

Architectural requirements for a next generation Trusted Operating System (TrOS)

Arun Viswanathan

Computer Science Department, University of Southern California

Abstract— With the growth of the internet and its usage in business and e-commerce, security and privacy have become the two major requirements for any application in the networked world. Also, as applications grow in complexity, so do their security requirements and today's common computer user is overburdened with too many controls to fiddle with to ensure security and privacy of his data. Current computing platforms cannot sustain the security and privacy requirements of the next generation internet and its applications and also cannot provide an easy way for users to manage their security. The Trusted Computing Group (TCG) was created to develop an open specification for an improved computing platform taking into account the security and privacy requirements of the next generation internet. Trusted Computing is a fundamentally new approach to address the security and privacy concerns of the future and is based around the concept of building and proving "trust" relationships. A lot of research has been done around building systems using the TCG primitives like sealed storage and remote attestation. There is also a lot of research and work done on using Virtual Machine Monitors to provide features like isolation and containment. This paper surveys the research done in both areas of Trusted Computing and Virtual Machines and presents a list of requirements for a next generation trusted operating system. Additionally, the paper tries to present a concept of "Plug and Play security" which implies security with minimal fuss and configuration. Broadly, the paper tries to nail down the specifications for a next generation Trusted Operating System with the following objectives: a) Provide high assurance b) Provide resilience to malicious code c) Provide isolation and containment for different applications d) Provide seamless and "plug and play security" to the common user.

Index Terms—Trusted Computing, Security, Virtual Machines, Operating Systems

I. INTRODUCTION

The growth of the internet and its applications has added an extra premium to security and privacy requirements. It is becoming increasingly evident from everyday reports on worm/virus outbreaks that the current computing infrastructure does not have the potential to carry the burden of next generation applications and their security requirements. As

Garfinkel et.al. state in [1]; "commodity computing systems have reached an impasse". A few of the reasons for this are poor assurance, poor isolation, high complexity architectures and complex configurations. A complete list of the flaws is discussed in Section III. As security and privacy will play a pivotal role in the future of internet and e-business, there is a very urgent need to investigate on secure architectures for computing.

As discussed in [1], to address the above problems, closed systems like cellular phones, game consoles and ATM's resort to specialized platforms with their own closed software stack and hardware. This makes it easy to be tailored for security requirements. These platforms use techniques like hardware tamper resistance and embedded cryptographic keys to ensure that the platform's software stack is not easily modified to make it misbehave. The keys permit these systems to identify their own software to remote systems, allowing them to make assumptions about the software's behavior. These capabilities allow closed platforms to offer higher assurance and address a wider range of threat models than current general-purpose platforms. The security benefits of starting from scratch on a "closed box" special-purpose platform can be significant. However, for most applications these benefits do not outweigh the advantages of general purpose open platforms that run many applications including a huge body of existing code and that take advantage of commodity hardware (CPU, storage, peripherals, etc.) that offers rich functionality and significant economies of scale.

Another issue which is very poorly researched but one which is of paramount importance is that of easy-to-use security or which I call as "Plug and Play Security". Current security systems suffer from the problem that the common layman user is expected to do just too much configuration to make his system secure. This has the unfortunate effect of users turning off their security controls to work around the inconvenience. For example, common users who are only interested in searching for information on the web get absolutely irritated when they are asked to configure the web-browser settings for ActiveX downloads and Signed/Unsigned installations. Common users do not want to spend their time configuring systems and would like it to be used out of the box as much as possible. Thus, what is really required is a system which is theoretically as close as possible to a "Toaster". In a clearer sense, we need to build secure computing systems which can

be treated as an appliance by users and which can be used out of the box with minimal fuss and configuration.

