Attacking Computer Security Using Peripheral Device Drivers

Michael Aaron King

ATTACKING COMPUTER SECURITY USING PERIPHERAL DEVICE DRIVERS

By

Michael Aaron King

A Thesis
Submitted to the Faculty of
Mississippi State University
in Partial Fulfillment of the Requirements
for the Degree of Master of Science
in Computer Science
in the Department of Computer Science

Mississippi State, Mississippi

May 2010

ATTACKING COMPUTER SECURITY USING PERIPHERAL DEVICE DRIVERS

By

Michael Aaron King

Approved:

Yoginder S. Dandass Assistant Professor of Computer Science (Major Professor)	David A. Dampier Associate Professor of Computer Science (Committee Member)
Rayford B. Vaughn Professor of Computer Science (Committee Member)	Sara A. Rajala Dean of the Bagley College of Engineering
Edward B. Allen Graduate Coordinator Department of Computer Science and Engineering	

Name: Michael Aaron King

Date of Degree: May 1, 2010

Institution: Mississippi State University

Major Field: Computer Science

Major Professor: Yoginder S. Dandass

Title of Study: ATTACKING COMPUTER SECURITY USING PCI-E HARDWARE PERIPHERALS

Pages in Study: 46

Candidate for Degree of Master of Science

Detection of malicious logic on a hardware device is difficult to detect. This thesis proposes a device driver that emulates a hardware device and that device's software driver. This device driver attacks the target system by accessing the hard disk in order to perform read and write transactions without the knowledge of the operating system or intrusion detection/prevention software. The attacks performed by the driver compromise the confidentiality, integrity, and availability of data on the target system's disk drive. The attacks performed by the device driver have a less than one percent impact on system performance. This thesis, while tested in a Windows environment, applies to other operating systems (such as Linux/Unix, etc.) and thus has major implications for a wide range of users.

.

DEDICATION

I would like to dedicate this thesis to my wife Emily, whose support, encouragement, and understanding have made all the difference in this endeavor. I would also like to dedicate this work to my father, whose passion for the sciences inspired me to study them as well.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER I

INTRODUCTION

1.1 Introduction

Digital security has gained much attention with regards to protecting confidential systems from outsider threats (i.e. threats coming from outside the network of resources that are being protected). This type of security, while important, makes critical assumptions about the systems that the security mechanisms are attempting to protect. One such critical assumption is that the systems that are being protected are themselves trustworthy. However, this is a dangerous assumption, and one that is not always correct. This is why protecting confidential systems from attackers has been expanding to encompass insider threats (i.e. threats that come from within the network of resources that are being protected).

Handling both insider and outsider threats provides a more comprehensive approach to securing computer systems. In most scenarios, insider threats are comprised of software running on a confidential system that compromises the integrity, or trustworthiness, of that system. However, an insider threat that is only recently receiving attention is compromised hardware running on a confidential system within a protected network.

The concept of Trusted Computing is one area of digital security research that attempts to address the problem of insider threats. Trusted computing relies on the assumption of a trusted computing base (TCB) in order to provide security assurances about the system for which it is protecting. A trusted computing base is the core set of hardware and software that are responsible for providing security to a trusted computing platform. However, this thesis shows that the research done into trusted computing, thus far, is less than sufficient for handling all attacks from a compromised hardware component.

Another area of security research that attempts to provide a certain level of confidence in a computer system is the research being done into developing a secure bootstrapping protocol. Secure bootstrapping is the process by which the computer system builds layers of trust up through the boot process by verifying secure hashes. This area of research also falls short in dealing with the problem of compromised hardware. The literature survey shows that due to the basic assumptions of this approach, no significant security assurances can be made with regard to the type of malicious hardware device that is being proposed in this thesis.

A third, and much older, area of research that could protect a target system from attack is encryption. Encryption is the process of distorting information in such a way as to be unintelligible by eavesdroppers. The process of encryption is usually carried out using one of a variety of encryption algorithms. In order to perform the process of decryption, an encryption key is required. An encryption key is a piece of data that allows the encrypted information to be decrypted back into a readable form using a decryption algorithm. In most cases where encryption is used, the confidentiality of the information being encrypted remains intact until it is again put into an unencrypted state. However, the information can still be compromised while the information is in its plain text (i.e. unencrypted) form. In addition, there are cases where some attacks can compromise the availability of information even while in an encrypted state. One system utilizing encryption that will be considered is the TPM (Trusted Computing Module) proposed by the TCG (Trusted Computing Group).

The following sections in this thesis discuss motivations to explore the implications of confidential systems running compromised hardware. In addition, an approach to the proof of concept attack on a system running compromised hardware will be formulated. A similar, but more limited, attack strategy will be considered as well as possible defenses to the attack strategy. The experimental portion of this thesis will involve emulating the functionality of a malicious hardware device and its' device driver in order to compromise information on a target system's hard disk drive. The attacks from the malicious device will have no effect on system stability and will cause less than five percent degradation in system performance. In addition, the attacks presented in this thesis will not be detectable by current application level intrusion/prevention security measures.

1.2 Motivation

With the globalization of corporations in America, system hardware vendors are increasingly outsourcing more work to other countries where their costs to perform that work are lower as opposed to having it completed in the United States. The implications of such a trend on the security of our nation may be greater than anyone could have initially anticipated when it comes to technology companies.

For example, if a technology company outsourced a manufacturing job to another company (Company X) in another country (Country Y) that is not on good terms with the United States, that country may now have an inroad to spy on our nation. It is conceivable that Company X, in Country Y, having the manufacturing contract with the U.S technology company, could include malicious logic in the hardware design they were contracted to manufacture. The most dangerous scenario is one in which the components manufactured by Company X end up in computer systems that contain government classified information. These compromised hardware systems could then potentially perform cyber attacks on the host system of which they are a part. These attacks could be used as follows: 1) to compromise the integrity of the system by altering the information contained on the host; 2) to compromise the confidentiality of the system by transmitting data out to a third party; and 3) to compromise the availability of the system by causing the system to not provide data as required to authorized systems.

The experiments portion of this thesis will show that this type of scenario is indeed possible on workstation computers. The following literature survey of this thesis shows that this type of attack is extremely difficult to defend against as well as even more difficult to detect.

CHAPTER II

LITERATURE SURVEY

2.2 Introduction

The literature survey section examines and discuss three separate topics related to this thesis. The first topic to be discussed is an experiment by Qingbao et. al [1]. In their experiment Qingbao et. al. [1] propose a malicious hardware device that acts as a network switch. In the literature survey it is explained how their approach is similar but very divergent from the approach that is being proposed in this thesis. The second topic to be discussed is an approach to securing the boot-up process of a computer system to ensure a secure state once the operating system is loaded. This approach is termed the secure bootstrapping process. A specific implementation of a secure bootstrapping process is discussed called AEGIS. In the literature survey section of this thesis it is shown that this approach can give very limited, if any, security guarantees against the attack vector that is being proposed in this thesis. The next section of the literature survey looks at the process of encryption and how it might be used to protect against malicious hardware attacks. A specific security solution using encryption is also be presented and its impact on malicious hardware attacks is examined. The final section discusses the feasibility of remote controlling the malicious device (MDEV) by a malicious entity. The focus of the section is on utilizing covert channels to pass information between the remote malicious entity and the MDEV.

2.3 A Similar Experiment

In their paper Qingbao et. al. propose that hardware is “just as susceptible as software to hacker attacks, through inclusion of malicious logic”[1]. Hardware is susceptible to attack and unfortunately it is an area that does not appear to be heavily scrutinized. Their thesis seeks to explore a part of the information security problem space with regards to compromised hardware.

In [1], the authors look at the problem from an embedded device perspective, specifically a network switch. Figure 1.1 at the end of this section gives an illustration of the device they designed. They propose a network switch that has a malicious piece of hardware, implemented in a Field Programmable Gate Array (FPGA) device. In their implementation, their malicious FPGA reads data as it comes in to the device to be buffered in the SGRAM, for retransmission. Quingbao et. al. designed a number of, what they refer to as, “threat commands”. The FPGA, as it is hooked into the incoming packet data stream, checks each incoming packet and tries to match it to one of the “threat commands”. If the packet contains one of the predetermined “threat commands”, the FPGA alters the data in the Electrically Erasable Programmable Read Only Memory (EEPROM) chip. This causes the device configuration to be altered, and the switch to change its behavior. For example, given a certain “threat command”, the switch configuration will be altered and the device configuration will change, causing the switch to broadcast the data stream between port 1 and 2 to all ports [1]. Qingbao et. al. make the claim that this type of attack is “undetectable using traditional testing and verification tools”. Due to the fact that it is the device itself that is malicious, and most, if not all, traditional techniques would be searching for an outside attack on the device, this claim definitely has merit. They go on to quantify their claim to extend to all malicious hardware insider threats when they make the statement, “As the appearance of hardware threat, information security is confronted with huge challenge, because there is no effective way to detect and defend hardware threat”[1].

This thesis explores a similar part of the information security problem space as [1], but with more flexibility about the type of target that can be compromised. This thesis proposes a malicious device with the capability to attack a workstation computer as opposed to an embedded device. Where the device proposed by [1] is very limited in its capability, the device proposed by this thesis targets any storage device that is connected to a PCI-E bus on a Personal Computer (PC) workstation. Where data over network is often encrypted, data flowing between devices on a PC is usually not; however, there are a few exceptions that will be discussed later in this literature review. This implies the device proposed by this thesis is more likely be able to compromise the confidentiality, integrity, and availability of the data that it is gathering, where often, the only attack the device proposed by [1] is able to perform is a Denial of Service (DoS) attack compromising only the availability of data flowing from it.

