Containers for Scientific Computing”. In: 8th Iberian Grid Infrastructure Conference Proceedings, p. 27
14. Felter, W., Ferreira, A., Rajamony, R., Rubio, J.: “An updated performance comparison of virtual machines and linux containers”. Technology **28**, 32 (2014).
15. Rudenberg, J.: “Docker Image Insecurity”. https://titanous.com/posts/docker-insecurity (2014). Accessed 07 06 2015
16. Nyantec: “Docker networking considered harmful”. https://nyantec.com/en/2015/03/20/docker-networking-considered-harmful/ (2015). Accessed 08 02 2015
17. Mutti, S., Bacis, E., Paraboschi, S.: “Policy specialization to support domain isolation”. In: Proceedings of the 2015 Workshop on Automated Decision Making for Active Cyber Defense 2015, pp. 33-38. ACM
18. Grosperrin, Q.: “Docker Secure Deployment Guidelines”. (2015).
19. Docker: “Docker security”. https://docs.docker.com/articles/security/ (2015). Accessed 08 29 2015
20. Goldmann, M.: “Resource management in Docker”. https://goldmann.pl/blog/2014/09/11/resource-management-in-docker/ (2014). Accessed 8 8 2015
21. Vieux, V.: “Announcing Docker 1.2.0”. http://blog.docker.com/2014/08/announcing-docker-1-2-0/ (2014). Accessed 08 12 2015
22. Graber, S.: “Introduction to unprivileged containers” https://www.stgraber.org/2014/01/17/lxc-1-0-unprivileged-containers/ (2014). Accessed 08 22 2015
23. Chou, D.C.: “Cloud computing risk and audit issues”. Computer Standards & Interfaces 42, 137-142 (2015).
24. Turnbull, J.: “Docker Container Breakout Proof-of-Concept Exploit” (2014).
25. Firshman, B.: “Announcing docker 1.8: content trust, toolbox, and updates to registry and orchestration”. http://blog.docker.com/2015/08/docker-1-8-content-trust-toolbox-registry-orchestration/ (2015). Accessed 08 01 2015
26. Docker: “Introduction to Container Security”. https://d3oypxn00j2a10.cloudfront.net/assets/img/Docker%20Security/WP_Intro_to_container_security_03.20.2015.pdf (2015). Accessed 08 09 2015