IJCRT.ORG ISSN : 2320-2882

INTERNATIONAL JOURNAL OF CREATIVE RESEARCH THOUGHTS (IJCRT)
An International Open Access, Peer-reviewed, Refereed Journal

REVIEW AND ANALYSIS ON VARIOUS APPLICATIONS OF SOPHISTICATED LINUX OPERATING SYSTEM (LOS)

Atif Ali Mohammed, Information technology, Charles Sturt University, aliphd59@gmail.com.

ABSTRACT

Linux is an advanced artificial system and has become one of the most popular OSes in the world. However, from a networking perspective, nothing is known about how LOS structures and functions have changed over time. This paper delves into the history of the LOS system and how it has developed. Focusing on the similarities and differences across IOS, Android, Mac, Windows, and Linux, this study examines the visual operating system in detail. The latest versions of Linux, Android, and Windows 10 are the most dependable, compatible, and stable options. While other operating systems struggle to gain popularity, Linux, Android, and Windows have enough users to encourage them to improve their user interfaces and create more useful applications. This study presents the reviews on various studies and their various applications of Linux under various applications.With its capacity to facilitate continuous development and boost the effectiveness and dependability of runtime environments, container-based virtualization is gaining traction across a variety of industries. In addition, several methods are offered for keeping tabs on containers' safety, and they are also discussed. However, there are no rules to follow while picking the best methods for the task.

1. INTRODUCTION

Nowadays, the scale of HPCs can reach thousands of nodes, and practically all of the world's top supercomputers use Linux on every single node. An OS may be responsible for distributing the application among the available hardware resources to provide concurrency with asynchronous parallelization of workloads. Thus, the OS is responsible for things like thread/process management, data synchronisation, and inter-process communication [1]. Therefore, the OS uses some of the available computing power to coordinate the allocation and management of hardware resources. Meanwhile, time lags between fast processes and slow ones began to affect system performance in large-scale applications utilising hundreds to thousands of nodes.

1.1 What do you mean by Linux? Explain its features.

Linux, an open-source OS modelled after Unix, controls the hardware and resources of a computer, including the central processing unit (CPU), memory, and storage, and the way in which software and hardware interact. First published on 5 October 1991, Linux is a computer operating system developed by Linus Torvalds. It is widely regarded as being safer and swifter than Microsoft's Windows. It is open source and commonly based on the Linux Kernel, and its distribution is unrestricted (low-level system software that is used to manage hardware resources for users). Additionally, it is compatible with a wide variety of mobile devices, computer systems, notebooks, etc. Ubuntu, Debian, SUSE Linux, Gentoo, etc. are all flavours of the Linux operating system [2].

