Secure Cloud Container: Runtime Behavior Monitoring using Most Privileged Container (MPC)

[1]Vivek Vijay Sarkale, [2,3] Paul Rad, [3]Wonjun Lee
[1]Department of Computer Science, University of Texas at San Antonio
[2]Open Cloud Institute (OCI), University of Texas at San Antonio
[3]Department of Electrical Engineering, University of Texas at San Antonio
viveksarkale@gmail.com, paul.rad@utsa.edu, wonjun.lee@utsa.edu

Abstract: Hypervisor-based virtualization rapidly becomes a commodity, and it turns valuable in many scenarios such as resource optimization, uptime maximization, and consolidation. Container-based application virtualization is an appropriate solution to develop a light weighted partitioning by providing application isolation with less overhead. Undoubtedly, container based virtualization delivers a lightweight and efficient environment, however raises some security concerns as it allows isolated processes to utilize an underlying host kernel. A new security layer with the Most Privileged Container (MPC) is proposed in this article. The proposed MPC layer exhibits three main functional blocks: Access policies, Black list database, and Runtime monitoring. The introduced MPC layer implements privilege based access control and assigns resource access permissions based on policies and the security profiles of containerized application user processes. Furthermore, the monitoring block examines the runtime behavior of containers and black list database is updated if the container violets its policies. The proposed MPC layer provides higher level of application container security against potential threats.

Keywords: Virtualization, Container based virtualization, Linux Kernel security, Container Security, Access Control Policies.

I. INTRODUCTION

Virtualization technology refers to the creation of a virtual version of devices or resources, such as servers, storage devices, networks, or operating systems where frameworks divide resources in one or more execution environments [1]. Server virtualization is a popular technology where the physical server is partitioned into smaller virtual servers called virtual machine [1]. Operating system virtualization, [2] as a sub category of server virtualization, allows users to run multiple operating system instances on the host. Popular service providers such as Amazon EC2, Google Cloud Engine, and Microsoft Azure provide multi-tenant on demand, API-driven virtualized resources to their cloud users. The virtualization technology solution is mainly classified into two categories: Hypervisor based virtualization [3] and container based virtualization. Of these two, container based virtualization provides a more lightweight and isolated environment [4].

Container based virtualization - Containerization allows multiple isolated user-space instances with separate filesystems that resemble complete isolated systems. Every instance runs as a single instance of the operating system and has its own IP-address, network interface, filesystem, and unique process id [28]. The positive side of containerization is that the host operating system has complete visibility to entities running on it. Fig.1 illustrates the general idea about container based virtualization where the host kernel runs multiple virtual instances referred as Linux containers. Containers provide a lightweight form of virtual environments with very limited performance overhead, better resource utilization, and easy to create and destroy [4]. Although container based virtualization is considered to be less secure as there are many existing potential threats however some security features are introduced as: (i) internal security features such as Linux kernel Namespaces [5], Cgroups [6], Linux capabilities [7] , and (ii) supporting Linux Security Modules (LSM) such as SELinux, AppArmor, SECcomp, and GRSec.

In this paper, we propose a security layer which aims to secure cloud container environment. As a part of this layer, we architected a privileged container called Most Privileged Container (MPC) as access control enforcer and behavior monitoring watchdog module. The main contribution of this paper is twofold. First, we propose a secure container framework called Most Privileged Container. Second, we present the functional elements of the proposed architecture: a) MPC access control b) Black list database c) Runtime monitoring. Proposed MPC manages security by scanning container profiles against defined policies and container black lists. The MPC's run time monitoring block examines process behavior continuously during runtime.

The rest of the paper is organized as follows: Section II provides information on container and container security background including threat models. In Section III, internal security features and Linux Security Modules (LSM) are discussed. Container security model and existing Linux security features against potential threats are inspected and discussed in the Section IV. The Most Privilege Container (MPC) security layer for access control security and behavior monitoring of

978-1-5090-6644-5/17 $31.00 © 2017 IEEE
DOI 10.1109/CSCloud.2017.68

other containers are vented in section V, and finally conclusion is provided in section VI.

