Security Analysis of Access Linux Platform

Andreas Sjöström[1], Kazuhide Fukushima, Shinsaku Kiyomoto, Wook Shin and Toshiaki Tanaka

KDDI R&D Laboratories Inc., Fujimino, Japan

Summary
The complexity of the mobile phones has led to that many mobile phone manufacturers have entered into projects where the Linux operating system is made as a base for their mobile platforms. In this paper, we present some potential vulnerabilities of ALP 2.4 originate in the Linux kernel and propose countermeasures against them. Our proposal to eliminate these vulnerabilities contributes secure mobile services on ALP 2.4 or subsequent LiMo.

Key words:
Access Linux, Security, Open Platform, Mobile, Vulnerability

1. Introduction

Advances in hardware manufacturing technology enable mobile phones to be equipped with various functional modules (e.g. touch screen, camera, accelerometer, GPS, etc.). The evolution of mobile hardware and services has resulted in alliances of handset manufacturers and telephone companies. They began to collaboratively define a common open framework, in order to become more adaptive, embracing rapid changes in customer demands, services, and hardware specifications. Moreover, an open framework allows them to avoid proprietary issues and to attract third party developers to their ecosystem.

The efforts of defining an open mobile platform have naturally led to use an open source operating system, Linux, and resulted in a number of products and projects [1, 15, 20, 21, 20, 27]. Those Linux-based mobile platforms tend to have a similar composition: fundamental operating system services are provided by Linux kernel, and a GTK+/Qt-based or Java-compatible application development environment is placed on top of the kernel. Although they have some architectures in common, detail specification could be different. For example, Android [21] is working quite differently from LiMo [15]. In case of the former, all the programs from third party developers are programmed in Java and each of them is running on a separate virtual machine, being isolated from each other. The latter platform is working in a more similar way as ordinary Linux and focuses on using open source components that have already been well tested and are known to be secure.

This paper is focusing on security analysis of the ALP version 2.4. This version of ALP can be seen as a predecessor of LiMo, having the Hiker Framework; the heart of both systems. ALP is developed and maintained by Access, a company that since then has become a main actor in the LiMo project. The latest version of ALP, ALP 3.0, is actually LiMo compliant [1]. Unfortunately, no development kits for LiMo have yet been released for public use. While we evaluate the security of ALP, known security issues of the Linux operating systems and the security mechanisms employed in ALP are taken into account. Possible security threats, countermeasures to those threats, and potential vulnerabilities in other platforms are discussed in the following sections.

The outline of this paper is as follows: Section 2 introduces security issues in Linux system and Section 3 explains security architecture of ALP. Thereafter we discuss some possible security threats of ALP and countermeasures to them in Section 4 and Section 5, and conclude the paper in Section 6.

2. Related Work

Although worries about security of the Linux-based mobile platforms have grown because of their inherent open features, the security of those systems has not been intensively studied. On the contrary, there are some research achievements on other mobile operating systems: Debbabi *et al.* provided a vulnerability analysis of J2ME platform [8], and showed that some mobile phones face serious threats such as the Siemens SMS attack. Kingpin and Mudge analyzed the design of the Palm OS and hardware platform with respect to data storage issues, improper security design, and malicious code threats [14]. Goovaerts *et al.* presented the in-depth analysis of different vulnerabilities in Palm OS [10]. Murmann and Rossnagel examined current mobile operating systems such as Symbian OS, with regard to their security and concluded that no operation system is secure enough for open environments [19]. Thereupon, in the following sections, we are going to discuss the security of Linux-based mobile framework, especially take an example of the framework, Access Linux Platform (ALP), into account.

[1] Currently, he belongs to Integrated Electronic System Design at Chalmers University of Technology in Gothenburg, Sweden.

Manuscript received May 5, 2010
Manuscript revised May 20, 2010

3. Security in Linux

In this section, we present the vulnerabilities that may be considered to be the dominating ones of which malware makes use on the Linux platform. We then demonstrate that some of them are available on ALP as well and some further design patterns can be seen as flaws. Finally, we discuss the influence exerted by these vulnerabilities.

3.1 Overview

Linux has a strict policy on which users are allowed to use the system based on their permissions according to the POSIX permission standard. When the kernel version 2.6 was released, an access control system, Linux Security Modules (LSM)[32] was merged into Linux kernel. This is a framework that combined with a security module which defines certain policies strengthens the security control over a Linux system.