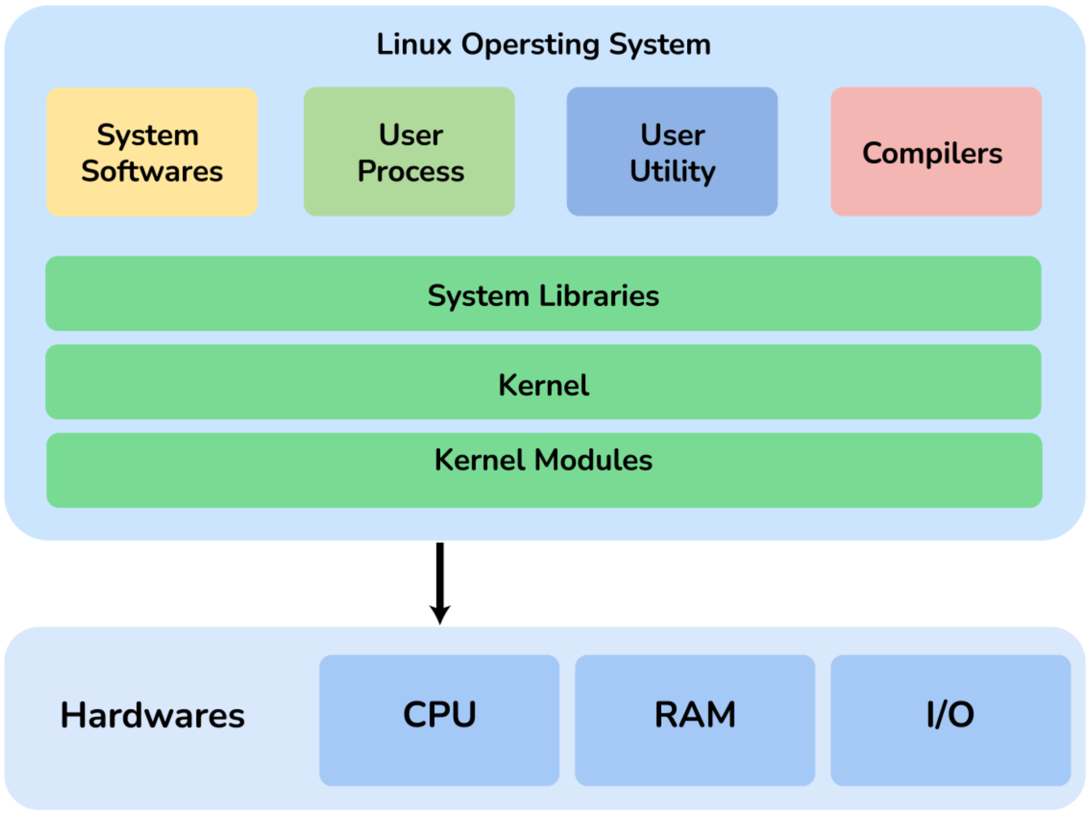

Fig 1: Linux Operating System.

Linux's OS has many useful features, including:

- It is available to anyone without cost or restriction, thanks to its "free and open source" status.
- Linux is known for its stability and versatility, as it can run for long periods of time without crashing and is hardly ever compromised by malicious software.
- It's safer since it has authentication features like password protection, audits of security, and restricted access to files.
- Multiple tasks or programmes can be executed on a single computer at the same time on a multiprogramming system.
- To help its users find and install the programmes they need, the platform includes a software repository, or central database.
- Allows for user-defined keyboard shortcuts Linux recognises the wide variety of languages spoken throughout the world and allows users to install keyboards for a wide variety of languages.
- The Graphical User Interface (GUI) allows users to interact with the system and run graphical programmes like VLC, Firefox, and others.

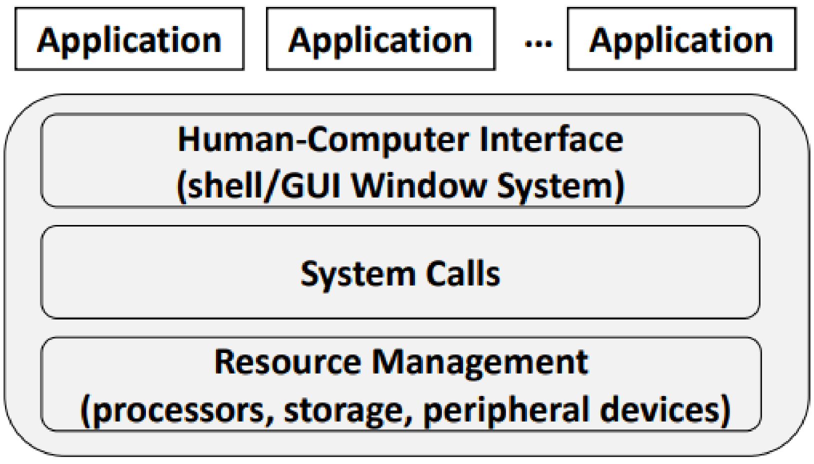
Application
Application
...
Application
Human-Computer Interface (shell/GUI Window System)
System Calls
Resource Management (processors, storage, peripheral devices)

Figure 2. The architecture of a traditional OS.