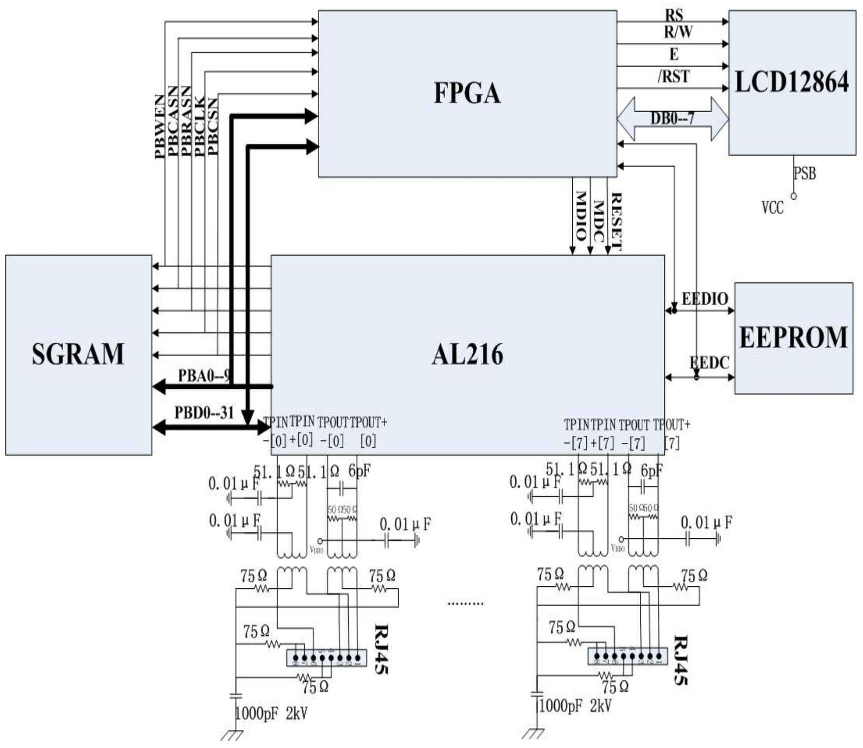


Figure 1 Illustration of the device built by (Li, Gao, & Xu, 2008)

2.4 Secure Bootstrap

One possible approach to the problem of compromised hardware on a security system, is securing the bootstrapping process of the system. Bootstrapping is a term assigned to the process of initialization as it relates to digital devices. From the time a device is powered on, up to the point the main software loop takes control is referred to as the bootstrapping process. In the field of digital security, the process of bootstrapping the operating system on workstations is usually considered a trusted process. However, [2] makes the case that a higher computational abstraction layer, such as the operating system, cannot be assumed secure if it was loaded by a lower computational abstraction layer, such as the BIOS, that has no security mechanisms in place. “Integrity of a layer can be guaranteed if and only if: 1. the integrity of the lower layers is checked, and 2.

transitions to higher layers occur only after integrity checks on them are complete”[2]. They go on to make the statement that if the two previously mentioned conditions are not met, “as they typically are not in the bootstrapping (initialization) of a computer system, no integrity guarantees can be made”[2]. The authors seek to make these security guarantees through the use of their proposed system, AEGIS.

The authors define AEGIS as a secure bootstrap process that is able to ensure bootstrap code integrity. AEGIS uses a “chain of integrity checks”[2] from power-on, all the way through the bootstrapping process up to the point that the operating system is loaded. The AEGIS boot process is split into two logical divisions. In the first phase, the AEGIS process performs a checksum on its address space to protect against ROM failures. If this process was successful, then a cryptographic hash of the second logical section is computed and the verified against a stored signature. Upon successful validation, control of the boot process is passed to the second logical division [2]. At this point the second logical division of the boot process begins. With each expansion ROM that needs to be executed, a hash function in computed and checked against a stored signature for that expansion code. When each expansion ROM has been verified, the BIOS passes control up to the operating system bootstrap code. The operating system core code (operating system kernel) then takes over after it has been verified by the boot block [2]. At this point the authors make the claim that the boot process is guaranteed to end up in a secure state. In addition, “no code is executed unless it is either explicitly trusted or its integrity is verified prior to its use” [2].

For most scenarios this process would seem to be able to provide reasonable assurances as to the security of the system when the operating system takes control.

However, if the attacker is a middle man somewhere in the supply chain, as is assumed in this thesis, this security procedure does not provide adequate protection from hardware-based attack vectors. For example, if the attacker has modified the logic of the hardware on a PCI-e device, the AEGIS system makes no provision to safeguard against this device. This means that the AEGIS process would assume that the operating system was in a secure state. However, the system as a whole would be compromised at a lower abstraction level, and thus the system user would have a false sense of security. In fact, the authors of the AEGIS system formally make the assumption that much of the hardware is trusted. “The first assumption upon which the AEGIS model is based is that the motherboard, processor, and a portion of the system ROM (BIOS) are not compromised, *i.e.*, the adversary is unable or unwilling to replace the motherboard or BIOS” [2]. In their assumptions, the authors of AEGIS did not explicitly mention that they assumed the trustworthiness of the system expansion devices’ hardware. Therefore, it is apparent that this scenario was not considered. Due to these observations, it can be seen that the approach taken in the experiments of this thesis would be able to exploit a system using the AEGIS methodology and, therefore, it is not a solution to the attack vector we are proposing.

An extension was proposed by [3], to the AEGIS architecture with the goal of securing the “firmware of all devices in the computer, not just devices that are accessible by the host CPU” [3]. The attack vector being proposed by the experiments of this thesis is one that is being taken into account by Hendricks, et. al. This makes their security mechanisms a possible solution to this thesis approach. It will be shown that in spite of the fact that the Hendricks, et. al. are attempting to protect the host system from compromised expansion devices, our attack vector is still a valid one. In fact, their approach may provide no security at all against an attack from our proposed system.

Hendricks et. al. make the statement that “Both the [AEGIS approach] and TCG based approaches share a CPU-centric view of the system that is inadequate for establishing a trustworthy system”. They go to mention that these approaches do not defend against attacks that utilize the processing abilities of peripheral devices. “Though peripherals and memory are implicitly proposed to be a part of the TCB, we do not believe they are currently adequately verified”[3]. The aim of Hendricks et al. is to protect against hardware compromise. They approach the problem by attempting to verify the authenticity of the firmware on peripheral devices of the host system. The system “must ensure that its firmware is signed and verified at startup just like the rest of the executable code, and it must verify its children”[3].

In spite of the fact that the extension proposed by Hendricks et al. is explicitly supposed to provide a layer of security against compromised hardware, it will not provide any protection against the attack vector being proposed here. For example, if the attacker has modified the logic of the hardware on a PCI-e device, then the checksum in the BIOS against the code for that firmware will pass. This is because that code would come from the trusted source from which the digital signature was originally calculated. It is the hardware logic that has been compromised, not the firmware on the device. Therefore, the AEGIS software with the extension proposed by Hendricks et al. will not provide protection against the hardware attack that is being proposed by this thesis.

2.5 Encryption

This section examines encryption, in general, as a possible defense against the attack vector proposed in this thesis. In addition, specific security approaches utilizing encryption that aim to guard against hardware attacks will be reviewed. However, it will be shown that in-spite of the fact that encryption is used to safeguard the host system; various attacks can still be performed that compromise the systems state of security.

Encryption is the process of changing information into an unusable state using an algorithm of some kind. While in the encrypted state, it is extremely difficult to be able to read the original information from the encrypted data (ciphertext). The characteristic of the data that encryption is protecting is confidentiality. The point of encryption is to make sure that the ciphertext is not read by anyone that does not have the special encryption information known as the cryptographic key. With any security measures in place that use encryption to protect the hardware, compromising the confidentiality of the data could prove difficult even with the sophisticated attack vector that is being proposed in this thesis. However, that does not mean that it would be impossible.

There are two types of encrypted data on a computer system. There is encrypted data that is saved on some type of storage device, and there is encrypted data that is in transit from one device to another. However, it is important to note that at some point, in order for the information contained within the ciphertext to be useful, the ciphertext is going to have to be unencrypted in order for the information to be processed by some device on the host system; it is at this point where the data is once again vulnerable.

For example, if the data is stored in an encrypted state on a hard disk it is unusable without the decryption key. However, when it is transferred to some device for processing, it must be unencrypted. If that device is trusted, as is the case in the proposed attack, then the host should have no problem passing the data to the device and allowing the device to process the data while it is unencrypted. We assume in the previous example that the data was in an encrypted state as it transitioned from storage on the hard disk to the device for processing. This scenario does not pose any security guarantees with regard to a compromised hardware device that is trusted on the host.

Trusted computing is one attempt to protect computing systems from insider threats as well as outsider threats. The Trusted Computing Group (TCG), formed from the Trusted Computing Platform Alliance in 2003, has proposed a number of specifications that could be implemented to form a root core of trust on a host [4]. The specification that is most relevant to the work being done here is the specification of the Trusted Computing Module (TCM). The TCM is described as a “low performance cryptographic coprocessor”[4], and is “assumed to be the ‘trust anchor’ in a computing platform”[5]. TPMs are usually shipped as a component of the motherboard, but can be located on expansion cards as well [5]. Together with the host CPU, the TPM provides the host system with ability to encrypt and decrypt data as it flows throughout the computer. The TPM has a cryptographic co-processor, with the ability to perform key generation, encryption/decryption, random number generation, and hashing as well as other cryptographic functionality [6]. “The TPM uses these capabilities to perform generation of random data, generation of asymmetric keys, signing and confidentiality of stored data” [6].

