Journal of Information Processing Vol. 16 157–164 (Sep. 2008)

Regular Paper

Lightweight Vulnerability Management System

Takeshi Okuda[†1] and Suguru Yamaguchi[†1]

To secure a network, ideally, all software in the computers should be updated. However, especially in a server farm, we have to cope with unresolved vulnerabilities due to software dependencies. Therefore, it is necessary to understand the vulnerabilities inside the network. Existing methods require IP reachability and dedicated software to be installed in the managed computers. In addition, existing approaches cannot detect vulnerabilities of underlying libraries and uniformly control the communication between computers based only on the vulnerability score. We propose a lightweight vulnerability management system (LWVMS) based on a self-enumeration approach. This LWVMS allows administrators to configure their own network security policy flexibly. It complies with existing standards, such as IEEE802.1X and EAP-TLS, and can operate in existing corporate networks. Since LWVMS does not require IP reachability between the managed server and management servers, it can reduce the risk of invasion and infection in the quarantine phase. In addition, LWVMS can control the connectivity based on both the vulnerabilities of respective components and the network security policy. Since this system can be implemented by a slight modification of open-source software, the developers can implement this system to fit their network more easily.

1. Introduction

Many software vulnerabilities have been reported, and remedies such as security updates are developed by software vendors almost daily. Applying these security updates or migrating to newer and safer software is a suitable way of securing servers, clients, and networks. Although it is important to keep the software components inside servers updated to remove security vulnerabilities, not all software can be updated because of dependencies. For example, a groupware may require a specific version of an enterprise database server that can run only on an older OS. In this scenario, the user cannot dispose of the existing groupware because building a new groupware will require extra time and money. Users must manage the vulnerabilities inside their networks. Ideally, all software should be kept up to date, but a more realistic solution is to recognize and understand network vulnerabilities [1)].

In order to understand these vulnerabilities, two main methods have been developed. The first is the active scan method, and the second is the agent-based method. In the active scan method, security scanners scan all ports on the target computer. This activity disrupts the services running on the target and generates numerous event logs on it. The scanning of the computer consumes resources such as CPU and memory. On the other hand, the agent-based approach does not affect the services, but requires dedicated software, referred to as an agent, to be installed on the target computers. These agents scan registries and the file system in the computer to search the installed software and libraries and employ proprietary protocols between agent and manager, which monitor and control the connection of computers to the network. The supported architectures and OSs are limited and the agent has to follow changes in the OS design. These methods are thought to be heavyweight methods from the viewpoints of the initial cost, the maintenance cost, and resources consumption. Small and medium business users cannot use these heavyweight methods to maintain the security of their networks and require lightweight methods to manage their networks.

Although almost all of these methods target the clients' security, securing servers in a server farm is relatively important because the servers are always online, and may include unresolved vulnerabilities due to software dependencies. As a result, an administrator of the network has to manage the risk of extrusion.

We focused on securing a server farm and proposed a lightweight vulnerability management system based on a self-enumeration approach that enables a vulnerability check and isolates computers that have vulnerabilities that violate the network security policy. Using this network security policy, the network administrator can control the vulnerability inside the network flexibly. In this architecture, standard protocols are used to maintain interoperability with the existing network facilities. Moreover, this architecture can incorporate an external vulnerability database to reflect the vulnerability information updates quickly.

The remainder of this paper is organized as follows. In Section 2, we analyze the requirements of vulnerability management. In Section 3, we describe the

†1 Graduate School of Information Science, Nara Institute of Science and Technology, Japan

proposed lightweight vulnerability management system. In Section 4, we evaluate the proposed system using the prototype system. In Section 5, we discuss related problems. In Section 6, we present conclusions.

2. Analysis of the Requirements

Generally, a vulnerability management system follows a procedure like the following. 1) It gathers the information on the software installed in the managed servers; 2) evaluates the vulnerability score; and 3) allows or denies connection. Existing methods, such as Cisco's NAC [2], use an agent approach that uses dedicated agent software installed on the computers. In the agent approach, when a computer connects to a network, the computer is authenticated by a switch and is assigned to the quarantine VLAN. In the quarantine VLAN, the computer is inspected for applied software updates and patches. If the computer passes the inspection, it is assigned the appropriate VLAN. The problem is that this approach requires IP connectivity between the computers and the quarantine server. If one computer has vulnerabilities and another is infected by malicious software, the vulnerable computer in the quarantine network may be infected by the malicious software. In addition, the agent software supports only major OSs and hardware. This approach is used mainly to check the vulnerability of clients.

