Security from the Transparent Computing Aspect

Guojun Wang†, Qin Liu‡, Yang Xiang§, and Jianer Chen†
†School of Information Science and Engineering, Central South University, P. R. China
‡College of Information Science and Engineering, Hunan University, P. R. China
§School of Information Technology, Deakin University, Australia
Correspondence to: gracelq628@126.com

Abstract—Transparent computing is an emerging computing paradigm where the users can enjoy any kind of service over networks on-demand with any devices, without caring about the underlying deployment details. In transparent computing, all software resources (even the OS) are stored on remote servers, from which the clients can request the resources for local execution in a block-streaming way. This paradigm has many benefits including cross-platform experience, user orientation, and platform independence. However, due to its fundamental features, e.g., separation of computation and storage in clients and servers respectively, and block-streaming-based scheduling and execution, transparent computing faces many new security challenges that may become its biggest obstacle. In this paper, we propose a Transparent Computing Security Architecture (TCSA), which builds user-controlled security for transparent computing by allowing the users to configure the desired security environments on demand. We envision, TCSA, which allows the users to take the initiative to protect their own data, is a promising solution for data security in transparent computing.

Index Terms—Transparent computing, user-controlled cloud computing, security architecture

I. INTRODUCTION

The vision of the disappearing computer presented by Mark Weiser [1] has been a source of inspiration and an important guideline for many researchers and practitioners working in the computer realm. One of the most promising technologies to realize this vision is transparent computing. With the emergence and rapid advance in transparent computing, the users can enjoy any kind of service without caring about the underlying deployment details.

Transparent computing was first proposed by Yaoxue Zhang in 2004 [2], [3]. One of its many aims was to allow the users to enjoy services transparently and seamlessly. The convening of Intel Developer Forum (IDF) 2012 in San Francisco indicated the era of transparent computing is coming. From the perspective of a system architecture, transparent computing extends the “stored program concept” in von Neumann architecture into the networking environments spatio-temporally. In transparent computing, computation and storage are separated between the clients and the servers. That is, all software resources (even the OS) are stored on the servers, while the execution of the programs is performed on the clients. Specifically, the clients can request the resources stored on the servers on demand for local execution/computation in a block-streaming way. From the angle of service delivery, transparent computing is similar to the Software as a Service (SaaS) model in cloud computing, where the software resources are provided as a service. However, transparent computing, which even delivers the OS as a service, carries out this model more thoroughly. The users can choose any kind of OS to run the desired applications.

Now, some people may ask “Are transparent computing and cloud computing the same thing?” We believe that transparent computing is user-controlled cloud computing, where the users have the ability to run the desired applications on different operating systems with different devices. They share many common characteristics, e.g., both of them deliver servers over networks, or both of them support the multi-tenant model. At the same time, they also have many differences. In general, transparent computing is a computing paradigm which focuses on technical extensions. It has an explicit technical route and can be clearly defined by its fundamental features. However, cloud computing is more like a business model, which focuses on commercial realization. Its definition and adopted techniques depend on who is using it.

In this paper, we address the security issues from the perspective of transparent computing. Same as in cloud computing, data leakage is one of the biggest obstacles hindering the use of transparent computing services. Furthermore, the unique features of transparent computing, such as separation of computation and storage in clients and servers respectively, independency of hardware and software, and block-streaming-based scheduling and execution, make security solutions suitable for traditional computing paradigm no longer adequate. Such security solutions need to be reconsidered to fit the new paradigm of transparent computing.

Therefore, we propose a Transparent Computing Security Architecture (TCSA), which builds user-controlled security for transparent computing by allowing the users to configure the desired security environments on demand. There are three stages in TCSA: Secure Booting, Data Classification, and Encapsulation. Secure booting, which allows the users to choose the desired OS as well as the security scheme, is safeguarded by the Extensible Firmware Interface (EFI). If the users choose the user-controlled security scheme rather than the centralized security scheme at this stage, they will be responsible for classifying their data from three dimensions: confidentiality, searchable ability, and sharable ability. Different data types correspond to different encapsulation mechanisms. The server will encapsulate data based on the data type, using the onion method [4]. The main advantage of TCSA is that the users can take the initiative to protect their own data, and thus it is...
more suitable for user-controlled clouds.

To devise an appropriate solution tailored for transparent computing, we must understand its principal characteristics, the new security challenges, how users will use these services, and how to safeguard them. Therefore, we first introduce the definition and features of transparent computing in Section II. Following this, we discuss the security issues in a transparent computing environment in Section III, before we provide the design of the transparent computing security architecture in Section IV. Finally, we will conclude this paper and discuss the future work in Section V.

