A Framework for Self-Verification of Firmware Updates over the Air in Vehicle ECUs

Dennis K. Nilsson
Department of Computer Science and Engineering
Chalmers University of Technology
SE-412 96 Gothenburg, Sweden
Email: dennis.nilsson@chalmers.se

Lei Sun, Tatsuo Nakajima
Department of Computer Science and Engineering
Waseda University
3-4-1 Okubo Shinjuku Tokyo 169-8555, Japan
Email: {sunlei,tatsuo}@dcl.info.waseda.ac.jp

***Abstract*—An upcoming trend for automobile manufacturers is to provide firmware updates over the air (FOTA) as a service. Since the firmware controls the functionality of a vehicle, security is important. To this end, several secure FOTA protocols have been developed. However, the secure FOTA protocols only solve the security for the transmission of the firmware binary. Once the firmware is downloaded, an attacker could potentially modify its contents before it is flashed to the corresponding ECU's ROM. Thus, there is a need to extend the flashing procedure to also verify that the correct firmware has been flashed to the ECU. We present a framework for self-verification of firmware updates over the air. We include a verification code in the transmission to the vehicle, and after the firmware has been flashed, the integrity of the memory contents can be verified using the verification code. The verification procedure entails only simple hash functions and is thus suitable for the limited resources in the vehicle. Virtualization techniques are employed to establish a trusted computing base in the ECU, which is then used to perform the verification. The proposed framework allows the ECU itself to perform self-verification and can thus ensure the successful flashing of the firmware.**

I. Introduction

A typical vehicle contains an in-vehicle network consisting of 40-60 embedded devices known as *electronic control units* (ECUs). As increasingly advanced functionalities are developed, there exists a need to update the software running on these embedded devices. Updates could be issued to improve existing functionality or to remedy discovered bugs. An emerging trend for vehicle manufacturers is to perform firmware updates over the air (FOTA) [1]. There are several benefits with this approach. First, it involves minimal customer inconvenience since there exists no need for the customer to bring the vehicle to a service station for a firmware update. Second, it allows faster updates; as soon as the firmware is released it could be downloaded and installed in the vehicle. The firmware is downloaded over a wireless network connection from a trusted portal to the vehicle and then flashed to the correct ECU. Since the firmware is responsible for the functionality of the vehicle, security is imperative. Protocols for secure download exist [2, 3] but ensuring proper firmware installation and memory verification is lacking.

As shown in [4, 5], attacks in the in-vehicle network could have serious consequences for the driver. If an attacker can install a malicious firmware, the attacker can virtually control the functionality of the ECU and perform arbitrary actions in the in-vehicle network. Vehicle virus emerges as an extremely potent threat. Thus, there is an imminent need for proper firmware update verification.

Since the ECU itself is an untrusted environment there exist challenges in how to verify the integrity of the firmware download and flashing. An attacker could take control of an ECU and modify any integrity checking functions stored on the unit itself. We therefore argue that a small trusted computing base, such as L4 [6, 7], which is secure and reliable, should exist on the embedded device.

We propose a framework where a firmware binary is downloaded to an ECU from a trusted portal using a secure FOTA protocol. We apply hardware virtualization techniques on the ECU, allowing two guest systems to run simultaneously on the underlying virtual machine monitor. The control system, consisting of L4, performs simple operations such as to provide memory verification of the flashed firmware. The functional system handles all the services or functionality of the corresponding ECU. The binary is then flashed to the ROM of the ECU, which constitutes the functional system. Next, the control system locally verifies the correct flashing on the ECU itself. The self-verification allows the ECU to assure that the proper firmware has been installed.

The main contributions of this paper are as follows.

- We have developed a framework for self-verification of remote firmware updates in ECUs.
- We have applied virtualization techniques to support a functional and a control system. The control system verifies the flashing of the functional system.
- We have designed an efficient protocol for memory verification of the flashed firmware. The verification is performed locally on the embedded device.
- The protocol has been analyzed in terms of security and cost overhead.

