Sandboxing in Linux: From Smartphone to Cloud

Imamjafar Borate
CSE Department
SGGSIE&T Nanded
India

R. K. Chavan
CSE Department
SGGSIE&T Nanded
India

ABSTRACT

In today's internet world, Malicious and malfunctioning contents from the internet are regular problems for host systems such as Smartphones, Desktops, Clouds etc. Almost all underlying operating systems provide security from most of the threats. However, we need to add some extra defense to our system. Sandboxing is an important security technique that lets programs run in its isolated environment. A sandbox is a tightly controlled environment where programs run. It provides access to a tightly controlled set of resources for programs, such as memory, scratch space on the disk, network access, and input devices. A program running in the sandbox has just as many permissions as it needs without having additional permissions that could be misused. Sandbox restricts a program to access resources outside the sandbox. Sandbox prevents malicious or malfunctioning programs from accessing rest of the system.

Nowadays, most of the mobile operating systems, desktop applications like web browsers, browser plugins, document viewers and cloud computing systems are using sandboxing mechanism to run applications. For the implementation of the sandboxing mechanism, software vendors rely on underlying operating system security features. There are different ways and approaches that can be used to implement sandbox mechanisms. This paper highlights the Linux security features such as Chroot, Cgroups, Capabilities, SCI, Namespaces, Seccomp, Resource Limit, LSMs such as SELinux, Virtualization and grsecurity that can be used in the implementation of the sandboxing mechanism.

General Terms

Sandbox, Virtualization

Keywords

Chroot, Namespace, Cgroups, Seccomp, Capabilities

1. INTRODUCTION

In today's globally networked society, security is a general term for everyone, from the individual user to the corporate giants. Security is important because the number of attacks against systems are increasing rapidly. Malicious content coming from the internet are a regular problem for host systems. Nature of problems is same for Smartphones, Desktops and Cloud Systems. The malicious content such as viruses, trojans, malwares, adwares, ransomwares etc. and vulnerabilities in the systems are major threats for Smartphones, Desktops and Cloud Systems. [28, 55, 32] Traditional operating system security features provide security from most of the threats. However, we need to add some extra defense to our systems. Sandboxing is an important security mechanism that lets programs and processes run in its isolated environment. Sandbox isolate running the program from host operating system and other running programs on the system. Sandbox is an environment where the program can only access restricted set of resources. It is a way to restrict a program's ability to access resources. It is different from access control applied to running processes. Typically sandboxes only apply to programs explicitly launched into a sandbox, where as traditional access control methods apply to all programs.[34, 46] The sandboxing is based on prevention instead of detection in the case of security. Most of the sandboxes provide isolation based approach where the program running in a sandbox is entirely isolated from resources outside the sandbox and programs running outside the sandbox. For this approach operating system features such as chroot, secomp, namespaces, cgroups, capabilities, hypervisor based virtualization, container based virtualization etc. can be used. Instead of completely focusing on isolating application, rule based approach aims to control what each application is authorized to do. This approach allows to applications to share the resources.[46]

For the implementation of the sandbox, software vendors rely on underlying operating system security features. So each software has different sandbox implementation for the underlying operating system. For example, Chrome has three different sandbox implementations for Linux, Mac and Windows[1]. It is believed that Linux systems are more protected and secure than Microsoft Windows. So Linux implementation of the sandbox is more powerful than windows. In this paper we discuss sandboxing mechanism on the Linux platform. Discussion on sandboxing on other platforms is not the part of this paper.

2. THREAT MODEL

As untrusted applications on the internet are increasing rapidly, it's extremely difficult to identify a benign application from a malicious application. Traditional operating system security features and antivirus solutions do not detect these malicious programs in real times. There are significant amount of applications on the internet claiming to be useful but contain viruses, trojans, malwares, ransomware and other malicious codes. These malicious contents can gain the access of host system, steal data on host machine such as userids , passwords, photos, videos etc, display unwanted contents such as advertisements , degrade system performance, crash system or other applications running on the system, make the network call such as send SMS, or access system resources such as camera of smartphone or computer for malicious purpose.[28, 55, 32]

Malicious programs can exploit the vulnerabilities in the system to attack a host system. Attacks such as Injection, Broken Authentication and Session Management, Cross Site Scripting, Buffer Overflow, Sensitive Data Exposure, Cross Site Request Forgery etc use the vulnerabilities in the system.[28, 55, 32] Malicious program injects its own code into the address space of the process. Thereafter it tries to execute the above said code with which it can get super user privilege either in process or kernel. Therefore the main task is:

—to prevent the malware from injecting code into the process or kernel.

—not to allow the execution of the malware code which may have been attached to some area of process or kernel memory.