3.2 Common Exploits in Linux

There have been several ways to subvert the Linux system during its lifetime of almost twenty years. Different exploits can be split up into two categories, those that only work in userland and those that take control in kernelland. The former ones can be protected from fairly easily as long as the kernel is still intact, whereas the latter one can cause a lot of trouble since the main control of the system is then compromised. Two well known ways of infiltrating the kernelland are to exploit kernel modules and to directly patch the memory data[18]. Especially, Linux Kernel Module (LKM)[24] and /dev/(k)mem files[5, 17] have been widely exploited in the Linux systems, and those techniques are often followed by malicious code injection[28]. They are explained in more detail below.

Linux Kernel Modules (LKM): In order to subvert a kernel one will have to get access to the area where the kernel is working in the memory [24].LKMs are programs written with the intention to extend the kernel [24]. They are mainly used for the integration of device drivers into the system in an easy way. Malware writers have, however, found more malign usage of this feature [24]. They have naturally discovered that it gives a good opportunity to write malwares such as rootkits, which have the possibility to partially or completely subvert and take over a system.

Access to Memory Device: The access to the memory device (/dev/mem and /dev/kmem) was originally meant for debugging purposes, but it has been up to debate for a long time about what their features are really useful for. The abuse of /dev/kmem was firstly proposed by Silvio Cesares in his paper ``Runtime Kernel Kmem Patching"[5]. What the feature gives is the possibility to map to the part of the memory where the kernel lives. Several rootkits came to use this feature, which led people to doubt whether it actually was useful for any other type of programs. Hence, /dev/kmem became disabled in version 2.6 or the Linux kernel. However, this modification could not quite stop the rootkits, since there was still the /dev/mem. As soon as /dev/kmem was deprecated, /dev/mem was introduced. Comparing to /dev/kmem which only mapped to the virtual memory of the kernel, /dev/mem maps to the whole memory in the system. Anthony Lineberry presented a method on how to use /dev/mem for malware purposes in his paper ``Malicious Code Injection via /dev/mem" [17], demonstrating how it still is quite possible to subvert the Linux kernel even now without too much effort. Some Linux developers have taken this problem seriously and several Linux distributions grant users limited access to the first megabyte in the memory. This memory area is actually used by some mainstream applications, but the limitation is not set as default in the mainline[17] which has consequently led to that distributions where this problem has not specifically been addressed are still vulnerable.

Code Injection: Rootkits often replace system hooks and redirect the system control flow in order to execute hidden malicious procedures. Code injection is one of the methods of hiding malicious codes in legitimate processes. While the Dynamic Link Libraries (DLLs) are usually victimized in Windows environments, the LKMs running in the kernel level is usual targets of the code injection in Linux environments. ELF stands for Executable and Linkable Format and is the file format that Linux uses for its executable files, including LKMs. By modifying the symbol table of an ELF, an attacker can execute an arbitrary function in the ELF-file during a function call. Combining this together with the Linux supported feature of uniting several ELF-files one can inject an ELF-file with harmful code.

4. ALP Security Framework

ALP supports various sorts of executables such as Linux native applications, Widgets, legacy Palm OS applications, and Java MIDlets. In result, the architecture of ALP has several sub-platforms on top of the Linux kernel such as NetFront Widgets, Garnet OS, and JV-Lite which supports the former executable formats, respectively. The task of coordinating all the different application types to give a common view into the system is the job of the Hiker Framework, which also directly handles the native applications and the communication between them. In this paper we only discuss the Hiker Framework [3] upon which the security architecture of the ALP system is based on.

Figure 1 shows an overview of the ALP Platform with a focus on the Hiker Framework, which consists of several supporting blocks for an application's execution. An application package (called a bundle) is added to the system by the Bundle Manager and runs while its life cycle is managed by the Application Manager. Interactions between applications are intermediated by a broker, the Exchange Manager which binds one application's request to another application. Events to an application are informed by the Notification Manager or the Alarm Manager whereas those going to the user are handled by the Attention Manager. To provide persistence in applications' status, they can store setting information in the Global Settings.
All the above behaviors are governed by the Security Policy Framework (SPF) which enforces fine-grained access control. The security policy can be provided by the device manufacturers or the network operators. The enforcement of the policy is based on an LSM specifically made for ALP [2] together with a policy file which is mapped with the standard POSIX permissions [30]. Each loaded application is launched with a unique user-ID, thus distinguished and separated from other applications.