Other methods, such as Nessus [3] and Retina [4], use an active scan approach that performs port scanning and detects the installed and running software to gather information about the vulnerabilities. The problem with this method is that there are some vulnerabilities that cannot be detected from outside of the computer. For example, if a server software uses vulnerable libraries, these vulnerabilities can be exploited even if the server software is not vulnerable in itself. In addition, the scan activity consumes resources such as the CPU and the memory and disrupts services running on the managed machines. This approach is useful for checking the vulnerability of servers.

Especially in the server farm, we have to continue to provide a service even if a server has some vulnerabilities when the server is providing an important service only if the vulnerabilities can be managed by operational tips. The existing approaches uniformly control the communication between computers based only on the vulnerability score, ignoring the importance of service and the security policy of the network.

These problems are summarized as follows.

P1 IP connectivity of the managed server is required, while gathering information about the software components. This may result in an infection by malicious software and the exploitation of software vulnerabilities.

P2 The active scan approaches cannot detect some vulnerabilities from outside of the server. In addition, the scan activity disrupts the legitimate services by consuming resources.

P3 There are barriers to developing, providing, and deploying agent software when the vulnerability management system uses proprietary protocols.

P4 The existing approaches control communication uniformly based only on vulnerability scores.

To solve the problems mentioned above, we have the following requirements for the vulnerability management system.

R1 The system does not require IP connectivity until vulnerability assessment completes.

R2 The system does not scan services running on the servers.

R3 The system complies with open standard protocols to be applied in the existing network environment.

R4 The system controls the connectivity based on both the vulnerabilities of respective components and the network security policy. This security policy should be customized site by site.

3. Proposal of a Lightweight Vulnerability Management System

We propose a lightweight vulnerability management system (LWVMS) that satisfies the requirements described in Section 2. This system assumes self-enumeration of the installed software and libraries, which does not involve active scanning. This LWVMS consists of the following four components.

LWVMS-C LWVMS-C is a software component installed on the managed server. This acts as an IEEE802.1X [5] supplicant that sends information about the software installed on the managed servers using standard IEEE802.1X EAP-TLS [6]. IEEE802.1X is an open standard for port-based network access control at network switches and wireless access points.

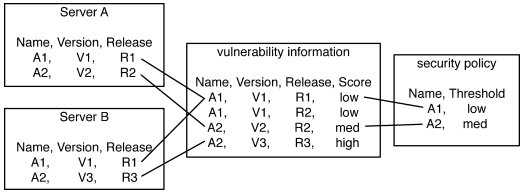

Fig. 1 Example policy.

LWVMS-S LWVMS-S is a vulnerability management server that can talk with LWVMS-C using RADIUS and EAP-TLS over authentication switches.

Authentication switch The authentication switch is an ordinary switch that complies with IEEE802.1X port authentication.

VWS VWS is a web service that can provide vulnerability information. The VWS stores the vulnerability information about a software, including the software's name, its version, its vulnerability description, and its severity score.

We assume the vulnerability score to be in the form of a Common Vulnerability Scoring System (CVSS) base score. The CVSS provides an open framework for communicating the characteristics and impacts of IT vulnerabilities. The CVSS consists of three groups: Base, Temporal, and Environmental. Each group produces a numeric score ranging from 0 to 10, and a vector, which is a compressed textual representation that reflects the values used to derive the score [7].

3.1 Goal of the LWVMS

The goal of the proposed LWVMS is to allow or disallow computers' connection to the network based on both the security policy and the vulnerability assessment. If the vulnerability score does not exceed the threshold in the security policy, the LWVMS-C is connected to the network. Otherwise, the LWVMS-C cannot connect to the network and cannot communicate with other computers in the network. For example, for the policy defined in **Fig. 1**, Server A can connect to the network because the security policy accepts the vulnerability scores of low and medium. On the other hand, Server B cannot connect to the network when the vulnerability score exceeds the threshold. This check is done regularly to maintain compliance with the network security policy. When vulnerability scores of the software are updated in VWS and a specific software exceeds the network security policy threshold, the server with that software is disconnected from the network.