This can be achieved by preventing the malicious program from getting detailed layout. This can be achieved by Address space layout randomization. Even if malware successful in attaching its code to process or kernel, this malicious code should not be allowed to execute by having non executable stack and heap segments. Also, exploitation of the vulnerability may lead to privilege escalation. Privilege escalation is an act of gaining additional privileges in which unprivileged user gains super user privileges.[45, 52]

A rootkit is special kind of malicious software that utilizes the privilege escalation.[3] Zero-day attacks are the another types of attack which is based on vulnerabilities that are not known or fixed yet. Various errors occur during writing the software and then eliminated by testing and real life usage. If software is insufficiently tested, cause number of problems specially operating system infections caused by malwares. These errors are often found after attackers have managed to abuse the fault. Subsequently, the software vender releases a patch and updates to remove the errors. The system remains infected because the attack has already happened.[13]

To enhance the display of websites or to make websites dynamic on the client side, JavaScript and Java Applets are used by web developers. JavaScript and Java Applets can also be used for malicious purpose. Nowadays, using JavaScript maliciously has become a major threat to client Machines.[9, 39, 35, 22, 14]

Nature of problems is same for Smartphones, Desktops and Clouds. All the above security related issues are major threats to Smartphones, Desktops and Clouds.

3. WHY SANDBOXING

The untrusted content from the internet needs to run in the restricted environment so that it can not affect the rest of the system. Sandbox is an important security technique that runs untrusted content in an isolated environment. To protect from malicious content, most of the mobile operating systems, Desktop applications such as Browsers, Document Viewers, Audio/Video players and Cloud systems use sandboxing mechanisms.

Smartphones have become pervasive due to the availability of Internet, Office Applications, Vehicle Guidance applications, Games, Multimedia services etc. The increased popularity of smartphones and associated monetary benefits attracted malware developers. Smartphones are vulnerable to spywares, viruses and phishing attacks as are home computers. The easiest way to compromise the security of the smartphone is downloaded Apps. Malicious downloaded Apps can steal the data of the other Apps or crash other Apps or the Operating System. Most of the mobile operating systems run their apps in a sandbox. A sandbox isolates apps from the host system as well as isolates app from other apps so that one app can not access the other App's data. Also, sandbox restricts unauthorized access to system resources by Apps.[20, 29]

The visited web sites or downloaded contents such as pdf documents or softwares can contain malicious contents such as malwares, viruses, adwares, spywares, ransomwares etc. Most of the Desktop applications such as web browsers, document viewers[24], browser plugins[43] etc run the content in the sandbox to restrict access to rest of the system. The web page can contain javascript or java applet code. This codes can be used for malicious purpose. The Browser sandbox restricts JavaScript code from accessing critical portions of the page's DOM, stealing cookies and navigating the page to the malicious site.[39, 35, 22] The JVM sandbox restricts the applets from performing many dangerous activities, such as file system or network access or the ability to execute applications.[22, 14] Browser plugins such as adobe flash player, displays content like videos, audios, online games and presentations that are made in proprietary formats. Mostly these content come from the internet. The plugins run the contents loaded by browser in sandbox.[43] Today, document viewers like adobe reader have their own sandbox implementation. In adobe all the pdf files run in sandbox, preventing them from leaving pdf viewer and tampering with the rest of your system.[24]

In Cloud computing computations and resources are shared at a low cost but at the same time it possesses many security risks. In a cloud computing scenario, different classes of participants are service instances, service users, and the cloud providers. Attack surface between service instant and user is vulnerable to all kinds of attacks possible in client server architecture such as injection, buffer overflow, or privilege escalation etc. Attack surface between service instance and cloud provider is also vulnerable to Denial-of-Services, resource exhaustion attacks, attacks on the cloud system hypervisor etc. Most of the Cloud Computing leverages virtualization for load balancing. Also, Virtualization provides some security. However, cloud security can be improved by integrating sandboxing mechanism in cloud infrastructure.[45, 23, 58, 19, 26, 27]

4. ARCHITECTURE OF SANDBOX

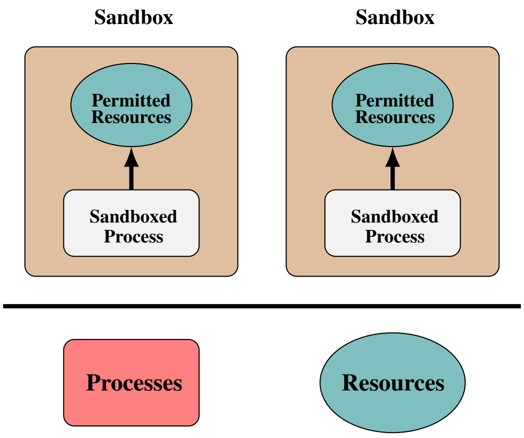

Fig. 1. Isolation based sandbox

Sandbox is an environment where the program can only access restricted set of resources. It is a way to restrict a program's ability to access the resources. It is different from access control applied to all running processes. Typically sandboxes only apply to programs explicitly launched into a sandbox. Programs that are outside the sandbox are free to access any resources.[46]