4.1 ALP LSM and Policies

The most essential part of the security framework is the ALP LSM with policies. Access has developed their own LSM for the Linux kernel, adapted specifically for the needs of ALP. The hooks in the kernel that the LSM provides connects to a policy file written in XML [2]. Whenever an application wishes to access a file or feature in the phone the kernel will call the LSM hooks which check with the policy file whether the request for the running application should be granted or not. The policy file describes three things:

PIN and PUK codes: These two fields contain information about the PIN and PUK codes. The respective fields contain descriptions about the iteration count the salt and finally also their encrypted passwords.

Policies: There can be several policies described, depended on the decisions from the mobile operator. In the default version from ALP there are six policies described. Each policy describes the privileges that should be given to the bundles granted the policy, including the GID (Group ID), SGID (Set Group ID), the level of trust and also specific access level to the components of the mobile framework.

Packages: Finally, the mapping of each bundle (Sometimes a bundle is referred to as a package in ALP) in the system to a policy is made. The unique user-id for each bundle is also listed here.

4.2 The /etc/passwd file

In the past, the password for each user in Linux was described in /etc/passwd. Eventually, the passwords were moved out, but the file is still there to describe the users in the system. Since ALP represents each application as a unique user, a lot of information is saved in this file. Each user-id and group-id is also described here.

4.3 Authentication Methods

According to source codes [3], two well-known standards, Password-Based Cryptography Specification (PKCS#5) and Secure/Multipurpose Internet Mail Extensions (S/MIME), are being used in ALP. PKCS#5 is used for the login procedure of the mobile phone. S/MIME is used for signing and certification of the installed applications.

PKCS#5: The dictionary attack is a common way to intrude on a password protected system. The PKCS#5 [26] suggests a solution to this problem by making sure that the pass phrase the user inputs are modified until it is no longer recognized as a non-random phrase when it finally arrives at the system [33]. This mechanism makes it very hard to brute force the password since an attacker has pretty much guess phrases randomly among a massive amount of possible values. The PKCS#5 consists of a key derivation function (KDF) which consists of a pass phrase, a salt and an iteration number. The pass phrase p is the password the user verifies herself with. ALP is used in two scenarios: when inputting the PIN code after the mobile has been booted and when inputting the PUK code in the case when the verification of the PIN code has failed too many times. The salt s is added to the KDF in order to enlarge the amount of possible values that can come out from the function. It is usually a wide number and the recommendations are to let it have a minimum width of 64 bits. Finally, the underlying function $H(x,y)$ of the KDF is executed sequentially a certain number of times, defined by the iteration count c. The recommended minimum value of c is 1000. The intent of the iteration count is to prolong the process of brute forcing with a dictionary attack whereas the procedure of computing a key legitimately is still feasible in time since it is only done once. In effect, the adding of an iteration count is equivalent to lengthen the width of the key with ($\log c$) amount of bits [3]. In PKCS#5 v2.1, there are specifications for two underlying password-based key derivation functions (PBKDF): PBKDF1 and PBKDF2. The first one applies a hash function, which is either MD2, MD5, or SHA-1. Since underlying hash functions are no longer recommended for new applications, PBKDF1 is now obsolete and should only be used for applications that should be compatible with existing ones. Furthermore, the derived key length is bound to 16 octets for MD2 and

MD5 and 20 octets for SHA-1. PBKDF2 is recommended for new applications. Despite the recommendation of using PBKDF2 the Hiker Framework is using the PBKDF1 key derivation function.

S/MIME: Each bundle should come together with a certificate and a signed key. It will run with the lowest privileges[30] without a certificate. The certificate and key are attained from a certificate coordinator, and are using the S/MIME standard.

5. Security Threats in ALP

The following security threats were found by evaluating the some of the most common known vulnerability classes in Linux systems, which were presented in the previous section. The following data has been attained by using the development kit, including a simulator for the platform supported by Access for ALP. The policies and the /dev/mem were modified through the simulator using basic Linux applications and the gcc compiler, whereas other vulnerabilities were found statically by examining the source code. The complete source code that was studied comes from version 0.9.1 of the Hiker Framework [3], whereas the actual version used in ALP 2.4 is a bit newer. The version number is not mentioned in the documents published by Access. There is, however, just a slight difference and a lot of it can be checked by comparing the code from version 0.9.1 with the header files in the Hiker Framework contained by ALP 2.4. In this section, five main concerns are addressed, in regard to the Linux vulnerabilities listed in Section 2 and the ALP security mentioned in Section 3:

- Root Access: The possibility for a user to become the super user.

