International Journal on Recent and Innovation Trends in Computing and Communication ISSN: 2321-8169
Volume: 5 Issue: 7 786 – 791

The Design of Convoluted Kernel Architectural Framework for Trusted Systems – CKA

Edward Danso Ansong[1], James Ben Hayfron-Acquah[2]
[1] Research Scholar, Kwame Nkrumah University of Science & Technology, Department of Computer Science
[2] Kwame Nkrumah University of Science & Technology, Department of Computer Science
[1] *edkan20002002@yahoo.com*, [2] *jbha@yahoo.com*

Abstract: This paper presents the overview of the Convoluted Kernel Architectural framework and a comparative study with the traditional Linux kernel. The architecture is specially designed for trusted sever environment. It has an integrated layer of a customized Unified Threat Management (UTM) and Stealth-Obfuscation OK Authentication algorithm, which is a highly improved and novel zero knowledge authentication algorithm, for secure web gateway to the kernel mode. The framework used is a combined monolithic and microkernel based (hybrid) architecture code-named – the integrated approach, to trade in the benefits of both designs. The architecture serves as the base framework for the *Trust Resilient Enhanced Network Defense* Operating System (*TREND*-OS) currently being experimented in the lab. The aim is to develop an architecture that can protect the kernel against itself and applications.

***Keywords*:** *Convoluted Kernel Architecture, TREND-OS, UTM, Stealth-OK Authentication, MBR*

*****

I. Introduction

An operating system (OS) kernel is the core of its architecture upon which all other modules orprogramming files(within the OS)are integrated. The kernel defines the architecture of the operating system and the hardware it supports. Over the past six decades, universities, research institutions, corporations and operating systemengineers have all contributed to the development and expansion of Kernels. Since the late 1990's,there has been a paradigm shift in OS development from distributed environment to OS Security. This paper introduces a novel Kernel architecture, dubbed the –Convoluted Kernel – which is designed with the goal of contributing to the on-going research on operating system kernel security to protect the kernel against itself from vulnerabilities such as un-authorized kernel modification and privilege escalation. The scope of this research is tailored to mechanisms in developing a novel security architectural framework that could easily retrofit into a monolithic kernel tofurther augmentthe already existing frameworks that falls under the Linux Security Module (LSM) to protect the kernel against itself and other applications.

The traditional OS architecture is generally made up of four major subsystems that work together to form a whole complete system which can be further classified into Kernel space and User space. The fundamental OS Architecture is made up of the hardware Controllers, which encompasses all the conceivable physical devicesin the OS installation such as the CPU, memory module, network devices, Hard Drives among others. The next upper layer is the OS Kernel which serves as the integral part of the entire OS. In this layer, the kernel abstracts and mediate access to the hardware resources as captured in the previous layer which completes the kernel space[1].

The proceeding layers forms the user space section of the model. It is however made up of an interface level between the kernel space and the user space called the OS Services layer. This layer of the model essentially has two key sides. The lower part interfacing the kernel, which has compiler tools, libraries etc.,and are considered part of the Kernel while the upper part interfacing the application, is considered part of the OSlike the command shells etc. The top most layer of the architecture is referred to us the User Application which consists of set of applications executed by clients and servers[2]. Users are more familiar with this layer since their day to day interactions with the OS is interfaced with the various applications they install. The decomposition of an OS into four main subsystem architecture is as shown below in figure 1.

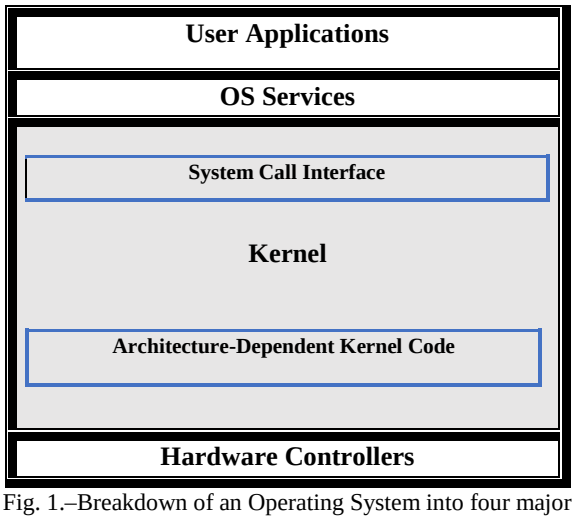

Fig. 1.–Breakdown of an Operating System into four major Subsystem.