The remainder of this paper is structured as follows. Section II discusses related work. In Section III, we present our proposed framework for self-verification of firmware updates over the air in vehicle ECUs. Section IV describes in detail the protocol used for self-verification. In Section V, we analyze the framework in terms of security and overhead. Section VI presents future work, and Section VII concludes the paper.

Authorized licensed use limited to: WASEDA UNIVERSITY. Downloaded on June 26, 2009 at 04:53 from IEEE Xplore. Restrictions apply.

II. Related Work

There exist similar work in software attestation, tamper resistance and remote code verification. SWATT [8] – software-based attestation for embedded devices – verifies the entire static memory contents of a device using an external verifier. This requires an external system to be connected to the device to be verified. SWATT has also been applied to vehicles [9]. Pioneer [10] verifies an executable on an untrusted system from an external network-connected dispatcher. SCUBA [11] provides secure code update by attestation in sensor networks where a trusted code base is established on the untrusted sensor node. BIND [12] is a fine-grained attestation service for securing distributed systems, which attests and protects the execution of code. Other software tamper-resistance solutions have been proposed including using secure hardware [13], checksums [14], security protocols [15], remote attestation [16], and timing [17] techniques.

ARM has developed the TrustZone solution [18] which is based on trusted execution of code and hardware isolation. It runs two execution environments: secure and non-secure. The TrustZone solution consists of both software and hardware elements. Intel has developed an authenticated flash approach [19] to prevent unwanted modifications to the flash memory contents by authenticating command requests to the flash memory using signatures. If the signature is correct, the flash is programmed with the program data. The authenticated flash approach can be integrated with the TrustZone solution.

With virtualization techniques becoming more popular, virtual machine (VM) based solutions both for high performance computing [20] and trusted computing [21] have been developed. In these VM-based systems, the VM provides an isolation environment for individual systems and also helps to manage the hardware resources. Moreover, virtualization techniques on embedded devices have shown to improve reliability and security [22, 23].

In [24], it is pointed out that current operating systems lack security isolation; therefore, once a fault occurs the whole system may crash. In contrast to traditional implementations of a monolithic kernel operating system, in a microkernel operating system all programs run in user mode and are protected against their own address space. Although the first-generation microkernels lead to rather discouraging results in system performance, L4 [6] known as a second-generation microkernel has drawn great attention due to its improvements. Due to its underlying security isolation, the microkernel architecture is chosen as the reasonable candidate platform to deploy our framework.

Previous research has mainly been concerned with an untrusted device or platform performing the verification and returning the result to a trusted remote unit. Instead, we consider the problem of self-verification where the device itself, split into a trusted (microkernel) and untrusted system by using virtualization, performs the verification and returns the result to itself.

III. Framework Design

In this section, we describe the overall design of the framework and discuss the problem it addresses. We also describe an attacker model for this scenario.

A. Problem Statement

- Assuming that a secure protocol can be used to download a firmware binary over the air to a vehicle ECU, before the firmware is actually flashed to the ECU there is a possibility that an attacker modifies the binary. Thus, there is a need to *verify the flashed memory contents after the firmware has been flashed to the ECU*.

In short, the binary is verified as it is downloaded using a secure protocol [2, 3]. However, even if the binary is verified before it is flashed it is still susceptible to modification in a time-of-check-to-time-of-use attack [11]. Attacks include creating vehicle virus that could potentially cause severe injury to the driver [4, 5]. Thus, it is necessary to verify the contents of the flashed area to assure that the correct binary has been flashed to the ECU.

B. Design Overview

Our proposed framework, depicted in Fig. 1, consists of two major parts: a portal and a vehicle. The portal contains the new firmware and is assumed to be trusted by the vehicle. The vehicle has a wireless gateway and an *in-vehicle network* consisting of interconnected ECUs. A secure FOTA (SFOTA) protocol ensures that the downloaded firmware is guaranteed to be authenticated and integrity-protected.