As can be seen in Figure 2, an OS provides system calls and human-computer interfaces for applications and end-users, as well as resource management capabilities on processors, storage, and peripheral device components. When we examine an operating system from several vantage points, we see different viewpoints:

• An operating system can be thought of as a resource manager from the point of view of a computer system. An operating system (OS) coordinates and manages the use of all of a computer's low-level hardware and software resources, allowing for their most effective and efficient deployment. Furthermore, an OS facilitates greater system-wide interoperability by bridging differences in hardware resources via hardware drivers.

• Users of the system can think of an operating system as a computer simulation. A layer of abstraction provided by an OS hides the granularity of the hardware resources underlying the software. However, it also provides user-friendly interfaces. An operating system also sets the standard for how applications built on top of it are programmed.

• An operating system (OS) can be thought of as a platform for creating and running applications. All other application software can be created and run successfully with the help of the infrastructure it provides. For instance, an OS typically consists of a software development and maintenance tool set, an execution environment for application software, and runtime resource management and scheduling.

2. LITERATURE REVIEW

R. Sairam and colleagues (2019) [3] With more and more smart devices and objects appearing in everyday life, the IoT is making strides forward. As a result, our lives have become increasingly dependent on these technological elements. Small-scale protective features and weaknesses in these systems are of great concern to these intelligent devices because of cyber thieves' advantage in complexity. Conventional centralised IT security techniques have limited scalability and expense. This type of smart gadget would function better if it could be managed at the edge of an IoT network, close to where it is located. On the network edge, various security measures can be applied to safeguard mobile devices in a smart home or corporate setting. Introducing network edge protection features necessitates the use of NFV, which we discuss in detail in this paper. To accomplish this, NETRA is developing a new lightweight, networkbased docking architecture for IoT security virtualization features. Using the suggested design, we show how it has advantages in terms of memory utilisation and latency, as well as performance, average load and scalability, over the present NFV system. We evaluate the proposed NFV-based IoT protection edge detection and show that threats with more than 95% precision may be estimated in less than one second.

S. Sultan and colleagues (2019) [4] In order to better support the design of microservices, containers were created as a lightweight replacement for VMs. Containers' market worth is expected to grow from 762 million dollars in 2016 to 2.7 billion dollars in 2020. Container health is a major worry for many businesses and a barrier to adoption, despite their status as a simplified approach for providing micro applications in rapidly developing sectors like cloud storage and application meshes, according to business research. Our focus in this study is on container safety and solutions. The threat posed by host containers prompted us to create four widely used programmes to solve the security concerns. For example, one may use it to defend a container from programmes running inside it, or to protect the container itself from those apps, or to guard against containers attacking the host computer (IV). Our software-based solutions in the first three situations all rely on Linux kernel capabilities (e.g.nameplaces, CGroups, and seccomp), as well as protection modules (e.g. AppArmor). TPMs and other hardwaredriven security solutions, such as trusted device support, are the primary emphasis of the latter framework (i.e., Intel SGX). We expect that this evaluation will aid researchers in gaining a better knowledge of container security vulnerabilities and threats. In addition, we point out unanswered scientific questions and potential routes of research that might lead to greater research in this field.

According to J et al (2020) [5], C Diekmann, J and others For complicated systems like Docker, Linux containers are becoming more and more frequent. However, for distributed microservice deployment, the primitive safety of network access control is frequently overlooked or left to the network operations team. Access control lists at the network layer aren't granular enough to enforce the security requirements of individual applications. Docker and network operators may work well together, but they still don't provide for granular control over networking between containers or in application creation. In this made-up story, we're following along as DevOp Engineer Alice builds a website. We show what Alice is meant to perform and help with the tools required for it all the way from the design and software engineering phases to network operations and automation. Alice, as a DevOps full stack, deals with issues of superior design and networking. By focusing on network access control and building out a tool-based solution, we have exposed the flaws in today's policy management. Academic research shows that a full stack engineer does not link many existing instruments between the various abstraction layers. With Isabell / HOL, our tools are open source and subject to regular evaluation.

3. REVIEWS ON SELECTION OF TECHNIQUES FOR MONITORING CONTAINERS SECURITY

To improve the effectiveness of monitoring and analysis of Linux containerized programmes, SPEAKER Lei et al. (2017) developed a general-purpose non-intrusive technique called speaker analysis. By eliminating unused system calls that could be used by malicious processes inside the container, SPEAKER greatly reduces the attack surface.