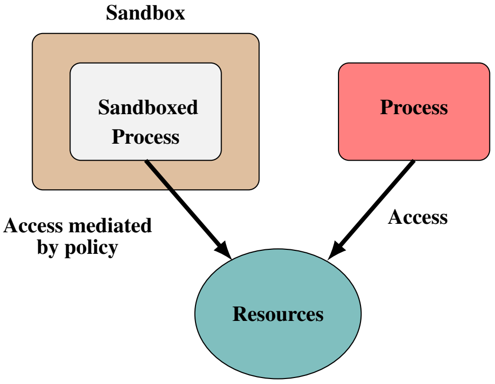

Fig. 2. Rule based sandbox

Most of the sandboxes provide isolation based approach where program running in the sandbox is entirely isolated from the resources and the programs running outside the sandbox. Instead of focusing on completely isolating the applications, rule based approach aims to control what each application is authorized to do. This approach allows applications to share the resources.[46]
The sandboxing is based on prevention instead of detection in the case of security. Usually, sandboxing is used to prevent applications from harming the host system or leaking information (for example, stealing credit card numbers or passwords). Below is a list of some of the sandbox design goals:

(1) Sandbox must restrict access to files outside the sandbox. This can be done by using access control methods or changing the root of the directory.[15]

(2) Sandbox must restrict resources particularly, memory and CPU time to prevent malicious content from blocking the whole system. A Sandbox must avoid exhaustion of memory, which could cause other parts of the system to fail. There is no traditional UNIX mechanism to control memory consumption by the process. Strlimit system call can be used to limit the total available memory space and CPU time.[4]

(3) Sandbox must limit the total number of processes to avoid overloading of the task scheduler.[4]

(4) Sandbox must monitor and control the Inter process Communication.

(5) Sandbox must control communication of the process over the network.

(6) There are many system calls which suspend the program until some event occurs. None of these system calls compromise system security, but they can lock up the grading system for the indefinite amount of time. To avoid that, sandbox limit not only execution time but also the time elapsed on a wall clock (independent clock measuring real time).

(7) Sandbox should control the process's access to system time.

5. EXISTING SANDBOXING MECHANISMS

For the implementation of the sandbox, software venders rely on underlying operating system security features. So each software has different sandbox implementation for the underlying operating system. For example, Chrome has three different sandbox implementation for Linux, Mac and Windows. It is believed that Linux systems are more protected and secure than Microsoft Windows. So Linux implementation of the sandbox is more powerful than Windows. In this paper we discuss sandboxing mechanisms on the Linux platform. Discussion on sandboxing on the other platform's is not the part of this paper.
There are different ways and approaches that can be used to implement sandbox mechanisms. Operating system security features for isolation and access control that can be used in sandboxing mechanisms are summarized as follows Chroot, Seccomp-bpf, System Call Interposition, Resource limit, Linux security modules, Namespaces, Cgroups, Capabilities, Hypervisor based virtualization and Container based virtualization.

5.1 Chroot

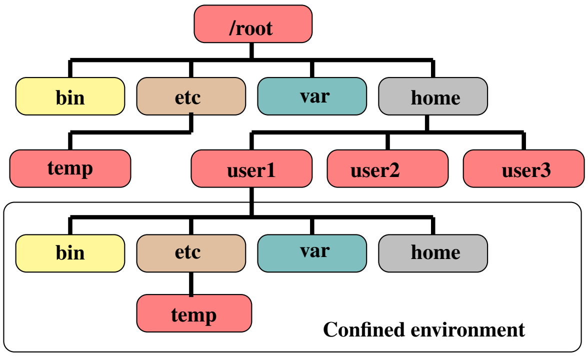

Fig. 3. Chroot

Chroot is a mechanism implemented in Linux system to change the root directory of a current running process to any specified directory. It is used to create confined environment also called as a jail. The process can change root directory to any specified directory by calling chroot system call. The program running in such a modified environment i.e. confined environment sees the given directory as its new root and cannot have access to files outside new root directory e.g. if process chroots to /user1 then /user1 becomes root directory of the current process and its children. If process tries to access /etc/temp then it will access /user1/etc/temp not etc/temp.(see figure 3) This mechanism works only with the file system and does not protect other sensitive resources like network sockets and system devices.
To start chrooted program normally, new root directory must be populated with a required minimum set of resources such as scratch space, device nodes, configuration files, and shared libraries. This makes it difficult to use chroot as a sandboxing mechanism.
During the development of Unix version 7,the chroot system call was introduced and on 18, March 1982 added to BSD by Bill joy. Creating sandbox using chroot system call is not easy and secure. The free BSD jail mechanism proposed by Watson and Kamp is a better option than chroot jail.[15]

5.2 seccomp

seccomp is a security facility built in a Linux kernel. It provides application sandboxing mechanism in the Linux kernel. It was included in the Linux kernel on 8 march 2008 in kernel version 2.6.12. This mode is enabled using system call seccomp(2) or prctl(2). The process running in this mode can make only four system calls which are exit(), sigreturn(), read() and write(). If the process attempts any other system call the kernel terminates the process with SIGKILL. This does not virtualize the system resources but isolates the process from other processes entirely. seccomp-bpf is an extension to seccomp in which system calls are filtered using Berkeley Packet Filter rules. This mechanism is used in Chromium browser which is an open source project of Google, Chrome os, OpenSSH and vsftpd.[11]

5.3 System Call Interposition

Malicious programs usually make system calls to harm the system. Application's interaction with the file system, network and other system sensitive resources need to be monitored and regulated. System call interface is a natural place to security checks and enforces the security policies. System call interposition(SCI) is a powerful mechanism for monitoring and regulating program behavior by intercepting system calls. It helps the programmer to take the control over a process. So SCI technique can be used in sandboxing.