II. Evolution of OS Security Framework

Traditional UNIX (the mother of Linux)architecture was created without adequate emphasis to security[3]. This was not however considered as a weakness to the architecture at the time since data protection and operating system level vulnerability did not exist at the time as a threat. In the early 90's, the first worm attack across the globeexposed the vulnerabilities of operating system and the debate of data protection. The only form of protection at the time was the Discretionary Access Control (DAC) which is user defined to protect user files and directories and are subject to the discretion of the user. The weaknesses of DAC became so highly evident when during the over reliance on networking and therefore led to the development of other frameworks. The MAC framework where the mandatory access control is managed was introduced to protect the kernel. Subsequently, other framework such as the, SESBD, Flask and SELinux frameworks were introduced to further enhance the architecture of the monolithic kernel. In the next section, we will delve into the various framework and their weakness to attacks.

III. Security Enhanced Framework

There has been several security framework that over the years have been developed to secure the Linux kernel and to improve the security of the architecture in general. Several kernel and operating system developers are beginning to adopt the use of virtualization to protect the core kernel structure from unauthorized manipulation from illegitimate users to expose its vulnerabilities[4].

OS-level integrated level virtualization technique was therefore implemented into the framework in order to enhance the security of the core kernel to improve the resilience of the architecture[5].

The diagram presented in figure 3 shows an expansion of the virtualization component of TrendOS System architecture which is the prototype of the convoluted architecture. Starting from the bottom, we installed Linux Containers (LXC) as the underlying technology behind the virtualization component. LXC (which is an abbreviated way of saying LinuX Containers) is an operating system-level virtualization method for running multiple isolated Linux systems which are called containers on a single control host. This creates a high performance environment for the VMS (sometimes referred to as containers).

Above this layer exists the virtual machines or containers which essentially achieves virtualization at the OS Level. This is achieved through Linux cgroups and is beyond the scope of our discussion[6].

As established earlier, the virtual machines share the host's kernel facilities. Above that is Bridge networking rules configured directly into every VM. This allows packets flowing from and to these containers to be analyzed by our UTM layer to protect application services from possible attack. Above this layer lies system libraries Application services make constant repetitive use during their execution. On top of this lies the actual application services that clients make use of. This is the ultimate goal of a secure server environment - To protect its application services [7].

IV. Traditional OS Kernel Design

OS kernel architecture is changing and expanding very fast to meet the ever changing complexities of computer hardware designs and ever improving sophistication of software applications. The multicore hardware designs of processes has made it possible for a complex programs which hitherto could only run on huge, high-end and expensive servers, to currently run on low-end personal computers. These developmenthas also been engineered by the numerous research by universities, corporations and computer engineers, to meet the growing need for secured yet fast kernel designs with minimal vulnerabilities.

However, core Linux architecture is monolithic by design and thereby,lack a resilient self-protection scheme when the security of the kernel space is breached[8].Due to this characteristic feature, a single bit exploit in the kernel could lead to a fissureof the entire kernel mode of the OS including the MBR, memory module and other resourcedependencies.Furthermore, vulnerabilities in most Linux based monolithic kernels have also made it predisposed to an array of kernel malware which exploits an internal kernel breach without any defense mechanism against internal attacks.

Even though copious developmentshave been made over the past decades to mitigate the drawback associated with theinitial architectural design - the monolithic kernels –the evolvement of a principal alternative architecture to the traditional design became imperative. The Microkernel therefore became a perfect substitute to the monolithic. The principal difference between the monolithic and the microkernel is that, in the former, every part of the kernel is executed in the unwieldly bottom-large kernel space which incidentally, happens to also run in the same address space. The key drawback therefore is a single process failure in any part of the kernel could have a grave consequence on the entire address space which frequently lead to a kernel panic.