By applying hardware virtualization techniques on the ECU, two guest systems, a *control system* and a *functional system*, are run simultaneously on the underlying virtual machine monitor. We have implemented a prototype system based on the virtual machine monitor SPUMONE [25] and the SH-4A platform [26]. SPUMONE provides CPU resource virtualization, memory isolation, and communication between the control and functional system by using virtual inter-process-interrupt. The functional system provides the functionality the ECU normally would provide. The control system, which is based on the microkernel L4, on the other hand handles the flashing and verification procedure of the functional system.

The functional system provides various functionality depending on the ECU: from opening and closing windows to turning on and off the headlights to providing break assistance and driver guidance. The functional system could contain complex functions to provide advanced functionality. Because of its complexity, the functional system could suffer from unexpected behavior due to software bugs.

The control system handles the flashing procedure. The flashing procedure is expanded to include a memory verification of the flashed data after the initial flashing of the binary to the ROM. Thus, the control system performs the verification procedure. Our proposal to provide memory verification of the flashed firmware includes a challenge-response protocol between the portal and the control system to verify the memory contents of the flashed binary. A simple approach is to read

Authorized licensed use limited to: WASEDA UNIVERSITY. Downloaded on June 26, 2009 at 04:53 from IEEE Xplore. Restrictions apply.

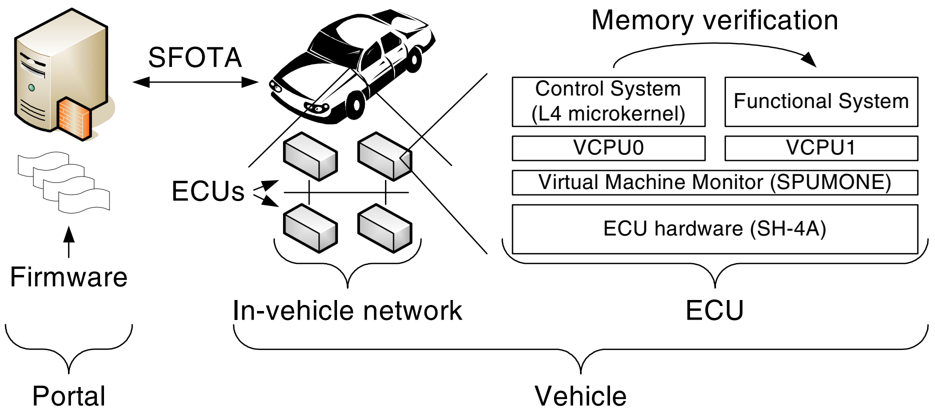


Fig. 1. Framework for self-verification of firmware updates over the air in vehicle ECUs.

the memory contents in blocks and run the contents through a checksum. However, checksums are vulnerable to active attacks where the contents can be modified and still yield a correct checksum [27]. Instead, we propose calculating a hash chain based on the memory contents and a challenge.

An overview of the framework operation is described as follows. First, the portal issues a new firmware binary for a specific ECU and calculates the corresponding verification code based on a random challenge and the firmware contents.

Using SFOTA, the firmware, the verification code, and the challenge are downloaded to the ECU. The verification code and the challenge are small in size and can be stored in the control system. The firmware is typically much larger and is thus stored in the untrusted RAM. The SFOTA protocol verifies the integrity and authenticity of the firmware, the verification code, and the challenge during download.

Next, the ECU commences the flashing procedure. The control system reads the binary from RAM and flashes it to the ROM (the functional system). The ECU then performs self-verification, where the control system verifies the flashed memory contents using the received verification code and challenge. Thus, the control system is assured that the correct functional system is running on the ECU.

This procedure addresses the previously mentioned threat. If an attacker modifies the binary in RAM after the download verification but before the binary is flashed to ROM, a malicious or incorrect binary is flashed to ROM. However, the control system verification detects this change.

