Virtualization at File System Level: A New Approach to Secure Data in Shared Environment

Thi Thu Giang Tran, Duc Quang Le, and Trung Dung Tran

***Abstract*—The growth of networking services has led to an increase of users using these services. Because of the limitation of network infrastructure, each physical server is typically used for deploying a networking service for a group of users. In this model, data is centrally stored; As a result, ensuring its privacy is a critical requirement. In current systems, an administrator could use access control mechanisms provided by operating systems to prevent illegal access. However, these solutions are difficult to deploy and operate. Because of this issue, the administrator has difficulty comprehensively handling data security issues in a shared environment. In this paper, we propose a new approach using virtualization concept for securing data in this shared environment. The approach has been successfully implemented in the Linux environment, and has shown positive results. It can properly solve data security problems in the shared environment and surmount the weaknesses found in current solutions.**

***Index Terms*—Access control, data security, shared environment, virtualization.**

I. Introduction

A. Shared Environment

There are two models for deploying the shared computer hardware resources: the Virtual Private Server (VPS) hosting model and the Shared Server hosting model. In the VPS hosting model, a single physical server is partitioned into a number of virtual machines which known as VPS by using virtualization technologies. Each VPS is allocated to a single user. The advantage of this model is total isolation between VPSs; problems with one VPS cannot affect the others. However, using this model can create a resource-consumption problem. Each VPS must have a full operating system to operate. Since the operating systems may be the same between VPSs, the cost of maintaining and running these redundant operating systems is extremely high and unnecessary. Consequently, the VPS hosting model cannot and should not be used when the number of customers is large. By contrast, only one operating system is needed in a shared hosting model. Applications will be installed in the operating system and multiple users will be served based on the support of these applications. The Shared hosting model is widely used because of its cost efficiency and ease of deployment and management. The Shared hosting model has these following characteristics:

1) Each customer is mapped to a user in the operating system and permission of the customer depends on permission of the user. All customer processes have to run with permission of his/her representative user.
2) Data of all customers is stored in the same file system. Each customer is allocated a folder for storing their data. From the operating system perspective, data of each customer is owned by its representative user.
3) Services operate under permission of a system user. In order to run services normally, a system user has to access to data of all other users.

B. Data Security Problem in Shared Environment

Because data of all customers is stored in the same file system, data of a customer can be accessed by other customers if the system administrator does not apply a proper security mechanism. For example, in a shared web hosting environment [1], if the attacker successfully compromises a website, other customers in the same server may also be vulnerable to data theft risk. After compromising the website, attackers may try to access data on the server by using various web attack techniques, such as: local file inclusion, remote file inclusion, directory traversal [2]. If the data of other websites can be accessed by the attacker's scripts, then he or she can search for sensitive data and use it for further the attack. The principle consequence and danger of these attacks is that one user can access all other users' data.

Nowadays, system administrators use access control mechanisms for preventing illegal access between users in the system. In this solution, data of each user will be assigned a permission according to the rule: data of a user must be only accessed by their owner and system users. There are three main access control models: discretionary access control, mandatory access control, and role-based access control. Each access control model has its limitation which make it difficult to completely solve the data security issues in a shared environment. In [3], authors pointed out some weaknesses of these access control models.

In discretionary access control model, each file/folder in system is owned by a user, either owner of a file/folder or root can assign file permission. If a user does not understand the access control mechanism and interaction between processes in the system, they cannot properly assign permission for their data. For example, in shared web hosting environment, if user is careless or does not fully understand file permission, their data can be not assigned truly permission. As a result, this user may also be vulnerable to data theft risk. In other case, user can secure their data by preventing all access from other users in

Manuscript received December 1, 2014; revised February 10, 2015.

Thi Thu Giang Tran, Duc Quang Le, and Trung Dung Tran are with the Department, Faculty of Information Technology, VNUHCM-University of Science, Vietnam (e-mail: {tttgiang2510, ducquang92}@gmail.com, ttdung@fit.hcmus.edu.vn).

system. However, this may lead to a mistake: the system user cannot access that data, then services are corrupted for this user. Therefore, overall security of system is determined by security knowledge of all users in the system.