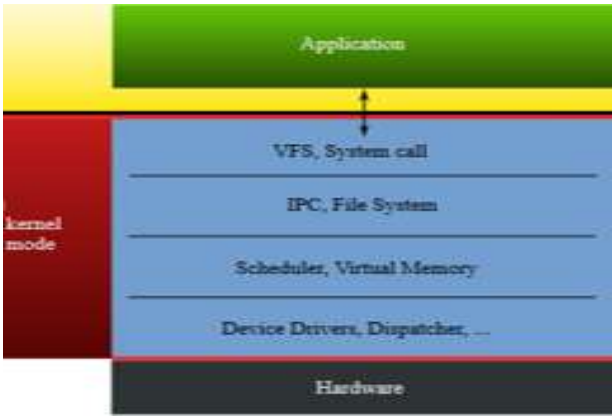

Fig 2. Monolithic Design

In the microkernel however, unlike the monolith's huge kernel space, part of it is moved from the risky kernel space into a convenient and safer user space which is not susceptible to frequent crash of the kernel. This approach is considered less dangerous for the reason that, in the user space, each process runs in an isolated mode (aka servers) and therefore any bug in this design will obviously have a far less consequence since the processes involved may crash but the rest of the kernel will be operating in a safe mode. Even though the microkernel appeared to have resolved the key challenges of the monolithic kernel, it also brought other limitations which are alien to the monolithic. Those weaknesses are code complexities which also leads to performance overheads[9]. Unlike the monolithic kernel's huge disproportionate kernel space with respect to the user space, the microkernel has the reverse forming its design. With this new design in the microkernel, it is therefore expedient to have effective communication of the various services which hitherto was located in the kernel space now situated in the user space. This service in the microkernel is referred to as the inter-process communication.

V. Linux Kernel Security Framework

LSM Framework creates an Application Programming Interface (API) to permit the Linux kernels to provide support to the several kernel security models that has implemented the framework's standard. The motivation behind its creation is to allow uninhibited selection of kernel security model of choice while avoiding a fixed hard-wired module.

The LSM project was developed to effectively implement Mandatory Access Control (MAC) without altering the base kernel yet augmenting the traditional Unix Discretionary Access Control (DAC) service already provided by the Linux kernel. The rationale behind the development of an additional security mechanism (MAC) to enhance the kernel security is because, the extent to which DAC is implemented relies on user discretion on access constraints. While DAC provides restrictions for file system access, the need for security mechanism to enforce defenseagainst threats of secured objects in systems such as Network Sockets, IPC among others which cannot be circumvented by users became inevitable.

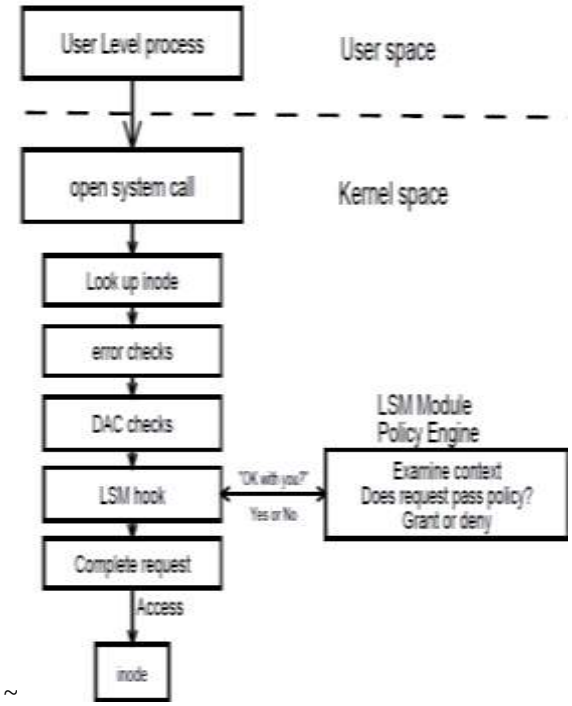

Fig 3. Linux security module architecture

A more intelligent implementation of the Linux Security Module architecture is the preventive access control mechanism. The architecture of the LSM is as shown in figure 3. It uses a technique of hooks by adding a security field to the Linux structure. In so doing it loads the credentials of the program in order to know the module to load and whether it meets the policy requirements [10].

VI. Drawback of Linux Security Module

With the adoption of LSM as a standard API for loadable access control modules for the kernel to enhance the security of the architecture, there were however some challenges associated with the implementation of the framework. While some engineers were considering the use of an integrated kernel structure, the founder of Linux and top maintenance group rejected such idea. Such kernel implementation were considered to be inflexible and uncompromising and as a result, some earlier development modules which could not adopt to the framework such as GRsecurity and RSBAC were obliterated from the list of standardized loadable modules because of their un-support for LSM API [11].

This therefore denies an otherwise potentially best approach to enhancing security. With this reason, it also denies majority of users the opportunity to experiment and select from their own best kernel security module. Reason for the deprecation of others could be attributed to inactivity and excessive inflexibility in allowing the modules to easily port and other compatibility issues [12].