Two technologies which have the potential of providing a solution for the above problems are Trusted Computing and Virtualization. Trusted Computing is a security solution developed by the Trusted Computing Group (TCG) to mitigate the risks of participating in an interconnected world while also ensuring interoperability and protecting privacy [3]. Trusted Computing is an open specification providing a framework with a set of primitives like sealed storage for keys, tamper resistance and remote attestation for devices to interoperate in a secure and trusted manner. Virtualization is a technique which presents an illusion of many smaller Virtual Machines (VMs), each running a potentially separate operating system instance [4]. Virtualization helps in realization of features like isolation and in combination with Trusted Computing has the potential of realizing a Trusted Operating System. This paper looks at laying down specifications to build a Trusted Operating System called *TrOS* by utilizing the above two technologies.

Specifically, the paper will try to achieve the following high level objectives in defining the requirements for a trusted operating system:

- 1) High assurance to the external world about the state of the system.
- 2) Resilience to malicious code.
- 3) Isolation and containment for different applications.
- 4) Seamless “plug and play security”.

The remainder of this paper is organized as follows. Section 2 discusses trusted computing and virtualization technologies and related work in both the fields. Section 3 presents a list of flaws in mainstream operating systems and builds up a base for the architectural requirements of a next generation Trusted Computing OS that follows in Section 4. A high level block architecture of a system which meets some of the requirements is presented in Section 5. Section 6 provides directions for further research and section 7 summarizes the achievements of this paper and concludes.

II. BACKGROUND AND RELATED WORK

This section explains in some details the concepts of trusted computing and virtualization and surveys the research done in both the fields. Section C below takes a look at two architectures which combine trusted computing and virtualization to build a secure architecture.

A. Trusted Computing

Trusted Computing (commonly abbreviated TC) is a technology developed and promoted by the Trusted Computing Group (TCG). In this technical sense, "trusted" does not necessarily mean the same as "trustworthy" from a user's perspective. Rather, "trusted computing" means that the

computer can be trusted by its designers and other software writers not to run unauthorized programs [5].

Trusted computing encompasses five key technology concepts, of which all are required for a fully trusted system [3] [5].

- 1) *Endorsement Key* - The endorsement key is a 2,048-bit RSA public and private key pair, which is created randomly on the chip at manufacture time and cannot be changed. The private key never leaves the chip, while the public key is used for attestation and for encryption of sensitive data sent to the chip.
- 2) *Secure Input and Output*- Secure input and output refers to a protected path between the computer user and the software with which they believe they are interacting. Secure I/O reflects a hardware and software protected and verified channel, using checksums to verify that the software used to do the I/O has not been tampered with. Malicious software injecting itself in this path could be identified.
- 3) *Memory curtaining / Protected execution* – Memory curtaining extends the current memory protection techniques to provide full isolation of sensitive areas of memory — for example locations containing cryptographic keys. Even the operating system doesn't have full access to curtained memory, so the information would be secure from an intruder who took control of the OS.
- 4) *Sealed storage*- Sealed storage protects private information by allowing it to be encrypted using a key derived from the software and hardware being used. This means the data can be read only by the same combination of software and hardware.
- 5) *Remote attestation*- Remote attestation allows changes to the user's computer to be detected by authorized parties. It works by having the hardware generate a certificate stating what software is currently running. The computer can then present this certificate to a remote party to show that its software hasn't been tampered with.

A few examples of systems built using trusted computing primitives are presented below. Research has been done on using the remote attestation feature of trusted computing to provide an attestation based policy enforcement for remote access to clients [6]. In this system, remote attestation is used to verify the client integrity properties and establish trust upon the client before allowing the client to remotely access the corporate network. In another instance, trusted computing principles of remote attestation and sealed storage have been applied to provide privacy protection to RFID enabled devices [7]. Remote attestation is used by readers to prove to concerned individuals that they are running a specific version of the reader software and that the RFID reader is not a compromised one.

B. Virtualization

As defined in [8], the term *virtualization* broadly describes the separation of a resource or request for a service from the underlying physical delivery of that service. The *virtual infrastructure* provides a layer of abstraction between computing, storage and networking hardware, and the applications running on it. The deployment of virtual infrastructure is non-disruptive, since the user experiences are largely unchanged.