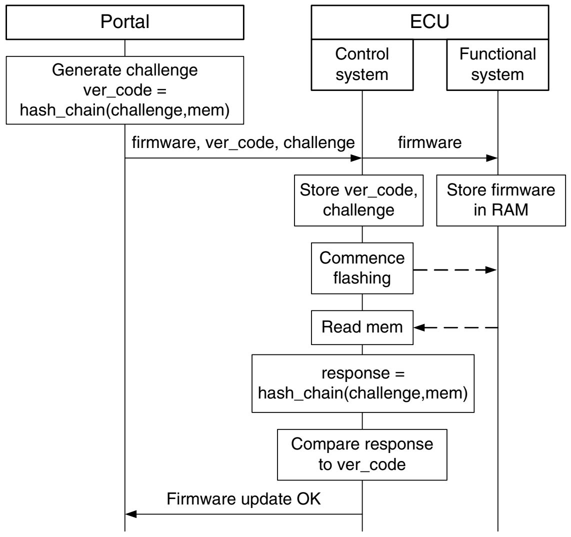
C. Attacker Model

It is assumed that an attacker has gained access to the in-vehicle network via the wireless gateway. The attacker cannot modify the control system (trusted area) which is implemented as a microkernel but can modify contents in the untrusted area (RAM). Thus, a typical attack constitutes replacing or modifying the downloaded binary before it is flashed to the ROM. The size of the attack window depends on how long it will take from a firmware is downloaded until it is flashed. Flashing is typically restricted to occur only at times where the driver is safe from potential injury whereas downloading of the firmware can occur at any time. Thus, the attack window can vary from a few seconds to several hours.

Moreover, we assume that after the binary has been flashed, i.e., installed as a new functional system, the attacker cannot modify the contents of the flashed area. The only way to modify the contents of the flashed area is to perform the flashing procedure again, and thus reinvoking the verification procedure.

IV. Verification Protocol

The verification protocol for the framework is depicted in Fig. 2. First, the portal generates a random challenge (nonce) and calculates a hash chain of the firmware and the challenge. The hash calculation procedure is illustrated in Fig. 3.



Portal
ECU
Control system
Functional system
Generate challenge ver_code = hash_chain(challenge,mem)
firmware, ver_code, challenge
firmware
Store ver_code, challenge
Store firmware in RAM
Commence flashing
Read mem
response = hash_chain(challenge,mem)
Compare response to ver_code
Firmware update OK

Fig. 2. Protocol for self-verification of ECU firmware update.

The firmware is divided into blocks of predetermined size, and each block is used as input to the hash function. The output value from the hash function is included in the next hash calculation in order to create a hash chain of memory

Authorized licensed use limited to: WASEDA UNIVERSITY. Downloaded on June 26, 2009 at 04:53 from IEEE Xplore. Restrictions apply.

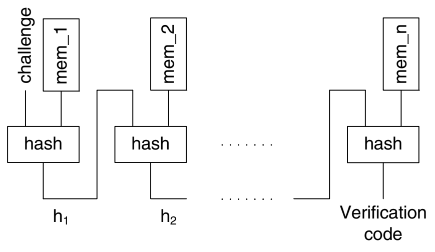


Fig. 3. Hash chain calculation on memory contents.

contents. A nonce is included to create randomness in the hash calculation to prevent an attacker from performing a second preimage attack [28]. The final hash output is used as the verification code. The nonce-seeded hash calculation is shown below.

$$h_i = \begin{cases} hash(challenge, mem\_i) & \text{if } i = 1 \\ hash(h_{i-1}, mem\_i) & \text{otherwise} \end{cases}$$

Next, the verification code together with the firmware binary and the challenge are sent to the vehicle using SFOTA, which guarantees the authenticity and integrity of the downloaded data. The control system stores the verification code and the challenge. These values are used later to verify the integrity of the memory contents of the flashed firmware (functional system). The firmware binary is stored in the RAM of the functional system.

Once the vehicle driver is safe from potential injury (e.g., when the vehicle is not in use), the control system of the corresponding ECU commences the flashing procedure. First, the functional system is shut down, which disables the functionality of that ECU. Then, the binary is read from the RAM, and the control system flashes the binary to the ROM.