There are different ways to implement SCI services. purelibc can be used to trace the system call generated by itself. ptrace system call can be used to monitor and control the execution of other processes. There are a number of sandboxes based on system call interposition. Janus is one of the sandboxing mechanism which is based on system call interposition. BlueBox uses ptrace to monitor system calls and enforces the specified security rules. [18, 37]

5.4 Cgroups

Cgroup is a Linux kernel security feature that limits and isolates resource's usage like network memory, CPU, disk io, etc. Basically cgroup is a group of processes with specific limits and usage counts assigned to the group. The uses of cgroup are accounting, limitation, prioritization and isolation. Accounting measures how much of a given resource is used, isolation provides different views of the resources available in the system to different processes, limitation asserts that process will not use too much of the given resources and prioritization makes sure that some processes have higher/lower priority in accessing certain resources. Many projects use cgroups as their basis, including Docker, Lxc, Hadoop etc.[5, 33]

5.5 Capabilities

The UNIX-style user privileges come in two varieties, privileged processes and unprivileged processes. Privileged processes are also referred as root users whose effective user ID is zero and unprivileged processes have effective UID as nonzero. Unprivileged processes undergo full permission checking based on the process's credentials (usually: effective UID, effective GID and supplementary group list) while privileged processes bypass all kernel permission checks. The power of unprivileged process is quite limited and the privileged processes are very powerful. If a process needs more power than those of unprivileged processes, the process generally run with the root privilege. Unfortunately, most of the times the processes do not need all the privileges. In other words, they become more powerful than what they needed to be. This can cause serious risk when a process gets compromised. Linux kernel (Since version 2.2) divides superuser privileges into groups called Capabilities These Capabilities allow the unprivileged process to run with restricted root privileges.[2, 6]

5.6 Resource Limit

This feature is introduced in the Linux to limit the process's use of system resources like CPU, memory, number of open file descriptors etc. The setrlimit() and getrlimit() are system calls to set and get resource limits respectively. The prlimit system call can be used for both get and set the resource limit of a process. The prlimit system call extends and combines the functionality of getrlimit and setrlimit. Each resource has associated limits called soft and hard limit, as defined by the rlimit structure. The stucture of rlimit provides a tool to the administrator for the resource limit.

```
struct rlimit
{
    rlim_t    rlim_cur;
    rlim_t    rlim_max;
};
```

The variable rlim_cur defines soft limit and variable rlim_max defines hard limit that the kernel enforces for the corresponding resource. An unprivileged process can change its soft limit to a value less than equal to the hard limit. An unprivileged process can lower its hard limit to a value greater than or equal to its soft limit. The only privileged process can raise the hard limit. [4]

5.7 Namespace

In Linux kernel, namespace provides the ability to isolate groups of processes. Each Linux process is associated with a namespace. The process can only see and access resources which associated to their namespace. Linux supports six types of namespaces:

5.7.1 IPC namespaces:. IPC namespaces isolate certain inter-process communication resources, namely, POSIX message queues and System V IPC objects. IPC namespace allows access to objects created in the namespace to all the processes associated with that namespace but not to processes associated with other namespaces. Thus, IPC namespace provides restrictions on the communication of processes associated with one namespace to the processes associated with other namespaces.

5.7.2 Network namespaces: . Network namespaces isolate system resources associated with networking such as IPV6 and IPV4 protocol stacks, network devices, IP routing tables, port numbers (sockets) and so on. A physical network device belongs to exactly one network namespace. Network namespace isolates network resources associated with one namespace from other namespaces.

5.7.3 Mount namespaces:. It is a set of file system mount points visible to processes associated with that namespace. It isolates the set of file system mount points, in other words processes in different mount namespaces can have different views of the filesystem hierarchy. Creating mount namespace has an effect same as doing Chroot. But Chroot does not give complete isolation and its effects are restricted to root mount points only.

5.7.4 PID namespaces:. Because of the Linux namespaces, it became possible to have multiple nested process trees. Each process tree can have its own isolated set of processes. Process in a child namespace doesn't know the existence of processes in the parent namespace and the sibling namespace. However, process in the parent namespace can see the process in the child namespace. PID namespaces ensure that the process in one PID namespace cannot inspect or kill the process in the other PID namespaces.

5.7.5 User namespaces:. A user namespaces isolate security related attributes and identifiers, such as UIDs and GIDs, the root directory, keys and capabilities. A process's group ID and user ID can be different inside and outside the user namespace. In particular, a process can have non-zero user ID outside name space i.e. unprivileged for operations outside the namespace while at the same time zero user ID inside the namespace i.e. full privileges for operations inside the namespace.

