Web Intelligence and Agent Systems: An International Journal 8 (2010) 203–217
DOI 10.3233/WIA-2010-0187
IOS Press

Information security underlying transparent computing: Impacts, visions and challenges

Yaoxue Zhang [a,*], Laurence T. Yang [b], Yuezhi Zhou [a] and Wenyuan Kuang [a]
[a] *Key Laboratory of Pervasive Computing, Ministry of Education*
Tsinghua National Laboratory for Information Science and Technology,
Department of Computer Science and Technology, Tsinghua University, Beijing 100084, P. R. China
E-mail: {zhangyx,zhouyz}@mail.tsinghua.edu.cn
[b] *Department of Computer Science, St. Francis Xavier University, Antigonish, NS, B2G 2W5, Canada*
E-mail: ltyang@stfx.ca

Abstract. The rapid development of computer network technologies and social informationalization has brought many new opportunities and challenges in information security. With improved information and service sharing enjoyed by more and more people, how to strengthen the information security has become an increasingly critical issue. In this paper, we propose a new network security mechanism based on a novel computing paradigm, i.e., transparent computing, which is based on the extended von Neumann architecture. This paradigm separates the program storage and execution, which is implemented in the network environment. It is realized by a new generation server and client BIOS, namely EFI BIOS, and coordinated with the MetaOS management platform and related switching and input/ouput devices of transparent computing. Through the interface between hardware and software, it conducts effective control, monitoring and management of data and instructions in a block-streaming way for the operating system and the application programs above it. At the same time, it adopts a security protection mechanism to prevent and remove prevalent malicious software such as worm and Trojan horse. Several demonstrated examples are described in detail to illustrate the attractive and promising security features and advantages.

Keywords: Transparent computing, extended von Neumann architecture, ubiquitous and pervasive services, Meta OS, service sharing

1. Introduction

In recent years, because of the rapid developments on computer network technologies and social informationalization, the security of computer information systems is becoming more and more important. Traditional ways mainly focus on such methods and techniques as encryption, decryptions or other enhancement strategies to insure the security of either the transmitted content or the smooth execution of operating systems and their application programs. Nowadays the scope of the security issue bas been extended ranging from physical network transmission, computer system execution through to massive data and content storage, etc., and correspondingly, various attack and defense mechanisms and techniques such as information masquerade, cheating, penetration and prorogation, disclosure, destruction, auditing and monitoring have been used widely in different computer and information networking systems.

Undoubtedly, these new mechanisms and techniques have significantly positive impacts on computer and network security. However, as users' demands and expectations on computer and networks are growing drastically, the computer systems have become more and more large and complicated. For example, in ubiquitous or pervasive environments, hardware and software are becoming more complex, harder to maintain and manage, and needs more frequent system upgrades. This may well lead to even more security holes

*Corresponding author. E-mail: zyx@moe.edu.cn.

in these systems. Therefore, it is more difficult to ensure the high security of the corresponding computer and network systems.

Currently the security issues can be classified, from the behavioral point of view, into the following categories [3,6,30]:

- Destructive attack and defense: this refers to attacking and counter-attacking behaviors via viruses, modifying data in transition or denial of service, and so on, aiming to interfere, destruct the victim computer and information systems.
- Information-based attack and defense: this refers to those behaviors via eavesdropping and monitoring traffic flow and data in transition, guessing secret keys and passwords, masquerading one entity as another, as well as other misconduct to obtain unauthorized data or resources from the host computer and information systems.
- Content analysis and filtering: this refers to such behaviors as identifying, retrieving and selecting massive Internet data to get useful information, prevent violent and pornographic contents, and protect intellectual property.
- Misplay protection: this prevents those authorized or legitimate users from operating without appropriate protocols and rules, which may bring about harmful effects to the system and other users, and leaving the system vulnerable to outside attackers.
- Information leakage protection: this prevents any internal system administrator and operator from disclosing any unauthorized or classified information by using networks, laptops and removable hard drives and other storage devices. Currently such counter measures mainly reply on non-technical law and management rules.

Correspondingly, the information security systems can be roughly divided into the following categories:

- Specific information security systems: examples include anti-virus software [32], firewalls [2,10], Virtual Private Networks (VPN) [7], various encryption systems and key management techniques [24], etc.
- Biometric identification systems [18,19]: in these systems user ID and privileges must be verified before her/his accessing designated information and resources.
- Trusted computer systems [33–35]: refer to those systems that, starting from the hardware and BIOS levels, are constructed by a trusted chain with user authorization and authentication.
- Content discrimination and filtering systems [21, 36]: refer to those systems that, on the application level, discriminate and filter all users' visited content to ensure them a suitable and safe access and protect the intellectual property right concerned.
- Secure operating systems [13]: starting from the classification on the data and information access control, these systems conduct tight monitoring and protection on the reading and writing operation on data and instructions, as well as I/O requests and interrupt handling, etc.

Originally designed for stand-alone machines, all the above security systems have one or another system architectural flaw. For example, it is very difficult to prevent information disclosure, for in networking environments, information is stored in a distributed manner, and basically all users on the client terminals may have their own hard disks and removable storage devices. They can easily access all classified information and leak them out; due to the lack of anti-virus and protection capabilities, attackers can easily inject various viruses and malicious software such as Trojan horses and Zombie into the victim servers and client operating systems via network. This would lead to many serious problems, like massive system crashes and information disclosure and leakage. A major reason for this is that the system architectures are designed without a network-based perspective. As the number of attacking methods and techniques significantly increases with the rapid development of computer network, the defense or counter-attacking measures still use those outdated ones intended for stand-alone machines lacking network-based considerations.

A new computing paradigm based on a network-based perspective, namely, Transparent Computing [38], has been proposed to address such problem. The core idea of this paradigm is to extend von Neumann's architecture based on the "stored program concept" into networking environments spatio-temporally. In this extended model, execution and storage of programs, including operating system and applications, are separately done in different computers: with the former performed in the clients and the latter, in the servers. Specifically, system/service programs are stored on the central servers, and fetched on demand in a block-streaming way instead of downloaded all at once, and automatically initiated/executed on the embedded devices or client systems with local CPU

and memory resources. Users can get different (or the same) services, including commodity OSes and applications, via the same (or different) embedded device(s) or client system(s). Furthermore users don't need to do or even care about the installation, maintenance and management of services. This computing paradigm totally changes the traditional system architectural basis, thus naturally provides extended mechanism to enhance the security and trust features of corresponding computer systems.