In mandatory access control model, only the system administrator can decide the access permission of users in the system. The system administrator defines a policy set which ensures the system's security. The complex interaction between components in system requires a tough policy definition set. One example of mandatory access control is SELinux [4] in Linux operating system. In SELinux, all subjects (processes) and objects (files, sockets) in system are assigned a security label. System administrator must define a security policies set that determines the interaction among security labels. For instance, subjects with a specific security label can access objects with the corresponding security label. When a subject (e.g. a process) accesses an object (e.g. a file), the kernel will check whether this access is valid or not based on the predefined rule sets. This strict policy model make SELinux difficult to deploy, so most system administrators disable it [5].

Role-based access control model is the permission model which combines several roles, each role is defined as a set of access permissions. Each user is assigned some particular roles which imply the user's access permissions. In a large environment, a role-based access control model meets the difficulty of assigning permission to each role, and mapping each user to his/her respective roles. Moreover, each user can belong to several roles which leads to complexity in managing overall system security.

In summary, the approach using access control mechanism to strengthen data security in a shared environment is not a comprehensive solution. These methods have several major disadvantages: effectiveness of solution depends on users' security level, complexity and difficulty in implementation. Moreover, these solutions are not specifically designed for use in a shared environment, so they must be implemented in a way that conforms to the security characteristics of the shared environment, which is not native and often unintentionally creates mistakes. One can rightly conclude that having a dedicated solution for data security in a shared environment is very important.

Through this paper, we propose a new approach which is designed for this environment: using virtualization at file system level techniques. Firstly, we clarify requirements for data security in a shared environment based on the provided characteristics of the session. Secondly, we propose a security model complying with these requirements. Third, an implementation of the suggested model in the Linux Operating System is introduced. In addition, we also explain how this implementation conforms to these requirements. Finally, we evaluate the approach and suggest some future works.

II. Related Works

Existing approaches have applied virtualization technology for solving data security issues because a major advantage of these techniques is resource isolation ability. If data is stored in two different virtualized environments, security of data will be guaranteed. Virtualization techniques can be classified into two groups: hypervisor-based virtualization including *VMWare*, *Microsoft Virtual Server* and *Xen*; and container-based virtualization including *Linux Container (LXC)* in Linux operating system, and *FreeBSD Jail* in FreeBSD Operating System.

However, in shared environment, container-based virtualization is more popular because its performance cost is much lower than hypervisor-based virtualization's [6].

In container-based virtualization, operating systems have some special features which allow running multiple isolated user space instances, known as containers. Each container can have some particular processes, and each process is only assigned to one container. Because all containers are in the same operating system, the performance of this model is much better than those of the hypervisor-based virtualization model. However, the purpose of these studies mainly focuses on isolating malicious processes [7], not for securing data in the shared environment; the object of these solutions is a group of processes from which the system administrator will manually choose a set of processes to run in a container. These processes are isolated and cannot affect any other resource in the system. In order to apply this model for securing data in the shared environment, we have to make a container for each user and put all processes of a user in his/her corresponding container. In a shared environment, all processes of users are created automatically by the application running services; we cannot manually set containers for these processes. The services need to be modified in order to integrate with this model. In practice, the cost of modifying source code discourages application developers from supporting this model.

III. Virtualization at Filesystem Level Approach

A. Problem Formulation

Basing on the characteristics of the shared environment, we propose a new approach which is designed with the purpose of ensuring users' data security and surmounting weak points with existing current solutions. The approach has to satisfy four following requirements.

1) Data is isolated between users in the system.
2) There is a flexible method of sharing data among users in group.
3) Impact on overall system performance is minimized.
4) Integration of the new solution can be made easily into the existing system without any changes in operations of applications on the system.