As classified in [8], Virtualization can apply to a range of system layers, including hardware-level virtualization, operating system level virtualization, and high-level language virtual machines. Hardware-level virtualization was pioneered on IBM mainframes in the 1970s, and then more recently Unix/RISC system vendors began with hardware-based partitioning capabilities before moving on to software-based partitioning. For Unix/RISC and industry-standard x86 systems, the two approaches typically used with software-based partitioning are

a) *Hosted architecture* - A *hosted* approach provides partitioning services on top of a standard operating system and supports the broadest range of hardware configurations

b) *Hypervisor architecture* - In contrast, hypervisor architecture is the first layer of software installed on a clean system (hence it is often referred to as a “bare metal” approach). Since it has direct access to the hardware resources, a hypervisor is more efficient than hosted architectures, enabling greater scalability, robustness and performance.

As can be seen from the above virtualization is a very powerful tool to provide security because it can provide a clean separation of domains for applications in different virtual machines. This is a very important property for achieving isolation between applications. Using virtual machines one can create multiple independent systems running on the same machine and which would not be affected by crashes or malicious behavior of other virtual machines.

XEN[4] is a hypervisor based architecture and it uses a form of virtualization called *paravirtualization* [4] to provide a virtual machine monitor which allows multiple commodity operating systems to share conventional hardware in a safe and resource managed fashion, but without sacrificing either performance or functionality. VMware [8] primarily provides a hosted architecture for virtualization using which one could run multiple operating systems on a single host OS. XEN is expected to be more scalable than VMware in terms of number of virtual machines that can be supported.

C. Hybrid systems built using Trusted Computing and Virtualization primitives

Of late, researchers have focused on building secure architectures using a combination of trusted computing and virtualization primitives. Two good examples of such an

approach are Microsoft’s Next Generation Secure Computing Base architecture (NGSCB) [9] and the Terra Trusted VM from Stanford [1], [2]. Section III on Trusted OS requirements will borrow a lot from these two architectures in defining the requirements for a Trusted Operating System.

As described in [10], [11] ‘Next Generation Secure Computing Base’ (NGSCB) is the name for Microsoft Inc.’s trusted computing program, formerly known as Palladium. NGSCB builds on the TCG specification and a number of significant planned modifications to the CPU and chipset. The chipset modifications are apparently designed to introduce a trusted path for keyboard, mouse and other input, and secure display output. The CPU modifications introduce a new mode akin to a ‘ring -1’ and a number of new instructions to support it. The NGSCB is equivalent of a VMM as an *isolation kernel*. The isolation kernel protects the memory and device resources of a domain from access by other domains. The isolation kernel is not itself a fully featured operating system, it merely manages the coexistence of multiple operating systems on the same machine. It appears to embody the important features that are required of a trustworthy computing platform. But there have not been any details about this architecture from Microsoft of late.

As described in [1], Terra provides a simple and flexible programming model that allows application designers to build secure applications in the same way they would on a dedicated closed platform. At the same time, Terra supports today’s operating systems and applications. Terra realizes this union with a *trusted virtual machine monitor* (TVMM), that is, a high-assurance virtual machine monitor that partitions a single tamper-resistant, general-purpose platform into multiple isolated virtual machines. Using a TVMM, existing applications and operating systems can each run in a standard virtual machine (“open-box VM”) that provides the semantics of today’s open platforms. Applications can also run in their own closed-box virtual machines (“closed-box VMs”) that provide the functionality of running on a dedicated closed platform. The TVMM protects the privacy and integrity of a closed-box VM’s contents. Applications running inside a closed-box VM can tailor their software stacks to their security requirements. Finally, the TVMM allows applications to cryptographically authenticate the running software stack to remote parties.

As can be seen from the above descriptions, both the architectures satisfy a lot of the requirements for a trusted OS especially in terms of partial assurance and isolation. But there are still certain requirements like providing total assurance and plug-and-play security that is missing from these architectures. Section IV will list down all requirements for a Trusted Operating System keeping the four high level objectives in mind.