This sandbox approach is used to thwart assaults and threats that include tampering, like malicious code and unauthorised access to a network. SystemTap is used in this method to monitor and record malicious software activity (SystemTap, 2020). SystemTap is an open-source software framework for efficiently capturing data on Linux processes. This method can effectively conceal the environment artefacts from the infection since it allows changing the values of syscall parameters. In conclusion, this method ensures that the containerized solution is able to fool sandbox evasion through the use of artefact obfuscation, network reorganisation, and system call introspection.

LiCShield is a tool developed by Mattetti et al. (2015) to keep an eye on and protect Linux containers running in the cloud. LiCShield protects against tampering, information disclosure, and privilege elevation. Targeted attacks include things like kernel exploits, attacks on shared kernel resources, incorrect configurations, malicious modules, and data leakage. With LiCShield, security profiles for a container's execution on the host and inside the container are generated automatically. This method uses a training environment's execution of activities and operations to mechanically define rules outlining the typical behaviour of containers. When abnormalities are detected, LiCShield can restrict the capabilities of containers by creating profiles of kernel security modules based on the applications' execution. This prevents criminal schemes from being carried out and spreads.

For real-time monitoring of programmes within Linux containers deployed locally or in the cloud, Abed et al. (2015) suggest a non-intrusive method.

Methods like malware injections, OS compromise, file system access, and brute force attacks are used in this strategy to spoof, tamper, and launch denial of service attacks. Bags of System Calls (BoSC) analysis is employed, which is a sliding window and frequency-based analysis technique that counts how often a given set of system calls occurs within a given window of size k, where k is the number of system calls being monitored at any given time instant. This method makes use of features available in Linux, such as strace, which records system calls and displays details such as the calling process's ID, the arguments used, and the value returned. The method learns how the containerized applications behave by monitoring these system calls, and it can then identify any abnormalities in the surrounding ecosystem.

To maintain containerized stateful applications running in a secure environment, researchers from Sayed and Azab's (2019) lab developed a general-purpose container monitoring approach called Time Machine (TM). The method takes aim at both tampering and denial-of-service attacks. For mission-critical infrastructures in particular, this prevents the triggering of logic bombs. Logic bombs are intentionally inserted lines of code that, once triggered by user input, unleash a slew of harmful behaviours.

4. A MEASUREMENT STUDY ON LINUX CONTAINER SECURITY

A. Xin Lin, 2018, The Linux container technique is gaining popularity and is being used more frequently to deliver business applications. Despite widespread agreement that the container mechanism is insecure because of its reliance on a shared kernel, there has been surprisingly little effort put into doing a thorough and systematic analysis of the system's vulnerability to actual exploits. In this paper, we compile a dataset of attacks utilising a 2-dimensional attack taxonomy to categorise 223 exploits that work on the container platform. Then, after removing common exploits from the dataset, we assess the safety of the current Linux container system with the help of 88 of the most severe ones. Fifty of the exploits we tested (54.82%) are able to successfully launch attacks from within the container when the default settings are used. Due to the fact that privilege escalation exploits can completely disable the container protection mechanism, we analyse them in detail. Although container isolation measures are useful, we discover that kernel security mechanisms like Capability, Seccomp, and MAC are more crucial in preventing privilege escalation (i.e., Namespace and Cgroup).

There are two main reasons why container technology is gaining popularity in the business world. Before anything else, container orchestration tools like Docker [Docker Inc. 2018] and Kubernetes make it easier to deploy, scale, and manage containerized applications. As a result, containerization is gaining traction in the manufacturing setting. Also, cloud providers like Amazon Fargate [2018], Microsoft Azure Kubernetes Service, etc., are starting to offer container services. Furthermore, the container mechanism is a lightweight OS-level virtualization technology, making it more appealing to the resource-constrained mobile platform. Many container-based BYOD solutions have been proposed and implemented [VMware Inc. 2018].