This paper focuses on the security enhancing mechanism based on Transparent Computing and demonstrates that this new computing paradigm can provide a potentially new approach to securer computer systems. The rest of this paper is organized as follows. We first present a short overview of Transparent Computing, including its concept, architecture and related implementation in Section 2. In Section 3, we introduce several secure enhance mechanisms based on the Transparent Computing, which lay the foundation for establishing securer information systems. Section 4 gives some examples we have implemented to illustrate the effectiveness in terms of system security of the Transparent Computing. We also discuss some related works that have been done to tackle the problems of information security in Section 5. Finally, we summarize our efforts and discuss the future work in Section 6.

2. Transparent computing

Transparent Computing [38] is a new pervasive/ubiquitous paradigm to realize users' service sharing. Based on the same philosophy of pervasive/ubiquitous computing, transparent computing involves a kind of network thinking. As illustrated in Fig. 1 and Fig. 2, service sharing means users can get the same services through different embedded devices, hardware platforms (such as MID, LID and digital home appliances) or computing devices (such as personal computers); or they can access different services through the same embedded device or platform or computing device. In order to support service sharing, the client systems must be able to support, run and display various operating systems and their associated software.

In addition to providing a new paradigm on client systems to support and display multiple operating systems, virtual machine technology [20,29,37] also incurs additional performance overhead due to its virtualization of all hardware resources, including CPU and memory. Therefore it demands to be executed on high performance hardware computing platform. Such virtualization solution cannot be efficiently applied to ubiquitous or pervasive applications since most of the clients are embedded in systems or devices which are not capable to run the host operating system to support other operating systems and their applications.

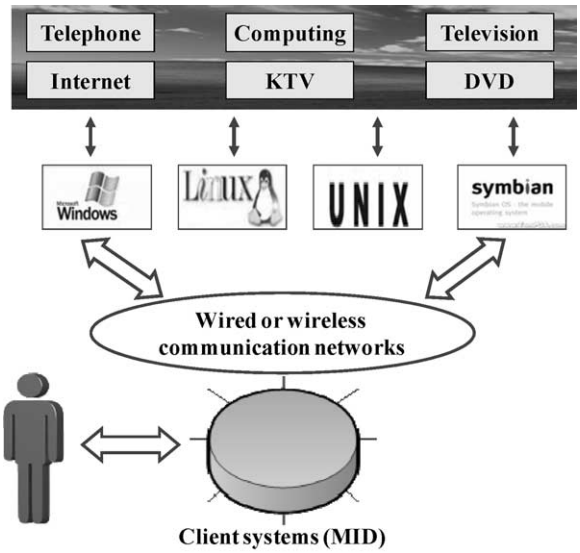

Fig. 1. Different services on the same hardware platform.

The essential idea of transparent computing paradigm is to extend the von Neumann "stored program concept" architecture into the networking environments spatio-temporally. In this extension, computation and storage of programs are separated in different computers. Specifically, the system/service programs (no matter which OS or application program) are stored on the central servers (like a warehouse), to be fetched on demand if needed, and automatically initiated/executed on embedded devices or client systems with local CPU and memory resources (like a factory). When users need the services but cannot access them through the client system, it will send interrupt and I/O requests through the network to the servers where all necessary data and instructions are stored. The server(s) will handle all corresponding interrupt and I/O requests accordingly, then send the data and instructions back in a block-streaming way to the client system for execution.

The basic idea of the separation of storage and execution in the transparent computing paradigm is illustrated in Fig. 3. In order to realize such separation via networks, the following two fundamental problems

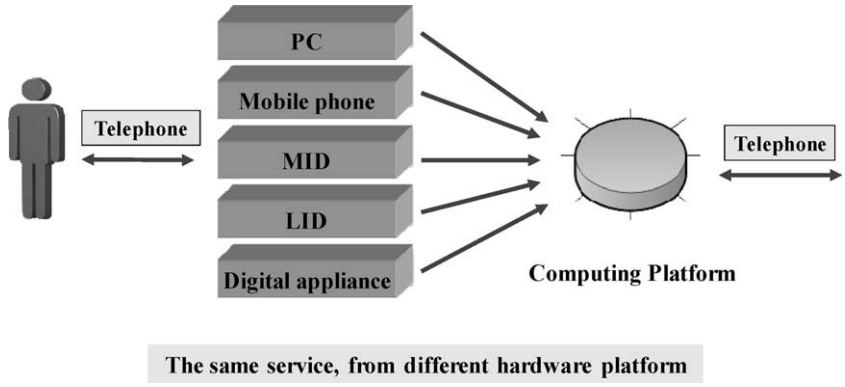

Fig. 2. The same services on different hardware platforms.

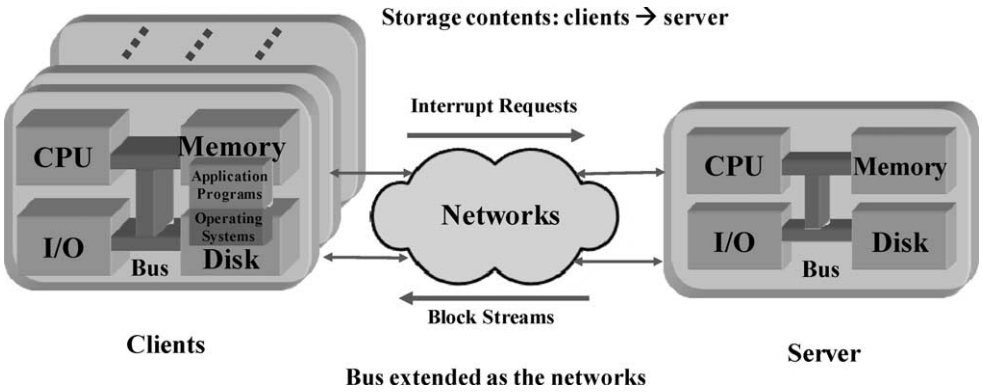

Fig. 3. An extended von Neumann architecture.

should be solved. Firstly, how to manage and distribute the data and instructions in the server, and secondly, how to solve the compatibility issue between different client system hardware and different operating systems supplied by the server. The first problem can be translated into how to ensure all necessary data and instructions to be sent to client systems on time, and the second, how all data and instructions sent to client systems can be executed efficiently. An interface between Meta OS and hardware/software such as EFI (Extensible Firmware Interface) [12] has been proposed to address the above two problems in [38]. Correspondingly, a transparent client and server system called 4VP$^+$ supporting both Windows and Linux has been successfully implemented [25] and widely used and deployed for daily use in such fields as education, industrial and electronic government in China.