In spite of the fact that the TPM has all of this cryptographic functionality, encryption, as was stated in the introduction to this section, cannot provide protection against all possible attacks. It would still be very easy for a malicious piece of hardware to overwrite data on the hard disk of the host, thus compromising the availability of the data. In addition, with the attack scenario presented here, we assume that the malicious device is trusted from the manufacturer, and thus the TPM will allow data to be read by the device. Therefore, the confidentiality of the device would also be compromised by the attack vector being presented. If the malicious device wanted to write data back to some storage device as well, that would also be permitted by the TPM. If there were, in fact, some policy that notified the host when data was altered, in memory or on a storage device, by some component other than the CPU, the integrity compromise would be discovered. However, in spite of this fact, the integrity of the data would still be compromised and it would be very difficult to discover the culprit. There are also vulnerabilities in the TPM itself. As [6] point out, “the communication channel between TPM and CPU is mainly unsecured”. A malicious piece of hardware could take advantage of the insecure data flowing between the TPM and CPU. However, this is not the approach that is taken in the presented attack scenario.

It can be seen that encryption alone is not the solution to the attack scenario that is being presented. Even in the case of hardware devices that implement security solutions using encryption, such as the TPM, the host system is still vulnerable.

2.6 Confidential Communication

Being able to transfer data to and from the MDEV is a high priority. This section will discuss the feasibility of confidential remote communication with the MDEV through the use of covert channels. A covert channel is a “path of communication that was not designed to be used for communication”. [10] In other words a covert channel is a mechanism for passing information over a media secretly where that information was not intended to flow. Due to the remote nature of the malicious entity utilizing the MDEV for exploitation, the TCP/IP stack will be considered the medium in which the covert channel shall be embedded.

A covert channel may be implemented in a variety of ways. The first type of covert channel that will be discussed here is a storage channel. A storage channel allows communication, “by modifying a stored object. This covert channel involves the direct or indirect writing to a storage location by one process and the direct or indirect reading of the storage location by another process.”[10] By modifying the data in a TCP/IP packet from the MDEV, and the remote entity retrieving the data, a covert storage channel is being created. [10] names multiple TCP/IP header fields that may modified to embed data for covert transmission. Those fields include “Type of Service”, “IP Identification”, and “TCP Sequence Number” to name a few.

In their paper Ray and Mishra [11] describe a covert storage channel they have implemented based on the internet protocol (IP). The authors insist that their covert channel could be built “over any IP-based channels” [11]; however, in their publication they focus on an ICMP Echo-Request packet. ICMP (Internet Control Message Protocol) packets are used to by networked computers to gather information about other computers on the network. An ICMP Echo-Request packet is sent when the “ping” program is run. These packets are returned to the requesting networked computer on which the ping program is being executed in order to gather status information about another networked computer. By choosing the ICMP Echo-Request packet as the storage channel, the authors of [11] hope to provide a covert channel that is very difficult to discover. Ray and Mishra use two covert channels in one ICMP Echo-Request packet in order to keep the size of the covert channel low to avoid detection. The authors go on to make the statement that "by simply inspecting all packets it will not be possible for the intermediary to decide with certainty that a particular packet is carrying some covert data and not normal traffic generated by the end hosts" [11]. Ray and Mishra go on to conclude that they have created a covert channel that allows for 5 bytes of data per ICMP Echo-Request packet and that according to preliminary analysis "it is not possible to detect this covert channel based on statistical analysis of sent data" [11].

In many cases the security policy on a network will disallow or flag ICMP packets flowing outside the local area network (LAN). In this case it may be more appropriate to design a covert channel that is based on another type of network traffic than ICMP. Sellke et. al. have developed a covert channel using TCP/IP that utilizes timing as opposed to storage space to implement the covert channel and is thus referred to as a timing covert channel. This type of covert channel would be able flow outside the LAN and would be difficult to detect due to the fact that the contents of the packet are not altered. Information is leaked based on the timing of the packet arrival on the receiving end of the channel. In fact, the authors make the statement that "the traffic patterns of the covert timing channel can be made computationally indistinguishable from that of normal traffic, which makes detecting the communication virtually impossible" [12].

It has been shown that there are multiple ways in which to construct a covert channel to facilitate leaking information from a secure system. The TCP/IP stack can be exploited to contain a covert storage channel, as seen in [11], or a covert timing channel, as seen in [12]. In both cases information was leaked from the compromised host to a remote host. In [12] it was shown that a low bandwidth timing channel was utilized to not only pass information but also be difficult to detect. These two examples show that information can passed from the MDEV to a remote entity.

CHAPTER III

APPROACH

3.2 Introduction

In this section, a high level view of the target system architecture is presented. Also to be discussed is the process of data flow throughout the target system and how that data is retrieved by the malicious hardware device thus compromising the data on the target system.

The target system for this thesis is an x86 general purpose computer. By using a common computing platform, it is apparent that the implications of this experiment are widespread. The target system is composed of a motherboard, Intel 32-bit processor, 2GB of random access memory (RAM), a serial advanced technology attachment (SATA) connected hard drive, and a network interface card (NIC). In a real world attack the NIC would act as the malicious hardware that will compromise the target system. For the purposes of these thesis experiments, the NIC (i.e. malicious device) is implemented using a Windows device driver (WDD). Figure 2 shows a conceptual representation of the target system with the proposed malicious hardware device already installed.

Figure 2 shows MDEV transactions with the host system’s host bus adapter (HBA), also deemed the disk controller. Each item in the image shown in Figure 2 represents some component of the x86 host (target) system. The largest item in the image is the target x86-system itself. Some of the more common components of the x86 computing platform include the item labeled ‘Memory’ which represents the actual RAM present on the target system and the item labeled ‘CPU’ which represents the central processing unit of the target. The lines connecting the CPU to the Root Complex represent the connection between the CPU and the North Bridge of the Root Complex. This connection is called the Front Side Bus (FSB). The lines connecting the hard disk controller, root complex, and MDEV represent the PCI-e bus. The lines connecting the Memory with the Root Complex are the memory bus. The component labeled ‘Root Complex’ does not actually correspond to a physical piece of hardware in the system. Instead, the Root Complex represents a subset of the functionality of the North Bridge and South Bridge which together comprise the chipset. Its relevance to this thesis has to do with the routing of disk accesses throughout the target system. The functionality of the Root Complex that is relevant to this thesis is that functionality with which the Root Complex takes a given memory address or an address range and routes the address to an address in RAM or to an address on a peripheral device on the host system. The item labeled ‘MDEV’ represents the malicious NIC. All of these components are relevant to how the target system is compromised.

In addition to showing the components involved in the experiments of this thesis, Figure 2 also shows how data flows between those components. In Figure 2 the MDEV initiates a read from the disk drive to the Root Complex. This initial disk read request is labeled as ‘1’ in the figure. The Root Complex then recognizes the address range specified by the MDEV as being memory mapped from the HBA. Then the Root Complex reads the contents of RAM at the address range, represented in the path labeled

‘2’, and makes the data available to the MDEV. Since the AHCI space of the HBA is mapped into system memory, the flow of data portrayed in Figure 2 occurs through the use of read/write transactions from/to host system memory. This concept will be elaborated upon in Chapter four.

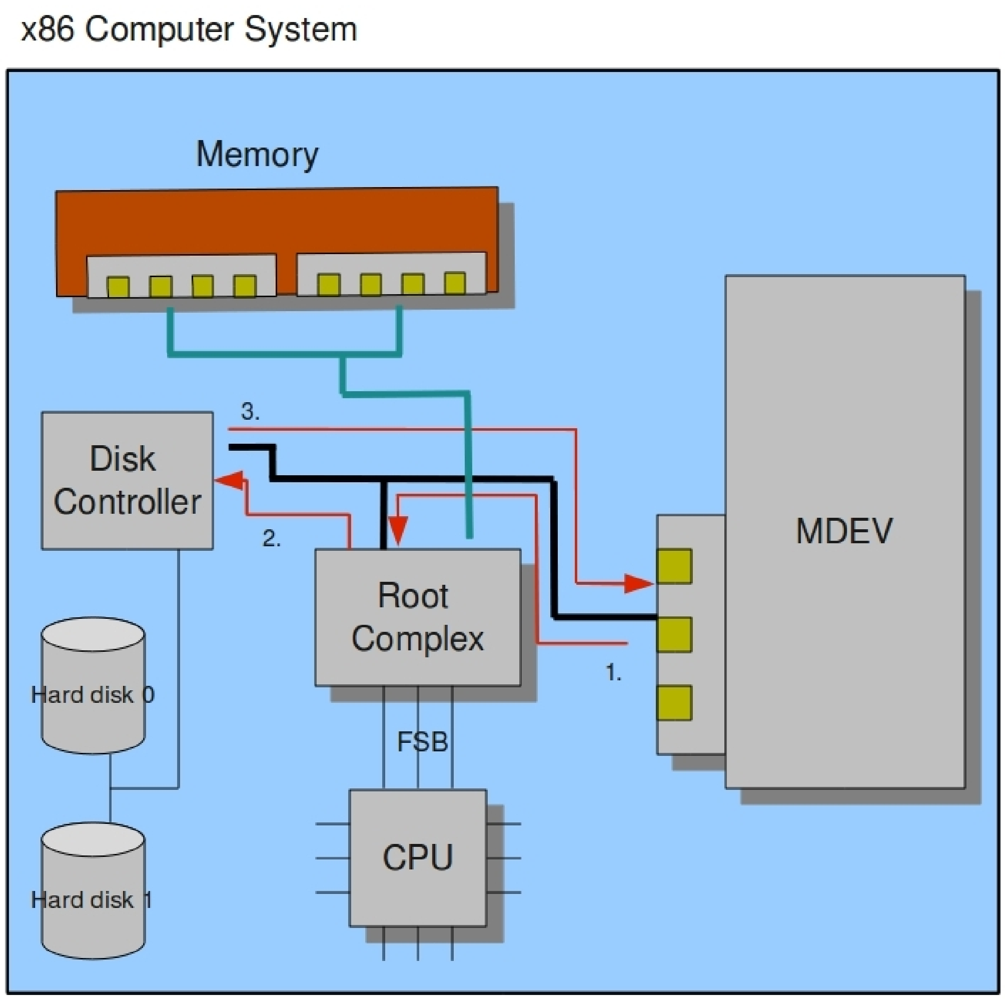


Figure 2 Host/Target System High Level Architectural View

Figure 2 also shows how data transactions can occur on the target system without the knowledge of the CPU and thus the OS as well. The fact that the CPU and OS have not been alerted to the memory/hard disk read is one reason why this malicious hardware is successful at not being detected.