Finally, the flashed binary is verified to assure that the flashed memory contains the correct data. The control system reads the memory contents in blocks and uses them as inputs to the hash function to create a hash chain. The challenge received from the portal is also included in the calculation. At the end of the calculation, the control system compares the calculated response with the received verification code. If it matches, the control system is ensured that the firmware flashing was successful and that the correct firmware is flashed to the functional system. Thus, the ECU itself performs self-verification. Once verified, the control system boots up the updated functional system, which enables the ECU functionality. Last, the control system sends a message to the portal indicating a successful firmware update. The entire protocol procedure is shown in the flowchart in Fig. 4.

The main advantages with this protocol are as follows.

- The ECU itself can perform self-verification of the firmware update without relying on an external device.
- The control system can detect if the binary has been modified between the downloading and flashing procedures.
- The control system can verify that the correct firmware binary has been flashed. Thus, when it starts the functional system it is assured that the correct functional system is started.
- The protocol uses only simple hash functions which generates a low overhead. Since the verification code is stored in the control system and verified when downloaded there is no need for digital signatures.
- The protocol is based on a challenge-response mechanism and includes random numbers to prevent preimage attacks.

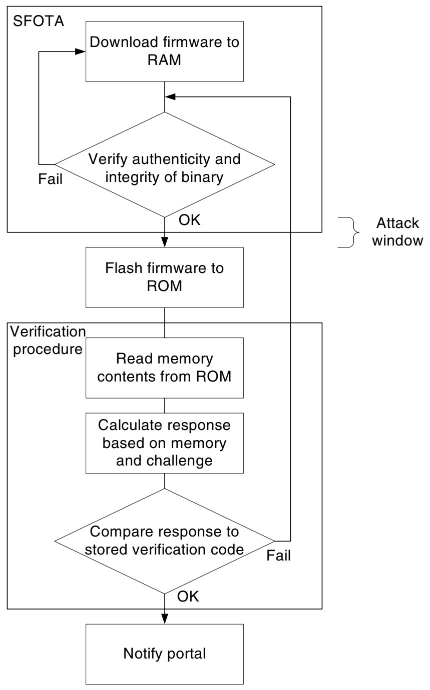

SFOTA
Download firmware to RAM
Verify authenticity and integrity of binary
Fail
OK
Attack window
Flash firmware to ROM
Verification procedure
Read memory contents from ROM
Calculate response based on memory and challenge
Compare response to stored verification code
Fail
OK
Notify portal

Fig. 4. Flowchart of firmware download, flashing, and verification.

V. Analysis

Based on our assumptions, our proposed framework provides secure self-verification of firmware updates in ECUs with relatively little overhead. We analyze the framework and the protocol in terms of security and overhead.

- Hash functions, instead of checksum functions which have been found to be vulnerable, are used to calculate the verification code.
- A random challenge is included in the hash chain calculation to prevent a second preimage attack [28].
- The verification code is stored in the control system, where an attacker cannot modify it (memory isolation).
- The protocol verifies the functional system (flashed memory), which an attacker cannot modify.
- After flashing little overhead is introduced, which consists of traversing the memory of the ROM and calculating a hash chain. Only simple hash functions are used, and there is no need for digital signatures since a secure protocol is used to verify that the binary comes from a trusted server.

Authorized licensed use limited to: WASEDA UNIVERSITY. Downloaded on June 26, 2009 at 04:53 from IEEE Xplore. Restrictions apply.

VI. Future Work

Future work involves expanding the verification procedure. Assuming that an attacker could modify the functional system after the firmware update, there is a need to develop methods for runtime verification of the functional system. In addition, the functional system often contains advanced and complex functions in which programming errors or bugs could easily be introduced. Therefore, the framework could be extended to improve reliability and dependability of the ECUs. Thus, based on virtualization techniques, it would be possible to introduce various maintenance and recovery functions for the functional system. Moreover, the verification procedure could be expanded to handle unsuccessful firmware updates by using rollback techniques. For example, when the control system detects a failed flashing, it automatically reverts the ECU to the last good known functional system version.