III. FLAWS IN MAINSTREAM OPERATING SYSTEMS

According to [12], “current security efforts suffer from a flawed assumption that adequate security can be provided in applications with the existing security mechanisms of mainstream operating systems”. They identify two important operating system mechanisms, namely trusted path and mandatory access control, that are necessary for effective application level security policy enforcement and since neither property is implemented in current mainstream operating systems they argue that the security of applications based thereon can be nothing more than “a fortress built on sand”. The work presented in [12] is thus an indication of the poor state of mainstream operating system for supporting the next generation security and privacy requirements. This section surveys all such limitations or flaws. These flaws in turn become the motivation for the requirements of a Trusted Operating System presented later in section 4.

- 1) *As stated in [1], mainstream operating systems are complex programs that often contain millions of lines of code, thus they inherently offer low assurance.*
- 2) *As stated in [1], mainstream operating systems poorly isolate applications from one another and thus the compromise of almost any application on a platform often compromises the entire platform.*
- 3) *As stated in [1], current platforms provide only weak mechanisms for applications to authenticate themselves to their peers and thus building robust and secure distributed systems is difficult.*
- 4) *As stated in [1], current platforms provide no way to establish a trusted path between users and applications. Conversely, human users have no way of establishing whether they are interacting with a trusted financial application or with a malicious program impersonating that application.*
- 5) *As stated in [11], a key limitation of mainstream operating systems that makes them inappropriate for applications like DRM is their enforcement of a identity based discretionary security policy rather than a mandatory security policy. The discretionary model does not give assurance to the content provider that the provided content will not be tampered with.*
- 6) *As stated in [11], mainstream operating systems do not implement the principle of least privilege. In today’s operating systems, the privileges are bound to the user IDs and access decisions are based on user identity. As a consequence, all of the users privileges are granted to each program running on behalf of the user. There is no way to give fine grained access control.*
- 7) *As stated in [11], another problem in today’s systems is that the device drivers present must be totally trusted but they are not necessarily trustworthy. Device drivers have complete access to system memory space and they can bypass most of security*

measures. Further more, device drivers can be loaded at run time thus allowing malicious/buggy drivers to be loaded. This problem is due to the incorrect usage of the ring structure in the x86 processor.

- 8) *A flaw which has till now been overlooked but which I feel is the root cause of a lot of security issues is that of a lack of “Plug and Play Security”. Today’s security systems are so complex that the common user tends to not use them to save his/her time. This results in either a shut down of the security system or a misconfiguration, both of which defeat the whole purpose of providing security. It is important the next generation security architectures take into account the ease of setup and use of security and provide a more “Toaster” like architecture for using computers securely.*

IV. TRUSTED OPERATING SYSTEM (TROS) REQUIREMENTS

It is assumed that the TrOS will have some form of trusted hardware as specified in [3] to run upon. The following list of requirements is derived from [1], [4], [10] and [12] plus additional requirements proposed by me. All the requirements are geared towards meeting the high level objectives set out in the introduction section. The requirements for the trusted operating system can be split into the following categories.

- A. Functional requirements
- B. Security requirements
- C. Ease of use / Plug and play requirements
- D. General requirements

It is important to note here that the requirements below refer to a concept called “domains”. Domains can be thought of as containers which have strong isolation boundaries. They can contain either be single applications or they can contain operating systems running various applications.

A. Functional requirements

- 1) **Domain Support:** The operating system must allow for creation/deletion and management of entities called domains. These domains could contain single applications or contain complete operating systems.
- 2) **Thin Layer:** The OS must not be very bulky with respect to the size of its code and must provide as thin a layer between the hardware and the domains. It must be written to provide minimal yet powerful abstractions of the underlying hardware. The layer must itself not support device drivers for any of the devices and must leave that support to the domain code.
- 3) **Scheduling:** The OS must provide for efficient scheduling of the domains based on priorities assigned to the domains.