5.7.6 UTS namespaces:. UTS namespaces provide isolation of two system identifiers, NIS domain name and host name. Each UTS namespace has its own UTS related information. In the context of the containers, the UTS namespace allows each container to have its own hostname and NIS domain name. [7, 41]

5.8 Hypervisor Based Virtualization

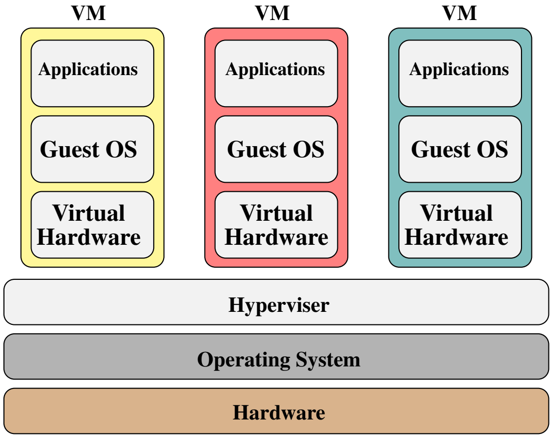

Fig. 4. Hypervisor Based Virtualization

Hypervisor based virtualization is considered to be the most secure and the robust mechanism for sandboxing applications. In this type of virtualization, physical hardware is multiplexed between multiple virtual operating environments called virtual nodes or virtual machines. In hypervisor-based virtualization, each virtual node contains a separate operating system called the guest operating system as shown in figure 4. Hypervisor is a layer between the virtual node and the host operating system. The guest operating system communicates to the host operating system through the hypervisor. Hypervisor such as Xen, vSphere or KVM etc isolates guest operating systems from the host operating system and from other guest operating systems. So malicious application running on a virtual node cannot affect the host system and the applications running on other virtual nodes. [42, 40, 50, 54]

5.9 Container Based Virtualization

Container-based virtualization is a lightweight alternative to the hypervisor-based virtualization. It is also called as operating system level virtualization. Container based virtualization virtualizes the user space resources, allowing a separate instance of the virtual environment called container to be created but shares the same one operating system kernel as shown in figure 5. Unlike hypervisor-based virtualization instead of running entire guest operating system, container-based virtualization isolates guest operating systems but doesn't virtualize the hardware. Even though all the virtual environments are running on top of the same kernel, they have their own memory, file system, processes, devices etc. One or more processes can run in a container. Containers confine the process along with required resources with no access to resources outside the container. Thus, container-based virtualization isolates the processes running in one container from the host system and the processes running in the other containers. [54, 50]
Many container-based virtualization platforms exist, such as lxc[31], Docker[53], OpenVZ [25], VServer [36], Google's container platform lmctfy.[16]

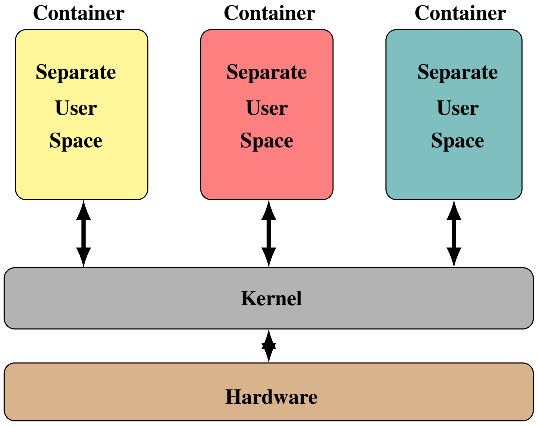

Fig. 5. Container based Virtualization

5.10 Linux Security Module

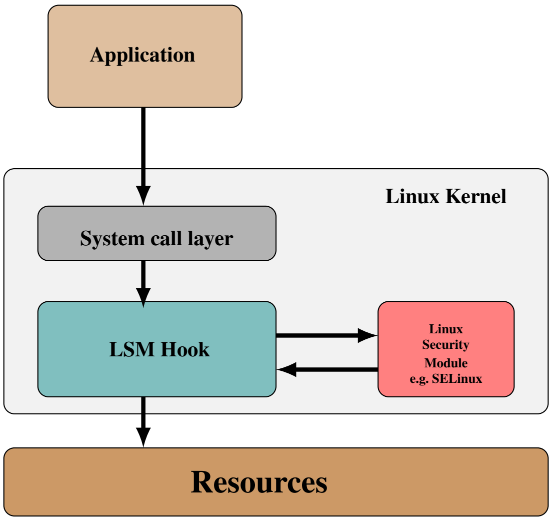

Fig. 6. Linux Security Module

The existing access control mechanisms in the operating systems are inadequate to provide strong system security. In DAC, owner of the resource controls access to the resource by other interested

applications. Whereas in MAC, system controls access to the resources. DAC provide more flexible environment than MAC but DAC has some weaknesses like it can not restrict setuid programs or system daemons that run with root identity.
LSM (Linux security module) is general purpose, lightweight access control framework that enables many different access control modules to be plugged into the Linux kernel as loadable kernel modules. LSM allows modules to mediate access to kernel objects by placing hooks at various points within the kernel(see figure 6). The existing access control modules include SElinux, TOMOYO, AppArmor, Yama, SMACK etc. that are accepted in the official Linux kernel. Most of the sandboxes use these modules to enforce MAC policy on sandboxed process. [38, 56, 12]