  Policy Modification: The possibility to change a policy by the user.

  /dev/mem: The possibility to access all the memory on the platform and what consequences that can lead to.

  Flaws in PKCS#5: The problems with using PKCS#5 in the way it is used for ALP.

  Code injection: The possibility for code injection in ALP.

5.1 Root Access

One of the strengths in any Unix system is its robust permission management system. A user is only allowed to work within the limitations of the scope it has been allowed to work in. The system has at least one user that is given super user capabilities, which is called *root*. The root user has the ultimate control over the system and is the one with the capabilities to decide over the other users' access over the system, the installation of applications and configurations of the system. ALP allows the shell access to the device via a terminal. This is simply done by connecting the device to a computer via USB, and opening a telnet session to the device [31]. In the simulator, super user privileges could be attained simply by connecting to the terminal. If this is the same in the actual product it can lead to serious matters. Besides the shell access gives an attacker a chance to modify the system, it simply reveals the phone owner's private information. Assets like contacts, mails, documents, and media files can be directly accessed via the shell. This makes losing a cell phone be more than the loss of the expensive handset product.

Some of the vulnerabilities we mention below assume that an attacker already has taken the super user privilege. Those vulnerabilities are worth considering, since we might want to compensate those defects by applying other techniques (like cryptography), even after the attacker acquired the super user privilege.

5.2 Policy and User Access Modification

Security policy enforcement is the main way of restricting processes' behavior; thus, the stored policy is indeed essential to the security in ALP. That is why it is very important to maintain the integrity of the policies. However, it is very easy for a user or application with root access to modify the policies, since they are simply represented as strings in an XML [3] file which is not at all protected from root. By modifying the policy file and the /etc/passwd file one can widen the limitations of an application and change its user id. If the new user id would be changed to 0 the application will work as a super user.

5.3 Access to /dev/mem

Modifying /dev/mem can as already mentioned give a user or application the ultimate control over a system. This is of course something that a mobile platform, with the purpose of protecting the integrity of the mobile operators, has to protect itself from. However, the only protection that ALP has is the access control mechanism which was already shown to be vulnerable in the last subsection. If an attacker has come as far as exploiting the possibility, he or she can modify the kernel memory and get full control of the system, by e.g, searching for and modifying the sys_call_table to run malicious code instead of the intended system functions [18]. This is of course unacceptable.

5.4 Flaws in PKCS#5

The point of using the PKCS#5 as an authentication method is to turn the password into a key phrase that appears to be just a random sequence. In PKCS#5 v2.1, the recommendations are to have an iteration count of at least 1000, a salt with the width of at least 64 bits and to use the PBKDF2 as its underlying encryption function. ALP is using the PBKDF1 as its underlying function which relies on hash functions that are no longer considered to be safe. Even though it in most cases is unlikely that a mobile will be the victim to an attack that possibly could break the authentication algorithm it makes little sense to just have an incomplete certification method, which the PKCS#5 in ALP correctly can be considered to be.

5.5 Code Injection

Although ALP does not support the dynamic LKM architectures, code injection exploits could still be in effect. ALP applications can be compiled and installed as shared object formats (*.so files). Some known variations of the attack exploit these files [25] and can well be used to attack ALP.

6. Consideration

In this section, some possible solutions to the problems mentioned together with some suggestions for improvement are presented.

Disabling root ccess: A good thing to consider is if there is actually any need for a user to ever run a root, or even to run in a shell. Most of the mobile phones out in the market do not support those features. If there actually is a need for this the user can at least be restricted access to the components that the Hiker Framework consists of. The system at least needs to hide user's private data, so that the confidentiality and the privacy could be guaranteed even when the phone is lost. This could be fundamentally supported by the cryptographic file systems.

Checking the integrity of the policies: The system can protect the integrity of the policies by calculating hash digests of each policy or whole policy file. Whenever a program needs to access the policies, the operating system compares the digests of them with the previous values.

Restricting access to /dev/mem: Many Linux desktop distributions disable access to /dev/mem even for the super user. Those applications that need access are limited to the first megabyte in the memory [17], where actually legitimately useful data lies. Not very long ago this was not supported by the mainline of the kernel [7], but since a while back it gives the option in the source code to enable the preprocessor directive CONFIG_STRICT_DEVMEM [7]. This is something the developers of ALP should have considered already from the start.