- 4) **Embedded OS:** The OS must ideally come embedded with a processor. This way there would be maximal trust in the OS because of the tamper resistant properties of the hardware.
- 5) **Trusted Configuration Path:** Configuration changes to the trusted OS must not be possible from any domains or any higher level management software. All configurations must be allowed only using a trusted application which should be packaged with the OS itself and must happen out-of-band. No changes should be possible when the OS is running.
- 6) **Privilege levels for domains:** The domains must be allowed to run at various levels of privileges (but always above the OS privilege). This is useful in building a network of domains spread across the internet in which each domain is running at a similar privilege level thus facilitating distributed collaboration. These privileges must restrict on how the domain uses its resources.
- 7) **QOS for domains:** The OS must have facility to provide assured quality of service to domains as configured. For example, the financial domain shown in section 5 diagram would have high priority on the network and CPU over the other domains.
- 8) **Authentication:** Authentication support is not really required and it must be up to the domains themselves to provide for various forms of authentication.
- 9) **Switching between domains:** The OS must provide secure methods for switching between the various domains running concurrently on a machine. A user must be able to figure out the domain he is working in without relying on the domain itself.
- 6) **Persistent storage:** A domain should be able to store data (e.g. cryptographic keys) persistently, such that the integrity and the confidentiality of the data are ensured. The OS must provide locations within memory which are accessible only by the OS and not by any other programs.
- 7) **Communication:** A program should be able to exchange data with another program, such that the integrity and the confidentiality of the data are ensured.
- 8) **Local authentication:** A local user should be able to determine the identity of a program.
- 9) **Root Secure:** Even the platform administrator should not be able to break the basic privacy and isolation guarantees the OS provides to the domains.
- 10) **Attestation:** This feature allows an application running in a domain to cryptographically identify itself to a remote party, that is, to tell the remote party what is running inside the domain. This allows that party to put trust in the application, i.e. to have faith that the application will behave as desired.
- 11) **Semantic Attestation:** Attesting on the basis of a cryptographic hash only tells the remote machine about the version of a program but does not say anything about the behavior of the program. As is shown in [13], understanding the behavior of a program is very important to make correct estimates about the level of trust to be placed. Extending the idea in [13], the OS must provide abstractions for semantic attestation to help domains to query the behavior of programs within domains from a third party perspective. This might allow in detecting several classes of malicious code.

B. Security requirements

- 1) **Secure boot of domains:** The OS must take care to boot the domains in a secure way using integrity protected booting as discussed in [15].
- 2) **Isolation:** The OS must provision for each domain to run independent of others. Malicious behavior in one of the domains must not result in crashes in other domains. Also, the OS must ensure the confidentiality and integrity of data in each domain.
- 3) **Mandatory Access Controls:** The OS must implement mandatory access controls to mediate accesses to the hardware domains. These are required so that remote entities are assured the client computers are not running on tampered device drivers.
- 4) **Least Privilege:** The OS must have facilities for fine grained access control mechanisms for domains. It must be possible to configure privileges for applications running within domains.
- 5) **Trusted paths:** A domain should be able to receive data from a local input device (keyboard, mouse), such that only the domain and the user of the input device share the data. Data integrity must be assured.

- 12) **Encrypted Storage:** The OS must have support for encrypting the storage for the various domains to prevent data theft across domains.
- 13) **Key Revocation and management:** The OS must maintain keys for all the domains and must have facility to support the key revocation protocols in future.
- 14) **Tamper resistance for code:** The OS must provide semantics to validate code integrity within domains. One way of doing this is using the system called Pioneer [14]. Pioneer using timing measurements taken during checksum calculations of trusted code to detect tampering and integrity violations.

C. Ease of Use / Plug and Play requirements

- 1) **Minimal configuration rule:** It must be possible to build domains which would come preconfigured with all the required settings. There should not be any need for users to configure explicit security policies in domains. Applications should be built on domain such that when a user buys an application all he needs to do is to just click on a few buttons to start the

application. It must be as simple as installing and running a “toaster”.

