http://ansinet.com/itj

ITJ

ISSN 1812-5638

INFORMATION TECHNOLOGY JOURNAL

ANSInet
Asian Network for Scientific Information
308 Lasani Town, Sargodha Road, Faisalabad - Pakistan

Information Technology Journal 9 (2): 395-398, 2010
ISSN 1812-5638

A Novel Protocol for Software Authentication

[1,2]Rongyu He, [1]Zheng Qin and [2]Shaojie Wu
[1]School of Electronic and Information Engineering,
Xi'an Jiaotong University, Xi'an, 710049, Peoples Republic of China
[2]School of Electronic Technology, Information Engineering University,
Zhengzhou, 450004, Peoples Republic of China

Abstract: The authenticity and integrity of software running on mobile equipment is relevant and important in m-commerce. Mobile trusted computing can solve the problem by using Reference Integrity Metric (RIM) certificate. But the RIM certificate stored in Mobile Trusted Module (MTM) is suffered to frequently renew while the software is updated or patched. In the study, a user-specific RIM, uRIM, is presented. Based on the uRIM, a novel software integrity verification protocol is proposed. It allows an easy management of RIM to support the secure boot as well as a low-cost on verifying of software authenticity.

Key words: Reference Integrity Metric (RIM), secure boot, trusted mobile module, authenticity, authentication factor

INTRODUCTION

The authenticity and integrity of software running on mobile equipment is relevant and important in the distributed applications with high security requirements, such as e-Bank. Secure boot introduced in Mobile Phone Work Group (MPWG) could provide a high-security environment that only legitimate and authorized software can be loaded. During secure boot process the integrity of a pre-defined set of system components is measured and these measurements are then compared against a set of expected measurements, known as the Reference Integrity Measurement (RIM). If, at any stage during the boot process, the removal or modification of a platform component is detected, the boot process is aborted. Here, the RIM of a software component is the cryptographic digest or hash of the component (Gallery, 2007). In order to validate the RIM itself, it must be signed by a trusted entity by using asymmetric cryptography. The signed RIM is RIM certificate, abbreviated as RIM_Cert.

Secure boot functionality can detect the malicious or accidental modification or removal of security-critical software at boot time. But the MTM, anchor of the mobile equipment, must be securely stored and accessed the RIM_Cert (Gallery and Mitchell, 2007) and must be able to get the new RIM_Cert while the software is updated or patched. However, secure boot is a new feature in the specifications of MPWG (TCG, 2007a, b) and the MPWG isn't specified how the RIM_Cert is deployed to the mobile equipment and where the boot-loader can locate these RIM_Cert.

The MPWG has suggested storing the RIM_Cert into the MTM and assuming that only authorized entities can update or store it. But the software components running on the platform is subject to frequently patching and updating and the corresponding RIM_Cert stored in MTM must be renewed frequently, so deploying valid values for them is non-trivial.

The system-wide global database proposed in Tripwire (Kim and Spafford, 1994) is a straightforward and widely used approach for storing and managing the RIM_Cert. All the RIM_Certs and keys required during the secure boot process are stored in the database. However, Such global database based approaches become very complex to handle when new software packages are installed or when existing packages are being updated (Dietrich and Winter, 2008) and RSA-based validation method for RIM_Cert is rather time consuming and slow on mobile devices. So, the global RIM database becomes a single point of failure and any database corruptions are likely to render the whole platform unbootable.

The database corruption issue can be solved by aggregating the measurements of all software components, which will be run on mobile equipment, by extend operation according their loaded order. The aggregated measurement named the mobile equipment

Corresponding Author: Rongyu He, School of Electronic and Information Engineering, Xi'an Jiaotong University, Xi'an, 710049, Peoples Republic of China

integrity certificate and stored a special PCR (Sailer *et al*., 2004). Unfortunately, the security of a newly executed piece of code depends on the security of all previously executed code. A single compromised piece of code may compromise all subsequent code (Mccune *et al*., 2008).