5.10.1 SELinux:. In SELinux, security labels are assigned to all objects on a system such as files and processes. LSM framework passes all the security related interactions between objects to SELinux module which checks its security policy and determines whether the operation should deny or allow. SELinux security policy is loaded from the user space. SELinux security policy can be customized to meet a range of different security goals. SELinux was created in Dec 2000 by NSA and available as part of RHEL 4 and all future releases. SELinux is adopted as a standard feature by Fedora-based distributions. Debian Linux kernel has support for SELinux but by default it is disabled. Ubuntu 8.04 and later versions have support for SELinux. OpenSUSE has support for SELinux but by default, AppArmor is enabled. Android operating system uses SELinux as MAC mechanism in its Sandbox.[49, 10, 47, 57]

5.10.2 SMACK:. Relative to SELinux, SMACK provides a simple form of MAC security. Like SELinux, it is also implemented as a label-based scheme. Also, smack security policy is customizable to achieve different security goals. SMACK has been officially adopted since the release of Linux 2.6.25. It is the main access control mechanism in the MeeGo mobile operating system. It is generally used in embedded systems such as Philips Digital TV products and Wind River Linux solutions. It is also used in the Tizen mobile operating sandbox.[10, 44, 17]

5.10.3 AppArmor:. It is a MAC mechanism for confining applications and designed to be simple to manage. Unlike SELinux and SMACK security policy is applied to pathname instead of direct labeling of objects. In learning mode of AppArmor security behavior of the application is observed and automatically converted into a security profile. AppArmor was created by Immunix inc. In 1998, AppArmor was first used in the Immunix operating system. AppArmor is included in openSUSE and SUSE, and default enabled in openSUSE 10.1 and in SUSE Linux Enterprise Server 10. AppArmor was first successfully integrated for Ubuntu in April 2007. AppArmor hardening continued to improve in Ubuntu as it is integrated with profiles for its guest sessions, libvirt virtual machines, the Evince document viewer, and an optional Firefox profile.[47, 12, 30]

5.10.4 TOMOYO:. It is another MAC mechanism. Similar to AppArmor, implements path based security instead of direct labeling of the objects. In learning mode of TOMOYO, similar to AppArmor behavior of the application is observed to generate the security policy. TOMOYO Linux is not for users expecting ready-made policy files supplied by others. It creates the policy from scratch, aided by the learning mode which can automatically generate policy files with necessary and sufficient permissions for a specific system. TOMOYO is proposed for the end users rather than system administrators, although it has not yet seen any appreciable adoption. It provides a more customized layer of security and co-exists with AppArmor. It is merged in the Linux kernel 2.6.30 as ”Tomoyo Linux 2.x”. [57, 48]

5.10.5 Yama:. Yama is not a MAC scheme. In Yama, miscellaneous DAC security enhancements are collected, typically from external projects such as grsecurity. In Yama, enhanced restrictions on ptrace are implemented. Yama module may be stacked with other LSMs in a similar manner to the capabilities module.[8]

5.11 grsecurity

grsecurity patches provide extensive security enhancement to the Linux kernel.
Pax is a main component of grsecurity which flags program memory as non-writeable, data memory as non-executable and randomly organizes the memory. This prevents executable memory from being overwritten with injected malicious code. This prevents many security exploits such as buffer overflow.
RBAC (role based access control system) is an another important component of grsecurity intended to restrict access to the system further than access control normally provided by UNIX access control list. It creates a fully least privileged system where processes and users have minimum privileges. This reduces the ability of the attackers to gain or damage sensitive information on the system.
Another feature of grsecurity includes changing root restriction that prevents attacks such as privilege escalation attacks. [51, 21]

6. CONCLUSION

Traditional operating system security features and antivirus solutions cannot protect operating system and processes from malicious programs. Sandboxing is an important security mechanism which along with traditional operating system security, adds extra defense to the system. It protects the applications from malicious programs in all the systems right from Smartphone, Desktops to the Clouds. For the implementation of the sandboxing mechanism underlying operating system features such as Chroot, Namespace, Seccomp, Cgroups, Capabilities, SCI, Resource Limit, Virtualization and LSM can be used. Beside this grsecurity patch provide security enhancement to the Linux kernel that can be used in the implementation of sandboxing mechanism.
To provide security to the Linux-based operating system steps need to be taken at two levels user level and kernel level. In the case of user level programs, executable files should be created in such a way that user stack and data area (such as heap area) should have no executable permission. Therefore compiler and linker-loader have to play an important role in order to provide above features. Even if malicious code gets executed, its activity can be limited to execute only limited less sensitive system calls by using seccomp mechanism. This disallows execution of system call like setuid(), thereby preventing super user privileges. Furthermore, malicious code can be prevented from accessing the root file system by having namespace mechanism. Linux extends concept of namespace to the other OS layers such as PIDs, users, IPC, networking etc., so a specific process can live in a container with a new group of pids, a new set of users, a completely unshared IPC system (semaphores, shared memory etc.), a dedicated network interface and its own hostname. At the kernel level, different memory areas are to be laid out using proper Address Space Layout Randomization techniques (ASLR). So that malicious program can not identify the various regions of

the process. If above said futures are incorporated in the system, the complete operating system should remain more secure.