II. DEFINITION AND FEATURES OF TRANSPARENT COMPUTING

Transparent computing aims to provide a cross-platform user experience, and make computer systems more secure, more reliable, and more flexible. The enabling techniques that contribute to the emergence of transparent computing include software interoperability standards, virtualization technologies, high-bandwidth communications, and Web 2.0. At present, more and more companies and scholars are paying attention to transparent computing, and consider it as an emerging computing paradigm. For example, the Intel Corporation has already initiated collaborative research with Tsinghua University and Central South University in China. They believe that the success of transparent computing will undoubtedly lead the computing paradigm into a new era.

The pioneering work [2], [3] formally defined transparent computing as follows:

**Transparent Computing.** Transparent computing is a computing paradigm, where the users can enjoy services over network on-demand with any kind of devices, while they don’t have to know the location of the OS, middleware, and applications.

Transparent computing paradigm extends the von Neumann “stored program concept” architecture into the networking environments spatio-temporally. The system model of transparent computing is shown in Fig. 1. The transparency reflects in three aspects: transparent client (T-Cient) held by the user, transparent server (T-Server) maintained by the service provider, and transparent network (T-Network) managed by the network operator. The T-Cient is rather simple and light-weight, almost like a bare computer. It does not equip an OS or hard disk, and only stores the underlying basic input output system (BIOS) plus a small fraction of the protocol and management program in the memory or cache. The T-Server is a regular PC or a high-end dedicated machine that stores all softwares (OS and applications) and data. The T-Network supports various kinds of network access methods, e.g., Ethernet, WLAN, 3G, etc. The T-Servers provide various services to the T-Clients over the T-Network in a streaming way as follows: the T-Clients send interrupts and I/O requests over the T-Network to the T-Servers; the T-Servers will handle corresponding requests and then send the programs and data back to the T-Clients in a block-streaming way over the T-Network; the T-Clients execute the programs locally.

Transparent computing supports only one kind of service models, i.e., SaaS, and carries out this model more thoroughly. All software, even the OS, are provisioned as a service. The fundamental features of transparent computing that differentiate it from cloud computing are as follows:

- Separation of computation and storage. Application programs, data, and system software (including OS) are stored in the T-Servers, while the execution of the application programs is performed on the T-Clients.
- Independency of hardware and software. End users can choose and run any desired OS and application programs on the T-Clients transparently.
- Block-streaming-based scheduling and execution. Application programs are dynamically scheduled and executed on the T-Clients in a block-streaming way.

Besides the same benefits as cloud computing, e.g., improved operational efficiency, flexibility, and reliability, transparent computing has the following desired features:

- Service transparency. The underlying details of services are transparent to the users. They use the applications as their accustomed usage in traditional paradigm.
- Cross-platform experience. Users can flexibly choose the desired OS and applications to run on different clients.
- Thin client. The client is rather simple and lightweight, almost like a bare computer. The software and data are delivered in a block-streaming way to clients instead of residing on the clients permanently.
- Centralized security. The system (including OS, applications, and data) is fully protected by the server in a centralized way. Since the client does not store any data permanently, the risk of information leakage caused by insecure clients is reduced.

III. SECURITY ISSUES IN TRANSPARENT COMPUTING

Due to the similarities between transparent computing and cloud computing, potential security risks in cloud computing [5] also exist in transparent computing, e.g., privileged user
access, data location, and data segregation. The main reason is that, when moving data to remote servers, the users face two main changes [6]. First, the data is moved from local machines to a perimeter-less environment, where the physical security boundary does not exist. Second, the data is moved from a single-tenant to a multi-tenant environment, where unexpected side channels (passively observing information) and covert channels (actively sending data) may occur. These changes may raise an important concern about data leakage, which has become one of the greatest security risks hindering the wide adoption of cloud computing and other emerging computing paradigms.

Meanwhile, transparent computing, due to its distinctive features, introduces a number of new security challenges [7]:

- Multi-OS remote booting. Transparent computing allows the cross-platform experience. However, different operating systems employ different security policies to ensure different levels of security. For example, the Windows family is apt to integrate most functionalities into the OS, which may cause more security breaches than the Linux family. Furthermore, the applications running on it continue to upgrade, and thus the old security policies may be missing or invalid. For example, Windows XP Service Pack2 enhanced Windows security by turning off many services that are originally opened by default. From the users’ point of view, they want to enjoy services unconsciously. Thus, the migration and upgrade of security policies should be done transparently and seamlessly.

- Virtual disk sharing. The T-Client does not equip any hard disk. If the data requested by the T-Client is not found in its main memory, the virtual I/O on the T-Client will trigger a page fault error and forward this request to the virtual disks on the T-Server over network. This process occurs frequently from system booting to system shutdown. Therefore, it requires the T-Server to have the ability to respond quickly. More importantly, the T-Server can deal with different security demands from multiple T-Clients simultaneously and continuously.