Meta OS [38] is a super operating system to control and manage different operating systems. It is distributed in every server and client system, and resides above the BIOS level and beneath different operating systems (called "Instance OS", such as Windows and Linux). In order to realize the separation of execution and storage, the Meta OS should have two basic functions. One is to enable users to select an operating system should be loaded and used, and then boot the needed OS remotely from server repositories. The other is to schedule programs demanded by users in a streaming way during or after the booting of operating system. This is vital because local storages are lack in the client system. The structure of Meta OS is shown in Fig. 4, which consists of four virtual views of I/O, disk, file and users, and two protocols of MRBP (Multi-OS Remote Boot Protocol) and NSAP (Network Service Accessing Protocol).

The implementation of Meta OS is through a 4VP$^+$ (shortly for four virtual layers and two protocols) distributed platform, which partially operates at the assembler instruction level. This 4VP$^+$ software platform is mostly installed in a management server, except for a part of MRBP that is burned into a BIOS EEPROM in the bare client. However, the other programs of 4VP$^+$ platform run in clients systems or server(s) according to their specific functions. Those parts of 4VP$^+$ run-

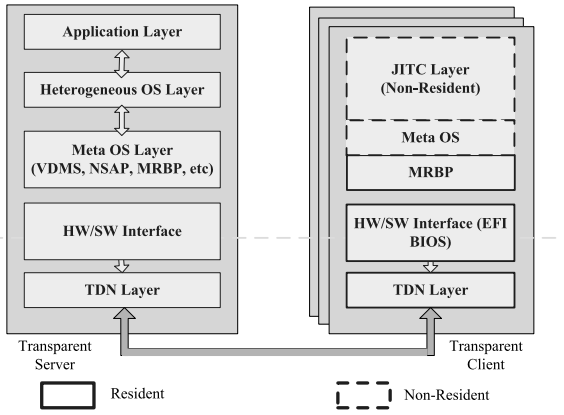

Fig. 4. Layered architecture of transparent computing.

ning in the client systems will be also fetched from the server repository along with Instance OSes.

The MRBP is used to boot bare clients remotely from servers. It enables users to select their desired OSes and applications and then installs an NSAP (Network Service Access Protocol) module which is written in assembler instructions to enable a virtual I/O for clients. Through this virtual I/O interface, clients can access the OS images located at servers and then load them as if with a regular I/O device. After the dynamically scheduled OS is started up, the OS-specific in-kernel modules for Virtual I/O Management (VIOM), Virtual Disk Management (VDM), Virtual File Management (VFM) and Virtual User Management (VUM) will function to help the users to access the software and carry out their tasks seamlessly. The above 4 modules can be further elaborated as follows:

- Virtual I/O Management (VIOM): it receives interrupt and I/O requests of user processes; analyzes the reasons of interrupts, then wakes up the interrupt handling routines and responds accordingly; analyzes the reasons of I/O requests, then allocates devices and buffers, starts I/O operations. In the transparent computing paradigm, the buffer queues are different from traditional paradigms due to the network extension, and interrupts and I/O requests are much more complex due to the use of network devices. The interrupts and I/O handling between clients and servers have to be synchronized and mutually exclusive.
- Virtual Disk Management (VDM): this module functions for the allocation, collection, and driving of virtual disks (V-disks), and virtual swapping and scheduling of programs and data streams between client systems and the server(s). V-disks are flat addressable block-based virtual storage devices located beneath file systems. A transparent client can be configured to access data from one or more V-disks, with each corresponding to a V-disk image located on the server. There are 4 categories of V-disks: System V-disk (S), used to store the "golden image" of OS and applications; Shadow V-disk (H), a user-specific COW disk of the system V-disk to enable write access to the system V-disk content; Profile V-disk (P), a profile V-disk to store user-specific persistent data such as customized user settings for OS and applications for each user; and User V-disk (U), used to store private user data.
- Virtual File Management (VFM): this module functions for the allocation and management of file space and directory, as well as file redirecting, access, control and retrieval; and to ensure the file consistency.
- Virtual User Management (VUM): this module functions for the configuration and management of user profiles, management of users' addresses and processes as well as addition and removal of users.

The Extensible Firmware Interface (EFI) [12], later as Unified Extensible Firmware Interface (UEFI), is a specification that defines a software interface between an operating system and platform firmware. EFI/UEFI is developed as a significantly improved version of the old legacy BIOS firmware interface historically used by all IBM PC compatible personal computers. It works to standardize two primary functions of the PC Basic Input/Output System (BIOS): firmware-to-OS interface and platform initialization. It consists of data tables that contain platform-related information, plus boot and runtime service calls that are available to the operating system and its loader. EFI/UEFI provides a clean and stable interface between operating systems and the platform at boot time and supports an architecture-independent mechanism for initializing add-ins.

As illustrated in Fig. 5, with the standardization of interfaces between operating systems and platform or embedded system firmware, it is anticipated that, in the transparent computing paradigm, various operating systems and applications will be able to run and be shared on different architectures, platforms or embedded systems, so as to accelerate the evolution of innovative and differentiated system designs.

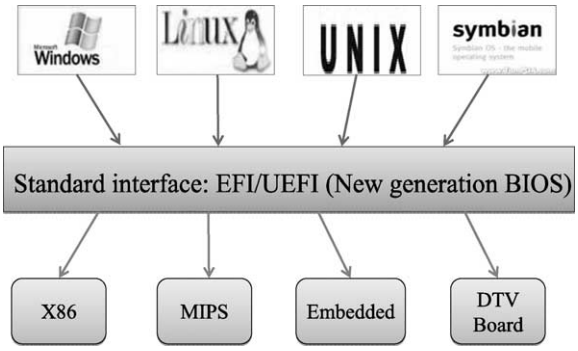

Fig. 5. Architecture-independent extensions with EFI/UEFI.

3. Impacts on information security

Because the transparent computing paradigm separates the storage of program instructions and data from execution via network, it not only makes possible the central management of data and instruction, but enables different operating systems to run on the same hardware platform in a harmonic way, which reduces the hardware and software complexity, costs and management difficulty. Next we will discuss and analyze the impact on information security from the viewpoints of system architecture, virus, information disclosure and leakage.