However, however, security worries have been the main obstacle to wider use of the container technique. Most notably, once the Linux kernel is compromised, the isolation given by the container system will be completely nullified, as all containers running on a single host share the same kernel. As a result, it's important to provide a thorough assessment and study of the container mechanism's safety. The majority of the existing research evaluates container security at the level of the system architecture or design principles.

The isolation techniques provided by Docker, LXD, and Rkt are compared and contrasted in a 2017 paper by M. Ali Babar et al. In a short discussion focused on system design, Thanh Bui et al. [2015] contrast the safety of hardware-based virtualization technology (such as XEN) with that of operating system-level virtualization technology (i.e., the container mechanism). Several OS-level virtualization technologies, including FreeBSD Jails, Linux-VServer, Solaris Zones, OpenVZ, LxC, Cells, etc., are theoretically analysed by Reshetova et al. [2014]. Potential vulnerabilities against container mechanisms like Docker are used by certain researchers [Tao Lu 2017, A Mouat. 2015] to assess container security.

4.1 Linux Kernel Security Mechanisms

All containers running on the same host share the same Linux kernel, which poses a serious security concern. The isolation afforded by the container technique is rendered useless if a process running within it affects the Linux kernel. Capability [Linux Man. 2018], Seccomp, and Mandatory Access Control (MAC) are just a few of the Linux kernel security techniques used to restrict the capabilities of the processes running inside containers. The Capability method breaks down the ROOT superuser access into 38 separate capabilities. The capacity to operate on a particular set of kernel resources is symbolised by a particular capability. Capabilities indicate what kinds of actions a user is allowed to take; for instance, the CAP NET ADMIN capability indicates network administration privileges. Docker containers come equipped with 14 features by default.

By limiting the kind of system calls a process may make, Seccomp helps ensure that it stays inside its boundaries. Docker uses a Seccomp profile called "le" to specify which system calls are exposed to a container; by default, this list is more than 300 system calls long [Docker Inc. 2018]. Seccomp and Capability are DAC techniques, while SELinux and AppArmorare MAC mechanisms used by containers. CentOS, RHEL, and Fedora have all incorporated SELinux, and Debian and Ubuntu have also incorporated AppArmor. To enforce policies, AppArmor uses a path-based paradigm, while SELinux uses labels.

4.2 CPU Protection Mechanisms

Linux kernel attacks can also be thwarted with the help of three CPU-level safeguards: Kernel Address Space Layout Randomization (KASLR) [Jake Edge. 2013], Supervisor Mode Access Prevention (SMAP), and Supervisor Mode Execution Prevention (SMEP) [Wikimedia Foundation Inc. 2018.]. Rather than always using the same base address for the kernel, KASLR will add a random slide to the base address at boot time. Supervisor Mode Access Prevention (SMAP) stops supervisor-mode programmes from accessing user space memory and Supervisor Mode Execution Prevention (SMEP) stops supervisor-mode code from executing user space code by accident. Turning on SMAP and SMEP required setting the 21st and 20th bits of the CR4 register.

4.3 Privilege Escalation Procedure through commit_creds().

Figure 3 depicts the fourth step of the procedure to gain administrative privileges by using the commit creds(), which is also depicted in Figure 3.