Our new approach is virtualization at the file system level over users in the system, which is illustrated in Fig. 1.

In this model, each user has a different view of the file system hierarchy from the others. Users are able to see only data that they have privileges to access. In order to satisfy the first three requirements: 1), 2), and 3), the file system is divided into three parts:

- The first part is shares the data of all users in the system, such as the OS, libraries, system files, data of applications, etc. All users in the system use this part of data together in order to reduce unnecessary data storage and data processing cost 3).
- The second part is to share data with other groups of

users. Defining how directories and files can be shared among users in a group. This must be flexible with configuration files and is dependent on the demands of the users; a user might belong to several groups 2).

- The last part is the privacy of user data. Each set of data must be totally isolated from the others. Moreover, the shared data of a group of users can be seen only by users in the group. This will guarantee the privacy of each user's data in the system 1).

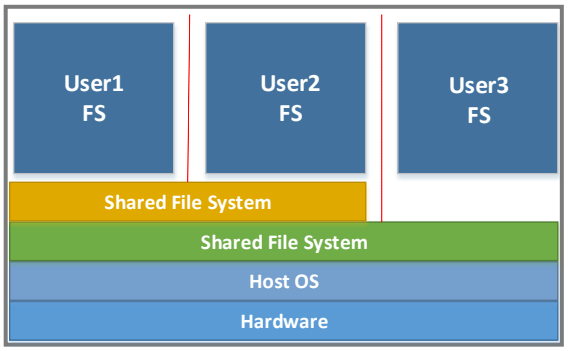

Fig. 1. Virtualization at file system level.

B. Our Solution

We deployed the model on the Linux operating system.

First of all, to create the isolated view of the file system hierarchy for each single user, we must use mount namespace [8], which is a feature of the Linux kernel. Mount namespace isolates the set of mount points seen by a group of processes. One process belongs to a unique mount namespace. The file system hierarchy is a set of mount points. Thus, processes in different mount namespaces can have different views of the file system hierarchy.

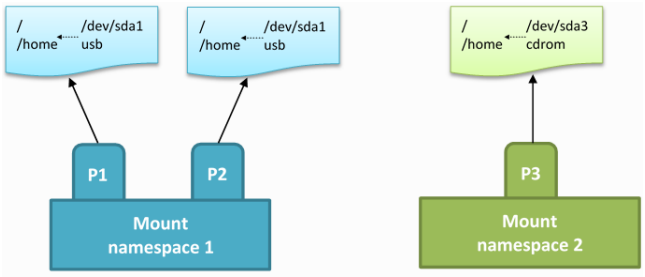

Fig. 2. Mount namespace.

Fig. 2 illustrates two different namespaces in a system which has its own mount points set. For example, mount point / in the first mount namespace is mounted from partition *sda1* and / in the other is mounted from *sda3*. Because the mount points set of each namespace is isolated from the other and the mount operations of them are dissimilar. Process P1 which belongs to namespace 1 has different view of system from P2 which belongs to namespace 2. In the other words, P1 and P2 will see different contents although they access to the same directory. In this case, P1 will see data of partition *sda1* and P3 will see data of *sda3*.

In the original Linux OS, all processes belong to the only namespace – global namespace. That is the reason why all processes in the system have the same view of the file system. Any operation within the file system, such as mounting, un-mounting or moving data, has the same effect on all of them. Hence, to isolate the view of file system of each user from the others in the system, each user must have its own dedicated mount namespace which distinguishes it from others' namespaces. This means all processes of a user belong to a unique mount namespace and the processes of different users are in their own, discrete namespaces. A process's UID is used to determine the system resources it can access, so that defining the namespace that a process belongs to is based on the UID of the process. In Linux, the init process is the parent of all processes. Its UID is zero (UID of root). When a new process is created, it copies the UID and namespaces from its parent process. In this case, there is no need to change a namespace for a child process. In order to ensure that all processes of a user are in the unique namespace which is different from other users' namespaces, we have to change namespace of a process whenever its UID is changed. The process can only change its UID by calling system calls set*uid()[1]. Therefore, we handle the transformation of processes' namespace only when they invoke set*uid(). We overwrite the system calls set*uid() in order to move processes to suitable namespaces. The new namespace is decided based on the new UID of process. The comparison between original system call and modified system call is shown in Fig. 3 and Fig. 4.