3.1. Secure system architecture

Traditional methods to ensure the security of computer systems mainly focused on the secure operating systems, identification authorization, authentication and monitoring, firewalls and gateways, various anti-virus and scanning systems, encryption and decryption, etc. In recent years, people start to think about building the trusted chain from BIOS, based on which to construct trusted computing systems. As information systems become increasingly complex and networked, the traditional security framework of system architectures and techniques are facing even more daunting challenges. For example, secure operating systems are considered a promising solution, and thus a BLP security model has been proposed to classify different security levels for different operating systems, but this solution still cannot solve all problems currently in front of us. For those users who do not have the source codes of secure operating systems, in particular, they cannot make any further changes and updates to enhance the security of their own systems. Additionally, in the traditional von Neumann architecture, all reading and writing operations on file blocks are done through system bus, disk file storages, data exchange areas or data blocks in the buffer queues. If users cannot monitor the reading and writing operations on files, attackers can easily by-pass the reading and writing protection mechanism, and cause destructive damages on users' files and the entire computer system.

The transparent computing paradigm provides a new and more secure system architecture to overcome the above issues. It extends the traditional stored program concept to networked/ubiquitous computers or devices spatio-temporally. Specifically, the storage and execution of programs are separated between different computers. Applications and operating systems are stored on the central computers (servers) instead of local storage devices in traditional architectures. Therefore all reading and writing operations on data and instruction blocks should go through the network buffers. This makes it possible for system administrators to monitor and control, via network switches and gateways, all data and instruction block streams including operating systems, without having to rely on the secure operating system entirely. This will shift the focus of secure system architecture from secure operating systems and their above applications to BIOS level and its above data and instruction streams between clients and servers. Since the BIOS is the level closest to hardware, and is relatively less complex compared with secure operating systems, such an approach can much easily solve all related challenges and difficulties we are facing.

3.2. The anti-virus mechanism

Typically most viruses start to spread and attack computer systems by one of the following two ways: changing the boot sector or modifying data during the reading and writing operations. Transparent computing provides an effective defense mechanism to address these issues, which is also realized by either of the two methods: adding a control and protection area in the Instance OS's boot sector [25], or changing the reading and writing operations on data and instruction blocks to prevent the virus attacks. Firstly, for those viruses trying to destroy the boot sector, this new paradigm can set up various control and protection parameters at the BIOS and EFI interface to prevent the booting sector from being infected and damaged. Figure 6 gives a detailed illustration on the struc-

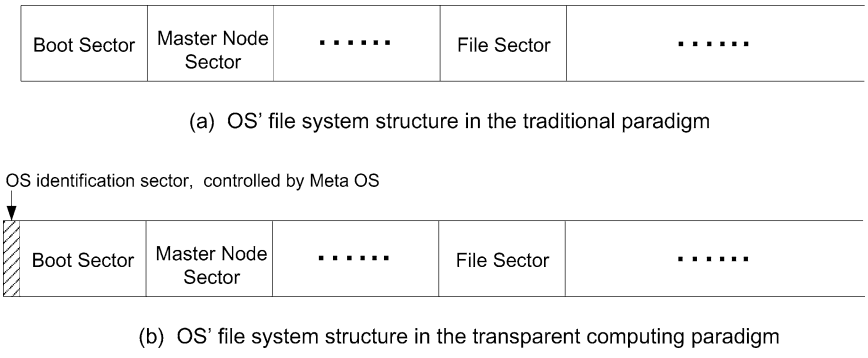

Fig. 6. OS file system structural changes.

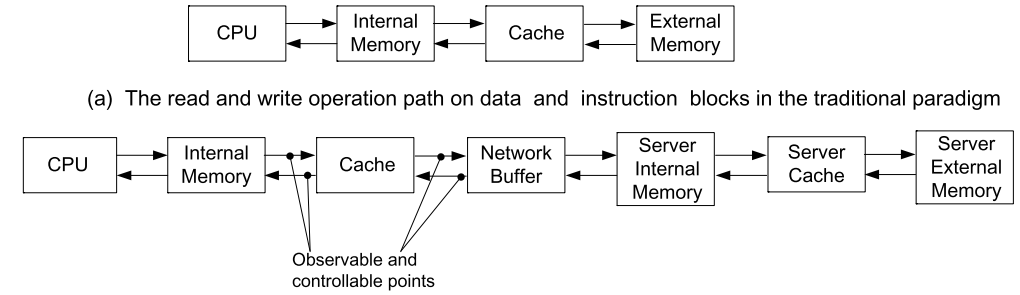

Fig. 7. Changes of the reading and writing operation path.

tural changes of the OS file system. The control and protection areas have the following advantages:

- After receiving the initial interrupt signals such as INT 13 signal from MRBP (Multi-Remote Booting Protocol), the Meta OS can retrieve and identify the corresponding Instance OS from the server.
- Through the EFI interface, some related control and protection parameters can be adjusted to effectively prevent the relevant viruses from rewriting and destroying the boot sector.

A detailed example on how to prevent the viruses from attacking the booting sector will be given in Section 4.

Further, this new computing paradigm changes the way of reading and writing file in Instance OSes. In Instance OSes, the reading and writing mechanism can be divided into 3 types: synchronous reading and writing, asynchronous reading and writing, and delayed reading and writing. All these three types are implemented and used among memory, cache and hard disk of the von Neumann architecture as illustrated in Fig. 7(a). In the transparent computing paradigm, the reading and writing mechanism is totally different. Since the client system does not have the hard disk (or the virtual disk image in the server), when an Instance OS reads or writes (by either synchronous, asynchronous or delayed ones) on the hard disk, the reading and writing path will be extended like the one depicted in Fig. 7(b). Correspondingly, Meta OS will revise the reading and writing program to adapt them to the above mentioned new reading and writing path of the transparent computing paradigm. Obviously, all viruses involving the reading and writing operations on data and instructions will fail to activate.

Furthermore, since all reading and writing operations in Instance OS on data and instructions become visible, once anything unusual happens during the process, the system administrator can adjust the control and protection parameters immediately to stop and prevent any further infection. A detailed example on how to prevent the read- and write-related viruses will be described in Section 4.

3.3. Information leakage prevention

The largest advantage of the transparent computing paradigm in terms of information security lies in its

effectiveness in information leakage prevention due to its central management feature. For example, since the client systems do not store any data, users cannot directly copy any data and instruction. If users want to access the server to get the relevant data and instructions, the transparent computing system can monitor and record the entire accessing process via Meta OS. If necessary, it can stop all illegal access requests. Since the client systems do not contain any of their own data and instructions, they are totally useless if they are not connected with the network and managed by the Meta OS. Therefore, this computing paradigm allows all information centrally stored, monitored and managed, so as to effectively avoid and prevent any information leakage.