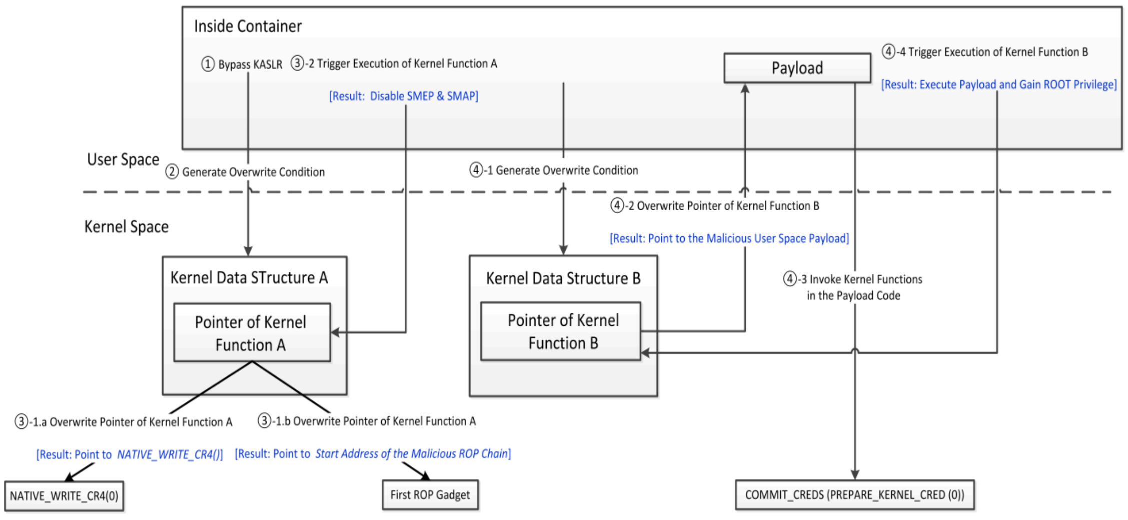
Inside Container
① Bypass KASLR
③-2 Trigger Execution of Kernel Function A
[Result: Disable SMEP & SMAP]
Payload
④-4 Trigger Execution of Kernel Function B
[Result: Execute Payload and Gain ROOT Privilege]
User Space
② Generate Overwrite Condition
④-1 Generate Overwrite Condition
Kernel Space
④-2 Overwrite Pointer of Kernel Function B
[Result: Point to the Malicious User Space Payload]
Kernel Data STructure A
Pointer of Kernel Function A
Kernel Data Structure B
Pointer of Kernel Function B
④-3 Invoke Kernel Functions in the Payload Code
③-1.a Overwrite Pointer of Kernel Function A
③-1.b Overwrite Pointer of Kernel Function A
[Result: Point to NATIVE_WRITE_CR4()]
[Result: Point to Start Address of the Malicious ROP Chain]
NATIVE_WRITE_CR4(0)
First ROP Gadget
COMMIT_CREDS (PREPARE_KERNEL_CRED (0))

Figure 3: Kernel Privilege Escalation Attack Model

In Linux kernel, the credential associated with a process is stored as two "elds inside the task_struct structure, i.e., cred and real_cred. The cred "eld represents the current privileges (including capabilities, GID, UID etc.) of the process, and could be temporarily modified during execution of the process. The real_credfield represents the highest privileges a process could reach, and normally could not be changed.

The KASLR mechanism has been proclaimed dead by many researchers, as current implementations of KASLR have fatal flaws [19]. Overwriting of the specific kernel functions' pointers is achieved by exploiting the vulnerabilities in the Linux kernel, such as UAF, race condition, improper verify, buffer overflow etc. And it is pretty unlikely to patch all vulnerabilities considering the large code size of Linux kernel. The CPU mechanisms SMAP&SMEP are easy to be disabled if the attackers compromise the KASLR mechanism and gain the ability to overwrite the pointers of some kernel functions. Therefore, we propose a defense system by forbidding the commit_creds() to be utilized to elevate the privilege inside the container.

CONCLUSION

Linux container is increasingly utilized by the industrial community. Although it is a consensus that container mechanism is not secure, a concrete and systematical evaluation is absent.Container-based virtualization is gaining popularity in different domains. Different techniques for monitoring containers security are proposed. However, there are no guidelines supporting the selection of suitable container monitoring techniques for the tasks at hand.We review the literature to identify relevant techniques for monitoring container-based virtualization environments with the goal to provide a comprehensive overview of these techniques. We further categorize the identified techniques to help developers understand their purpose, technical characteristics, applicability, and effectiveness.

Future work

To enhance the current version of the framework, we will address the comments that we received via the evaluation. Moreover, to complement the conducted evaluation and get further insights, we plan to evaluate the framework using more scenarios and involving more participants.

Furthermore, we plan to provide an interactive presentation of CONSERVE to assist the exploration and selection process of container monitoring techniques. We also plan to continuously maintain and update the framework to reflect eventual evolution of the considered monitoring techniques as well as include newly developed techniques.