 ISSN: 2321-8169
786 – 791

Even though modules such as SELinux, AppArmor, SMACK, TOMOYO and YAMA are among others are highly rated, the first two appears in some distributions as the default kernel security module precompiled, they also have their own vulnerabilities that keeps operating system developers a bit apprehensive on the best option to choose from as far as kernel security modules were concerned. The key challenge arising out of the use of Linux Security Modules is the capability to disable and enable the module as and when it becomes required [13].

VII. The Convoluted Kernel Architecture

Due to various concerns raised on the adoption of a single framework (Linux Security Module) to interface kernel security modules, the need for the development of an integrated kernel framework to provide enhanced security and resilience became indispensable. Even though some efforts have been made by operating system engineers and scientists such as capsicum and Secure Virtual Architecture (SVA) which are the two widely pronounced kernel architectural framework, however, their emphasis do not involve amalgamation of other useful features of kernels such as High Availability and cryptography using zero knowledge [14].

In the absence of a kernel which could harness these functionalities to support server environment, the idea of an integrated kernel architecture with sandbox-virtualization and its prime focus on High Availability and secured authentication scheme was conceived. This kernel with these framework was referred to as the Convoluted Kernel architecture.

Beginning from the bottom "Kernel mode" division. The Linux Kernel Layer contains the various subsystems that make up the kernel namely, the IO Manager, the Device Drivers, the Process manager, Virtual Memory Manager and more.

Above this layer is the TrendOS Linux Security Module (TLSM) which aims at enabling the efficient and concurrent use of multiple LSMs in a well-coordinated manner. This module is included as a built-in module in the TrendOS Linux Kernel. TLSM enables capabilities that allow the coexistence of multiple LSMs (e.g. SELinux, AppArmor etc.) which greatly enhances system wide security [15].

Above this layer is the system call interface where standard system call function (e.g. exec) are evaluated and handled. This layer denotes the beginning of the Kernel Mode Division. Moving upwards, the User mode Division also known as Secure Trust Level 1 (STL1). This layer begins with System binaries the libraries that eventually make contact with the System call interface (Bridge).

Above this Layer lies the High Availability Monitoring Unit which lies as a backbone to the Sandbox Environment. This layer is responsible for ensuring the all sandboxes are available 99.9% of the time. The services and operation of the High Availability technology ensures that continuous service is available between the private and secondary services at all times.

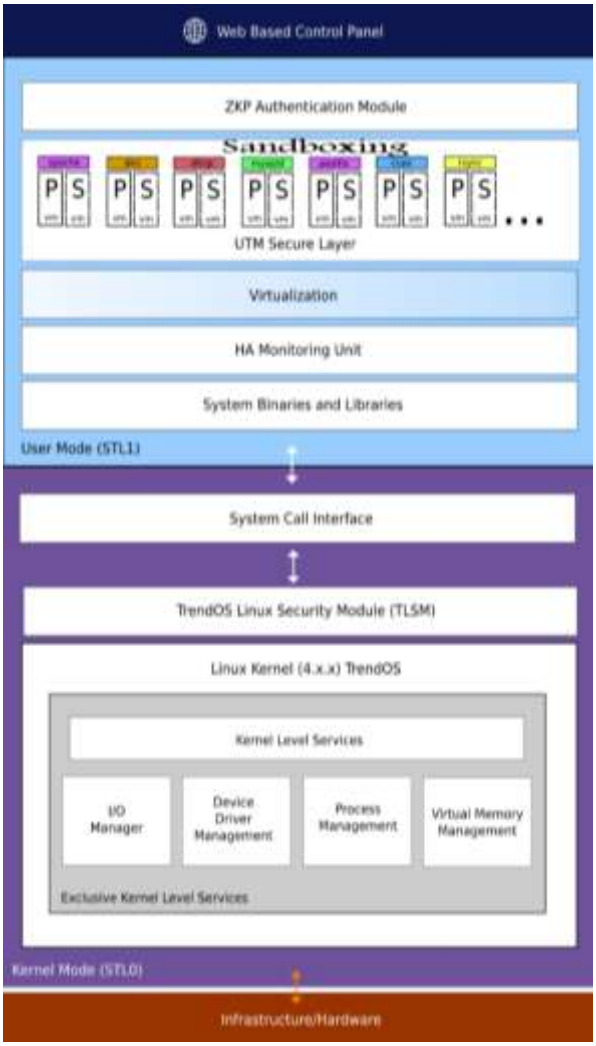

Fig 4. Convoluted Kernel Architectural Design

Techniques including the Heart Beat mechanism are utilized efficiently to ensure downtime and recovery time is greatly reduced (Shahapure, 2015).Next above the HA Monitoring Unit is the Sandbox Environment where Application services run in a safe isolated environment that is actively protected by UTM enabled system-default sandboxes.