3.4. Ubisafe computing

With the rapid advance of information technology and the spread of information services, the IT disparity between different groups of people in terms of age, social standing, and race has been expanding and has become a critical social problem of the 21st century. Ubisafe [23] is a novel and inclusive paradigm proposed to study and provide possible solutions with a unified methodology to satisfy the needs of people in any situation, any place and any time. The ultimate goal is to build a computing environment in which all people and organizations can benefit from ubiquitous services anytime anywhere with satisfaction, without worrying or thinking about safety. It covers such issues as reliability, security, privacy, persistency, trust, risk, uncontrollability, and other watchfulness while considering the novel, essential ubiquitous and pervasive features of unobtrusive computers, diverse users/people and life-like systems. For example, since so many sensors and embedded systems are deployed in our daily life, several wailful accidents, due to the diverse and inappropriate software designs and other reasons, have been reported, such as a boy got stuck to death by the revolving hotel door, the temperature of the programmed closestool is too high, etc., which may lead to very negative attitude towards new technologies among people. The traditional way to deal with these tragic problems is to re-design the corresponding software systems, but it is very expensive. The updates and upgrades on those sensors and embedded systems may not only result in tremendous economic loss, but also lead to further safety and security problems; in the worst case, the whole system may have to be rebuilt from the scratch.

Transparent computing provides a new solution to update and upgrade these embedded systems and sensors effectively. When system administrators find any missing functionalities and technical flaws, they can get the latest and revised resources from the network to get the updating and upgrading done. This is a simpler, safer and more feasible approach to realize a Ubisafe computing environment.

4. Demonstrated examples

The current prevalence of Internet renders various malicious software one of the major threats to computer systems. Early computer viruses propagated and spread by embedding harmful codes in the executable program or hard disk's system sector, which were set to "explode" when certain conditions were met. From the mid-1990s, macro viruses has become the most prevalent type of virus, and they are particularly threatening because they are platform independent, infecting documents instead of executable codes, and are easily spread. Macro viruses take advantage of the macro feature found in Word and other applications. Besides self-replication, many viruses carry function codes to send copies from computer to computer across network connections. Upon arrival, they may be activated to replicate and propagate again, and usually secretly perform disruptive or destructive activities without users' awareness.

Currently there are more than one hundred thousands active malicious software, including around 6–7 thousands major ones. There has been a constant arms race between virus writers and antivirus software programmers, with the most significant types of viruses being:

- Boot sector virus: it infects a master boot record and spreads when a system is booted from the disk containing the virus.
- Parasitic virus: this is the most traditional and still most common form of virus, attaching itself to executable files and replicates when the infected program is executed.
- Macro virus: a prevalent type of viruses, which is particularly threatening since they are platform independent, infecting documents instead of executable code, and are easily spread.
- Memory-resident virus: it lodges in the main memory as part of a resident system program, and infects every executed program.

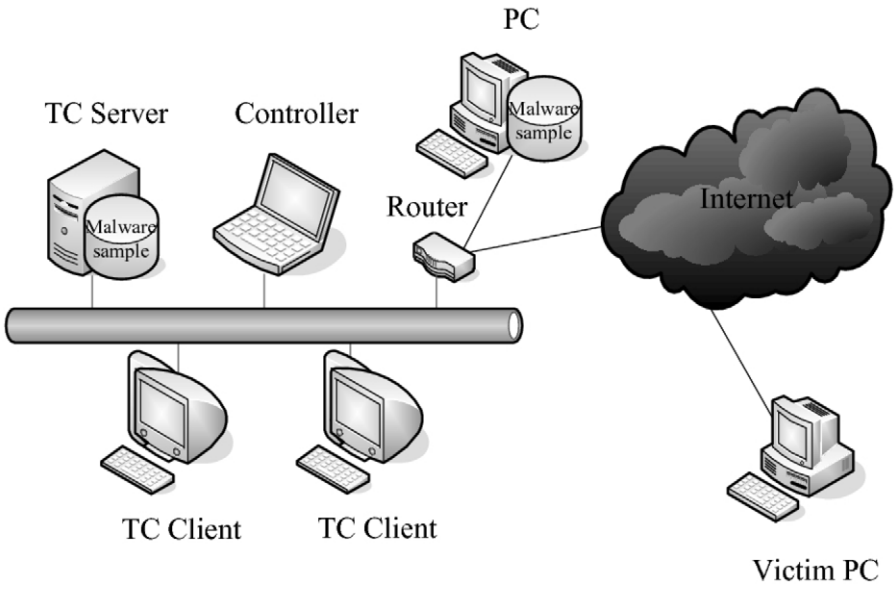

Fig. 8. System configuration for demonstrated examples.

- Trojan horse: it is a useful, or seemingly useful, program or command procedure (e.g., game, utility, software upgrade, etc.) containing hidden codes that perform some unwanted or harmful functions. These functions could not be accomplished directly by an unauthorized user.
- Worm: a program that can replicate itself and send copies from computer to computer across network connections. It actively seeks out more machines to infect and each infected machine serves as an automatic launching pad for attacks on other machines. To replicate itself, a network worm uses a kind of network vehicle such as email, remote execution, or remote login. Once activated within a system, a network worm can behave as a computer virus or bacteria, or it could implant Trojan horse programs or perform any harmful and unwanted actions.

In order to verify the security protection mechanism in the transparent computing paradigm, we have conducted some preliminary experiments on a transparent computing system connected by LAN, as illustrated in Fig. 8. The experimental system consists of the experimental controller, the malicious software sample depository, the victims in the transparent computing system and the PC system. From the propagation point of view, the malicious software can be divided into passive and active ones. Passive software attempts to learn or make use of information from the system but does not affect system resources. The aim of eavesdropping or monitoring of data transmission is to obtain message content, or monitor traffic flows. Such software is difficult to detect because it does not involve any alteration of the data. For example, W32.Sasser.Worm is such a worm, which exploits the vulnerability of MS Window LSASS buffer. Active software attempts to alter system resources or affect their operation. By modification of the data stream, it masquerades one entity as some other one, replays previous messages, modifies messages in transiting or involves denial of services. The characteristics of active software are opposite to those of passive ones. Whereas passive software is difficult to detect, measures are available to prevent their successful attacking. On the other hand, it is quite difficult to prevent active ones absolutely, owing to the potential presence of physical, software, and network vulnerabilities. So that in the latter case, the goal is to detect and recover from any disruption or delays caused by them.