VII. Conclusion

We have proposed a framework for self-verification of firmware updates over the air in vehicle ECUs. The framework consists of a trusted portal and a vehicle. The portal issues firmware updates including a corresponding verification code which the vehicle downloads using a secure FOTA protocol. Virtualization techniques are applied to the ECU hardware to allow a control system and a functional system to run simultaneously. The functional system provides the ECU functionality, and the control system, built on a microkernel architecture, flashes the firmware to the functional system and verifies the correctness. We have provided an efficient protocol for memory verification of the flashed firmware. Using our approach it would be possible to detect malicious changes to the downloaded firmware and to verify the correct flashing of the firmware binary. An upcoming trend for vehicle manufacturers is to perform firmware updates over the air. For such scenarios is it essential to not only verify that the firmware has been downloaded correctly but also to verify that the firmware has been properly flashed to memory.

References

[1] D. K. Nilsson, U. E. Larson, and E. Jonsson, "Creating a Secure Infrastructure for Wireless Diagnostics and Software Updates in Vehicles," in *Proceedings of the 27th International Conference on Computer Safety, Reliability and Security (SAFECOMP)*. Springer, 2008, pp. 207–220.

[2] S. M. Mahmud, S. Shanker, and I. Hossain, "Secure Software Upload in an Intelligent Vehicle via Wireless Communication Links," in *Proceedings of IEEE Intelligent Vehicles Symposium*. IEEE, 2005, pp. 587–592.

[3] D. K. Nilsson and U. E. Larson, "Secure Firmware Updates over the Air in Intelligent Vehicles," in *Proceedings of the First IEEE Vehicular Networking & Applications Workshop (Vehi-Mobi)*. IEEE, 2008, pp. 380–384.

[4] ——, "Simulated Attacks on CAN Buses: Vehicle virus," in *Proceedings of the Fifth IASTED Asian Conference on Communication Systems and Networks (ASIACSN)*. ACTA Press, 2008.

[5] D. K. Nilsson, U. E. Larson, F. Picasso, and E. Jonsson, "A First Simulation of Attacks in the Automotive Network Communications Protocol FlexRay," in *Proceedings of the First International Workshop on Computational Intelligence in Security for Information Systems (CISIS)*. Springer, 2008.

[6] "The L4 micro-Kernel Family," http://os.inf.tu-dresden.de/L4/.

[7] K. Elphinstone, G. Heiser, R. Huuck, S. M. Petters, and S. Ruocco, "L4cars," in *Embedded Security in Cars (ESCAR)*, Cologne, Germany, 2005.

[8] A. Seshadri, A. Perrig, L. van Doorn, and P. K. Khosla, "SWATT: SoftWare-based ATTestation for Embedded Devices," in *IEEE Symposium on Security and Privacy*. IEEE Computer Society, 2004.

[9] A. Seshadri, A. Perrig, L. van Doorn, and P. Khosla, "Using Software-based Attestation for Verifying Embedded Systems in Cars," in *Embedded Security in Cars (ESCAR)*, 2004.

[10] A. Seshadri, M. Luk, E. Shi, A. Perrig, L. van Doorn, and P. Khosla, "Pioneer: Verifying code integrity and enforcing untampered code execution on legacy systems," in *SOSP '05: Proceedings of the twentieth ACM symposium on Operating systems principles*. New York, NY, USA: ACM Press, December 2005, pp. 1–16.

[11] A. Seshadri, M. Luk, A. Perrig, L. van Doorn, and P. Khosla, "SCUBA: Secure code update by attestation in sensor networks," in *ACM Workshop on Wireless Security (WiSe)*, September 2006.

[12] E. Shi, A. Perrig, and L. Van Doorn, "BIND: A fine-grained attestation service for secure distributed systems," in *SP '05: Proceedings of the 2005 IEEE Symposium on Security and Privacy*. Washington, DC, USA: IEEE Computer Society, 2005, pp. 154–168.