 ISSN: 2321-8169
786 – 791

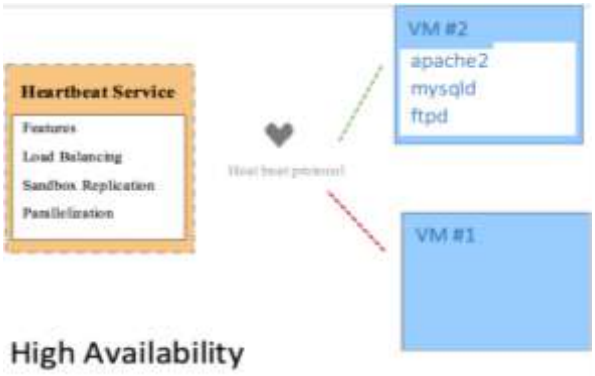

Fig. 5 Heartbeat architecture of the of the Convoluted Architecture.

This Layer allows the creation of as many application sandboxes as desired that will run in a Secure Sandbox environment that is protected by system-default UTM enabled sandboxes using state-of-the-art techniques such as Content Inspection as shown in Figure 6.

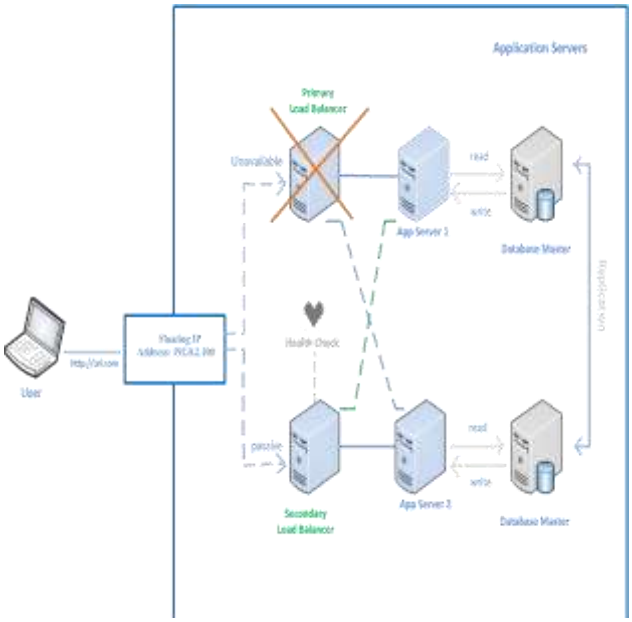

Fig. 6. Simplified High Availability (HA)

The Depiction shows the Architectural breakdown of the Secure UTM Layer of the Main Architecture. In the design, several Unified Threat Management (UTM) modules that make up the UTM stack. This stack consists of applications like snort and ipfw.

What happens is that when network traffic arrives from the internet to or from the virtual machines. The UTM layer filters all traffic to ensure all network traffic is safe and provides some level of protection to the sandboxes. It does so through a chaining mechanism that allows packets to be filtered thoroughly before they are finally routed to their final destinations.

The UTM stack has been built in such a way that, it is able to co-locate with third party security tools without any conflict. This UTM stack works transparently with the virtual machines and sandbox technique.

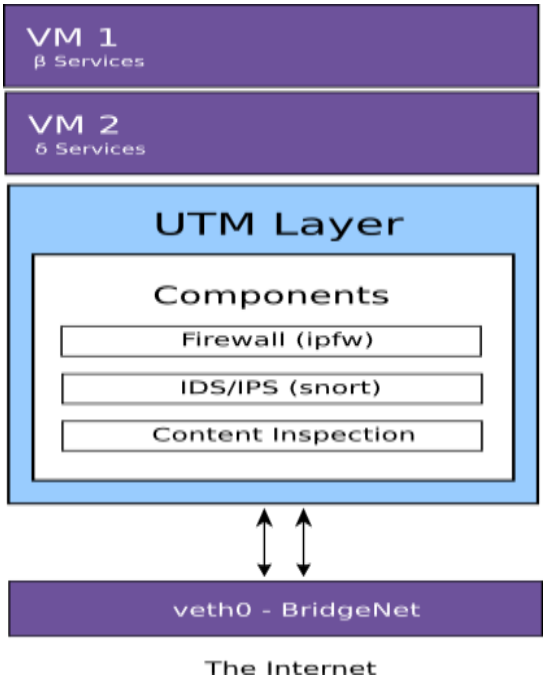

Fig. 3.6 Expanded Unified Threat Management

Above these levels is the ZKP Security module. This layer is essentially a Pluggable Authentication Module (PAM) responsible for System-wide authentication using the Zero-Knowledge Authentication technique. Using such a transparent framework, the all great benefits of using the ZKP technique can be realized seamlessly through PAM-aware system-based application like "ssh" (Soares, 2013).