7. REFERENCES

[1] Chromium developers guide. `https://www.chromium.org/developers/design-documents/sandbox`.

[2] How linux capability works in 2.6.25. In *SEED Document*.

[3] Rootkits- symantec security response.

[4] Linux programmers manual. `http://man7.org/linux/man-pages/man2/setrlimit.2.html`, 2014.

[5] Linux programmers manual. `http://man7.org/linux/man-pages/man5/systemd.cgroup.5.html`, 2014.

[6] Linux programmers manual. `http://man7.org/linux/man-pages/man7/capabilities.7.html`, 2014.

[7] Linux programmers manual. `http://man7.org/linux/man-pages/man7/namespaces.7.html`, 2014.

[8] Yama lsm. `https://www.kernel.org/doc/Documentation/security/Yama.txt`, 2014.

[9] Jason Ansel, Petr Marchenko, Úlfar Erlingsson, Elijah Taylor, Brad Chen, Derek L Schuff, David Sehr, Cliff L Biffle, and Bennet Yee. Language-independent sandboxing of just-in-time compilation and self-modifying code. *ACM SIGPLAN Notices*, 46(6):355–366, 2011.

[10] Irfan Asrar. Attack surface analysis of the tizen os.

[11] Enrico Bacis, Simone Mutti, and Stefano Paraboschi. Apppolicymodules: Mandatory access control for third-party apps. In *Proceedings of the 10th ACM Symposium on Information, Computer and Communications Security*, pages 309–320. ACM, 2015.

[12] Mick Bauer. Paranoid penguin: an introduction to novell apparmor. *Linux Journal*, 2006(148):13, 2006.

[13] Leyla Bilge and Tudor Dumitras. Before we knew it: an empirical study of zero-day attacks in the real world. In *Proceedings of the 2012 ACM conference on Computer and communications security*, pages 833–844. ACM, 2012.

[14] Douglas R Dechow. A brief history of java and java security.

[15] Wenliang Kevin Du. Security education. 2009.

[16] Rajdeep Dua, A Reddy Raja, and Dharmesh Kakadia. Virtualization vs containerization to support paas. In *Cloud Engineering (IC2E), 2014 IEEE International Conference on*, pages 610–614. IEEE, 2014.

[17] Olga Gadyatskaya, Fabio Massacci, and Yury Zhauniarovich. Emerging mobile platforms: Firefox os and tizen.

[18] Tal Garfinkel, Ben Pfaff, Mendel Rosenblum, et al. Ostia: A delegating architecture for secure system call interposition. In *NDSS*, 2004.

[19] Nils Gruschka and Meiko Jensen. Attack surfaces: A taxonomy for attacks on cloud services. In *2010 IEEE 3rd international conference on cloud computing*, pages 276–279. IEEE, 2010.

[20] Tao Guo, Puhan Zhang, Hongliang Liang, and Shuai Shao. Enforcing multiple security policies for android system. In *2nd International Symposium on Computer, Communication, Control and Automation*. Atlantis Press, 2013.

[21] Olsson Hall. Selinux and grsecurity: A case study comparing linux security kernel enhancements.

[22] Mohammad Shouaib Hashemi et al. Security issues of the sandbox inside java virtual machine (jvm). 2010.

[23] Purui Su Jun Jiang, Meining Nie and Dengguo Feng. Vccbox: Practical confinement of untrusted software in virtual cloud computing. Trusted Computing and Information Assurance Laboratory, Institute of Software, Chinese Academy of Sciences, Beijing.

[24] Jarle Kittilsen. Detecting malicious pdf documents. 2011.

[25] Kirill Kolyshkin. Virtualization in linux. *White paper, OpenVZ*, 3:39, 2006.

[26] Flavio Lombardi and Roberto Di Pietro. Secure virtualization for cloud computing. *Journal of Network and Computer Applications*, 34(4):1113–1122, 2011.

[27] Shengmei Luo, Zhaoji Lin, Xiaohua Chen, Zhuolin Yang, and Jianyong Chen. Virtualization security for cloud computing service. In *Cloud and Service Computing (CSC), 2011 International Conference on*, pages 174–179. IEEE, 2011.

[28] Shinsuke Miwa, Toshiyuki Miyachi, Masashi Eto, Masashi Yoshizumi, and Yoichi Shinoda. Design and implementation of an isolated sandbox with mimetic internet used to analyze malwares. In *DETER*, 2007.

[29] Tiwari Mohini, Srivastava Ashish Kumar, and Gupta Nitesh. Review on android and smartphone security. *Research Journal of Computer and Information Technology Sciences,[online]*, 1(6):12–19, 2013.