REFERENCES

1. A. Silberschatz, P. B. Galvin, and G. Gagne, Operating system concepts essentials. John Wiley & Sons, Inc., 2014.
2. https://www.makeuseof.com/practical-applications-of-linux/
3. R. Sairam, S. S. Bhunia, V. Thangavelu and M. Gurusamy, "NETRA: Enhancing IoT Security Using NFV-Based Edge

Traffic Analysis," in IEEE Sensors Journal, vol. 19, no. 12, pp. 4660-4671, 15 June15, 2019. doi: 10.1109/JSEN.2019.2900097

4. S. Sultan, I. Ahmad and T. Dimitriou, "Container Security: Issues, Challenges, and the Road Ahead," in IEEE Access, vol. 7, pp. 52976- 52996, 2019. doi: 10.1109/ACCESS.2019.2911732
5. C. Diekmann, J. Naab, A. Korsten and G. Carle, "Agile Network Access Control in the Container Age," in IEEE Transactions on Network and Service Management, vol. 16, no. 1, pp. 41-55, March 2019. doi: 10.1109/TNSM.2018.2889009
6. Lei L., Sun J., Sun K., Shenefiel C., Ma R. , Wang Y., Li Q.SPEAKER: Split-phase execution of application containersInternational Conference on Detection of Intrusions and Malware, and Vulnerability
Assessment, Springer (2017), pp. 230-251
7. SystemTap S.A free software (GPL) infrastructure to simplify the gathering of information about the running linux systemSystemTap (2020)https://sourceware.org/systemtap. (Accessed December 2020)
8. Abed A.S., Clancy C., Levy D.S.Intrusion detection system for applications using linux containers
9. International Workshop on Security and Trust Management, Springer (2015), pp. 123-135Sayed M.M., Azab M.The time machine: Smart operation-resilience in presence of attacks and failures2019 IEEE 10th Annual Information Technology, Electronics and Mobile Communication
Conference, IEMCON, IEEE (2019), pp. 0127-0132
10. Xin Lin, "A Measurement Study on Linux Container Security: A!acks and Countermeasures", 2018.
11. Docker Inc. 2018. WHAT IS DOCKER. https://www.docker.com/what-docker
12. Amazon Company. 2018. AWS Fargate. https://aws.amazon.com/fargate/?nc1=h_ls
13. VMware Inc. 2018. VMWare Airwatch BYOD. http://acestandard.org/zh-hans/solutions/bring-your-own-device-byod
14. M Ali Babar and Ben Ramsey. 2017. Understanding Container Isolation Mechanisms for Building Security-Sensitive Private Cloud. Technical Report, CREST, University of Adelaide, Adelaide, Australia (2017)
15. Thanh Bui. 2015. Analysis of Docker Security. CoRR abs/1501.02967 (2015). arXiv:1501.02967 http://arxiv.org/abs/1501.02967
16. Elena Reshetova, JanneKarhunen, Thomas Nyman, and N. Asokan. 2014. Security of OS-Level Virtualization Technologies. In Secure IT Systems - 19th Nordic Conference, NordSec 2014, Tromsø, Norway, October 15-17, 2014, Proceedings. 77–93. https://doi.org/10.1007/978-3-319-11599-3_5
17. Tao Lu and Jie Chen. 2017. Research of Penetration Testing Technology in Docker Environment. (2017)
18. A Mouat. 2015. Docker Security Using Containers Safely in Production.
19. Linux Man. 2018. Capabilities - overview of Linux capabilities. http://man7.org/linux/man-pages/man7/capabilities.7.html
20. Docker Inc. 2018. Docker Seccomp Pro"le. https://github.com/moby/moby/blob/master/pro"les/seccomp/default.json
21. Jake Edge. 2013. Kernel address space layout randomization. https://lwn.net/Articles/569635/
22. Wikimedia Foundation Inc. 2018. Supervisor Mode Access Prevention. https: //en.wikipedia.org/wiki/Supervisor_Mode_Access_Prevention