The most approaches for the update issue are embedded the RIM_Cert into the software itself, such as (Dietrich and Winter, 2008) authors proposed a decentralized approach which directly embedded certificates and keys into the executable images found on the platform. However, directly porting these methods to mobile devices are not practical due to its high computation overhead during booting and runtime.

In this study, we propose a new concept, the user-specific RIM_Cert, uRIM, by introducing a shared secret into tranditional RIM_Cert and then propose a new protocol for software integrity certification based on the user-specific RIM_Cert. The protocol solves the problem that the Reference Integrity Metric (RIM) stored in MTM must be frequently renewed while the critical-application is updated or patched. The authenticity verification in our protocol is based on computational-cheap operations, hash and XOR and it is thus more computationally efficient than others that based on asymmetric cryptography.

THE SOFTWARE VERIFICATION PROTOCOL BASED ON uRIM

The basic definition: Let a software P is composed of n performance-critical-codes and it is expressed as $f_i(0,<i\leq n)$. Here the performance-critical-code may be the key functions or the components of the software. Then the integrity metric, S, of the software P defined as:

$$S = SHA1(f_1 \| f_2 \| \cdots \| f_n) \quad (1)$$

where, S is used as integrity metric of software P. If S is provided and signed by a trusted entity, such as Software Provider (SP), it is used as reference integrity metric, or RIM_Cert (Martin, 2008).

When a mobile equipment u_i will run a software P, it must register to the Server and get the user-specific RIM of the software. The process of generating an uRIM certificate for the software P is as follows.

Assumption: A random number, x_i, is generated as a shared secret between the Server and the MTM mounted on the mobile equipment u_i. The random number is named as Authentication Factor (AF), then the Server calculates the following secretly:

$$v = hash(P_{id}, x_i), \quad e = v \oplus S \quad (2)$$

$$c = h(e \oplus S) \quad (3)$$

Definition 1: The user-specific RIM, rim_{Pi}, of the software P is defined as a tri-tuples:

$$rim_{Pi} = < P_{id}, e, v > \quad (4)$$

where, the P_{id} represents identity of the software P and e is used as encrypted RIM, v is used as a signature for the integrity of the software.

Definition 2: The authentication vector, $\overrightarrow{AV_{Pi}}$, of the software P for u_i is defined as:

$$\overrightarrow{AV_{Pi}} = \{x_i, P_{id}\}sign_{Server} \quad (5)$$

The $\overrightarrow{AV_{Pi}}$ is signed by the Server to attest its authenticity.

Definition 3: The RIM certificate of the software P for mobile equipment u_i is represented as $uRIM_{Pi}$ and it is defined as following:

$$uRIM_{Pi} : \{rim_{Pi}, \overrightarrow{AV}_{Pi}\} \quad (6)$$

The protocol for software verification: Based on the uRIM, we propose a new protocol for integrity verification of software. In our protocol, the rim_{Pi} will be embedded in the software and the $\overrightarrow{AV_{Pi}}$ will be downloaded into MTM securely. The protocol is defined as following:

$$ME \rightarrow SP : P_i, u_i$$

A Mobile Equipment (ME) register for software Pi, the Software Providers (SP) determines if it is a legal user by user identity u_i. If it is a legal user, the SP will generate a user-specific RIM for software Pi using Eq. 2, 3, 5 and 6.

$$SP \rightarrow ME : \{P_i, rim_{Pi}\}$$

The software Pi with its rim_{Pi} is sent to ME. Here the rim_{Pi} is embedded into the software Pi. Executable and Linkable Format (ELF) file is a good candidate for embedding rim_{Pi} into binaries due to its inherent capabilities of adding metadata to the binary images.

$$SP \rightarrow MTM : \{\overrightarrow{AV_{Pi}}\}Sign_{SP}$$

SP sends the authentication vector $\overrightarrow{AV_{Pi}}$ into MTM securely. The SP may be an authorized entity for MTM and it has the privileges to update or store $\overrightarrow{AV_{Pi}}$ into MTM.