The Last Layer that ends the STL1 is the Web Based Control Panel. This layer presents a Web GUI that provides a general overview of system performance and enables system administrators to regulate TrendOS system functionality in as simple a manner as just turning knobs and setting values. This Control

panel is the official TrendOS dashboard that Trend System Administrators will be familiar with.

VIII. Conclusion

In conclusion, we presented *J*discussed the overview of the Convoluted Kernel Architectural framework and a comparative study with the traditional Linux kernel. This architecture is specially designed for trusted sever environment. It has an integrated layer of a customized Unified Threat Management (UTM) and Stealth-Obfuscation OK Authentication algorithm, which is a highly improved and novel zero knowledge authentication algorithm, for secure web gateway to the kernel mode. The framework used is a combined monolithic and microkernel

 ISSN: 2321-8169
786 – 791

based (hybrid) architecture code-named – the integrated approach, to trade in the benefits of both designs. The architecture serves as the base framework for the *Trust Resilient Enhanced Network Defense* Operating System (*TREND*-OS) currently experimented in the lab. The aim is to develop an architecture that can protect the kernel against itself and applications

References

[1] B. Spengler, "SSTIC 2016 Keynote," Grsecurity, Rennes, France, 2016.

[2] I. T. Bowman, Hybrid Shipping Architectures, Waterloo: University of Waterloo, 2001.

[3] C. Alm, Analysis of Manipulation Methods in Operating System Kernels and Concepts of Countermeasures , Considering FreeBSD 6 . 0 as an Example, Hamburg: University of Hamburg, 2006.

[4] D. C. B. D. W. K. S. W. M. S. J. M. P. G. N. J. W. Robert N.M. Watson, Capability Hardware Enhanced RISC Instructions: CHERI User's guide, Cambridge: University of Cambridge, 2014.

[5] T.-c. C. X. W. Zhiyong Shan, "Virtualizing System and Ordinary Services in Windows-based OS-Level Virtual Machines," in 2011 ACM Symposium on Applied Computing, Cornel Univerisity Library, 2011.

[6] Intel Security, "Security Best Practices. Moving to Office 365," Osterman Research, Inc., Black Diamond, Washington , 2015.

[7] Information Assurance Technology Analysis Center (IATAC), Information Assurance Tools Report: Vulnerability analysis, Information Assurance Technology Analysis Center, 2009, 2009.

[8] T. K. W. D. J. C. V. A. Nathan Dautenhahn, "Nested Kernel: An Operating System Architecture for Intra-Kernel Privilege Separation," in ACM, Turkey, 2015.

[9] M. v. Q. F. B. M. H. S. K. Nikolaus Huber, "A Method for Experimental Analysis and Modeling of Virtualization Performance Overhead," Karlsruhe Institute of Technology (KIT), Karlsruhe, Germanay, 2011.

[10] C. C. ,. J. M. ,. S. S. ,. G. K.-H. C. Wright, "Linux security modules: general security support for the linux kernel," in Linux security modules: general security support for the linux kernel, Los Alamitos, CA,, 2003.

[11] C. C. J. M. S. S. G. K.-H. Chris Wright, "Linux Security Module Framework," in Ottawa Linux Symposium, Ottawa, 2002.

[12] E. Karlsson, "Evaluation of linux security frameworks," Umeå University, Umeå, 2010.

[13] P. M. &. B. M. S. S. Backes, "Android Kernel Extension Login on Lab Machines," SAARLAND UNIVERSITY, 2013.

[14] W. Mauerer, "Linux Kernel Architecture Auding," Linux, 2008.

[15] M. Gorman, Understanding the Linux Virtual Memory Manager, Dublin: Bruce Peren's Open Source Series, 2004.

[16] A. S. a. M. Salama, "Cloud Computing: Paradigms and Technologies".