3.2 Features

This LWVMS has the following features. 1) The vulnerability assessment is performed without IP connection to other computers. This can eliminate the risk of virus infection and invasion. This feature satisfies requirement R1. 2) Because LWVMS uses a self-enumeration approach to collect the vulnerabilities inside servers, LWVMS can get the vulnerability information about the libraries used by server software. Since LWVMS does not have to scan servers to check for vulnerabilities, legitimate services are not disrupted. This feature satisfies requirement R2. 3) LWVMS complies with standard IEEE802.1X EAP-TLS. This contributes to easing barriers to developing a customized LWVMS-C. As a result, LWVMS can be deployed in existing network environments equipped with retail IEEE802.1X compliant switches. This feature satisfies requirement R3. 4) The network security policy can be configured in LWVMS-S. An administrator can configure the security policy threshold for each software component. LWVMS controls the IP connectivity based on both the vulnerability scores of respective components and the network security policy. This feature satisfies requirement R4. 5) Managed servers are checked for compliance with the network security policy regularly. When a managed server violates the network security policy, it is disconnected from the network. Thus, LWVMS satisfies all the requirements described in Section 2.

3.3 Self-enumeration Approach

In the agent approach, the agent software must scan the registry and file system to search installed software and libraries. In the active scan approach, the scan activity consumes computing resources regularly. In addition, the scanned results are not always correct in both approaches. To address these problems, we developed a self-enumeration approach in which the administrator or package system reports installed components. This approach relies on the administrators of each PC to keep their networks secure.

3.4 Composition of the Certificates

First, all managed servers must have valid certificates: one certificate as their own server certificate and other certificates for their software components, such

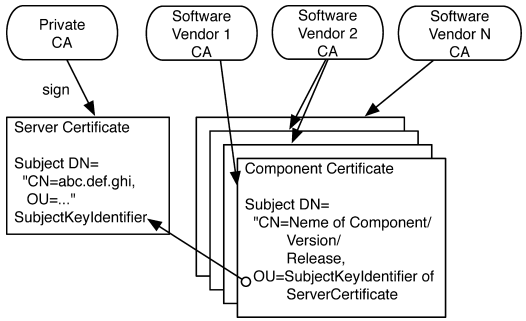

Fig. 2 Composition of certificates.

as server programs and libraries. LWVMS-C uses a series of the server certificate and component certificates. The server certificates are signed by the authority or network administrator (Private CA) to guarantee the identity of the server. The component certificates are signed by the distributor or developer to guarantee the identity of the component. These component certificates are issued to each component in each managed server. We assume these certificates comply with the X.509 [8),9)] specification. The server certificate is an ordinary Public Key Certificate (PKC) that has a subject DN pointing to the server and is issued by a trusted authority. Since in LWVMS, the name, version, and release of the components are used to obtain the vulnerability score, the component certificates have a CN including the component's name, its version, and its release in the subject DN field. Although the TLS session is executed based on the server certificate and the corresponding private key, the corresponding private keys of the component certificates are not used in TLS sessions. To prevent the abuse of the component certificates, they should be bound to the server certificate. To bind the component certificates to the server certificate, the component certificates have an OU pointing to the server certificates (**Fig. 2**). Since the component certificates are signed by the distributor or the developer, verifying these component certificates can prevent their abuse. Although these component certificates can be seen as Attribute Certificates (ACs) [10)], we used the PKC in our implementation because the PKC is easy to handle.

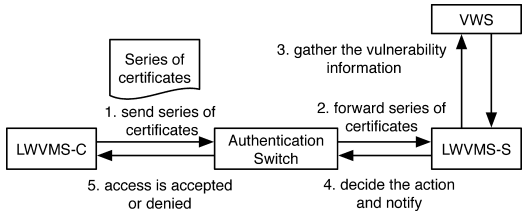

Fig. 3 Overview of workflow.

3.5 Workflow Overview

The managed servers (LWVMS-C) have to have their own server certificate as well as the signed PKC of LWVMS-S beforehand. In addition, LWVMS-C has to obtain component certificates issued to the software components. LWVMS executes the following steps to check the compliance with the network security policy regularly (**Fig. 3**).

(1) When the LWVMS-C connects to the network, it should submit a series of a server certificate and component certificates to the LWVMS-S through the authentication switch by using IEEE802.1X EAP-TLS. At this point, the LWVMS-C cannot communicate with other computers on the network. Because the LWVMS-C cannot access the vulnerability information, LWVMS-C only has to submit a series of certificates and delegate vulnerability assessment to the LWVMS-S.