- 2) **Plug and Play domains:** Domains should be easily transferable across machines and their configuration requirements must not change drastically across machines.
- 3) **Template driven domains:** Domain configuration must be template driven and it must be possible to select security templates from a list of available templates. The templates must not assume anything about the users and must be intuitive to use and apply. A common user should never have to bother about what the template actually does.
- 4) **Secure management of domains:** It must be possible for machine admins to create/delete/add/start/stop domains using a management domain. The management domain must possess a highly intuitive and easy interface and must not have any connections to the outside world. It must be accessible only physically. The management domain must not use any special privileges to run and must interact to other domains through the OS.

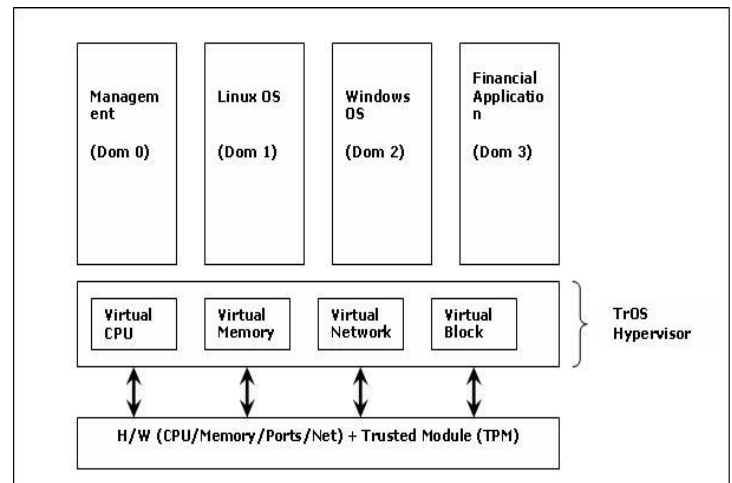
D. Advanced requirements

- 1) **Extensibility:** As discussed in [1], any “one size fits all” approach to providing an operating system for a trusted platform greatly limits a platform’s flexibility because it ties all applications to one interface. If this interface is too complex, it compromises the simplicity of the system, forcing many applications to deal with an unacceptably low level of assurance. Conversely, if it is too simple, it compromises the performance and functionality of the system, severely limiting the variety of applications that can usefully take advantage of it. The OS must provide powerful abstractions to developers such that they can tailor their applications for specific security requirements. One example would be building of a trusted financial application running all by itself within a domain without any underlying OS.
- 2) **Efficiency:** The OS layer must not introduce any overhead of its own and must not make any significant performance difference to applications running within the domains.
- 3) **Scalability:** It must be possible to run as many domains as possible on a single hardware depending on the current resource availability.
- 4) **Compatibility:** Existing operating systems and their applications must be able to run on TrOS without any major modifications to their software.

V. HIGH LEVEL ARCHITECTURE OF TROS

To meet a lot of the stated requirements above one could use the existing XEN[4] or VmWare[8] architectures as a starting point to build a trusted OS. Terra[1] has shown that such an architecture is possible but it is a research project and does not implement most of the requirements stated above.

To help the reader in better understanding the requirements, below is an architecture diagram much like XEN [4].



As seen in the figure, there are four domains running on the hardware. Two domains are running a separate operating system while one is running a standalone financial application. The domains are shown as boxes thus indicating that there is a clean separation between them. Each of these domains has a virtual CPU/Mem/Network and storage provided by the hypervisor. The hypervisor is a very thin layer above the physical hardware. The hypervisor also talks to the TPM module for all the trusted computing related requirements of sealed storage and remote attestation.

E. FUTURE WORK AND CONCLUSION

A lot of requirements in this paper are sort of wish lists and need a good amount of work to be researched and implemented. But, they do provide directions for thinking. The following areas are open research topics: Semantic attestation, plug and play security and tamper resistance for code. The approach to realize all the requirements will be to take a robust implementation like XEN [4] and research on filling in the gaps within XEN.