Once the data has been retrieved from the host system and stored in the on-board memory of the MDEV, it can then be transmitted to a malicious entity outside the network. The malicious device also has the capability to request instructions from a malicious entity remotely. Due to the fact that the malicious hardware device is acting as the host system's network interface card, it could implement any number of protocols to communicate with someone outside the network on which it resides. For example, one such communication protocol in which the malicious device receives instructions could be in the form of a simple web GET request. The FPGA/NIC could send out intermittent GET requests from a specified source. The response could be encoded with the instructions for the FPGA/NIC to perform some action (i.e. read data from memory), thus compromising the host system. By using the hyper text transfer protocol (HTTP) the routers and switches will simply allow the transaction to pass through the network as legitimate web traffic. To further hide the actions of the malicious device, the secure hyper text transfer protocol (https) could be used. This would encrypt the payload data of the communication packets flowing between the malicious device on a compromised host system and the malicious entity providing the device with instruction.

3.3 Implementation details

3.3.1 PCI Express

PCI Express (PCI-e) is a data transmission media used in computers systems to send and receive data between devices (i.e. CPU, Hard disk, Video card). The overall PCI-e bus architecture works much like a switched Ethernet network. The physical PCI-e bus itself is composed of wires, switches, and endpoints (i.e. PCI-e devices). The wires connecting two PCI-e devices are referred to as a Link. “A Link consists of either x1, x2, x4, x8, x12, x16, or x32 signal pairs in each direction”[9]. It is up to the designers of the system as to how many signal pairs to use as the interconnect on the PCI-e bus. Switches act in much the same way that an Ethernet switch would act. On a high level, the switches on a PCI-e bus route PCI-e traffic between endpoints and between endpoints and the Root Complex. Each switch is composed of multiple ports. One upstream port, and multiple downstream ports. Each downstream port connects to either an endpoint or another switch. The upstream port connects the switch directly to the Root Complex or indirectly to the Root Complex via another switch. The endpoints in the PCI-e architecture refer to the actual peripheral devices that want to communicate over the PCI-e bus. For example, many video and sound cards can be designed in such a way as to use the PCI-e bus of a computer system to communicate with the host.

The traffic that passes over the PCI-e bus is composed of packets. “PCI Express employs packets to accomplish data transfers between devices”[9]. These packets are called Transaction Layer Packets (TLPs). The TLPs are composed of multiple fields, each appended by various stages of transmission. The fields of a TLP are the Start, Sequence Number, Header, Data, ECRC, LCRC, and End fields. Figure 3 shows a conceptual representation of a TLP.

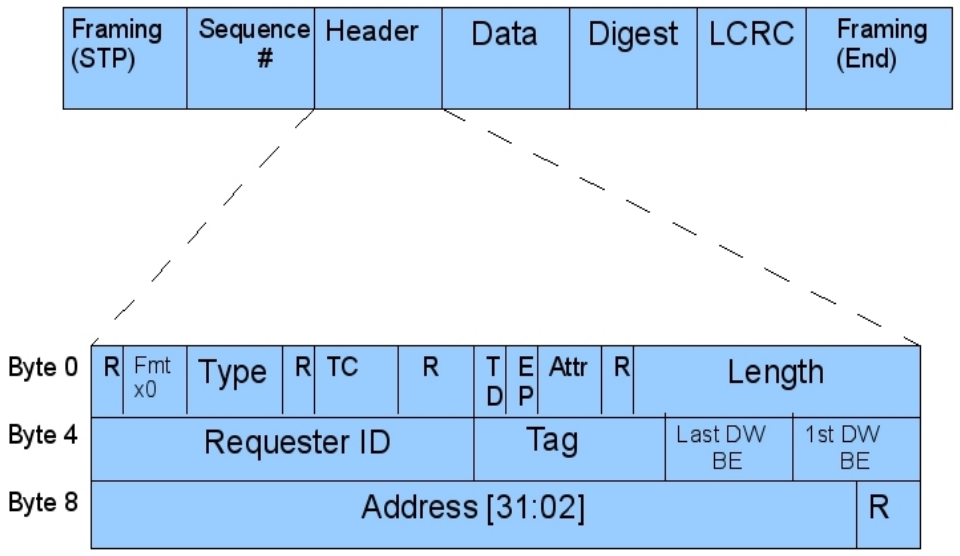


Figure 3 PCI-e Transaction Layer Packet [9]

The header field of the TLP contains important information for routing the packet over the PCI-e bus. The TLP header consists of three double words (DW). When talking about PCI-e, a word refers to a 16-bit value; therefore, a DW is a 32-bit value. As Figure 3 shows, the header field contains the address of the device that the TLP is to be routed to as well as the address of the requester, or sender, of the packet. For example, if a read request is sent to device B from device A, the address of device B will be in the address field and the A's ID will be in the Requester ID field.

3.3.2 ATA Disk Controller

ATA is a data transfer interface for various storage devices. There are two major categories of ATA interface. The first is Parallel ATA and the second is Serial ATA (SATA). SATA is the type of interface that is used to connect the hard drive to the PCI-e bus on the target system. The SATA disk controller connects the hard disks on computer systems to the PCI-e bus. It is the disk controller (Host Bus Adapter) that provides the interface to the PCI-e bus. The host bus adapter (HBA) provides multiple block address registers (BARs) that are used for various things. BAR5, also called ABAR, is the AHCI base address register. It is this BAR that the MDEV manipulates in order insert commands to be executed by the HBA without the knowledge of the operating system.

In addition to the base address registers, AHCI also requires that the host operating system have a number of data structures setup. These data structures include the command list and command table. The command list is a queue of command headers. Figure 5 provides a conceptual representation of the command list structure. The command headers, shown in Figure 6, contain information such as the base address of the command table, the bit to determine a read or a write, and the length and size of the physical region descriptor table. The command table, shown in Figure 7, contains the Command Frame Information Structure (CFIS) as well as the physical region descriptor table. The role of the physical region descriptor table is to keep track of the location in memory where data from a read transaction is stored. Figure 4 displays a conceptual layout of the HBA's AHCI base address register. Here it becomes apparent that the ABAR contains offsets 100h – 1100h that are labeled Port 0 to Port 31. Each port corresponds to a hard disk connected to the HBA. Every port address contains pointers into system memory where each port's command list is stored.

The target system being used employs a protocol called Native Command Queuing. Native Command Queuing (NCQ) enables the hard disk to service multiple commands from the host. When setting up commands for an HBA that supports NCQ, queued commands and non-queued commands cannot be mixed. In the AHCI BAR, each port address points to a command list. That command list contains the queue of commands that the device executes.

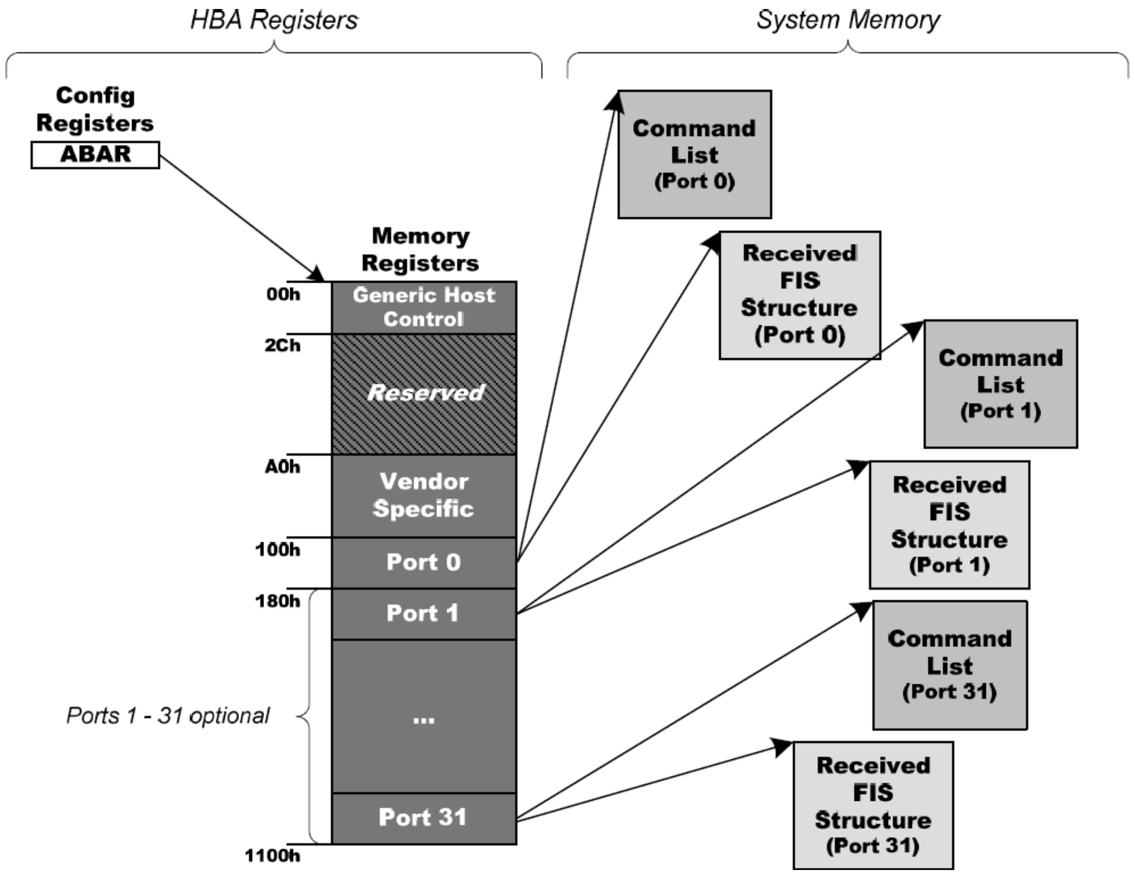


Figure 4   HBA ABAR Layout [13]