(2) When the LWVMS-S accepts certificates from LWVMS-C, it separates them into the server certificate and the component certificates. Next, LWVMS-S verifies the validity of the server certificate using the PKC of private CA and the component certificates using the PKCs of software vendors.

(3) LWVMS-S gathers the vulnerability information for each component from the VWS.

(4) LWVMS-S gathers the vulnerability scores inside the LWVMS-C and compares these scores to the network security policy. If the vulnerability score matches the network security policy, LWVMS-S opens the port on the authentication switch to which LWVMS-C connects; otherwise, the port is closed.

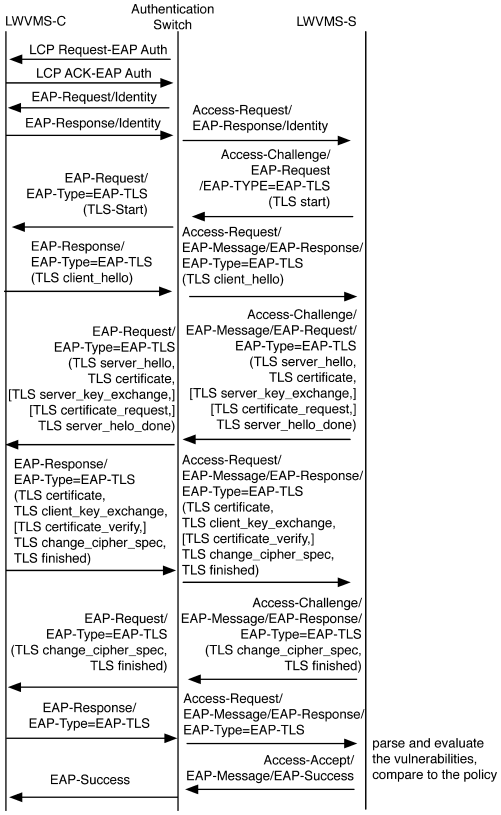

Fig. 4 Sequence overview.

(5) LWVMS-C determines from the authentication switch whether it can communicate with other computers through the authentication switch.

Note that communication between LWVMS-C and the authentication switch is performed by EAP, and does not require IP connectivity of LWVMS-C. Communication between the authentication switch and LWVMS-S is performed by RADIUS/EAP [11] (**Fig. 4**). As shown in Fig. 4, this sequence complies with ordinary IEEE802.1X EAP-TLS. Since the communication between the authentication switch and LWVMS-S has the RADIUS attribute EAP-Message, the authentication switch does not need to understand the contents of the packet. The authentication switch has to forward the packet from/to LWVMS-S and LWVMS-C. This enables existing IEEE802.1X compliant switches to be used with the proposed mechanism and the existing IEEE802.1X supplicant and authentication server (RADIUS server) to be used in this mechanism with slight modification.

4. Evaluation

To evaluate the proposed LWVMS, we implemented a prototype system and conducted experiments. We show the details of the implementation of the prototype system, along with quantitative and qualitative evaluation.

4.1 Prototype Implementation

We used open-source software to implement the proposed LWVMS. LWVMS-C was based on an open-source IEEE802.1X supplicant called Xsupplicant [12]. Since the component certificates are not relevant to each other, the original Xsupplicant cannot send a series of certificates that does not construct the CA path. We modified Xsupplicant to enable this series of certificates to be sent.

LWVMS-S was implemented based on an open-source RADIUS server called FreeRADIUS [13]. We added approximately 600 lines of code to the EAP-TLS module in FreeRADIUS to handle the series of the server certificate and the component certificates. The obtained certificates are validated based on the trusted CA described in the configuration file and are checked for the linkage with the server certificate. Then, LWVMS-S gathers the vulnerability score related to the component name, version, and release from the VWS. Finally, LWVMS-S compares the vulnerability score to the security policy to decide the action.

As an external VWS, we used Project SIX's Vulnerability Web Services (SIX-VWS) [14]. Since SIX-VWS provides an XML-RPC interface for external applications, we implemented an XML-RPC client in the LWVMS-S. Instead of retrieving the vulnerability information from SIX-VWS every time LWVMS-C connects, the VWS-watchdog module in the LWVMS-S fetches vulnerability information and caches it to a local database every 30 minutes. This interval can be changed in the configuration file. The architecture of this prototype is shown in **Fig. 5**.