The work presented in this paper has listed out requirements which meet all the high level objectives set forth at the beginning. These are not the final requirements and surely there would be modifications required to cater to the ever changing world of computing.

ACKNOWLEDGEMENTS

The concept of building a computer system analogous to a “Toaster” appliance was born out of discussions that I had with Dr. Clifford Neuman during his Security Systems lectures at the University of Southern California. His analogy of the “Toaster” spurred my interest in looking at simplifying security for the common user and hence this work.

REFERENCES

- [1] T. Garfinkel, B. Pfaff, J. Chow, M. Rosenblum, and D. Boneh; Terra: A virtual machine-based platform for trusted computing; 19th Symposium on Operating System Principles(SOSP 2003), October 2003.
- [2] T. Garfinkel, M. Rosenblum, and D. Boneh; Flexible OS support and applications for Trusted Computing; 9th Workshop on Hot Topics in Operating Systems (HotOS- VIII), May 2003.
- [3] Trusted Computing platform Alliance, TCPA main specification v.1.1b.
<http://www.trustedcomputing.org/>
- [4] Barham, P., Dragovic, B., Fraser, K., Hand, S., Harris, T., Ho, A., Neugebauer, R., Pratt, I., and Warfield, A. 2003. Xen and the art of virtualization. In *Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles* (Bolton Landing, NY, USA, October 19 - 22, 2003). SOSP '03. ACM Press, New York, NY, 164-177.
- [5] Trusted Computing -
http://en.wikipedia.org/wiki/Trusted_Computing
- [6] R. Sailer, T. Jaeger, X. Zhang, and L. van Doorn. Attestation-based policy enforcement for remote access. In *Proceedings of ACM Conference on Computer and Communications Security (CCS)*, Oct. 2004.
- [7] David Molnar, Andrea Soppera, and David Wagner. Privacy For RFID through Trusted Computing. WPES 2005, November 7, 2005.
- [8] Virtualization Overview (VMware)
[http://www.vmware.com/pdf/virtualization.pdf#search=h=%22hypervisor%20VMware%20virtualization%20layer%22](http://www.vmware.com/pdf/virtualization.pdf#search=%22hypervisor%20VMware%20virtualization%20layer%22)
- [9] Microsoft next-generation secure computing base—technical FAQ.
<http://www.microsoft.com/technet/treeview/default.asp?url=/technet/security/%news/NGSCB.asp>, February 2003.
- [10] Peinado, M. Chen, Y. et al. (2004), NGSCB: A Trusted Open System. In *Proceedings of 9th Australasian Conference on Information Security and Privacy ACISP*, Sydney, Australia, July 13-15.
- [11] Reid, Jason F and Caelli, William J (2005) DRM, Trusted Computing and Operating System Architecture. In *Proceedings Australasian Information Security Workshop (AISW) 2005*, Newcastle, Australia.
- [12] Loscocco, P.A. Smalley, S.D. et al. (1998). The Inevitability of failure: The flawed assumption of security in modern computing environments. In *21st National Information Systems Security Conference*, Arlington, VA., August.
- [13] Vivek Haldar, Deepak Chandra, and Michael Franz; Semantic Remote attestation: A Virtual Machine Directed Approach to Trusted Computing; USENIX Virtual Machine Research and Technology Symposium, May 2004.
- [14] Arvind Seshadri , Mark Luk , Elaine Shi , Adrian Perrig , Leendert van Doorn , Pradeep Khosla, Pioneer: verifying code integrity and enforcing untampered code execution on legacy systems, *Proceedings of the twentieth ACM symposium on Operating systems principles*, October 23-26, 2005, Brighton, United Kingdom
- [15] Arbaugh, W.A. Farber, D.J. et al. (1997). A secure and reliable boot strap architecture. In *Proceedings of 1997IEEE Symposium on Security and Privacy*, pp 65-71, May 1997.