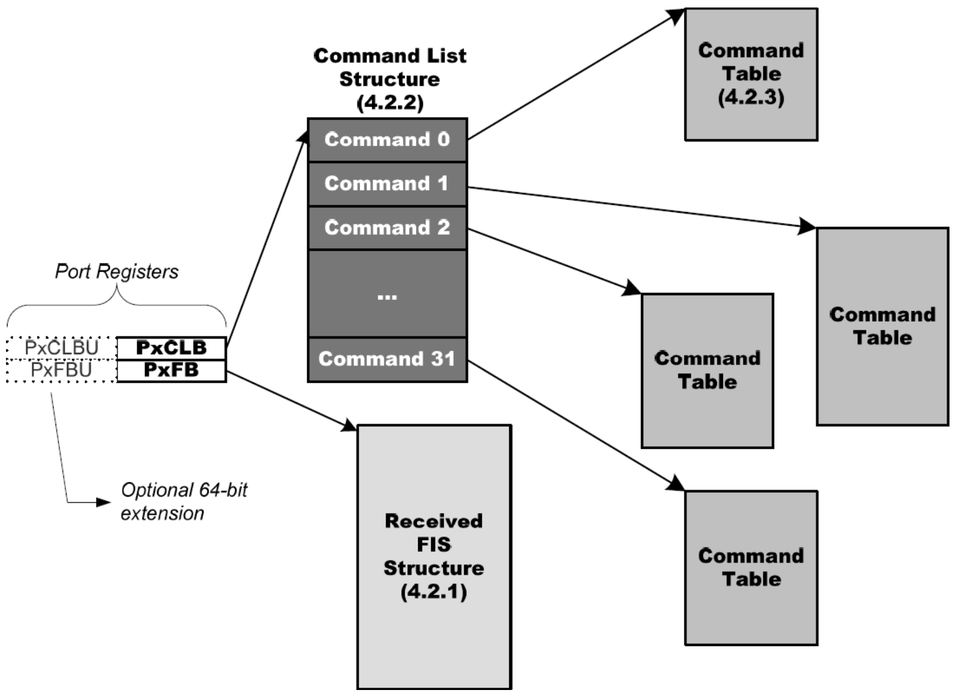
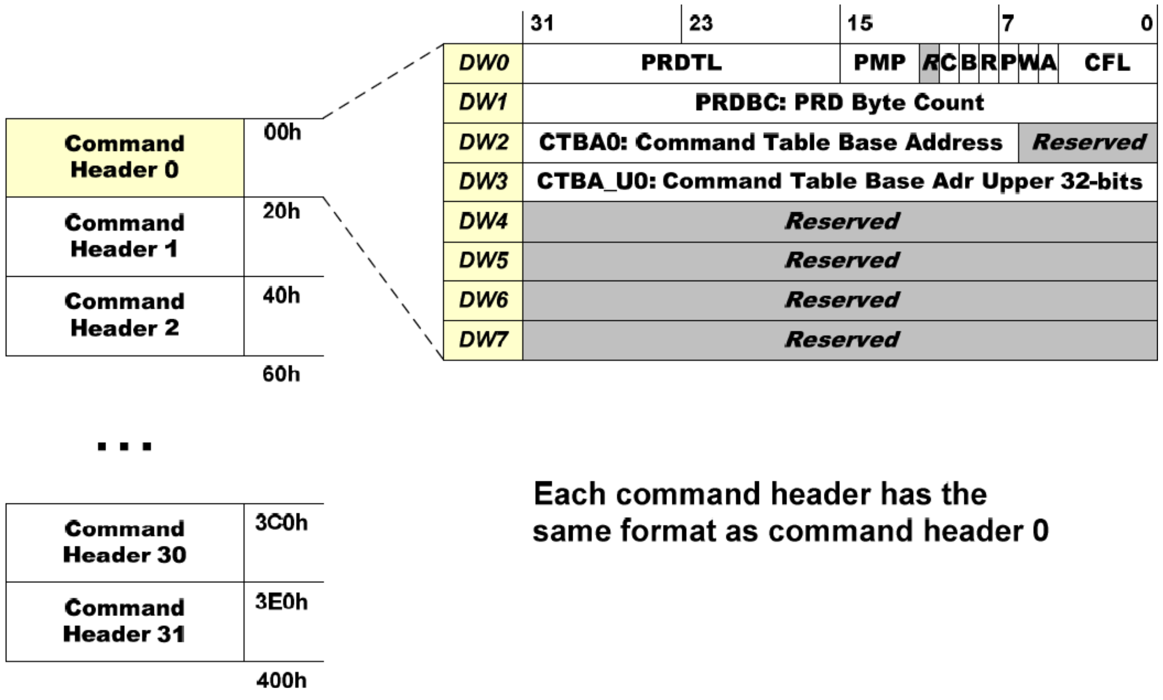


Figure 5 AHCI Command List Structure [13]



	31	23	15	7	0
DW0	PRDTL		PMP R C B R P W A	CFL	
DW1	PRDBC: PRD Byte Count				
DW2	CTBA0: Command Table Base Address				Reserved
DW3	CTBA_U0: Command Table Base Adr Upper 32-bits				
DW4	Reserved				
DW5	Reserved				
DW6	Reserved				
DW7	Reserved				

Figure 6 AHCI Command Header Structure [13]

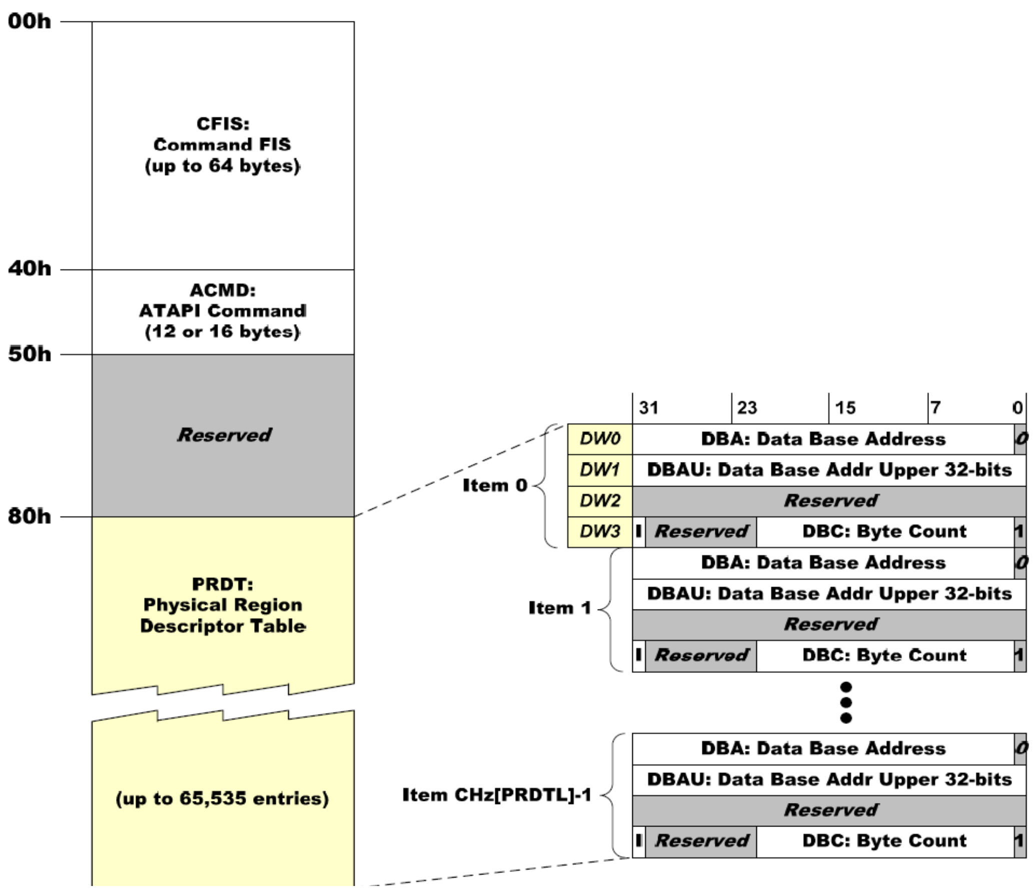


Figure 7 AHCI Command Table Structure [13]

3.3.3 *Malicious Hardware Device*

The malicious device that could be used in to perform the attacks on the host system could be implemented in a number of ways. By using this device, it can be shown that an attacker with the intention of compromising a system using the approach presented in this thesis will be able to do so with little chance of discovery.

The MDEV would have all of the logic to perform these system attacks built directly into the logic. It is the job of the MDEV to setup up the PCI-e interface and initialize the base address registers to be used for communication with the host machine. The MDEV could potentially receive instructions from the remote malicious entity and then execute the specified attack on the host system. If the MDEV is instructed to read a chunk of host memory or a sector of the hard disk, then it will do so via a direct memory access (DMA) read. This type of read transaction performs a read on a portion of host memory that could be greater than one 32-bit value. The data would be read into the MDEV's base address registers that have been mapped to the host memory address space. At this point the system has been compromised. Confidential data has been read into the MDEV, which can then do any number of things with the data.

For the experimental portion of this thesis, the MDEV is emulated using a device driver (Windows terminology)/kernel module (Linux terminology). This allows for a more controlled and simplified testing environment. The kernel module provides software functionality that cannot be performed by normal programs running on a computer. For these thesis experiments, the kernel module does not perform or execute any functionality that could not be performed by hardware logic with direct access to the host system.