II. RELATED WORK

Container based virtualization usually gets compared with hypervisor based virtualization which is considered to be more secured –In which any application running on the guest operating system is only able to approach the guest OS kernel. If any malicious application running on the guest OS cannot directly break down to the host as it has to bypass the guest kernel and Virtual Machine Manager which makes hypervisor based virtualization more secure. On the other hand, in container based virtualization, lightweight Linux containers run on the container engine and are easily able to approach the host kernel by running malicious programs inside containers that actually create a possibility of the host kernel exploitation and compromises the security of the whole system. Both the application's image security and container's internal security must be considered in order to secure the host Kernel [10]. Here are some security threats which are faced by Container based virtualization.

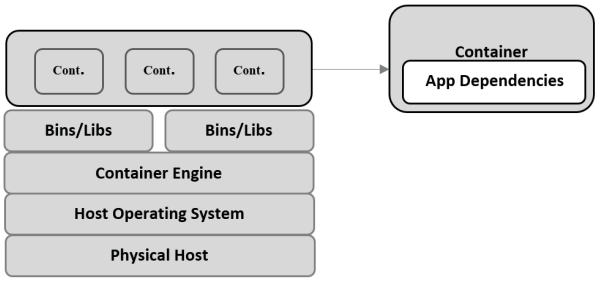

Fig.1: Container Based Virtualization

A. Potential threats

Security is the major challenge when running a service in virtual environment, especially in the multitenant Cloud system [8]. Containerization is not an exception and thus has several security issues. Our analysis considers various threats, security issues, and vulnerabilities in container based virtualization, such as Linux kernel exploit [9], leaks to other Containers, Container Engine Attack Surface[10], Fake System calls [11], and other Issues are discussed in the rest of this section.

Linux Kernel Exploit

Unlike hypervisor based virtualization running VM's, the kernel is shared among all containers and the host. Running malicious containers may cause panic to the kernel. It can take down the host and typically damages the whole system. In VMs, the situation is much better; an attacking program would have to route an attack through both the VM kernel and the hypervisor before being able to touch the host kernel, but this is not the case with containerization. Containers which demand extra privileges to run are called privileged containers where container user id (UID) 0 (root privileges) is mapped to the host's UID 0. This indicates that the root inside the container is also the root outside the container (on Host). In this case the host is really at higher risk. We need to worry about potential privilege escalation attacks where the user gains elevated privileges such as those of root users. Container break outs are unlikely however possible in certain cases. Mechanisms like Mandatory Access Control (MAC) [12] typically prevent the host from accidental damages like reconfiguration of host hardware, host kernel or filesystem access. Furthermore, vulnerable environments exist which will allow such containers to escape and get full root privileges on the host. Un-privileged containers are safe by design. Any container UID 0 (root) inside container is actually a non-zero UID (non-root) outside container. However, there are still general security issues such as container escape and resource abuse which are considered threats to the system.

Container Engine Attack Surface

While running LXC on container engine, the user is required to have root privileges therefore only trusted users should be allowed to control the container engine. Containers management can be done in web interface so that administrators and container service users can access through the web interface. In such case, container engines face attacks like cross-site scripting; enables attacker to inject client side script into web pages and also to bypass weak access control policies [10].

B. Other security issues

Several other security threats are commonly addressed in containerization, such as poison images, fake system calls, and other attacks –such as Denial of Service. How can we know that the images being used are safe, there is a possibility that attacker can trick us into running his image hence our data and host are at higher risk.It is necessary to ensure that images that we run are up-to-date and do not contain software with vulnerabilities Malicious software contained in the image can manipulate */dev/mem* device to access system memory. Finally the malicious software is injected and access kernel memory. From such scenario, a fake system call made inside the container may cause injury to host configuration. Containers share kernel resources. If any containers take over access to major resources e.g. memory, process, UID – they can starve out other container which would result into Denial of Service (DoS) [13], where authorized user processes would be unable to access resources of the system.