- Centralized security. For the users, they may lose the direct control over their data or even the OS. It is difficult for the users to ensure that right security measures are in place and also difficult to audit whether the service provider performs correctly according to the signed service level agreements (SLAs), or not. The users have to depend on the service provider to carry out the security measures. The service provider not only needs to prevent the outside attackers from peeking at the user data, but also needs to keep multiple users from seeing the data of each other.

Actually, all of the above security threats, vulnerabilities and challenges can be subject to the CIA yardstick, where CIA is an abbreviation for confidentiality, integrity, and availability.

- Confidentiality. Confidentiality refers to the prevention of intentional or unintentional unauthorized disclosure of information. A pivot approach is the encryption of sensitive data before storing. Reinforcing access control, authentication, and authorization can also benefit the preservation of data confidentiality.

- Integrity. Integrity needs to ensure that unauthorized modifications are not made to data. The Message Authentication Code (MAC) and Digital Signature (DS) are two main approaches to achieve integrity. In a transparent computing environment, the users do not store any data copies locally, thus how to efficiently achieve dynamic, blockless, and stateless verification is a further issue [8], [9].

- Availability. Availability needs to ensure the reliable and timely access to data or resources. To provide ubiquitous and continuous access, a service provider should maintain multiple replicas for all data on distributed T-Servers. A key problem of using the replication technique in transparent computing is that it is very expensive to achieve strong consistency to ensure that a user always sees the latest updates [10].

CIA are the important pillars for ensuring security in either traditional computing paradigm or the emerging transparent computing paradigm. In TCSA, we mainly address the problem of confidentiality. For confidentiality, existing research suggests to store only encrypted data on untrusted servers [11]. In this way, only the authorized entities can decrypt the data with appropriate keys. The unauthorized entities, even if the service provider cannot know data contents. In a transparent computing environment, data is generally shared by many users with different roles and attributes, thus how to achieve fine-grained access control on ciphertexts becomes a burning question. Furthermore, the T-servers have been building up the capacity to store huge amounts of digital data. Users often want to retrieve only the files of their interests without leaking any information, thus how to protect user privacy while performing searches is another key problem.

IV. TRANSPARENT COMPUTING SECURITY ARCHITECTURE

As we demonstrated, transparent computing introduces new security challenges, which are amongst the biggest obstacles
when considering the use of such a service. Due to its key features, i.e., separation of hardware and software, separation of computation and storage, as well as block-streaming execution, security solutions suitable for traditional computing paradigms are no longer adequate, but need to be rethought to fit this new paradigm. In this section, we propose a transparent computing security architecture (TCSA) to resolve these problems. TCSA allows the users to take the initiative to customize the desired security environments for program execution and data storage. In a sense, the conception of user-controlled cloud evolves as user-controlled security.

TCSA involves a three-stage process: Secure Booting, Data Classification, and Encapsulation, as shown in Fig. 2. Secure Booting happens in the system booting phase, where the users choose the desired OS as well as the desired security schemes. Two optional security schemes are available: a user-controlled security scheme and a centralized security scheme. The default setting is the centralized security scheme, which entrusts the T-Server to arrange all the security stuffs. In the user-controlled security scheme, the users can classify their data before storing it to the T-Server, which will adopt different methods to encapsulate the data based on the data type.

A. Secure Booting

Unlike the traditional booting process, where all requests are sent to a local hard disk, transparent computing adopts a remote booting mode. The key idea is to initialize a virtual I/O device on the T-Client, where each virtual disk is mapped to a virtual disk image on the T-Server. To boot a system, the T-Client will send the request over network to the T-Server, which will execute the request and return programs and data in a block-streaming way.

Step 1: Selection. All of the operating systems and security schemes available from the T-Server are displayed on the screen of the T-Client. The users can choose the desired OS as well as the security scheme for the system, and send the corresponding requests to the T-Server.

Step 2: Instantiation. The T-Client downloads a universal OS loader (MetaOS) from the T-Server. The main functionality of the MetaOS is to instantiate a BIOS-enabled virtual I/O device, with which the T-Client can redirect all I/O access requests to the T-Server.

Step 3: Booting: The MetaOS on the T-Client will help to find the boot sector and load the OS-specific loader. Then, the OS takes control and continues to boot up as normal.