When the ME loads the software, it abstracts the user-specific RIM, rim_{Pi}, from the software and then computes the integrity metric, S', of the software according the Eq. 1.

$$ME \rightarrow MTM: \{S', rim_{Pi}\}$$

The ME sends the rim_{Pi} and the current integrity metric, S', into MTM by tpm_quote command.

The MTM verifies the authenticity of the software firstly, then verifies the integrity of the software and get the result that if the verified software is modified or not.

$$MTM \rightarrow ME: \{Y/N\}$$

MTM returns the result to ME as response of the tpm_quote command. The software will run if it is success.

The verification of the software: While the software is loaded on mobile equipment at boot time, it is measured. The measurement, $S'(SHA1(f_1||f_2||\cdots||f_n)$ and the rim_{Pi}, which is embedded in the software, are passed to MTM for verifying. The MTM computes the secrets as following:

$$v' = hash(P_{id}, x) \tag{7}$$

$$c = hash(e \oplus S') = hash(v \oplus S \oplus S') \tag{8}$$

From Eq. 7, if v' = v, the MTM can prove the authenticity of rim_{Pi} because only MTM and the SP know the shared secret x.

Form Eq. 8, if the software hasn't been modified, then S = S', we will get:

$$c = hash(v) \tag{9}$$

If the Eq. 9 is satisfied, the integrity of the software is verified.

Verification for updated software: If the software is updated or patched, the RIM of the software is changed. Supposed that the RIM for the updated software is S*,

$$S^* = SHA1(f_1 \| f_2 \| \cdots \| f_n) \tag{10}$$

The SP calculates the following:

$$v^* = hash(P_{id}, x), \quad e^* = v \oplus S^* \tag{11}$$

The new RIM certificate for the updated software $uRIM\_Cert'_{pi}$ is:

$$uRIM\_Cert'_{Pi} : \{rim'_{Pi}, \overrightarrow{AV'_{Pi}}\}$$

Where:

$$rim'_{Pi} = < P_{id}, e^*, v > \text{ and } \overrightarrow{AV'_{Pi}} = \{x_i, P_{id}\}sign_{Server}$$

Because the authentication vector, $\overrightarrow{AV'_{Pi}}$, is not changed, the elements $\overrightarrow{AV_{pi}}$ in the MTM need not to be modified. The only thing the SP will be done is to embed the new RIM, rim'_{Pi}, into the updated software.

The updated software embedded with its the user specific RIM, rim'_{Pi}, is distributed to the mobile equipment u_i and it is verified as following:

The loader abstracts the user-specific RIM from the updated software firstly, then measures the updated software to get the currently measurement S', passes the rim'_{Pi} and S' into MTM by tpm_quote.

The MTM verifies the authenticity of the rim'_{Pi} according Eq. 7, then verifies the integrity of software as following:

$$c' = h(e \oplus S') = h(v \oplus S^* \oplus S') = h(h(P_{id}, x) \oplus S^* \oplus S') \tag{12}$$

If the updated software isn't modified, S* = S'. From Eq. 3 we can get:

$$c' = c$$

So the integrity of the updated software is verified.

THE ANALYSIS

Security for the uRIM_Cert: The way that the RIM is distributed with its software makes the RIM is subject to attack such as tampered or counterfeit.

In our scheme, the attacker cannot get the AF from the mobile equipment because it is stored in the MTM. The security features of MTM guarantee that the AF cannot be observed out of the chip and only the verification function, an inner function of MTM, can read it. The attacker either cannot get the AF from the SP because it is encrypted with a protect key.

The uRIM of the software is embedded into the software in plaintext. But the reference integrity metric S is hided and signed, as Eq. 2 shown, by the secret AF which is shared between SP and mobile equipment (MTM). The attacker cannot generate a fake RIM because he cannot generate the signature, v, of the uRIM. A tampered RIM cannot be detected but the integrity verification of the software is failure and the software can no longer be loaded. So, it guarantees the security of the mobile equipment.