3.3.4 Linux Kernel Module

Typically, Linux device drivers for PCI-e peripherals are implemented as loadable Linux kernel modules. A Linux kernel module (LKM) is a software program that can be dynamically loaded into the running Linux kernel while the kernel is running. "Each piece of code that can be added to the kernel at run-time is called a module. Each module is made up of object code that can be dynamically linked to the running kernel by the *insmod* program and can be unlinked by the *rmmod* program"[8]. Figure 4 shows how using the *insmod* program inserts the kernel module object code dynamically into the kernel at runtime. The object code that is inserted into the kernel is built using the Linux kernel API, and thus must be compiled against the kernel source tree present on the development system. A kernel module is code executed by the processor in 'kernel mode' as opposed to 'user mode', which is the execution domain for application level programs. By running in kernel mode and having access to kernel functionality, the kernel module has unrestricted access to system utilities and information. The fact that a kernel module has unrestricted access to the host is important only in the capacity that the LKM has access to the Linux kernel application programming interface (API).

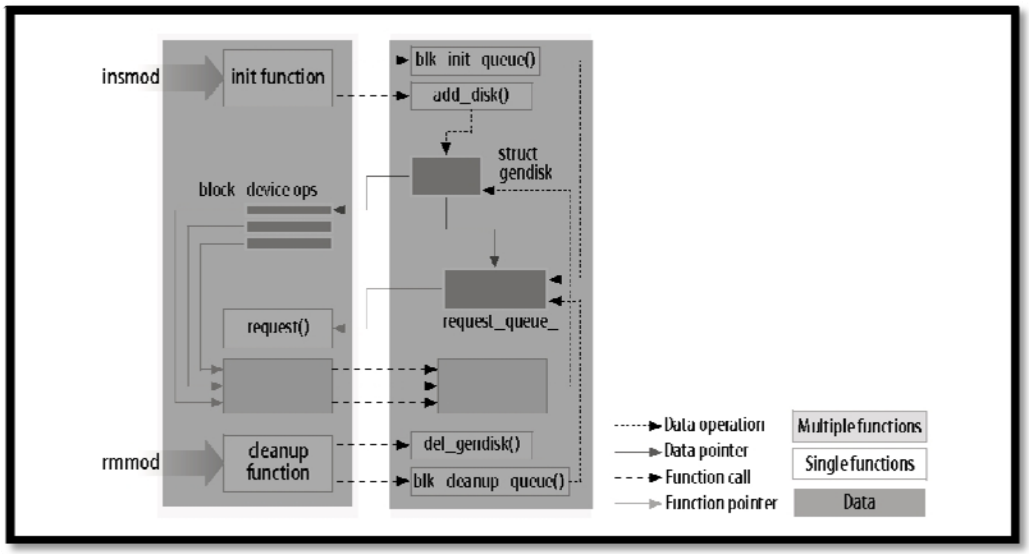


Figure 8 Example of insmod and rmmod usage[8]

3.3.5 Windows Device Driver

The high level concepts of WDDs are very similar to that of the Linux kernel modules. Like the LKMs, WDDs can be loaded by the running kernel. The API functions for performing such tasks as inserting and removing the module are different, but both the LKM and WDD require function callbacks that are called when the module code is dynamically loaded into the running kernel. Like the LKM, the WDD runs in kernel mode and is given access to all data structures built by the Windows kernel as well as having access to all the functionality of the Windows kernel. It is for this reason that a WDD can be used in place of a hardware device for the experimental portion of this thesis.

3.3.6 PCITree

PCITree is a Windows graphical shareware utility that allows users to have read/write access to each pci connected device's memory mapped base address registers. While it is possible to obtain the memory location of the BAR's of a pci device without using PCITree, this utility simplifys the experimental portion of the thesis.

3.4 Hypothesis

The hypothesis of this research is that a compromised PCI-e peripheral device, emulated by a WDD, can maliciously access disk drives on a host computer system with no discernable impact on the stability of the host system and less than five percent degradation of system performance while capturing host system data at full capacity. Furthermore, the attacks will not be detected by popular intrusion detection/prevention security measures such as anti-virus and intrusion detection software such as SNORT.

CHAPTER IV

DESIGN OF EXPERIMENTS AND RESULTS

4.2 Introduction

The experimental portion of this thesis consists of two major testing components. Part one involves compromising data from the host system's serial ATA AHCI hard disk. Part two consists of performing a number of timing experiments which test the impact that the attacks have on the target system performance. In the first series of experiments, the confidentiality, integrity, and availability of the data are compromised as follows: a) read transactions are issued to the disk drive to compromise the confidentiality of the data on the host; b) write transactions are issued to the host system's disk drive to compromise the integrity of the data on the host; c) large targeted write transactions are issued to the data storage unit in order to compromise the availability of data on the host system by overwriting critical operating system data or the confidential data itself.

In order to better control the environment of the thesis experiments, there is a WDD playing the role of the MDEV. The WDD allows for commands to be sent to the disk drive. Various commands can be sent to the disk drive using the WDD, with each command allowing for the execution of a different attack. The WDD gives the disk drive commands by writing to the HBA's command registers. Upon receiving a command, data flows through the system as presented in Figure 2.

The WDD in these experiments is used to give commands to the HBA as well as verify that those commands have been carried out as expected. In order to give commands to the HBA, host memory address space must be allocated which maps to the AHCI base address register (ABAR) on the HBA. By using the WDD to write data into host memory that has been mapped to the HBA's control registers, the WDD is able to give commands to the HBA. AHCI ABAR memory space on the HBA that has been mapped to the host system's RAM for communication between the WDD and the HBA itself. For example, when a read transaction has been performed by the HBA on a specific disk drive, the WDD is able to read the contents of the address space that has been mapped from the HBA's ABAR and verify that a read has occurred. Figure 9 shows a conceptual representation of the mapping from the HBA's memory space into the host system's memory space. In order for the WDD to have access to the HBA's registers, it must first determine where those registers have been mapped into the host's system memory. In order to simplify the process for the purposes of the experiments, the AHCI BAR of the HBA is determined with the use of PCITree. This address is then hard coded into the WDD. From this point forward this chapter will refer to the WDD and the MDEV interchangeably.

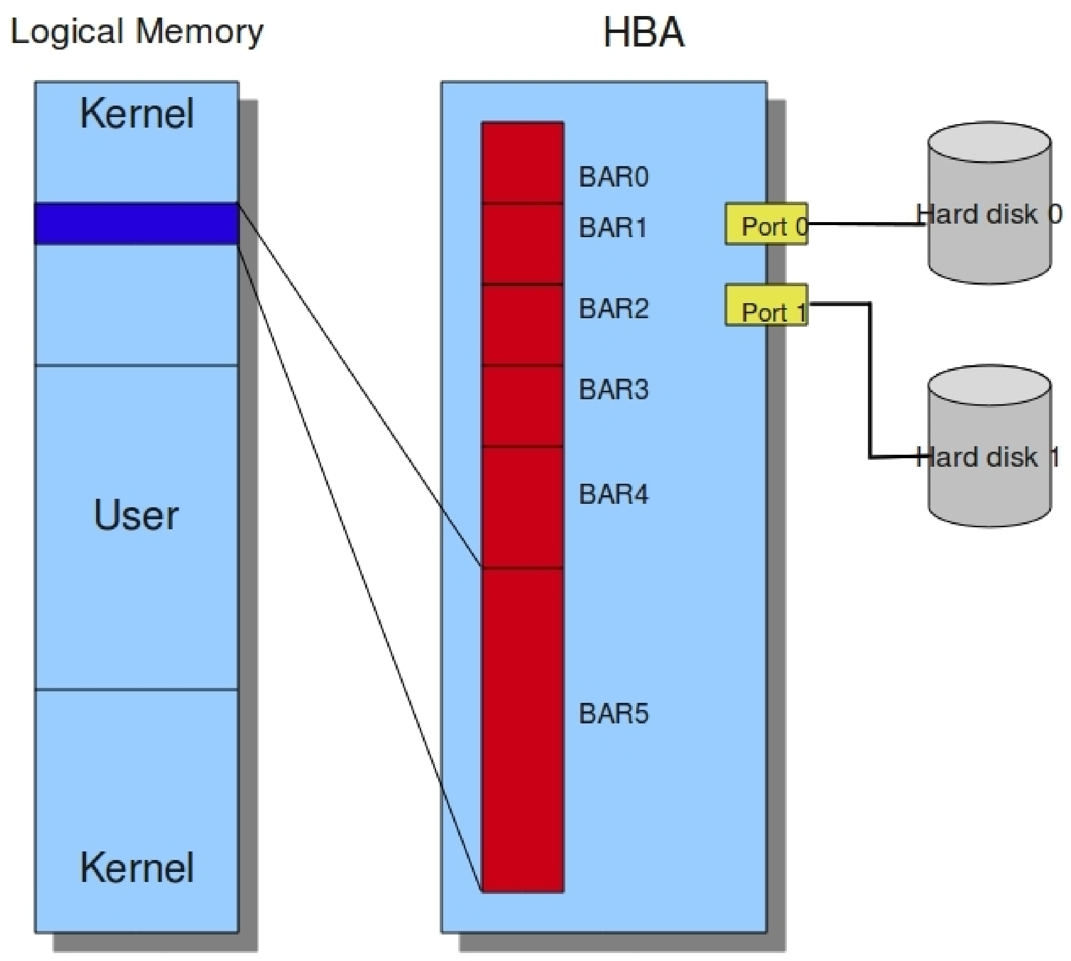


Figure 9 Mapping HBA ABAR register to host memory

In order to control the MDEV, a user space program has been compiled which will provide commands to the MDEV. In addition, the user space program provides data for the MDEV to write to the hard disk as well as maintain a data buffer so data may be read back from the hard disk into user space for verification.