We have set up two passive malicious software sample depositories on a transparent computing system server and a PC system, respectively, and an active one on a victim PC system outside the LAN. The passive sample depositories contain file viruses, macro viruses such as Mellisa, boot sector virus such as Polyboot, Trojan horse viruses including info-stealing trojan Netchief and disk killer trojan HDBreaker, and so on. The active one includes different worm viruses such as W32.Sasser.Worm and mass-mailing worm

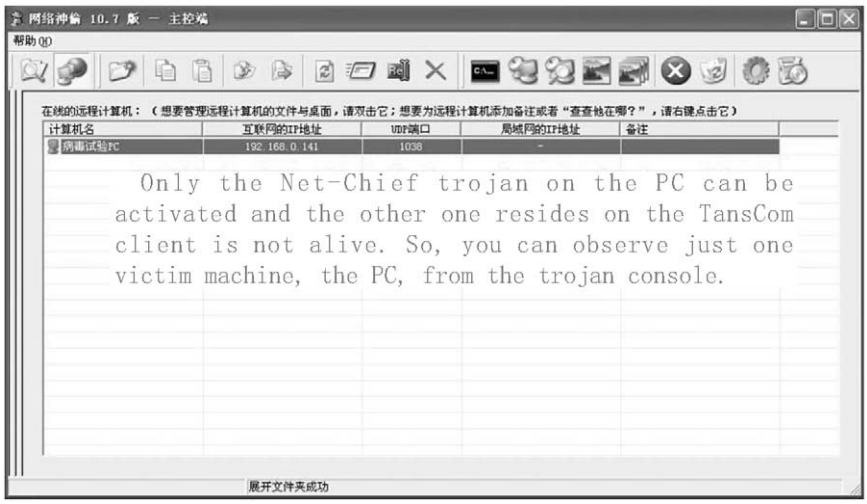

Fig. 9. Example of info-stealing Trojan on the PC and the transparent computing system.

W32.Netsky.C. First we install a virus control program into the experimental controller, where it communicates via a customized lightweight protocol to instruct the activation programs in both the transparent client and the PC respectively to initiate the malicious software. We then compare how the viruses infect both machines. The traditional PC is either disk destroyed or totally infected, but the transparent client in transparent computing system runs normally, and even it is infected, it will run normally after reboot. This clearly demonstrates the inherent advantage of the proposed computing paradigm in terms of system security.

Two viruses, namely an info-stealing Trojan Netchief and a disk killer Trojan HDBreaker, are used as the demonstrated examples as follows. Figure 9 shows how the info-stealing Trojan Netchief infects the PC and the transparent computing system. From the control console we see that the virus can only run on the PC system, but we cannot trace any running virus on the transparent computing system.

Figures 10 and 11 give the examples of the disk killer Trojan on the transparent computing system and the PC system, respectively. Figures 10(a) and 11(a) show the main experimental interface for the malicious software to be activated. In case the disk killer Trojan HDBreaker is selected, Figs 10(b) and 11(b) show the interface on how to operate the experiments, which involves injecting the virus either to the transparent client or to the PC. Figures 10(c) and (d) indicate the cases in which the virus is being and has been activated in the transparent client, respectively. Figure 11(f) shows clearly that on the PC the Window operating system has been totally destroyed, so that the system administrator has to clean the hard disk to reinstall the operating system. On the contrary, because of the protection mechanism of Meta OS in the boot sector, the virus cannot infect the boot sector of the transparent client. Users just restart the system again to go back to work normally as usual, as described in Fig. 10(f).

5. Related work

Extensive researches on distributed and pervasive computing platforms have been reported. Our work is mostly related to such systems as network computers, thin-clients, network (distributed) file systems, and virtual machine based systems.

In order to deal with the management challenge of personal computers, network computers, such as the Java Station by Sun [15], are proposed to replace personal computers. This solution only supports WWW & Java applications, and does not support general commodity OSes or other applications such as Microsoft Office.

Thin-client systems have been very popular, and by providing a full featured desktop to users with low management costs, they are deemed as pervasive computing platforms. Exemplary systems include Microsoft RDP [11], Citrix ICA [4], Sun Ray 1 [31], VNC [26], and MobiDesk [1]. In the thin-client system paradigm, all computing tasks are performed at the

Fig. 10. Example of disk killer Trojan on the transparent computing system.

central server, while a client works only as a user interface by performing display, and keyboard/mouse input/output functions. Although such systems also conduct centralized management, they greatly increase the server resource requirements, and feature very limited scalability. Applications with heavy computing requirements (e.g., multimedia applications) usually cannot be supported by thin-client systems efficiently. Furthermore, there is no isolated user performance or security guarantee: one user can easily interfere with another user when they are sharing the server.

Network file systems and devices, such as NFS [27], AFS [16], and NAS [14], are popular solutions for sharing data in a distributed enterprise environment. Although these systems can be used to share user files flexibly, they generally do not support sharing operating systems or application files for the reason that the running of software will need to write to the location that they resides to save configurations or user-specific data.

Our idea of centralizing storage while distributing computing is similar to the concept of diskless com-

