제 19 회 한국정보처리학회 춘계학술발표대회 논문집 제 10 권 제 1 호 (2003. 5)

The Designs and Implementation of Trusted Channel between Secure Operating Systems

Joon-Suk Yu*, Jae-Deok Lim*, Jeong-Nyeo Kim*, Sung-Won Sohn*
*Information Security Research Division, ETRI
e-mail : jsyu92@etri.re.kr

Abstract

Trusted channel provides a means of secure communication and it includes security services such as confidentiality, authentication, and so on. This paper describes the implementation of trusted channel between secure operating systems that integrates access control mechanisms with FreeBSD kernel code[1]. The trusted channel we developed offers confidentiality and message authentication for network traffic based on the destination address. It is implemented in the kernel level of IP layer and transparent to users.

1. Introduction

There exist lots of security holes in multi-user environment such as Unix or Linux and the concentrated rights on root user and Trojan horse are the main weaknesses among those. These weaknesses may cause serious problems but we can cope with the m effectively by using access control mechanisms such as MAC(Mandatory Access Control) or RBAC(Role-Based Access Control).

[1] presents the secure operating system that integrates various access control mechanisms with FreeBSD 4.3 kernel. The integration provides reasonable solution to the problems caused by the system security holes mentioned above. However, it doesn't provide any security against network-based attack such as spoofing or sniffing.

These days, most of systems are connected to the open network such as Internet. It means that the network traffic between the systems can be easily accessed by anyone. Considering that the value of communicated information is getting increased, a means to protect network traffic from unauthorized access is essential in order to guarantee the security of the whole system.

In this paper, we describe how to construct the secure communication channel, called trusted channel, between two systems which secure operating system is installed on. Generally, trusted channel provides confidentiality, authentication, and so on and the trusted channel we developed provides confidentiality and message authentication[2].

This paper is organized as follows. Chapter 2 gives the overview of the functionalities of the system. The design and implementation issues are presented in chapter 3 and we compare the performance of the system in chapter 4. Lastly, the conclusion and future work is given in chapter 5.

2. System Overview

As mentioned earlier, the purpose of the trusted channel we developed is to offer confidentiality and message authentication for the network traffic between secure operating systems. To provide confidentiality and message authentication service, sender encrypts outgoing packet and adds authentication information to the packet. In other hands, receiver decrypts the received packet after verifying the authentication information of it.

Sender decides if outgoing packet requires the trusted channel service based on the destination of the packet. In other words, the system, which secure operating system is installed on, has the list of IP addresses and if the destination address of outgoing packet is in the list, trusted channel is applied. If trusted channel is applied, the next_protocol field of IP header is replaced with new value representing that trusted channel is applied to the packet.

When the packet arrives at the destination, the receiver can recognize the TC-applied packet by checking the next_protocol filed of IP header and performs an appropriate process.

We assume that all the systems have shared the algorithm and key for encryption and authentication in advance. The details of the system will be given in the rest of the paper.

3. Designs and Implementation

3.1 Architecture and General Issues

The trusted channel we present is applied to the network traffic between two secure operating systems, which enhance the security by combining access control mechanisms with normal operating system, FreeBSD 4.3. The trusted channel is implemented in IP layer and provides confidentiality and message authentication service for specific outgoing packets. 64 bits symmetric encryption algorithm, blowfish, is used for confidentiality and HMAC-MD5 is used for message authentication. It is known that blowfish has good performance comparing with DES or IDEA[3]. And blowfish can use variable size of key from 32 bits to 448 bits[3, 4]. However, we fixed the size of key for convenience of implementation. Figure 1 shows the architecture of the system.

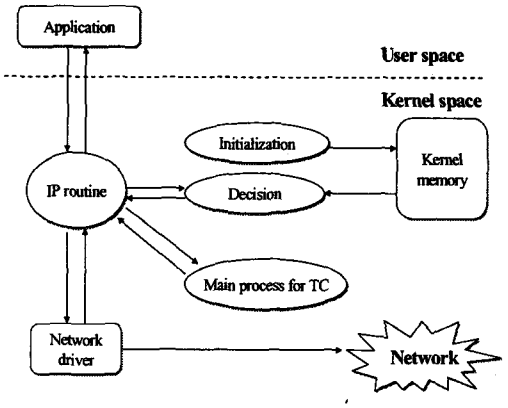

Figure 1. Architecture of the trusted channel

3.2 Configuration files and initialization

There are three configuration files for the trusted channel service and they are listed below.