For the MDEV to give commands to the HBA, all AHCI protocol data structures must be setup. The operating system already has the data structures setup; therefore, one way to have the hard disk execute commands is to have them inserted into the operating system's command list. However, in order to avoid potential corruption, the data structures are setup by the MDEV instead. The data structures that will be setup include the command list and the command table. These data structures will be created and stored within a buffer on the MDEV. The next step will be to set the correct values in the Command Frame Information Structure (CFIS), located at offset zero from the command table.

4.3 Experiment: Compromising the Disk Drive Overview

This series of attacks in the experimental portion of this thesis will involve compromising data on the host system's hard disk drive. The confidentiality of the host system's hard disk data will be compromised by performing a read transaction on the host system's hard disk. The integrity of the data on the hard disk drive is compromised by having the MDEV write to the hard disk at some specified location. To compromise the availability of data on the host system, the MDEV again performs write transactions on the hard disk.

In order to compromise the confidentiality of the information on the disk drive, a read transaction is performed by the MDEV. In order to perform a read transaction, the proper values must be placed in the CFIS. One important value that must be placed in the CFIS is the logical block address (LBA) of the data that you want to read. This is also the data structure that the command to 'read' is specified. Once this is done, the MDEV must replace the host system's AHCI port 0 command list pointer with the pointer to the command list built by the MDEV. In addition, the SACT and CI bit of the AHCI register must be set. At this point, the MDEV waits until the disk drive completes the request. Upon command completion the data read from the disk is located in the MDEV buffer specified by the PRD table (part of the command table). At this point, the confidentiality of the information located on the hard disk has been compromised.

To compromise the integrity and availability of the information on the disk drive, a write transaction is performed by the MDEV. As with the read transaction, the correct values must be written to the CFIS of the command table. The queued write command must be specified in the CFIS as well as the LBA of the location on the hard disk that is to be written. In addition, the write bit in the command header must be set. At this point the MDEV must replace the host system's AHCI port 0 command list pointer with the pointer to the command list built by the MDEV. As with the read transaction, the SACT and CI bit of the AHCI register must be set. The MDEV now waits for command completion to occur. Once the command has been completed, the write to the disk drive is a success. It is by this process that writing to corrupt integrity of data and availability of data happens.

Finally, in order to compromise the availability of information on the disk drive, a target write transaction is performed by the MDEV. By having the capability to write to a specified region sector on the disk drive, any information on the disk could be over written. The availability of the information on the target system could be attacked and compromised in various ways. One option is to attack the entire system by over writing critical operating system structures and data stored on the disk. This would cause the entire system to fail, thus making it impossible for authorized entities to get access to the data. A second option would be to compromise the confidential data itself by overwriting it completely. By destroying the data that authorized entities need , the availability of the data is compromised.

4.3.1 Compromise Information Confidentiality

To test if the confidentiality of the hard disk had been compromised using the MDEV, a file of known data was created. This file was then written to disk on the target system via a simple text file. The file was composed of 16K (16,384) bytes of data. The data was composed of 32 512 bytes divisions. Each division was composed of a byte pattern. The first 512 byte section was composed of the pattern "BAAD0000" 64 times, where the last four bytes ("0000") represented the current 512 byte division of data. Therefore, the last 512 byte division was composed of the string "BAAD0031" 64 times. A user application program was written that accessed the physical hard drive directly using the Windows API. This file determined the exact sector number of the data on the disk and reported it back to the user. That sector number was then used to determine the LBA of the data. When a disk is using 48-bit LBA addressing (the target system was using 48-bit LBA addressing), the sector number corresponds directly to the LBA for that sector. That LBA number was then specified in the CFIS for a read transaction. Another user space program was written to act as a controller for the MDEV. That user program provided the MDEV with the command to read the hard disk. When the MDEV executed the read command from the hard disk the data did correspond with what was written in the file. Therefore, the confidentiality of the information located on the specified sector of the hard disk had been compromised.

4.3.2 Compromise Information Integrity

To test if the integrity of the hard disk had been compromised using the MDEV, the same 16K data file from the previous procedure was used. The same user application program that scanned the hard disk for sector numbers of the data was also used. The user space control program setup a buffer with a specific signature to write to the hard disk at the specified LBA. Once the LBA of the sector to write was set in the CFIS, the write transaction was ready to take place. When the MDEV executed the write command to the hard disk, new data was written to the 16K file. This was verified by opening the 16K file with a text editor and viewing the contents of the file where the sector signature of “BAADXXXX” was previously located. The new contents of the file did correspond with that of data buffer from the user space control program.

4.3.3 Compromise Information Availability

In order to test if the information availability of the data on the hard disk had been compromised using the MDEV, a very similar procedure to the compromise of disk integrity was used. The same 16K file was used as the test file for this experiment. The same user application that scanned the hard disk for the sector numbers of the data was used as well. As with the compromise of information integrity experiment, the user space control program setup a buffer with a specific signature to write to the hard disk at a specified LBA. Once the LBA of the sector to write was set in the CFIS, the write transaction was ready to take place. The first write was specified to overwrite data in the test file. This was to show that the system could compromise availability by destroying the data, thus preventing access. When the MDEV executed the write command to the hard disk, new data was written to the 16K file. This was verified in the same way that it was with the previous compromise of integrity experiment. The file was opened with a text editor and viewing the contents of the file where the sector signature of

"BAADXXXX" was previously located. The new content of the file did correspond with that of the data buffer from the user space control program. The second test to determine that the availability of data on the hard disk had been compromised was to destroy the boot sector of the hard disk, thus preventing the system from booting up. This was accomplished by overwriting the hard disk MBR located at sector 0 on the hard disk. The data that was written to that location was 512 bytes all containing the character "A". The contents of the boot sector, when read by the MDEV, were in fact all A's. Upon rebooting the system, the BIOS posted an error message and the system failed to boot into the operating system.

4.3.4 Experiment Metrics

In order to gather measurable data from the experiments described in the previous three experiment sub-sections, two system qualities have been examined and measured. Those three qualities include system stability and system process execution timing. In addition, another metric was gathered based on the success with which the system is able to detect changes made by the device.

System stability was measured by monitoring a select number of processes on the system. These processes have been monitored during the execution of experiments 1 and 2 described in the previous two sub-sections. In addition, these same processes were examined prior to executing commands from the MDEV in order to gather control data with which to compare and contrast the experiment data. System process execution time was monitored by running a large task and measuring its execution time during transactions with the disk drive. The processor intensive task of sorting a million integers was performed during the MDEV’s attacks on the system. The same process was executed prior to running the attacks in order to once again gather control data. Finally, metrics were gathered concerning the success rate with which the host system was able to detect the malicious attacks performed by the MDEV. Intrusion detection software, virus scanning software, and malware detection software have been run from the host system during the execution of these experiments to aid the host in detecting malicious activities.

4.3.5 Experiment Results

In compromising the information security qualities of the disk drive, the attacks presented in this thesis were all successful. The confidentiality, integrity, and availability of the information on the target system disk drive were all compromised. However, target system stability was affected. When performing a disk transaction data must be written to the AHCI memory mapped register of the HBA. The data is written to the port 0 memory range of the AHCI register. The problem occurs when the operating system also attempts to write to the same slot in the register. When this occurs, the operating system’s command times-out while the device services the MDEV’s request. This event is logged by the system as a ‘hard disk command expired’ event. If this event occurs multiple times in a row, the target system will crash. In order to reduce the probability that this will happen, a delay has been inserted into the MDEV’s disk transaction algorithm. Tests were run in order to determine the optimal time delay between MDEV disk transactions in order to maintain system stability and MDEV stealth while maximizing the amount of data that could be compromised. The time delays between MDEV attacks that were tested were 10ms, 100ms, 500ms, 750ms, and 1000ms. The relationship between the MDEV attack time delay and stability of the system maintain an inverse relation. As the time delay was increased, the frequency with which the system crashed was decreased. Impact on system performance had a direct relation to MDEV attack time delay. As the attack time delay was increased, system performance also increased. The graph in Figure 10 shows the results of the tests that were run. Each displayed point at the time delays of 10ms, 100ms, 500ms, 750ms, and 1000ms are an average of 5 for that delay value. The control for these experiments was gathered using just the sort without any running attacks. The control average sort time was calculated to be 8.24023 seconds.

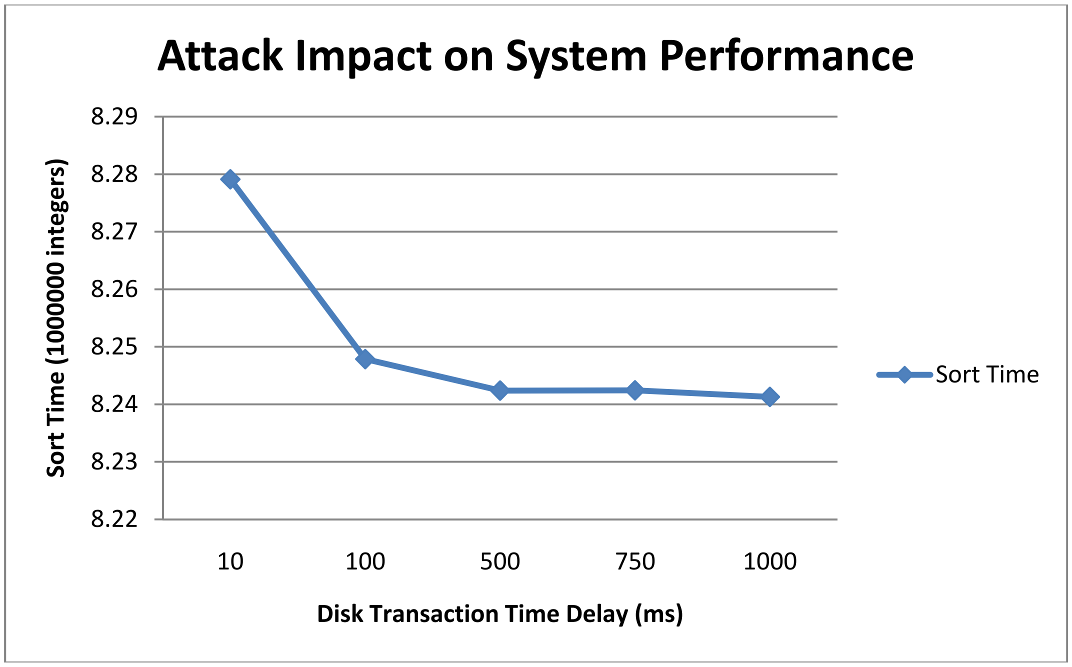


Figure 10   Attack Effect on System Performance

The results here show the time delay between MDEV attacks on the target system, while a processor intensive task is being run. The processor intensive task was sorting a million integers. As the delay between MDEV attacks increases, the time to sort a million integers decreases. This shows the direct relationship between system performance and MDEV attack time delay. The data in Figure 11 show the percent system performance degradation as the MDEV attack time delay is increased.