[13] B. Chen and R. Morris, "Certifying Program Execution with Secure Processors," in *HOTOS'03: Proceedings of the 9th Conference on Hot Topics in Operating Systems*, Berkeley, CA, USA, 2003, p. 23.

[14] R. Kennell and L. H. Jamieson, "Establishing the Genuinity of Remote Computer Systems," in *SSYM'03: Proceedings of the 12th conference on USENIX Security Symposium*. Berkeley, CA, USA: USENIX Association, 2003, pp. 295–310.

[15] T. Park and K. G. Shin, "Soft Tamper-Proofing via Program Integrity Verification in Wireless Sensor Networks," *Mobile Computing, IEEE Transactions on*, vol. 4, no. 3, pp. 297–309, 2005.

[16] D. Schellekens, B. Wyseur, and B. Preneel, "Remote Attestation on Legacy Operating Systems With Trusted Platform Modules," *Electron. Notes Theor. Comput. Sci.*, vol. 197, no. 1, pp. 59–72, February 2008.

[17] J. A. Garay and L. Huelsbergen, "Software Integrity Protection Using Timed Executable Agents," in *ASIACCS '06: Proceedings of the 2006 ACM Symposium on Information, Computer and Communications Security*. New York, NY, USA: ACM, 2006, pp. 189–200.

[18] ARM, "TrustZone: Integrated Hardware and Software Security, White Paper," http://www.arm.com/pdfs/TZ_Whitepaper.pdf, 2004.

[19] ARM and Intel, "ARM Security Solutions and Intel Authenticated Flash, White Paper," http://www.arm.com/pdfs/Intel_ARM_Security_WhitePaper.pdf, 2007.

[20] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield, "Xen and the Art of Virtualization," in *Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles (SOSP)*, 2003, pp. 164–177.

[21] T. Garfinkel, B. Pfaff, J. Chow, M. Rosenblum, and D. Boneh, "Terra: A Virtual Machine-Based Platform for Trusted Computing," *SIGOPS Oper. Syst. Rev.*, vol. 37, no. 5, pp. 193–206, 2003.

[22] S.-M. Lee, S.-B. Suh, B. Jeong, and S. Mo, "A Multi-Layer Mandatory Access Control Mechanism for Mobile Devices Based on Virtualization," in *Consumer Communications and Networking Conference, 2008. CCNC 2008. 5th IEEE*, 2008, pp. 251–256.

[23] L. Sun and T. Nakajima, "A Lightweight Kernel Objects Monitoring Infrastructure for Embedded Systems," in *The 14th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, Kaohsiung, Taiwan, August 2008, pp. 55–60.

[24] A. S. Tanenbaum, J. N. Herder, and H. Bos, "Can We Make Operating Systems Reliable and Secure?" *Computer*, vol. 39, no. 5, pp. 44–51, May 2006.

[25] W. Kanda, Y. Kinebuchi, Y. Yumura, and T. Nakajima, "SPUMONE: LightWeight CPU Virtualization Layer for Embedded Systems," in *Proceedings of the 2008 IEEE/IFIP International Conference on Embedded and Ubiquitous Computing (EUC)*, 2008.

[26] "SH7780 SuperH RISC Engine Family," http://america.renesas.com/fmwk.jsp?cnt=sh7780_series_landing.jsp&fp=/products/mpumcu/superh_family/sh7780_series/.

[27] G. Wurster, P. C. van Oorschot, and A. Somayaji, "A Generic Attack on Checksumming-Based Software Tamper Resistance," in *Security and Privacy, 2005 IEEE Symposium on*, 2005, pp. 127–138.

[28] S. Halevi and H. Krawczyk, "Strengthening Digital Signatures via Randomized Hashing," in *Proceedings of the 26th Annual International Cryptology Conference (CRYPTO)*, August 2006.

Authorized licensed use limited to: WASEDA UNIVERSITY. Downloaded on June 26, 2009 at 04:53 from IEEE Xplore. Restrictions apply.