- ***Encryption key file***: this file stores 128 bits key for packet encryption/decryption.
- ***Authentication key file***: this file stores 128 bits key for message authentication.
- ***Host file***: this file contains the IP addresses of hosts that trusted channel service to be applied to.

All the configuration files are generated by a special user who has security manager role, called security manager, at system installation time and they are commonly used among all of secure operating systems that are supposed to provide trusted channel service. The configuration files are maintained in a special directory protected by RBAC and only security manager of the system can access the directory. It means that the configuration files are secure from unauthorized access only if RBAC is robust.

The configuration files are automatically loaded into kernel memory at system booting time and the system refers the loaded configuration file data during system running time. Using kernel memory increases system efficiency by reducing the overhead raised from frequent disk access. Moreover, it is more secure than using other memory space.

3.3 TC Header and Packet Output/Input

When the trusted channel service is applied to a packet, TC header is added to the packet. Figure 2 shows the structure of the TC-applied packet and the service area provided. The shade is TC header in the figure 2. Followings are the description of each TC header field.

- **Authentication data field**: this field presents the hash value of the packet.
- **IV(Initial Vector) field**: this field is used for packet encryption/decryption.
- **Next_hdr. field**: this field contains the next_protocol field of IP header.
- **HLEN/PLEN field**: this field presents the TC header length and padding length.
- **MAC class and MAC category field**: These fields can be used for setting up the security information at remote system. These are reserved for future use.

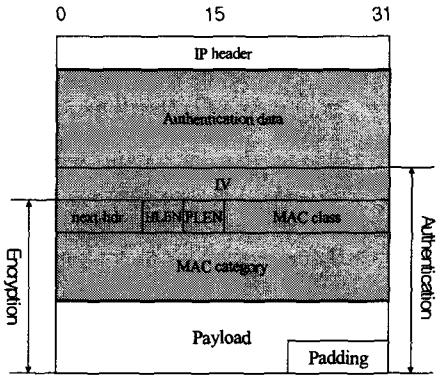

Figure 2. Packet structure

The procedures for outgoing packet at sender and incoming packet at receiver are different and we describe them hereafter.

When a user process requests to send data, the request is passed to the output interface of IP layer, ip_output(), through the interface of upper layer[5, 6]. Ip_output() performs not only all the tasks for IP output processing that includes routing and fragmentation but also trusted channel related tasks[6, 7].

Making a decision over the trusted channel application and applying the trusted channel to the packet are done just before packet fragmentation. Making a decision is based on the destination address of outgoing packet. If the destination address of the packet is in the address list loaded into kernel memory from host file, a specific variable indicating that the packet requires trusted channel service is set. Ip_output() calls a routine for the trusted channel if the variable is set. Otherwise, ip_output() just skips the procedure for the trusted channel service and continues normal IP processing. The routine for the trusted channel performs the following tasks.

① It generates TC header and reconstructs the outgoing packet.
② It encrypts the packet.
③ It computes authentication data for the packet and fills up the appropriate field of TC header with the authentication data.

The next_protocol field of IP header is set up with a specific value so that the receiver is able to recognize TC-applied packet in the procedure ㉠. The original value in the next_protocol field is maintained in the next_hdr. field of TC header and figure 3 depicts this.

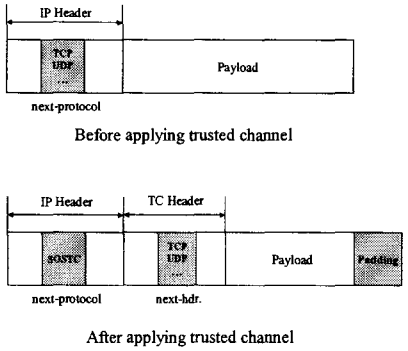

Figure 3. TC header field set up

After finishing all the procedures, the control is returned to ip_output() and it fragments the reconstructed packet if the packet size is larger than the maximum packet size for the outgoing interface. And it passes the packet to lower layer.

When a TC-applied packet arrives at destination, the packet is passed to the input interface of IP layer, ip_input(), through the interface of lower layer[5, 6]. Ip_input() performs all the tasks for IP input processing that includes checking the packet size, reassembly and so on[6, 7]. It also performs the trusted channel related tasks just before passing the packet to upper layer protocol.