Changing the underlying function of PKCS#5: In order to strengthen the cryptographic function of PKCS#5 the recommendations to apply an underlying function of PBKDF2 should be followed since the authentication mechanism in ALP is not depended on any older application.

Checking the integrity of the system: We can check the integrity of the ALP components by using checksums. If some files are modified, the new checksum of the system does not match the previous checksum. This process should be done regularly, e.g. during every system boot. Conveniently, checksums come together with most bundles when they are installed into the system and can hence be used for this purpose. Alternatively, a system program (e.g. Tripwire integrity checker) can be used, which saves a hash of each file to use as a reference when performing a regular system check. The hashes can be made for a component as soon as it is added to the system to make sure they are made from a ``safe'' component. This is a good method to prevent the before-mentioned code injection technique.

7. Conclusion

We presented some potential vulnerabilities of ALP 2.4 originate in the Linux kernel and proposed countermeasure against them. We showed that an access control mechanism provides a secure environment in ALP 2.4. Without an access control mechanism, an attacker can easily compromise the security mechanism by modifying the policy files that are essential for ALP LSM. Next, an attacker can control the whole system of ALP by accessing the /dev/mem, which directly maps physical memory. As the result, an attacker can inject malicious code to other executables. This vulnerability can be eliminated by limiting shell access to mobile phones. An access control system that prohibits access to /dev/mem and permits partial access to the first megabyte of the memory is also beneficial; it is already used for several Linux distributions. Furthermore, we found a potential vulnerability in the authentication mechanism; its password-based key derivation PKCS#5 uses an obsolete function PBKDF1 including old hash functions MD2, MD5, and SHA-1. PBKDF2 is recommended for new applications and systems. Although it may be hard for an attacker to get physical access to another person's phone via these vulnerabilities, they still can be exploited to ``hack'' his or her own mobile phone, for example, compromise the copyright protection mechanism. Thus, the elimination of

these vulnerabilities is essential to provide secure mobile services on ALP 2.4 or subsequent LiMo.

References

[1] ACCESS Co., LTD. The access linux platform.
[2] ACCESS Co., LTD. Access application framework - Technical overview. (Version 0.9.1), Dec 2006.
[3] ACCESS Co., LTD. hiker-0.9.1 source codes, 2006.
[4] Owen Arden and Charlie Miller. Opencore insufficient boundary checking during mp3 decoding. oCert #2009-002, http://www.ocert.org/advisories/ocert-2009-002.html, 2009.
[5] S. Cesare. Runtime kernel KMEM patching. http://biblio.l0t3k.net/kernel/en/, Nov 1998.
[6] corbet. Complete coverage in linux security modules. http://lwn.net/Articles/154277/, Oct 2005.
[7] corbet. Who needs /dev/kmem? http://lwn.net/Articles/147901/, Aug 2005.
[8] M. Debbabi, M. Saleh, C. Talhi, and S. Zhioua. Java for mobile devices: A security study. Proc. of the 21st Annual Computer Security Applications Conference (ACSAC 2005), pages 234–244, 2005.
[9] Antony Edwards, Trent Jaeger, and Xiaolan Zhang. Runtime verification of authorization hook placement for the linux security modules framework. In ACM conference on Computer and Communications Security, pages 225–234. ACM, 2002.
[10] T. Goovaerts, B. D. Win, B. D. Decker, and W. Joosen. Assessment of palm os susceptibility to malicious code threats. Proc. of the 9th IFIP TC-6 TC-11 Conference on Communications and Multimedia Security (CMS2005), 3677, LNCS:240–249, 2005.
[11] Michael Ihde and Tom Brown. An experimental study of file permission vulnerabilities caused by single-bit errors in the selinux kernel policy file. UIUC SIGMIL Meeting, 2004.
[12] Trent Jaeger, Antony Edwards, and Xiaolan Zhang. Consistency analysis of authorization hook placement in the linux security modules framework. volume 7, pages 175–205. ACM, 2004.
[13] Kingcope. Debian openssh selinux privilege escalation vulnerability. SecurityFocus, Bugtraq ID: 30276, Jul 2008.
[14] Kingpin and Mudge. Security analysis of the palm operating system and its weaknesses against malicious code threats. Proc. of the 10th conference on USENIX Security Symposium, pages 135–151, 2001.
[15] LiMo Foundation. Limo. http://www.limofoundation.org.
[16] LiMo Foundation. Security policy enforcement framework (SPEF) foundation api. (Version 1.0), Mar. 2008.
[17] A. Lineberry. Malicious code injection via /dev/mem. 2009.
[18] Anthony Lineberry. Alice in kernel land. Black Hat Europe, Apr 2009.
[19] T. Murmann and H. Rossnagel. How secure are current mobile operating systems? Proc. of the 8th IFIP TC-6 TC-11 Conference on Communications and Multimedia Security (CMS2004), pages 47–58, 2004.
[20] Nokia. Maemo. http://maemo.org/.
[21] Open Handset Alliance. Android. http://developer.android.com/.
[22] Openmoko, Inc. Openmoko. http://www.openmoko.org/.
[23] Alfredo Ortega. Android web browser gif file heap-based buffer overflow vulnerability. Bugtraq ID 28005, http://www.securityfocus.com/bid/28005, Mar 2008.
[24] Rául Siles Peláez. Linux kernel rootkits: protecting the systems ``ring-zero". GIAC Unix Security Administrator (GCUX), May 2004.
[25] Quake2th. PLT redirection through shared object injection into running process. http://www.codeproject.com/KB/cpp/shared_object_injection_1.aspx, Dec 2008.
[26] RSA Laboratories. PKCS #5 v.2.1: Password-based cryptography standard. 1999.
[27] The Linux Foundation. Moblin. http://moblin.org.
[28] truff. Infecting loadable kernel modules. Phrack Issues #61, Volume 0x0b, Issue 0x3d, Aug. 2003.
[29] US-CERT/NIST. Linux 2.6 kernel capability lsm module local privilege elevation National Vulnerability Database, CVE-2004-1337, Dec 2004.
[30] Greg Wilson. Access linux platform application programming. (Document Number3136-001), Jul 2008.
[31] Greg Wilson and Jeffrey Osier-Mixon. Development tools. (Document Number 3151-001), Jul. 2008.
[32] C. Wright, C. Cowan, and J. Morris. Linux security modules: General security support for the linux kernel, 2002.
[33] F. F. Yao and Y. L. Yin. Design and analysis of password-based key derivation functions. 51(9):3292–3297, 2005.