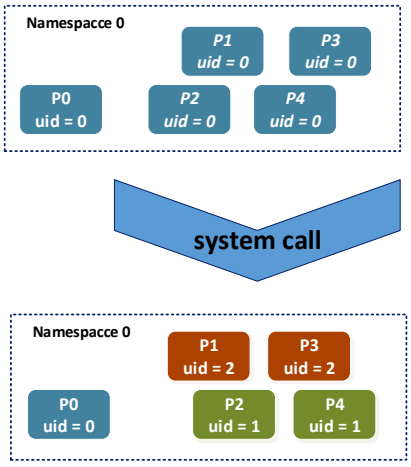

Fig. 3. Original system calls.

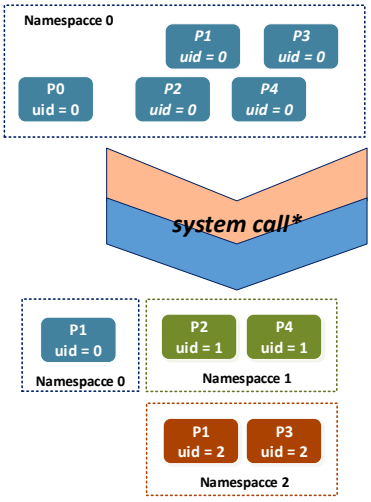

Fig. 4. Modified system calls.

[1] set*uid() includes all system calls that used for changing at least one of UID types (RUID, EUID, SUID) such as setuid(), setresuid(), setreuid().

The result of a modified system call is all processes of each user belong to the dedicated mount namespace for that user.

After the first milestone — creating dedicated namespace for each user mount points set of each user are isolated from the others. However, mount operations set in dedicated mount namespace of each user are still the same with the original mount operations set of root so all of the users still see the same view of the file system. Consequently, we need a second milestone — remounting all necessary mount points in user's own namespace to make data that the user does not have the privilege to access become invisible. The unprivileged data of a user includes data of other users and data which is not shared with them. User cannot see or access this data because it is totally invisible to them. How mount points are remounted depends on the users' demands. Which directories and files have to be isolated from other users and which ones are shared among several users can be configured easily in the configuration file. Necessary mount points are remounted after following these three steps.

Step 1: In directories which need to be virtualized, we mount all data that user is allowed to access to a corresponding directory: /.vdir/<UID>/<name of original directory>. After finishing this step, directory /.vdir/<UID> includes all data which that user with the appropriate UID is allowed to access.

For example, a simple web server in which */home* and */var/www* contain data of all users (Fig. 5). After finishing the first milestone — isolating mount points set of user, we need to remount data in */home* and */var/www* in the user's namespace in order to ensure that the private data of users cannot be seen by any others. In terms of user *ldquang*, its own data is in directories named *"ldquang"*. In addition, directory named *"shared data"* is configured to share with *ldquang*. In the first step, all data on which *ldquang* has access to is mounted to */.vdir/1* (1 is the UID of *ldquang*). Data in */var/www/ldquang* is mounted to */.vdir/1/var/www/lquang*, and that in */var/www/shared_data* to */.vdir/1/var/www/shared_data*. As a result, */.vdir/1* includes all privileged data of *ldquang*.