III. CONTAINERIZATION SECURITY

A. Internal Security Features

1) Linux namespaces

Linux namespaces mainly provide forms of isolation which creates isolated environments for every running

container on the host. Process running within container cannot have visibility to processes which are run by other containers. Single underlying host runs n number of Linux containers and with the help of namespaces containers are assigned to their own independent runtime environments. Each get it's own PID-Process isolation [14], Network namespace [15], Mount Namespace-Filesystem isolation [16] & IPC Namespace-Inter process communication isolation [17] which mainly contribute to container security.

2) PID Namespaces -Process Isolation

PID Namespaces isolate the process id number space, a process in different PID namespaces can have same process id. PID namespaces ensure runtime process isolation. Containerization ensures process isolation by limiting their permissions and visibility to process running in the other containers, and the host. It allows the containers to suspend and resume a set of processes in the container, live migration of container to other host while the process inside container stays with same PID. Use of PID namespaces requires a kernel that is configured with CONFIG_PID_NS option [14].

3) *Network Namespace –Network Interface Isolation*

Network namespaces provide independent network stacks to every instance and isolate system resources such as network devices, port numbers (sockets), firewalls and IP routing tables. This allows the container to establish communication between each other or host by using virtual network bridge, however network traffic of one container cannot be manipulated. Use of network namespaces requires a kernel that is configured with the CONFIG_NET_NS option [15].

4) IPC –Inter Process Communication Isolation

IPC namespaces give containers their own inter process communication resources, and also exhibit set of objects for data exchange among processes. Each IPC namespace has its own system v IPC objects and POSIX message queue file system. One process in a certain IPC namespace cannot read and write resources in another IPC namespaces. Member of one IPC namespace have no visibility to process in another IPC namespace [16].

5) Mount Namespaces-File System Isolation

Every running container gets associated with its own filesystem, in order to protect container and host's file system from unaccredited access, mount namespaces provide complete filesystem isolation. The processes in different mount namespaces can have different view of the file system. Mounted events inside container are restricted within particular container. Mount namespaces keeps compromised container away from making threats by removing containers write permission to the filesystem and also restrict from remounting of filesystem within container.[18]

6) Cgroups

Cgroups turn to be a key component in containerization as it limits and controls resources. They provide limiting and accounting and also ensure every instance gets its fair share of memory, CPU, and I/O devices. Cgroups usually prevent system from getting down by restricting exhaustive usage and reject privileges to create a new device node which guarantees better security in the system. They are essential to fend off some denial-of-service attacks. Cgroups are crucial on multi-tenant platforms, such as public and private PaaS, to ensure consistent uptime to guarantee performance when some applications start to misbehave [6].

B. Linux kernel security modules

1) Linux Capabilities

Unix system contains two types of processes: privileged processes with user id 0 or root user and, unprivileged processes with nonzero user id or non-root. Privileged processes do not require passing all kernel permission checks while other unprivileged processes undergo through full permission check. Root privileges are further divided into different forms and named as Linux Capabilities. Linux kernel is central authority to enable or disable capabilities of the root. If capabilities more than required are assigned to any container, it creates possibility of potential threat to the underlying host. As security principle describes 'less is more secure' it is always recommended to not provide capabilities other than necessary. List of major capabilities implemented on Linux [7] is introduced in Table 1.