Table 1: Comparisons of computation costs

Cost	Global database	Dietrich's scheme	eRIM_Cert scheme
RIM computing	1T(RSA$_s$)	1T(RSA$_s$)	21T(h)
	1T(h)	1T(h)	2T(⊕)
Authenticity verifying	1T(RSA$_v$)	1T(RSA$_v$)	1T(h)
Integrity verifying	1T(h)	1T(h)	1T(h)
			2T(⊕)

T (h): Computation cost of one-way function; T (⊕): Computation cost of exclusive-OR operation; T(RSAs): Computation cost of RSA-based signature; T(RSAv): Computation cost of RSA-based signature verification

The mobile equipment with the authorized MTM can verify the authenticity and integrity of software and this provides copyright protection for the software in some extent since running software without verification causes the user at great risk for Trojan and viruses.

Efficiency for the uRIM_Cert: We made comparisons among the previous RIM-based software verification schemes and our proposed scheme. The compared schemes are global database scheme and Dietrich's scheme (Dietrich and Winter, 2008). Table 1 is the comparisons in the computation cost and the results show that our scheme is cheaper and more efficient than others.

CONCLUSIONS

The purpose of this work is to provide software verification both in integrity and authenticity when it is loaded. With the user-specific RIM_Cert we proposed in the paper, a novel protocol for software validating is given. It solves the problem that the RIM must be renewed after the software is updated or patched. Our scheme improves the software verification specified in MPWG in the three aspects:

- The value stored in MTM is a secret instead of the RIM of the software. When the software is updated or patched, there is no need for renewing the secrets
- Using shared secret x and hash operator hash(P_{id}, x) instead of RSA-based signature operation for uRIM, this is suitable for mobile platform which is resource-constrained, such as mobile phone or PDA
- Integrity measurement implicitly embedded in the software, no one can tamper it without knowing the secret x and the MTM can verify the authenticity of the software. In the meanwhile, only the platform with the authorized MTM can run the software

ACKNOWLEDGMENT

This study was partially supported the National High Technology Development Program of China (No. 2007AA01Z479).

REFERENCES

Dietrich, K. and Winter, J., 2008. Secure boot revisited. Proceedings of the 9th International Conference for Young Computer Scientists, Nov. 18-21, Hunan, pp: 2360-2365.

Gallery, E., 2007. Authorization issues for mobile code in mobile systems. Technical Report RHUL-MA-2007-3, Department of Mathematics, Royal Holloway, University of London.

Gallery, E.M. and Mitchell, C.J., 2007. Trusted mobile platforms. In: Foundations of Security Analysis and Design IV Lecture Notes in Computer Science, A. Aldini and R. Gorrieri (Eds.). Springer-Verlag, Berlin, ISBN: 978-3-540-74809-0, pp: 282-323.

Kim, G.H. and E.H. Spafford, 1994. Experiences with tripwire: Using integrity checkers for intrusion detection. Proceedings of the System Administration, Networking and Security Conference (III Usenix), Aug. 9-13, San Diego, CA, USA., pp: 1-13.

Martin, A., 2008. The ten-page introduction to trusted computing. CS-RR-08-11, Oxford University Computing Laboratory, http://www.comlab.ox.ac.uk/files/1873/RR-08-11.PDF.

Mccune, M.J., B.J. Parno, A. Perrig, M.K. Reiter and H. Isozaki, 2008. Flicker: An execution infrastructure for TCB minimization. Proceedings of the 3rd ACM SIGOPS/EuroSys European Conference on Computer Systems, April 1-4, Glasgow, Scotland, UK., pp: 315-328.

Sailer, R., X. Zhang, T. Jaeger and L. van Doorn, 2004. Design and implementation of a TCG-based integrity measurement architecture. Proceedings of the 13th USENIX Security Symposium, Aug. 9-13, San Diego, CA, USA., pp: 223-238.

TCG, 2007a. Trusted computing group: TCG mobile reference architecture. Specification Version 1.0. Revision 1, June 12 (2007), http://www.trustedcomputinggroup.org.

TCG, 2007b. Trusted computing group: TCG mobile trusted module specification. Specification Version 1.0. Revision 1, June 12 (2007), http://www.trustedcomputinggroup.org.