It should be noted, the whole booting process will be safeguarded by the Extensible Firmware Interface (EFI) [12], later referred to as Unified Extensible Firmware Interface (UEFI). EFI/UEFI was originally promoted by Intel in 2007, as a specification that defines a software interface between an operating system and a platform firmware. It can be regarded as an improved version of the old legacy BIOS firmware interface historically used by all IBM compatible personal computers. It supports secure booting by utilizing cryptography to ensure that only the OS loaders or drivers with an acceptable digital signature will be loaded by the firmware. Specifically, a public key known as the “Platform key” (PK) is written to the firmware in the “setup” mode initially. Then, secure boot enters the “User” mode, where only drivers and loaders signed with the PK can be loaded by the firmware. The PK is regarded as a root private key, and plays an important role in the generation of other “Key Exchange Keys” (KEKs).

B. Data Classification and Encapsulation

If the system runs in the user-controlled security scheme, users can classify their data before storing it to the T-Server. Based on the data type, the T-Server will adopt appropriate strategies to encapsulate the data. The data access process in the user-controlled scheme is shown in Fig. 3.

For data storage, the user first conducts data classification based on Table I, and then sends the data together with the data type to the T-Server. On receiving the message from the T-Client, the T-Server will encapsulate data according to its type and store the encapsulated data in an appropriate location. For data retrieval, if the data cannot be found in the main memory of the T-Client, the virtual I/O on the T-Client will trigger a page fault error which will be forwarded to the T-Server over network. Upon receiving a request from the T-Client, the T-Server will first look up its cache to see if the requested data exists. If the data doesn’t exist in the cache, the T-Server will fetch the contents from its virtual disk. Then, the T-Server

<table>
<thead>
<tr>
<th>Confidentiality</th>
<th>Sharable ability</th>
<th>Searchable ability</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>No</td>
<td>/</td>
</tr>
<tr>
<td>II</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>III</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Fig. 4. Encapsulation.
will check whether the data is encapsulated appropriately. If not, the T-Server needs to encapsulate the data based on its type. Finally, the T-Server will reply the T-Client with the encapsulated data and the corresponding decapsulation codes, with which the T-Client can decapsulate the data locally.

Data is classified from three dimensions: confidentiality, searchable ability, and sharable ability, with each dimension corresponding to an encapsulation layer. For confidentiality, data should be encrypted before outsourcing to the servers. How to choose the encryption schemes depends on whether the data owner wants to share the data or not. For searchable ability, searchable encryption [13]–[15] allows the servers to perform keyword-based searches on ciphertexts without knowing user keywords and the file contents. Therefore, the users can retrieve only the files of their interests while preserving privacy. For sharable ability, attribute-based encryption (ABE) [16]–[18] is an effective cryptosystem for fine-grained access control. Meanwhile, as the encrypted data is shared by multiple users, how to effectively achieve user revocation [19]–[21] becomes a serious question. As shown in Table I, for Type-I data, the confidentiality dimension is set to No, which means that the data of this type will not be encapsulated at all; for Type-II data, both the confidentiality and searchable ability dimensions are set to Yes, and thus data of this type will be encapsulated with a confidential layer and a searchable layer; for Type-III data, all dimensions are set to Yes, and thus a three-layer encapsulation is imposed on data of this type.

The data is encapsulated in an onion way as shown in Fig. 4. From Type-I to Type-III, the number of layers for encapsulation steadily increases. Type-I data will be stored in the plaintext form. This kind of data has less relation about user privacy and thus can be shared with others. Furthermore, the plaintext is easy to support keyword-based searches, and the users can retrieve files of interests on demand. Type-II data needs a two-layer encapsulation. The data of this type is very sensitive and cannot be shared by others. Therefore, the secure layer encrypts data with the user’s public key, and the searchable layer encrypts the keywords with public key based searchable encryption. Furthermore, since the data is not shared by other users, the issues related to user revocation do not need to be considered here. Type-III data needs a three-layer encapsulation. Data of this type can be shared by only authorized users. Therefore, the secure layer encrypts data with a symmetric key, the sharable layer encrypts the symmetric key with ABE over a specific access structure, and the searchable layer encrypts the keywords with searchable encryption that supports multi-user performing searches. For secure data sharing, effective user revocation mechanisms should be applied in this case.

V. Conclusion

In this paper, we investigate the definition and features of transparent computing, and discuss the security challenges in such an environment. Specifically, a transparent computing security architecture (TCSA) is proposed to achieve user-controlled security. Although TCSA allows the users to choose the desired security environment, it does not provide a way for the users to audit whether the service provider carries out the promised protocols and policies or not. In transparent computing, it remains an open challenge to achieve thorough auditing without impairing performance. In our future work, we aim to explore how to audit transparent computing services in a bilateral or multifacational fashion.

ACKNOWLEDGMENTS

This research was supported in part by NSFC grants 61272151 and 61073037, ISTCP grant 2013DFB10070, the China Hunan Provincial Science & Technology Program under Grant Number 2012GK4106, and the “Mobile Health” Ministry of Education-China Mobile Joint Laboratory (MOE-DST No. [2012]311).

REFERENCES