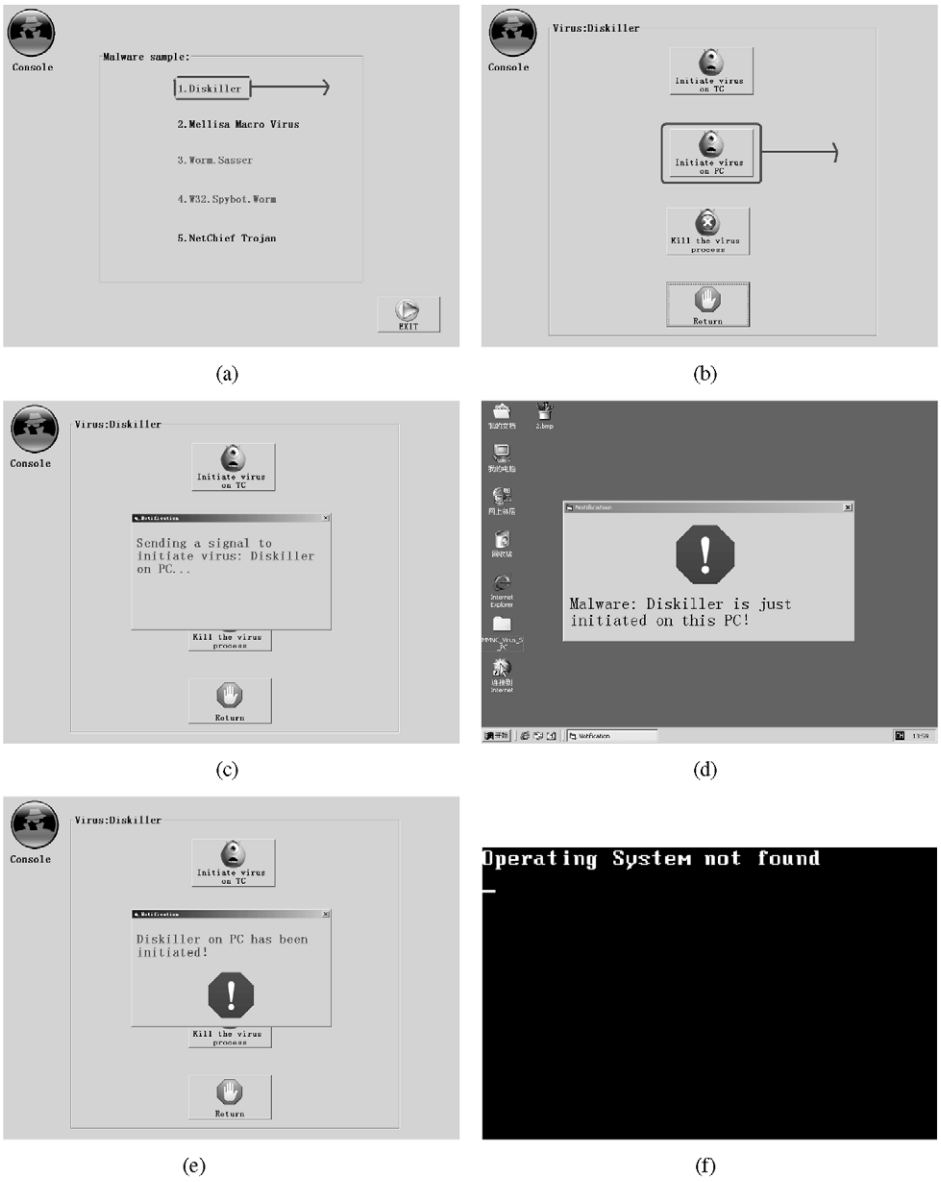

Fig. 11. Example of disk killer Trojan on the PC system.

puters (e.g., [9,22]) in early years. Without local hard disks, a diskless computer usually downloads an OS kernel image from the remote server. Thus it cannot support OSes that do not have clear kernel images, such as Windows, neither does it support booting from heterogeneous OSes. Further, V-disks perceived by users can be flexibly mapped to V-disk image files on the server. Such flexibility allows transparent computing system to share OS and application software across clients to reduce the storage and management overhead, while still isolating personal files for user privacy.

The iSCSI protocol has been used to access disk blocks via network [28]. In particular, the iBoot [17] Project at IBM has proposed a method that can remotely boot a commodity OS through iSCSI. An iBoot client needs a special type of BIOS ROM to carry out the OS boot process, so that it is not generally applicable. For better performance, transparent computing system can potentially adopt iSCSI to replace the cur-

rent V-disk access protocol, but it may need to modify the protocol implementation in order to fit the small size client BIOS memory.

The concept of resource virtualization has been introduced long ago, and recently, it has been adopted to address such issues in computer systems as security, flexibility, and user mobility. For example, commercial products such as VMware [37] have extended the concept of virtual machine to support multiple commodity platforms. The disks in these virtual machines are also virtualized, but reside in a local host machine and accessed through the file system of the Host OS. In contrast, virtual disks in transparent computing system is located in the remote server, with different types of V-disks for sharing and isolating data among users.

VM-based stateless thick client approaches, such as ISR (Internet Suspend/Resume [20]), use virtual machine technology (e.g., VMware) together with a network file system (e.g., Coda [29]) to support user mobility. Each ISR client runs OS and applications on top of a preinstalled VMware on the host OS. The use of virtual machines supports heterogeneous OSes as well, but it also incurs additional performance overhead due to virtualization of all hardware resources, including CPU and memory, while in transparent system, client OSes are running directly on top of the CPU, memory and graphics resource.

SoulPad [5] is another project that uses virtual machine concept for mobility, with a portable storage device to store the entire virtual machine image. The Collective Project [8] proposed a cache-based system management model based on virtual machines to reduce the management tasks of desktop computing. Similar to transparent computing system, it also uses different types of virtual disks, among which there is an immutable system disk to protect it against outside threats. Compared with Collective, the transparent computing system uses a COW file semantic instead of COW disks. Moreover, it adopts on-demand block level disk access instead of using network file systems (such as NFS) to access and cache disk images.

6. Conclusions and future work

This paper tries to give a detailed description of the impact of the transparent computing paradigm, based on the spatio-temporally extended von Neumann architecture, on the information security, which has been a significant issue since computer and information technology came into full existence in our daily life. This new computing paradigm brings a revolutionary change on the computer system architecture since the execution and storage of programs, including operating system and applications, are separated between client systems and the server(s); specifically, system/service programs are stored on the central servers, while fetched on demand in a block-streaming way, and automatically initiated/executed on the client system with local CPU and memory resources. This radically reduces the work load on the client systems, improves dramatically the performance, efficiency and capability running heterogeneous operating systems on various hardware platforms, and significantly enhances the central management and service. Obviously its impacts on information security are enormous. In the traditional von Neumann model, all security discussions are relied on the secure operating system and the security application programs above it, such as different access control program, etc. This is because users and system administrators do not know exactly how the data and instructions are operated (such as read, write and transmission operations) within operating systems, whereas all security problems such as virus and information leakage are just tightly coupled with these reading and writing operations within operating systems.

The transparent computing system allows system administrators to fully understand and control all the reading, writing and transmission operations of these internal data and instructions. Furthermore, it can adjust the control parameters of Meta OS at the BIOS level to manage and control the operating system (Instance OS) to identify the users' access privileges and to prevent virus infections. Also it can avoid users' and administrators' intentional and unintentional information leakage.

The transparent computing paradigm has its inherent advantages, which not only perfectly suit educational, industrial, military, governmental and entertainment applications, but are particularly useful in mobile, family and home applications. Currently more than one hundred thousands transparent computing systems have been massively deployed in the above fields in China. As this computing paradigm is more widely used in the society, its security feature and advantages will become more and more attractive. In order to further improve the system, we plan to conduct even more in-depth research in the following directions:

- The secure and trusted system architecture: the paradigm has shifted the focus of essential security issues from the secure operating system down to the Meta OS and BIOS (such as EFI) levels. Although the implementation based on these levels is simpler and easier, how to build the security model, define system structure and classify functional blocks, etc., still needs our further systematic study.
- How to control and implement the reading and writing operations between Meta OS and Instance OS is still an open problem. This control mechanism will directly influence the protection and prevention of viruses. This is also closely related with the issue of how to build the trusted transparent computing system.
- How to detect and control the data and instruction streaming is a very challenging research issue. At the server and gateway, this new paradigm has provided a mechanism to make the detection and control possible, but further details and techniques of its implementation need comprehensive investigations in the future.
- The integration of the existing security techniques into the transparent computing paradigm would be an interesting topic.
- The further studies on the Ubisafe computing and other emerging areas.