Table.1: Listed few capabilities may enable or disable in containerization

Capability	Permitted Behavior
CAP_SETCAP	Modify process capabilities
CAP_SYS_ADMIN	System administration operations
CAP_SYS_RESOURCE	Resource limits
CAP_SYS_RAWIO	I/O Port Operation
CAP_SYSLOG	Check syslog for information on which operation require privilege
CAP_SYS_TTY_CONFIG	Employ privilege IOCTL Operations on virtual terminal
CAP_NET_ADMIN	Configure Network

2) SELinux

Security Enhanced Linux (SELinux) [18] is a security architecture built into Linux kernel which provides flexible mandatory access control and running user process has access permission to files, processes, and sockets under Discretionary Access Control (DAC) [19]. Linux kernel running with Mandatory Access Control (MAC) [20] ensures protection to the host from malicious applications running on it. SELinux defines access policies for every user, application, process, and files. It is a system administrator who decides to implement

policies to the server in that policies can be strict or lenient based on the situation. SELinux policies are categorized as: Enforcement policies, Multi-level security enforcement policies, and Multi category security enforcement policies. MAC works by assigning labels to each file system object. System administrator refers label to write policies in order to control access between subjects and objects. In DAC user has complete control over programs and objects that it owns, in such cases attackers get access to all objects if user is compromised, But SELinux co-operate Linux kernel to manage access control to provide high security level. Fig.2 states SELinux decision making process, when a process attempts to access any labeled object. The policy enforcement server refers an access vector cache (AVC), where access control permission policies are defined. If access decision cannot be taken based on policies in AVC, then security server handles access request, which performs complete security check and convey the grant or access denied message.

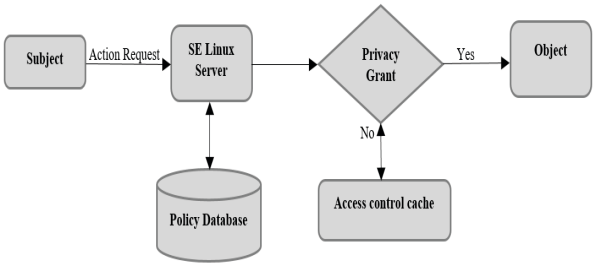

Fig.2: SELinux Decision Process

3) APPARMOR

AppArmor [21] is Mandatory Access Control (MAC) system which uses Linux Security Modules (LSM) enhancement to restrict processes to certain resources. AppArmor loads profiles in every program when system starts and it decides which files and resources program can access. AppArmor is installed by default and has two types of profile modes: Enforcement and Complain. Enforcement mode enforce policies on program and also reports policy violation attempt if any. Complain mode don't enforce any policies but it keeps log of policy violation attempts. AppArmor security model prevent application from turning evil. If an application running inside a container is compromised, the enforcement profile mode restricts compromised container and prevents from accessing important filesystem on the host. AppArmor acts as access control system that protects from attacks by allowing application to access only files and folders that are mentioned in AppArmor Profile.

4) *SECCOMP*

Secure Computing method [22] is Linux kernel feature, mainly used to restrict actions within the container. SECCOMP provides system call filtering and means for a process to specify a filter for incoming system calls. Filter is expressed as Berkeley Packet Filter (BPF) program [23] with the socket filters. This allows filtering of system calls using filter program language with some history and formal data sets. Linux SECCOMP is enabled via system call using PR_SET_SECCOMP argument.

5) *Access Control Mechanisms*

Access Control plays crucial role in security by restricting access to certain resources. Container based virtualization modifies an existing OS to provide extra isolation, this involves adding a unique id to every process and adding new access control checks to every system calls. The Linux container can be seen as another level of access control as the user and group based access control. [24] following traditional access control mechanisms are popular.

a) Discretionary Access Control[19] - The access control model is based on user's discretion. i.e. the owner of the resource can give access rights to other users based on discretion. Access Control List (ACL) is an instance of DAC. in short it is Owner/User decision oriented mechanism. DAC is defined by the trusted computer evaluation criteria as means of restricting access to objects according to identity of subjects or groups to which they belong

b) Mandatory Access Control [20] - In this mechanism every Subjects (users) and Objects (resources) are assigned with security label. The security label of the subjects and objects along with the security policy is determined if the subjects can access the objects.