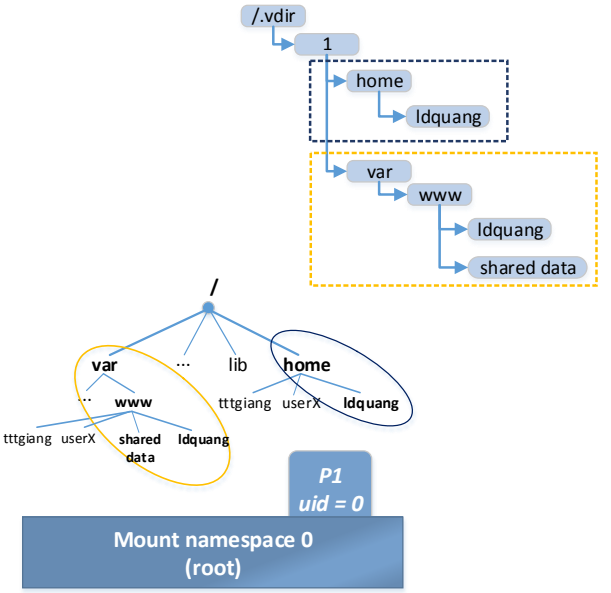

Fig. 5. Step 1 — Mount data to/.vdir.

Step 2: We remount data in /.vdir/<UID>/<*name of directory need to be virtualized*> into the correlative directories. As a result, the private data of other users is concealed from this user.

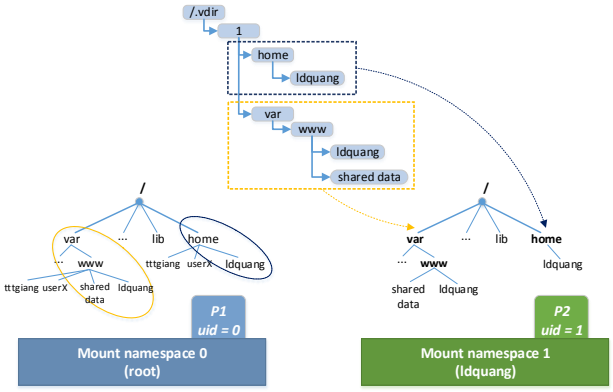

Fig. 6. Step 2 — Remount to virtualized directories.

As illustrated in Fig. 6, data in */.vdir/1* is remounted to */var/www* and */home/ldquang* respectively inside the dedicated namespace of *ldquang*. Consequently, *ldquang*'s view of the file system hierarchy is different from its original one and *ldquang* can only access its own data and shared data.

Step 3: We create an empty directory /.vdir/<UID>/EmptyDir and then remount it to */.vdir*. Therefore, all current data in */.vdir* disappears.

Thus, view of file system hierarchy of a user is changed though two milestones with many steps, which is demonstrated in Fig. 7.

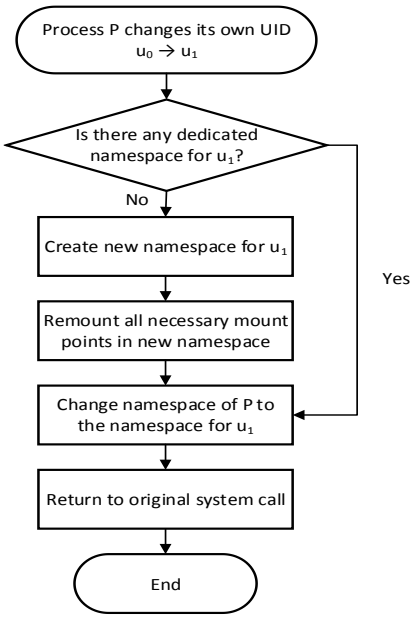

Fig. 7. Process of the approach.

Whenever a process change its own UID from u_0 to u_1, the modified system call will change the process's namespace to namespace of u_1 immediately if there is a dedicated namespace for u_1. Otherwise, a new namespace will be created for u_1, then all necessary mount points of the new namespace will be remounted. After that, the process's namespace will be change to a dedicated namespace for u_1. In the final step, the result will be returned to the original system call to continue processing.

In summary, our implementation meets requirement (1) based on its virtualization characteristic. Because a user can only access his data, data is isolated among users in system.