References

[1] R.A. Baratto, S. Potter, G. Su, and J. Nieh. MobiDesk: Mobile Virtual Desktop Computing. In *Proceedings of the 10th Annual International Conference on Mobile Computing and Networking*, 2004.

[2] S.M. Bellovin and W.R. Cheswick. Network Firewalls. *IEEE Communications Magazine*, 32(9):50–57, 1994.

[3] M. Bishop. *Computer Security: Art and Science*. Addison-Wesley, 2003.

[4] I. Boca. Citrix ICA Technology Brief, Technical White Paper, 1999.

[5] R. Caceres, C. Carter, C. Narayanaswami, and M. Raghunath. Reincarnating PCs with Portable SoulPads. In *Proc. of ACM/USENIX MobiSys*, pages 65–78, 2005.

[6] J. Carter and A. Ghorbani. Towards a Formalization of Value-centric Trust in Agent Societies. *Web Intelligence and Agent Systems*, 2(3):167–184, 2004.

[7] M. Carugi and J.D. Clercq. Virtual Private Network Services: Scenarios, Requirements and Architectural Constructs from a Standardization Perspective. *IEEE Communications Magazine*, 42(6):116–122, 2004.

[8] R. Chandra, N. Zeldovich, C. Sapuntzakis, and M.S. Lam. The Collective: A Cache-Based Systems Management Architecture. In *Proc. of NSDI*, pages 259–272, May 2005.

[9] D.R. Cheriton and W. Zwaenepoel. The Distributed V Kernel and its Performance for Diskless Workstations. In *Proceedings of the 9th ACM Symposium on Operating Systems Principles*, pages 128–140, Bretton Woods, N.H., October 1983.

[10] W.R. Cheswick, S.M. Bellovin, and A.D. Rubin. *Firewalls and Internet Security: Repelling the Wily Hacker*. Addison-Wesley, 1996.

[11] B. Cumberland, G. Carius, and A. Muir. *Microsoft Windows NT Server 4.0, Terminal Server Edition: Technical Reference*. Microsoft Press, 1999.

[12] Extensible Firmware Interface. `http://www.uefi.org/`.

[13] R.J. Feiertag and P.G. Neumann. The Foundations of a Provably Secure Operating System (PSOS). In *Proceedings of the National Computer Conference*, pages 329–334, 1979.

[14] G.A. Gibson and R.Y. Meter. Network Attached Storage Architecture. *Communications of the ACM*, 43(11):37–45, 2000.

[15] R.G. Herrtwich and T. Kappner. Network Computers – Ubiquitous Computing or Dumb Multimedia? In *Proceedings of the Third International Symposium on Autonomous Decentralized Systems*, 1997.

[16] J.H. Howard, M.L. Kazar, and S.G. Menees. Scale and Performance in a Distributed File System. *ACM Transactions on Computer Systems*, 6(1):51–81, 1988.

[17] iBoot, Remote Boot Over iSCSI. `http://www.haifa.il.ibm.com/projects/storage/iboot/index.html`, 2008.

[18] A. Jain, L. Hong, and S. Pankanti. Biometric Identification. *Communications of the ACM*, 43(2):91–98, 2000.

[19] A.K. Jain, R. Bolle, and S. Pankanti. *Biometrics: Personal Identification in Networked Society*. Kluwer Academic Publishers, 1999.

[20] M. Kozuch and M. Satyanarayanan. Internet Suspend/Resume. In *Proceedings of the 4th IEEE Workshop Mobile Computing Systems and Applications*, 2005.

[21] P.Y. Lee, S.C. Hui, and A.C.M. Fong. Neural Networks for Web Content Filtering. *IEEE Intelligent Systems*, 17(5):48–57, 2002.

[22] R. Linlayson. Bootstrap Loading Using TFTP. RFC 906, 1984.

[23] J. Ma, Q. Zhao, V. Chaudhary, J. Cheng, L.T. Yang, R. Huang, and Q. Jin. Ubisafe Computing: Vision and Challenges (I). In *Proceedings of the 3rd International Conference on Autonomic and Trusted Computing (ATC-06)*, pages 386–397, September 2006.

[24] D. Maughan, M. Schertler, M. Schneider, and J. Turner. Internet Security Association and Key Management Protocol (ISAKMP). RFC 2408, 1998.

[25] RedFlag Linux. `http://www.redflag-linux.com/eindex.html`, 2003.

[26] T. Richardson, Q. Stafford-Fraser, K.R. Wood, and A. Hopper. Virtual Network Computing. *IEEE Internet Computing*, 2(1):33–38, 1998.

[27] R. Sandberg, D. Goldberg, S. Kleiman, D. Walsh, and B. Lyon. Design and Implementation of the Sun Network Filesystem. In *USENIX Association Conference Proceedings*, 1985.

[28] J. Satran, C.S.K. Meth, M. Chadalapaka, and E. Zeidner. Internet Small Computer Systems Interface (iSCSI). RFC 3720, 2004.

[29] M. Satyanarayanan. The Evolution of Coda. *ACM Transactions on Computer Systems*, 20(2), 2002.

[30] W. Stallings. *Cryptography and Network Security, 4 Ed.*, Prentice Hall, 2005.

[31] Sun Ray Overview, White Paper, Version 2. http://www.sun.com/sunray/whitepapers.html, December 2004.
[32] P. Szor. *The Art of Computer Virus Research and Defense*. Addison-Wesley, 2005.
[33] TCG PC Client Specific Implementation Specification for Conventional BIOS, Version 1.20.
[34] TCG PC Specific Implementation Specification, Version 1.1.
[35] TCG Specification Architecture Overview, Revision 1.4. https://www.trustedcomputinggroup.org/groups/TCG_1_4_Architecture_Overview.pdf, 2007.
[36] Z. Tian, M. Hu, B. Li, B. Liu, and H. Zhang. Defending Against Distributed Denial-of-Service Attacks with an Auction-Based Method. *Web Intelligence and Agent Systems*, 4(3):341–351, 2006.
[37] VMware GSX Server. http://www.vmware.com/products/gsx, 2001.
[38] Y. Zhang and Y. Zhou. Transparent Computing: A New Paradigm for Pervasive Computing. In *Proceedings of the 3rd International Conference on Ubiquitous Intelligence and Computing (UIC-06)*, pages 1–11, September 2006.