To ensure security, the following to mandatory rules must be followed:

Rule 1: Subjects can read object only if $x(s)>x(o)$.
This rule is referred as "*no read up rule"*

Rule2: Untrusted Subject s can write object o only if $x(s)< x(o)$.
This rule is referred as "*no write down rule*"

c) Role-Based Access Control Model (RBAC) [25] - Access to resources governed based on the roles that the subject holds within an organization. RBAC is also known as no-discretionary access control because the user inherits privileges that are tied to his role. The user does not have control over the assigned roles. Each of the above access control models has their own pros and cons. The selection of appropriate access control mechanism as security model is based on type security policies that need to be enforced

d) Privilege Based Access Control Model (PBAC)

We are proposing Privilege Based Access Control mechanism with most privilege container which decides what privilege that user process can have and what type of resources that process can access on real host. Privileges and access permissions are assigned to each and every user process (container) running up on host. In section V PBAC is discussed in more detail.

IV. INSPECTION

Container based virtualization security mainly relies on Linux internal security features and other Linux Security Modules. Namespace and Cgroups contribute to security by providing process isolation and resource limiting. Other Linux Security Modules (LSM) like SELinux and Apparmor security architectures make use of Mandatory Access Control (MAC) in order to protect Linux kernel from external threats but, they enforce access control policies on processes at the lower level of the system. Fig.3 illustrates the existing security architecture which mainly focuses on Linux kernel protection from malicious attackers. There is no access control policy enforcement and runtime activity monitoring which can co-assist with the container engine in order to provide some extra security measures at the top. A security architecture that enhances security level by enforcing improved access control policies and runtime security checks which can continuously monitor all the containers.

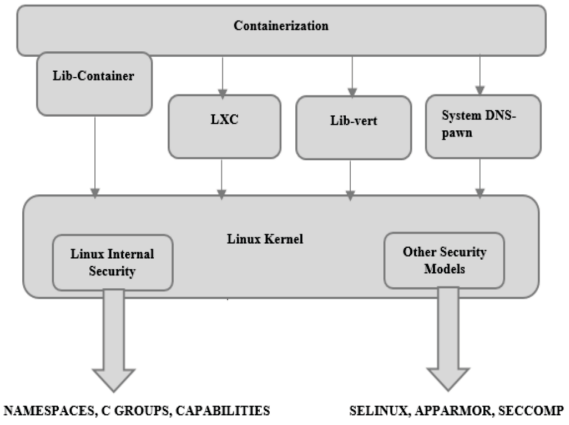

Fig.3: Container based Security architecture

V. PROPOSED SECURITY ARCHITECTURE

A new security layer with extra security features is proposed. This layer is added to existing model which ensures high level security. There is no access control enforcement and monitoring system at the top level of architecture. We introduce two important security features (1) Container Security Profiles (2) Most Privilege Container which mainly includes three functional blocks as follow: Access control policy, Black list database and, Runtime monitor.

Producers are mandatory to provide the unified security profile for every container. Security profile should remain within container wherever it runs which include information such as, what resources that the container can access and what are the privilege that the container can have on host. In another word, container profile should be enclosed with information that describes the minimum resources requirements, runtime behavior and extra privileges if required. Container's security profiles are used as unique identifier and assists the container engine in order to ensure fine grained control over kind of resources that container can access.

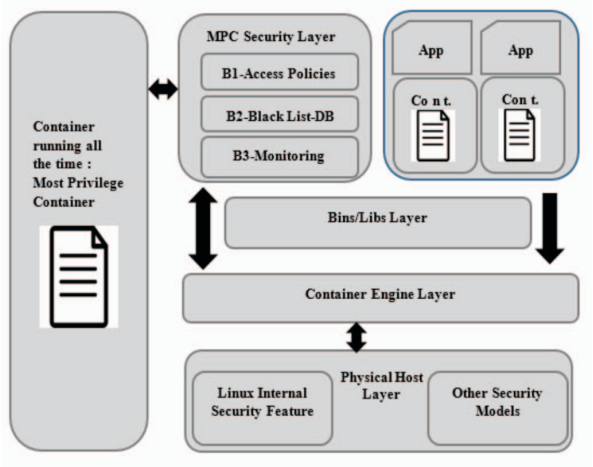

Fig.4: MPC with Container Security profile

The Container Engine with access control policies may get overloaded however the proposed system is completely separate authority in order to provide access control over every process. The proposed security layer itself is a container named as Most Privileged Container (MPC) which runs all the time on the container engine. Proposed container response to a request workflow:

Step 1 *- Request sent to the container engine*

Step 2 *- The container engine forwarded the request to proposed MPC security layer*

Step 3 *- MPC Security Layer*

a) Check the security profile of the requesting container against defined access control policies.

b) Check the requesting container against black listed database.

Step 4 *– Pass the security decision response to the container engine after security check*

Step 5 *– MPC behavior monitoring – Monitor the runtime behavior of the container against allowed resource usage permissions*

We'll discuss about MPC in next section of this section.

Most Privilege Container

Proposed Most Privilege Container runs all the time on the container engine. Most Privilege Container mainly includes three functional blocks as follows:

a) Access control policy
b) Black list database
c) Runtime monitor

a) Access control policy block

Access control policies are defined using introduced privileges based access control mechanism. Fig.4 illustrates detailed design. When the container initiates run request, the container engine allowed to look up inside its security profiles. Container engine forwards the request packet (packed with security profile) to Most Privileged Container (MPC). The MPC makes use of policies to verify privileges (root or non-root) that process is demanding on the host. If the processes claims for root privileges, then request is denied due to security concern because containerization security policy specifies that 'very few processes are allowed to run as a root'. Processes demands to run without root privileges are assigned as a non-root.

b) Black list database

MPC includes continuously updating database, it has list of black listed containers which are marked as anomalous. Secondly, MPC looks into the security profile in order to check what resources that process claimed for and verifies the behavior database (List of black listed containers).

After getting done with background information check, permissions are assigned to respective processes. MPC performs a two phase security check and forwards its decision packet to container engine. Based on the received decision packet, the container engine makes decision about requesting processes (container).

c) Runtime monitor

Once the container is permitted to run on the host, the MPC provides runtime monitoring and checks for the behavior. If the process tries to act against assigned permissions or tries to show behavior which is not supposed to, then the MPC immediately sends an alert message to the container engine. The container engine can completely halt that process and add that process to the black list.

Our architecture not only ensures high level security at the upper level of containerization but also assure the safety of underlying Host.

VI. CONCLUSION

Container based virtualization technology benefits security concerns. The paper presented potential threats to the system and we conducted an analysis on existing security solutions in order to discover the most secured approach. Our research implies that container based systems are at severe risk due to attacks such as kernel exploits (privilege escalation) and container breakout. In conclusion, Privilege based access control ensures appropriate privilege to user process, limits resource access based on some written policies and security profiles as well as provide process monitoring to keep watching on the runtime behavior of every container. This way we have enabled an additional security layer which ensures safety and protection in container based virtualization.

ACKNOWLEDGEMENT

We gratefully acknowledge the following:
(i) Support by NSF grant CNS-1419165 to the University of Texas at San Antonio; and (ii) time grants to access the Facilities of the Open Cloud Institute of University of Texas at San Antonio.