With regard to the trusted channel, ip_input() decides if the trusted channel is applied to the packet first by checking the next_protocol field of IP header. If the next_protocol field of IP header indicates that the trusted channel is applied to the packet, ip_input() calls a routine for the trusted channel processing. Otherwise, ip_input() skips the trusted channel processing just like ip_output().

The routine for the trusted channel processing performs the following tasks in turn.

① It verifies the authentication of the packet. If the packet is not authentic, the packet is discarded.
② It decrypts the packet only if the packet is authentic.
③ It removes TC header and padding from the packet.

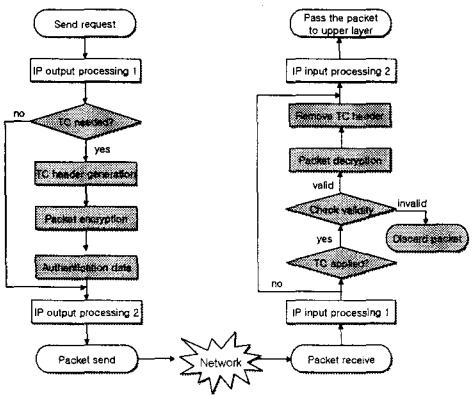

Figure 4. Procedures for packet output and input

The next_protocol field of IP header is replaced with original value maintained in the next_hdr. field of TC header in the procedure ✦. The control is returned to ip_input() and it passes the packet to the upper layer protocol. Figure 4 shows the procedures for packet output and input.

4. Performance

Packet encryption is essential to achieve the purpose of the trusted channel service and because encryption is time-consuming task, it always reduces the performance of system. We transferred same data files in two different environments respectively to measure the transmission time differences. One is the data transmission between secure operating systems and the other is the data transmission between normal FreeBSD systems. Figure 5 shows the comparison result of transmission time for test data. We used 50MB and 100MB text files as test data.

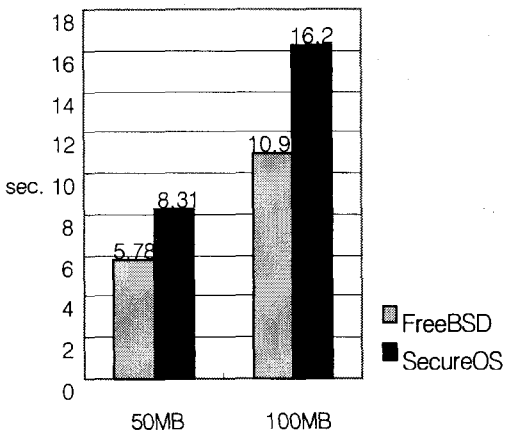

Figure 5. Comparison of performance

As shown in figure 5, when the trusted channel is applied, the performance goes down to about 70% comparing with the transmission between normal FreeBSD systems. Because the size of most user data transferred through the network is usually smaller than that of test data, the performance of the system is acceptable to the system users.

5. Conclusions and Future Work

Secure operating system that integrates various access control mechanisms with normal operating system at kernel level offers reasonable solution to the security problems caused by centralization of rights or Trojan hose. However, it doesn't provide any means to protect communicated traffic from sniffing or spoofing.

We implemented and described the trusted channel providing confidentiality and message authentication for network traffic between secure operating systems. The trusted channel we developed is implemented in kernel level and provides packet encryption and message authentication based on the destination address of the packet. Users'

intervention is no required during system running time and users can transfer their data transparently. However, it might be cumbersome that security manager must set up the configuration files manually in advance. Even though the developed system is initial version, more automated and easy configuration method is required for convenience.

References

[1] J. G. Ko, J. N. Kim, & K. I. Jeong, "Access Control for Secure FreeBSD Operating System", *Proc. of WISA2001, The Second International Workshop on Information Security Applications*, 2001.

[2] "Common Criteria for Information Technology Security Evaluation, Part 2: Security functional requirements, Version 2.1", 1999.

[3] B. Schineier, "Description of a New Variable-Length Key, 64-Bit Block Cipher(Blowfish)", Fast Software Encryption, Cambridge Security Workshop Proceedings, Springer-Verlag, 1994.

[4] B. Schineier, *Applied Cryptography*, John Wiley & Sons, 1996.

[5] M. K. McKusick, K. Bostic, M. J. Karels and J. S. Quarterman, *The Design and Implementation of the 4.4BSD Operating System*, Addison-Wesley Publishing Company, 1996.

[6] FreeBSD 4.3 Source Code.

[7] Behrouz A. Forouzan, *TCP/IP Protocol Suite*, McGraw Hill, 2002.