Andreas Sjöström has a B.E in Computer Engineering and is currently studying his M.E in Integrated Electronic System Design at Chalmers University of Technology in Gothenburg, Sweden. He was a trainee at the Information Security lab in KDDI R & D Laboratories, Inc. in the fall of 2009.

Kazuhide Fukushima received his M. E. in Engineering from Kyushu University, Japan, in 2004. He joined KDDI and has been engaged in research on digital rights management (DRM) technologies, including software obfuscation and key-management schemes. He is currently a researcher at the Information Security Lab. of KDDI R&D Laboratories, Inc. He received his Doctorate in Engineering from Kyushu University in 2009. He is a member of Institute of Electronics, Information and Communication Engineers, the Information Processing Society of Japan, and ACM.

Shinsaku Kiyomoto received his B.E. in Engineering Sciences, and his M.E. in Materials Science, from Tsukuba University, Japan, in1998 and 2000, respectively. He joined KDD (now KDDI) and has been engaged in the research on stream cipher, cryptographic protocol, and mobile security. He is currently a senior researcher of the Information Security Lab. in KDDI R&D Laboratories, Inc. He received the Dr. degree in engineering from Kyushu University in 2006. He was a visiting researcher of the Information Security Group, Royal Holloway University of London from 2008 to 2009. He received the Young Engineer Award from IEICE in 2004. He is a member of the Physical Society of Japan and Institute of Electronics, Information and Communication Engineers.

Wook Shin is a researcher at KDDI R&D Laboratories, Inc. His research interests include secure operating systems, web services, and healthcare systems. He received his BA from Dongguk University, Korea and MS and PhD from Gwangju Institute of Science and Technology, Korea. He was a postdoctoral research assistant at University of Illinois at Urbana-Champaign, U.S.A.

Toshiaki Tanaka received B.E. and M.E. degrees in communication engineering from Osaka University, Japan, in 1984 and 1986 respectively. He joined KDD (now KDDI) and has been engaged in the research on cryptographic protocol, mobile security, digital rights management, and intrusion detection. He is currently an executive director in KDDI R&D Laboratories, Inc. He received his doctorate of engineering from Kyushu University in 2007. He is a member of Institute of Electronics, Information and Communication Engineers, the Information Processing Society of Japan.