[30] Jeroen Ooms. The rapparmor package: Enforcing security policies in r using dynamic sandboxing on linux. *arXiv preprint arXiv:1303.4808*, 2013.

[31] Oracle. Linux containers (lxc),consolidate with oracle linux containers.

[32] Leena Patel and Divya Sharma. Cyber triangle. *International Journal For Technological Research In Engineering*, 1:799–807, 2014.

[33] Martin Prpi Rdiger Landmann Peter Ondrejka, Douglas Silas. Red hat enterprise linux 7 resource management and linux containers guide. Redhat, 2014.

[34] David S Peterson, Matt Bishop, and Raju Pandey. A flexible containment mechanism for executing untrusted code. In *Usenix Security Symposium*, pages 207–225, 2002.

[35] Phu H Phung and Lieven Desmet. A two-tier sandbox architecture for untrusted javascript. In *Proceedings of the Workshop on JavaScript Tools*, pages 1–10. ACM, 2012.

[36] Marc E Fiuczynski Herbert Potzl. Linux-vserver: Resource efficient os-level virtualization. In *Proceedings of the Linux Symposium*, volume 2, pages 151–160, 2007.

[37] Niels Provos. Improving host security with system call policies. In *Usenix Security*, volume 3, page 19, 2003.

[38] Markus Quaritsch and Thomas Winkler. Linux security modules enhancements: Module stacking framework and tcp state transition hooks for state-driven nids. *Secure Information and Communication*, 7, 2004.

[39] Charles Reis and Steven D Gribble. Isolating web programs in modern browser architectures. In *Proceedings of the 4th ACM European conference on Computer systems*, pages 219–232. ACM, 2009.

[40] DON REVELLE. Hypervisors and virtual machines.

[41] Rami Rosen. Linux containers and the future cloud. *Linux J*, 240, 2014.

[42] Farzad Sabahi. Secure virtualization for cloud environment using hypervisor-based technology. *International Journal of Machine Learning and Computing*, 2(1):39, 2012.

[43] Paul Sabanal and Mark Vincent Yason. Digging deep into the flash sandboxes.

[44] Casey Schaufler. Smack in embedded computing. In *Proc. Ottawa Linux Symposium*, 2008.

[45] Steffen Schreiner. *The Impact of Linux Superuser Privileges on System and Data Security within a Cloud Computing Storage Architecture*. 2009.

[46] Z Cliffe Schreuders, Tanya McGill, and Christian Payne. The state of the art of application restrictions and sandboxes: A survey of application-oriented access controls and their shortfalls. *Computers & Security*, 32:219–241, 2013.

[47] Z Cliffe Schreuders, Tanya Jane McGill, and Christian Payne. Towards usable application-oriented access controls: qualitative results from a usability study of selinux, apparmor and fbac-lsm. *International Journal of Information Security and Privacy*, 6(1):57–76, 2012.

[48] Himanshu Shukla, Vivek Singh, Young-Ho Choi, JaeOok Kwon, and Cheul-hee Hahm. Enhance os security by restricting privileges of vulnerable application. In *Consumer Electronics (GCCE), 2013 IEEE 2nd Global Conference on*, pages 207–211. IEEE, 2013.

[49] Stephen Smalley, Chris Vance, and Wayne Salamon. Implementing selinux as a linux security module. *NAI Labs Report*, 1(43):139, 2001.

[50] Stephen Soltesz, Herbert Pötzl, Marc E Fiuczynski, Andy Bavier, and Larry Peterson. Container-based operating system virtualization: a scalable, high-performance alternative to hypervisors. In *ACM SIGOPS Operating Systems Review*, volume 41, pages 275–287. ACM, 2007.

[51] Bradley Spengler. Increasing performance and granularity in role-based access control systems, 2005.

[52] Michael Treaster, Gregory A Koenig, Xin Meng, and William Yurcik. Detection of privilege escalation for linux cluster security. In *6th LCI International Conference on Linux Clusters*, 2005.

[53] James Turnbull. *The Docker Book*. Lulu. com, 2014.

[54] Jeroen van Kessel, Arie Taal, and Paola Grosso. Power efficiency of hypervisor-based virtualization versus container-based virtualization. 2016.

[55] Dave Wichers. Owasp top-10 2013. *OWASP Foundation, February*, 2013.

[56] Chris Wright, Crispin Cowan, James Morris, Stephen Smalley, and Greg Kroah-Hartman. Linux security module framework. In *Ottawa Linux Symposium*, volume 8032, pages 6–16, 2002.

[57] Kenji Yamamoto and Toshihiro Yamauchi. Evaluation of performance of secure os using performance evaluation mechanism of lsm-based lsmpmon. In *Security Technology, Disaster Recovery and Business Continuity*, pages 57–67. Springer, 2010.

[58] Kazi Zunnurhain and Susan V Vrbsky. Security attacks and solutions in clouds. In *Proceedings of the 1st international conference on cloud computing*, pages 145–156. Citeseer, 2010.