REFERENCES

[1] N. Regola and J.-C. Ducom. Recommendations for virtualization technologies in high performance computing. In 2010 IEEE Second International Conference on Cloud Computing Technology and Science (Cloud-Com), pages 409–416, Nov. 2010.
[2] Soltesz, S., Pötzl, H., Fiuczynski, M. E., Bavier, A., & Peterson, L. (2007, March). Container-based operating system virtualization: a scalable, high-performance alternative to hypervisors. In ACM SIGOPS Operating Systems Review (Vol. 41, No. 3, pages. 275-287). ACM.
[3] Uhlig, R., Neiger, G., Rodgers, D., Santoni, A. L., Martins, F. C., Anderson, A. V., ... & Smith, L. (2005). Intel virtualization technology. Computer, *38*(5), 48-56.
[4] Vaughan-Nichols, S. J. (2006). New approach to virtualization is a lightweight. Computer, 39(11), 12-14.
[5] Namespaces: http://man7.org/linux/manpages/man7/namespaces.7.html
[6] C Groups: http://man7.org/linux/manpages/man7/cgroups.7.html
[7] Linux Capabilities: http://linux.die.net/man/7/capabilities
[8] Shen, Z., Subbiah, S., Gu, X., & Wilkes, J. (2011, October). Cloudscale: elastic resource scaling for multi-tenant cloud systems. In Proceedings of the 2nd ACM Symposium on Cloud Computing (page. 5). ACM.
[9] Bui, Thanh. "Analysis of docker security." arXiv preprint arXiv:1501.02967 (2015).
[10] Docker Security: https://docs.docker.com/engine/security/security/
[11] Reshetova, E., Karhunen, J., Nyman, T., & Asokan, N. (2014, October). Security of OS-level virtualization technologies. In Nordic Conference on Secure IT Systems (pages 77-93). Springer International Publishing.
[12] Lindqvist, H. (2006). Mandatory access control. Master's Thesis in Computing Science, Umea University, Department of Computing Science, SE-901, 87.
[13] Liu, H. (2010, October). A new form of DOS attack in a cloud and its avoidance mechanism. In Proceedings of the 2010 ACM workshop on Cloud computing security workshop (pages 65-76). ACM.
[14] Rosen, R. (2013). Resource management: Linux kernel Namespaces and cgroups. Haifux, May, 186.
[15] Network Namespace:http://blog.scottlowe.org/2013/09/04/introducing-linux-network-namespaces/
[16] Linux IPC Namespaces: https://blog.yadutaf.fr/2013/12/28/introduction-to-linux-namespaces-part-2-ipc/
[17] MountNamespace:http://man7.org/linux/manpages/man7/namespaces.7.html
[18] Wright, C., Cowan, C., Smalley, S., Morris, J., & Kroah-Hartman, G. (2002, August). Linux Security Modules: General Security Support for the Linux Kernel. In USENIX Security Symposium (Vol. 2, pages 1-14).
[19] Sandhu, R. S., & Samarati, P. (1994). Access control: principle and practice. Communications Magazine, IEEE, 32(9), 40-48.
[20] Osborn, S. (1997, November). Mandatory access control and role-based access control revisited. In Proceedings of the second ACM workshop on Role-based access control (pages 31-40). ACM.
[21] AppArmorProjectWiki : http://wiki.apparmor.net/index.php/Main_Page
[22] SECCOMP: http://man7.org/linux/man-pages/man2/seccomp.2.html
[23] Roesch, M. (1999, November). Snort: Lightweight Intrusion Detection for Networks. In LISA (Vol. 99, No. 1, pages. 229-238).
[24] Sandhu, R. S., & Samarati, P. (1994). Access control: principle and practice. Communications Magazine, IEEE, 32(9), 40-48.
[25] Sandhu, R., Coyne, E.J., Feinstein, H.L. and Youman, C.E. (August 1996). *"Role-Based Access Control Models"*
[26] P. Rad, M. Muppidi, S. S. Agaian and M. Jamshidi, "Secure image processing inside cloud file sharing environment using lightweight containers," 2015 IEEE International Conference on Imaging Systems and Techniques (IST), Macau, 2015, pp. 1-6.
[27] M. Muppidi, P. Rad, S. S. Agaian and M. Jamshidi, "Container based parallelization for faster and reliable image segmentation," 2015 IEEE International Conference on Imaging Systems and Techniques (IST), Macau, 2015, pp. 1-6.
[28] P. Rad, R. V. Boppana, P. Lama, G. Berman and M. Jamshidi, "Low-latency software defined network for high performance clouds," System of Systems Engineering Conference (SoSE), 2015 10th, San Antonio, TX, 2015, pp. 486-491.