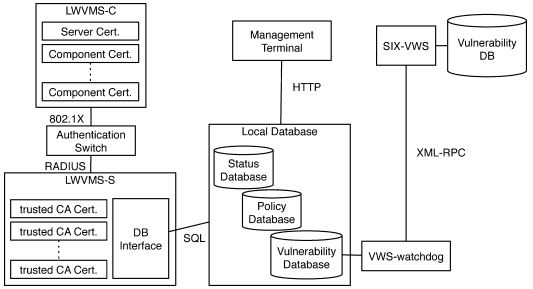

Fig. 5 Architecture of the prototype LWVMS.

Table 1 Configuration of prototype system.

CPU	Intel® Core Duo 2.0 GHz
memory	1 GB
OS	Fedora Core 6
Database	PostgreSQL
Authentication Switch	HP® ProCurve 3400cl

Table 2 Components used in the experiment.

Name	Version	Release
Apache HTTPd	2.2.3	0
PostgreSQL	8.1.3	0
Tomcat	5.5.9	0
BIND	9.3.0	0
Postfix	2.1.3	0
Qpopper	4.0.7	0

4.2 Experimental Set-up

The prototype system consists of four PCs, one for LWVMS-S, one for LWVMS-C, one for SIX-VWS, and one for the management console. All of the PCs have an Intel Core Duo 2.0-GHz CPU, 1 GB of memory, and Fedora Core6 and are connected to HP's ProCurve 3400cl authentication switch using 1000Base-T Ethernet. The configuration of the prototype system is listed in **Table 1**.

We used several component certificates to check the functionality of the prototype system, as listed in **Table 2**. In addition, we created a security policy that allows and disallows network connectivity, as listed in **Table 3** and used vulnerability information from NVD [15]. For the purpose of demonstration, we categorized the vulnerability score into three levels: low, medium, and high.

Table 3 Security policy used in the experiment.

Name	Threshold
Apache HTTPd	med
PostgreSQL	high
Tomcat	med
BIND	low
Postfix	med
Qpopper	high

Table 4 Comparison between LWVMS and existing approaches.

	LWVMS	agent approach	active scan approach
R1	yes	no	no
R2	yes	yes	no
R3	yes	no	no
R4	yes	no	no

4.3 Experimental Results

Although during the vulnerability assessment, managed servers cannot communicate with any other computers using IP, LWVMS-S can gather information about the installed software components. This satisfies requirement R1. Since managed servers have component certificates and have to report these certificates to LWVMS-S, there is no need to scan services from outside the managed servers. This satisfies the requirement R2. Based on the experimental results, we confirmed that the prototype system can control the connectivity according to the vulnerabilities of the respective components. When the managed server has vulnerable software components that violate the security policy threshold, it cannot communicate with any other computer. When the vulnerability information is updated, the information is reflected in the refresh interval of the local database. This satisfies the requirement R4. This experiment can be performed using an ordinary authentication switch and open source software because LWVMS complies with standard protocols. This satisfies the requirement R3. From the results, this LWVMS satisfies all the requirements. The comparison between the LWVMS and the existing approaches is summarized in **Table 4**.

Although the processing time of the vulnerability assessment may vary due to the implementation of the RADIUS server, the required time is at most one

second. The variation stems mainly from fragmentation and the interval of acceptance of the RADIUS packet. We conclude that the proposed mechanism satisfies all the requirements and is practical.

5. Discussion

Through this study, we found several issues to be considered. From an operational viewpoint, VWS should be trusted by the network administrator. LWVMS should authenticate VWS and the traffic from the VWS must be encrypted to ensure the consistency of the information using IPsec and SSL. Additionally, issuing certificates is a very cumbersome task. To reduce the overhead, we provided a web interface to issue server certificates and component certificates semiautomatically.

In this proposal, we assumed that the administrators of each server in a server farm are willing to cooperate to secure their network environment. The key point of the self-enumeration approach is how honest every administrator can be. As mentioned in Section 3.3, the LWVMS trusts the administrators of each server and does not require tight coupling of a component and its certificate. The component certificates are bound to the server certificate, and the paired secret keys of the component certificate are not used in the process. There is no means to guarantee the integrity of the component and its certificate. We hope this matter can be solved by using attestation [16] and the integrity check function of TPM [17], but other mechanisms must be developed in order to guarantee the integrity of the installed software and libraries.