The flexible configuration feature is best suited to the requirement (2). In our module, administrator can decide which directories will be virtualized. In the best practice, directories storing common data (library, system file) should not be virtualized, so they can be shared between all users. Moreover, in each virtualized directory, administrator can specify users who are not affected by virtualization, so these users can access data normally.

Because all users use the same operating system and essential data is shared between them, the cost for maintaining and running new system is minimized, which satisfies requirement (3).

The requirement (4) is satisfied by using kernel module to modify necessary system calls. Our implementation operates in kernel space, so all applications in system do not need to be modified and normal operations of the system is not affected.

IV. Evaluation

A. Security

The new approach — virtualization at file system level has been deployed successfully on two popular Linux distributions as CentOS and Fedora. As can be seen in the real experiments, systems on which have deployed the new kernel module can completely isolate users' file systems. On the new system, users are not able to see or access data which does not belong to them, or is not shared with them by other users.

In order to prove the effectiveness of our solution on real-world attack scenarios, we have built a web server running PHP framework on CentOS operating system and hosting two websites called website A and website X. Website X has some security vulnerabilities. As an attacker, we exploited the vulnerabilities of website X, and successfully compromised it. Then, we uploaded a popular malicious PHP script named C99 to website X and executed the script to find all sensitive files of website A. Before deploying our solution, all files of website A and some sensitive system files can be accessed by the malicious script. However, after applying the solution, the attacker could only browse files of website X and could not access files of website A. We have also tried two other web-based attack techniques including file inclusion attack and directory traversal attack on website X and obtained the same result: attacker cannot access files of website A by performing these attacks.

B. Adaptability

Moreover, the easy integration of the new module and existing shared servers which are providing networking services to customers is one of its most striking features. We have successfully deployed our solution to existing Linux servers running commonly Linux distributions such as CentOS, Fedora, and Debian. There is no need to change any component in the servers, as well as no error detected in these systems. Besides that, working well with existing applications, especially applications that are usually used to deploy services for customer such as cPanel, DirectAdmin is also ensured by this new module.

C. Performance Impact

One of the most concerns about the solution is whether adding codes to an existing system affects its performance or not. Therefore, we have conducted several experiments on different common Linux distributions to measure the performance impacts after integrating our solution to the existing systems.

As can be seen from these experiences' results, there are not any considerable impacts on system performance after applying this new approach. For more details, the performance comparison between the original servers and the integrated servers had been implemented and shown positive results. The test scenario on a CentOS server is detailed below.

A web server is implemented on a virtual machine created by Vmware. Virtual machine specification includes CentOS operating system, 1GB RAM, and 20GB HDD. Virtual hosts' names in turn are www.host1.local… www.host20.local, and these virtual hosts are granted to users named host1... host20, respectively.

We use Apache Jmeter to evaluate web server's performance. For more details, HTTP requests are constantly sent and HTTP Responses are constantly received. After that, throughput and error rate will be measured. This process will cover two cases involving the web server before and after integrating with our new module. In each case, there are eight stages with different numbers of HTTP Requests vary between 600 requests and 2400 requests.

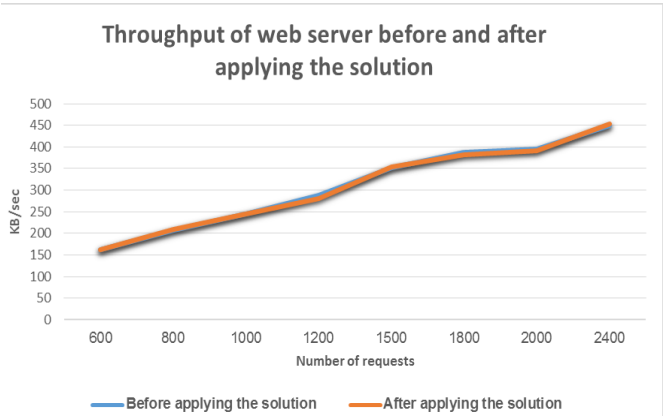

Fig. 8. Performance comparison.