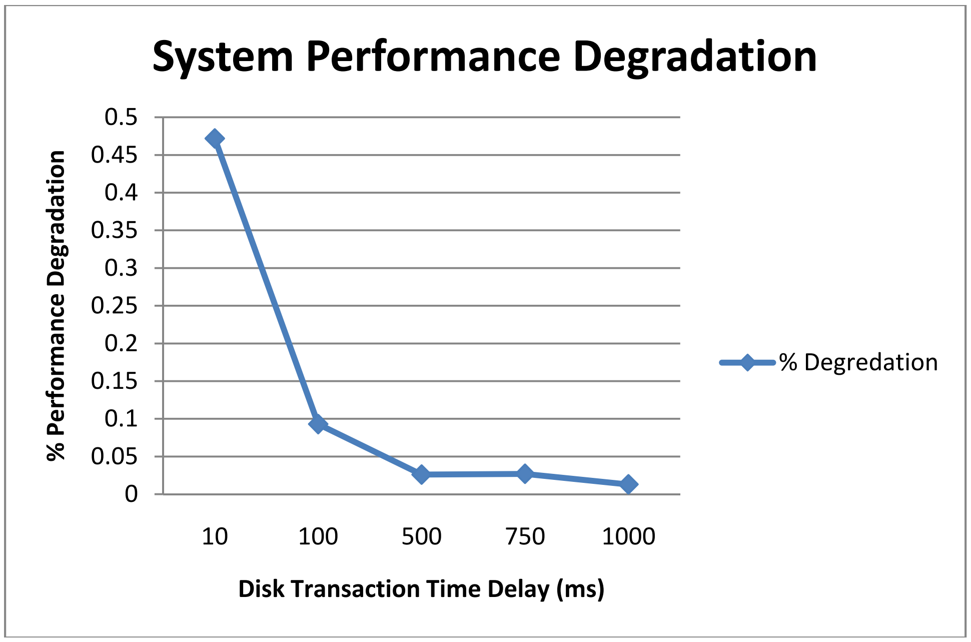


Figure 11   System Performance Degradation

Here the data shows that as the MDEV attack time delay increases, the percent degradation of system performance decreases.

CHAPTER V

CONCLUSIONS

In the hypothesis there were multiple claims as to the outcome of the experimental portion of this thesis. Those claims were as follows:

1. A malicious device (MDEV) will be able to access disk drives on a host computer
2. Disk accesses by the malicious device will have no discernable impact on the stability of the host system
3. Disk accesses will have less than five percent degradation of system performance
4. The disk accesses by the malicious device will not be detectable by intrusion detection/prevention software or by the host system itself

Claim number one was proven to be true, as the confidentiality, integrity, and availability of data on the target system hard disk were all compromised. Claim number two was not entirely correct. In fact, the attacks on the system's disk drive did have an impact on the stability of the host system. When a conflict occurs between the operating system's disk driver and the MDEV over which process will write to a command slot in the AHCI register of the HBA, the system could potentially crash altogether. To compensate for this fact, a time delay between attacks on the host system's hard disk was inserted into the MDEV's attack algorithm. Tests show that as the time delay increases the system has a smaller probability of failing. The most optimal attack time delay to balance the minimizing of system failure probability and the amount of data that is compromised was determined to be 500 ms. The 500 ms delay has only a 0.026% system performance degradation (see Figure 11) and maintains a very close probability of system failure to the 1000 ms delay. The following formula calculates the amount of time it takes to compromise 100GB of data on the hard disk given any MDEV attack time delay.

$$Time(Rate) = \frac{Quantity}{Rate} \tag{Eq 1}$$

Using this formula, the time to compromise 100GB of data using the optimal MDEV attack delay of 500 ms can be calculated as follows:

$$Time\left(\frac{512\ Bytes}{500\ ms}\right) = \frac{100\ GB}{512\ Bytes/500\ ms}$$

$$Time = 97656250000\ ms$$

$$Time = 3.14\ years$$

In spite of the apparent long time it takes to compromise the data, an attack by a real world entity would be more targeted, with the purpose of retrieving specific data. In this case the quality of the information would be very high while the size of the data containing that information could be low, thus decreasing the amount of data that needed to be compromised.

Claim number three of the hypothesis states that the attacks on the disk drive will have less than a five percent degradation of system performance. This statement was proven to be true. This statement encompasses the system performance with regards to processor and memory usage. It does not take into account the performance of the system during a disk I/O intensive task. Figure 11 shows the system degradation during the processor intensive task of sorting a million integers. It is apparent that as the MDEV attack time delay is increased, the system performance gets closer to the control. No average system degradation displayed in Figure 11 is greater than even one percent, with the highest system degradation recorded being only 0.472 percent.

Finally, claim number four of the hypothesis states that attacks by the MDEV will not be detectable by intrusion detection/prevention software or by the host system itself. The host system ran Symantec Endpoint intrusion detection/prevention software. This software was also an anti-virus software as well. At no point during any of the test runs did Symantec Endpoint detect the MDEV attack activity. When the operating system device driver and the MDEV both attempted to write to command slot 0 in the AHCI register, the operating system would log the event as a ‘hard disk command expired’. This log event does not in any way point to the MDEV or its activity. The log simply records the fact that there was a timeout with one of the operating system’s disk transactions.

REFERENCES

1. Li, Qingbao, Gao, Hongbo and Xu, Bing**, "** Hardware Threat: the Challenge of Information Security**",** *Proceedings: International Symposium on Computer Science and Computational Technology,* Shanghai , 2008, IEEE Computer Society, pp. 517 - 520.

2. Arbaugh, William A., Farber, David J. and Smith, Jonathan M, " A Secure and Reliable Bootstrap Architecture", *Proceedings IEEE Symposium on Security and Privacy*, Oakland, 1997, IEEE Computer Society, pp. 65- 71.

3. Hendricks, James and van Doorn, Leendert, " Secure Bootstrap Is Not Enough: Shoring up the Trusted Computing Base ", *Proceedings: ACM SIGOPS European Workshop,* Leuven, Belgium, 2004, ACM.

4. Weiss, Aaron, "Trusted Computing: Will the Open, Unrestricted PC Soon Become a Thing of the Past", *Journal of Trusted Computing, September 2006, pp. 18-25.*

5. Kursawe, Klaus, Schellekens, Dries and Preneel, Bart, "Analyzing Trusted Platform Communication", Computer Security and Industrial Cryptography, https://www.cosic.esat.kuleuven.be/publications/article-591.pdf (current 22 March 2010).

6. Eisenbarth, Thomas, Guneysu, Tim, Paar, Christof, Sadeghi, Ahmad-Reza, Schellekens, Dries, Wolf, Marko, "Reconfigurable Trusted Computing in Hardware", *Proceedings: Conference on Computer and Communications Security,* Alexandria, Virginia, 2007, pp. 15 - 20.

7. Trusted Computing Group, " Developers Trusted Platform Module Specifications ", *TPM Main Specification*, http://www.trustedcomputinggroup.org/developers/trusted_platform_module/specifications (current 22 March 2010).

8. Corbet, Jonathan, Rubini, Alessandro and Kroah-Hartman, Greg, *Linux Device Drivers,* O'Reilly Media, Sebastopol, CA, 2005.

9. Budruk, Ravi, Anderson, Don and Shanley, Tom, *PCI Express System Architecture,* Mindshare Inc., 2004.

10. Bishop, Matt, *Introduction to Computer Security*, Addison-Wesley, Boston, MA , 2005.

11. Ray, Baishakhi, Mishra, Shivakant, "Secure and Reliable Covert Channel", *Proceedings: 4th annual workshop on Cyber security and information intelligence research: developing strategies to meet the cyber security and information intelligence challenges ahead*, Oak Ridge, Tennessee, 2008, ACM.

12. Selke, Sarah, Wang, Chih-Chun, Bagchi, Saurabh, Shroff, Ness, “TCP/IP Timing Channels: Theory to Implementation”, Rio de Janeiro, 2009, IEEE Computer Society.

13. Boyd, James, "Serial ATA Advanced Host Controller Interface (AHCI) 1.3", *Intel: Serial ATA*, http://www.intel.com/technology/serialata/pdf/rev1_0.pdf (current 22 March 2010).

14. Oney, Walter, *Programming the Microsoft Windows Driver Model*, Microsoft Press, Redmond, Washington, 1999.

15. Intel Corporation, “Serial ATA II Native Command Queuing Overview: Application Note”, Intel, http://www.intel.com/assets/pdf/whitepaper/252664.pdf (current 22 March 2010).