Vulnerability assessment must be done regularly. In LWVMS, configuring the re-authentication interval of the authentication switch enables vulnerability checks to be performed regularly.

6. Conclusions

To solve the problems of existing heavyweight vulnerability management systems, we proposed a lightweight vulnerability management system. This system assumes a self-reported method to gather information about the components installed in the computers. Since this system complies with standard protocols and technology, it can be applied to existing network environments. Through an experiment using a prototype system, we demonstrated the usefulness of the proposed LWVMS. To make reproducibility experiment easier, we made the prototype software used in our experiment available on our web site [⋆1]. The next step is to integrate other technologies such as attestation and trusted computing technologies.

⋆1 http://iplab.naist.jp/research/#pki

References

1) Martin, R.A.: Managing Vulnerabilities in Networked Systems, *Computer*, IEEE, pp.32–38 (2001).
2) Cisco Systems, Inc.: Network Admission Control (NAC) Framework. http://www.cisco.com/
3) Tenable Network Security: Nessus Vulnerability Scanner. http://www.nessus.org/
4) eEye Incorporated: Retina Security Scanner. http://www.eeye.com/html/products/index.html
5) IEEE: 802.1X-2004: IEEE Standard for Local and metropolitan area networks Port-Based Network Access Control (2004).
6) Aboba, B. and Simon., D.: PPP EAP TLS Authentication Protocol, RFC 2716 (Experimental) (1999).
7) Chandramouli, R., Grance, T., Kuhn, R. and Landau, S.: Common Vulnerability Scoring System, *IEEE Security and Privacy*, Vol.4, No.6, pp.85–89 (2006).
8) Housley, R., Polk, W., Ford, W. and Solo, D.: Internet X.509 Public Key Infrastructure Certificate and Certificate R evocation List (CRL) Profile, RFC 3280 (Proposed Standard) (2002). Updated by RFCs 4325, 4630.
9) Housley, R. and Santesson, S.: Update to DirectoryString Processing in the Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile, RFC 4630 (Proposed Standard) (2006).
10) Farrell, S. and Housley, R.: An Internet Attribute Certificate Profile for Authorization, RFC 3281 (2002).
11) Aboba, B. and Calhoun, P.: RADIUS (Remote Authentication Dial In User Service) Support For Extensible Authentication Protocol (EAP), RFC 3579 (Informational) (2003).
12) Open 1X project: Open1X.org. http://open1x.sourceforge.net/
13) The FreeRADIUS Server Project: FreeRADIUS The world's most popular RADIUS Server. http://www.freeradius.org/
14) Project SIX: SIX Vulnerability Web Services. http://staff.aist.go.jp/nakamura-akihito/six/
15) National Institute of Standards and Technology: National Vulnerability Database.

http://nvd.nist.gov/
16) Maruyama, H., Seliger, F., Nagaratnam, N., Ebringer, T., Munetoh, S., Yoshihama, S. and Nakamura, T.: Trusted Platform on Demand (TPod), IBM Research Report RT0564 (2004).
17) Trusted Computing Group: TPM Main Part 1 Design Principles (2007).

(Received November 30, 2007)
(Accepted June 3, 2008)
(Released September 10, 2008)

Takeshi Okuda received the M.E. degree in information science from Osaka University in 1998. He is currently an assistant professor in the Graduate School of Information Science, Nara Institute of Science and Technology, Japan. His research interests include network security, mobile agent technology and multimedia application. He is a member of the IEEE.

Suguru Yamaguchi received the M.E. and D.E. degrees in computer science from Osaka University, Japan, in 1988 and 1991, respectively. From 1990 to 1992, he was an Assistant Professor at the Education Center for Information Processing, Osaka University. From 1992 to 1993, he was with the Information Technology Center, Nara Institute of Science and Technology (NAIST), Japan, as an Associate Professor. Since 1993, he has been with the Graduate School of Information Science, NAIST, where he is now a Professor. Since April 2004, he has been an Advisor on Information Security in the Cabinet Secretariat of the government of Japan. His research interests include technologies for information sharing, multimedia communication over high-speed communication channels, network security and network management for the Internet.