Result of the test shows that the error rate is 0% in both cases and the throughput of the server before and after integrating with our new solution has been illustrated in the line chart above (Fig. 8). As shown in the chart, disparities between the two test cases are insignificant. Hence, the integration of our new solution causes neither error nor considerable influence on the overall system's performance

In order to measure the repercussion of the new module on regular operations of the kernel and system services, we used LTP (Linux Test Project) test tool [9, 10]. Results of testing on server before and after deploying our new solution are similar. Consequently, this proves that the modified system calls on our kernel module do not cause any problem in kernel functions and system services.

V. Conclusion

The virtualization at file system level approach not only ensures security for user's data but also overcomes the problem of users' carelessness with a low cost. Besides that, our new kernel module is able to be integrated into existing server without moving data and changing it from current applications' operations. This is a completely new approach which can be considered by service providers or companies in order to increase the privacy of users' data in shared server.

In the future, this solution will be fully developed to become a more comprehensive solution with the lowest cost. The development involves virtualizing files containing users' information on system such as /etc/passwd and /etc/shadow, and limiting resources used by each user.

Acknowledgment

Our thanks to DTS Communication Technology Corporation for supporting us real testing environment.

References

[1] C. Jame. (November 2013). White paper: Hosting solutions compared. [Online]. Available: http://www.cjonlinedesign.com
[2] Imperva Corporation. (2013). White paper: Web Application Attack Report. [Online]. Available: http://www.imperva.com
[3] A.-C. Ryan (2006). Methods for access control: advances and limitations. [Online]. Available: http://www.cs.hmc.edu/~mike/public_html/courses/security/s06/projects/ryan.pdf. Accessed 9 Oct 2013.
[4] S. Stephen, C. Vance, and W. Salamon. "Implementing SELinux as a Linux security module," NAI Labs Report 1, vol. 43, 2001.
[5] V. Fernando, T. Horie, and T. Harada, *The Need for Setuid Style Functionality in SELinux Environment*, 2004.
[6] P. Padala, X. Zhu, Z. Wang, S. Singhal, and K. G. Shin, "Performance evaluation of virtualization technologies for server consolidation," HP Labs Tec. Report, 2007.
[7] S. Soltesz, H. Pötzl, M. E. Fiuczynski, A. Bavier, and L. Peterson, "Container-based operating system virtualization: A scalable, high-performance alternative to hypervisors," in *Proc. the 2nd ACM SIGOPS/EuroSys European Conference on Computer Systems*, 2007.
[8] M. Kerrisk, *Namespaces in Operation, Part 1: Namespaces Overview*, August 2014.
[9] P. Larson, "Testing linux with the linux test project," in *Proc. Ottawa Linux Symposium*, 2001, pp. 265-273.
[10] N. Hinds, "Kernel korner: The Linux test project," *Linux Journal*, August 2014.

Thi Thu Giang Tran was born in Vietnam in 1992. She received the B.S. degree in information technology from VNUHCM-University of Science, Vietnam, in 2014. She has worked as a developer in Miracle Linux Corporation-Asianux HCMC development center % UTS, since 2014. Her research interests are focus on network security, cloud platform and distributed systems.

Duc Quang Le was born in Vietnam in 1992. He received the B.S. degree in information technology from VNUHCM-University of Science, Vietnam in 2014. He has worked as a security instructor in Saigon Institue for Technique and Technology, a local vocational training center in Vietnam, since 2014. His research interests are focus on network security and system security.

Trung Dung Tran was born in Khanh Hoa, Vietnam in 1978. He received his PhD degree from UT at Dallas and his Msc degree from SUNY at Buffalo in 2010 and 2006, respectively. He is the head of Computer Networks and Telecommunications Dept., University of Science since 2012. His research interests are focus on network performance, network protocols, and the line between Computer networks and Game Theory.