Testing and Evaluation of a Secure Integrity Measurement System (SIMS) for Remote Systems

Shadi Aljawarneh[1] and Abdullah Alhaj[2]
[1]Faculty of Information Technology, Isra University, Jordan
[2]Faculty of Information Technology, University of Jordan/Aqaba, Jordan

Abstract: *We have designed a novel system called a Secure Integrity Measurement System (SIMS) to provide a practical integrity for flexible and traditional remote systems. SIMS is not only targeted for Linux, but it can also be used for different operating systems such as Windows, and UNIX. All and executable content that are loaded onto any operating system is measured before execution. These measurements are protected by a secure Database Management System (DBMS) rather than using Trusted Platform Module (TPM) that is part of the Trusted Computing Group (TCG) standards. The proposed system can measure the executable content from the BIOS and the content that is generated at the application layer. Note our system does not require any special hardware such TCG or a new CPU mode or an operating system. In this paper, a set of experiments are carried out to meet the security and performance objectives. We have shown with the system evaluation that the SIMS can provide a tamper detection, and recovery to different kinds of content. The SIMS can efficiently and correctly determine if the executable content has been tampered with.*

Keywords: *Data integrity, tampering, remote systems, trustworthiness, survivability.*

Received October 12, 2009; accepted March 9, 2010

1. Introduction

Due to the openness and distribution of computing systems, security becomes an important challenge in order to ensure the confidentiality, integrity, and availability of data. Therefore, the need to be able to securely identify the software stack that is running on remote systems is increased [18]. For example, in grid computing, we are concerned that the services advertised really exist and that the system is not compromised. In demand computing, we may be concerned that the outsourcing partner is providing the software facilities and performance that have been stipulated in the service level agreement. Yet another scenario is where we are interacting with the home banking or book-selling web services application and we need to make sure it has not been tampered with.

The main issue with the scenarios above is that we cannot trust the program itself because it could be modified and gives incorrect answers. For the same reason we cannot also trust the kernel or the BIOS on which these programs are running since they may be tampered with too. Instead we need to investigate an immutable root to provide that answer. This is essentially the secure boot problem [12, 16], and so we are interested in integrity of the software stack.

The Trusted Computing Group (TCG) has declared a set of standards TCG that describe how to take integrity measurements of a system and store the results in a separate trusted coprocessor called a Trusted Platform Module (TPM) whose state cannot be compromised by a potentially malicious host system. This mechanism is called a trusted boot. Unlike a secure boot, the trusted boot only takes measurements and leaves it up to the remote party to determine the trustworthiness of system. The secure boot works when the system is powered on and it then transfers control to an immutable base. This base will measure the next part of BIOS by computing a SHA1 secure hash over its contents and secure the results by using the TPM. This procedure is then applied recursively to the next portion of code until the operating system has been bootstrapped [18].

In this work, we will focus to maintain the chain of trust measurements up to the application layer, but unlike the bootstrap process, an operating system handles a large variety of executable content (such as kernel, kernel modules, libraries, scripts, and plug-in).

The previous research focused on measuring code and associating integrity semantics with the code. For example, the IBM 4758 defines that the integrity of a program is determined by the code of the program and its ancestors [2, 5]. However, the IBM 4758 environment is limited to a single program with a well-defined input state and the integrity is enforced with the secure boot process. However, the applications running on the 4758 which cannot handle low integrity inputs properly could be compromised [8]. Furthermore, because the IBM 4758 is most costly, most computers do not include a secure coprocessor, such as the IBM 4758 [3, 10, 13].

In this paper, we focus on the survivability of mission critical contents. We assume that all content are susceptible to malicious software attacks. The proposed system (consists of three components: registry, integrity checker, and recovery) will attempt to address the following concerns.

1. We have modified the Windows kernel and the runtime system to take integrity measurements as soon as executable content is loaded into the system, but before it is executed. This concern is similar to TCG-Based Integrity Measurement system [18].
2. We will use Database Management System (DBMS) table to store the hashing measurements rather than holding measurements directly in a special hardware such as TCG.
3. The proposed integrity checker mechanism can then determine whether the executable content has been tampered with and, once the content is verified, a kind of trust will be satisfied.
4. Otherwise, if any tampering occurs, the integrity checker sends alert to the recovery component to recover the tampered content by the original content.
5. Where we make a backup copy for all list of content in the BIOS.

This paper is organized as follows: section 2 describes the design of the proposed SIMS architecture and its components. In section 3, we present experimental studies to evaluate the reliability and performance of the SIMS. This section includes:

1. Experimental study to evaluate whether the SIMS is able to detect and recover from the tampering attacks.
2. Another case study for micro-benchmarking performance to measure the performance of the proposed registry mechanism (using SHA-1 and SHA1-extended). Related work is made in section 4. Finally, conclusions and future work are offered in section 5.

2. Related Work

There are sixth key issues in prior work including:

1. The distinction between secure boot and authenticated boot.
2. The semantic value of previous integrity measurement approaches.
3. Secure object identification.
4. Kiosk computing.
5. CD-boot linuxi.
6. Atomic measurement using a memory copy-on-write strategy.

As discussed above, the secure boot process enables a system to measure its own integrity and terminate the boot process if an action compromises this integrity. Arbaugh [12] has developed the AEGIS system, based on signed hash values to provide a practical architecture for deploying secure boot on a PC system. The validation process of the AEGIS is run for each layer in the boot process. It will abort booting the system if the hashes cannot be validated. However, secure boot does not enable a challenging party to validate the integrity of a boot process such as an authenticated boot process because it simply measures and checks the boot process, but does not ensure attestations of the integrity of the process.

The IBM 4758 secure coprocessor [20] includes both secure boot and authenticated boot in a limited environment. It promises secure boot guarantees by validating (hash) partitions before activating them and by enforcing valid signatures before loading executables into the system. A mechanism called outgoing authentication [5] enables attestation that links each subsequent layer to its predecessor. The predecessor attests to the subsequent layer by generating a signed code that includes the cryptographic hash and the public key of the subsequent layer. To protect an application from flaws in other applications, only one application is allowed to run at a time. Thus, the integrity of the application depends on hashes of the code and manual verification of the application's installation data. This data is only accessible to trusted code after installation. Whereas our web server's example runs in a much more dynamic environment where multiple processes may access the same data and may interact. Further, the security requirements of the challenging party and the attesting party may differ such that secure boot based on the challenging party's requirements is impractical.

The Trusted Computing Group [20] is a consortium of companies that together have developed an open interface for a TPM, a hardware extension to systems that provides cryptographic functionality and protected storage. The TPM enables the verification of static platform configurations, both in terms of content and order, by collecting a sequence of checksums over target code. However, the integrity of applications running on the operating system does not deal by TCG and TCG-measurements system.

Various TPM-based systems have been proposed, including the Integrity Measurement Architecture by Sailer *et al*. [18], and the more recent, late-launch-based Flicker [11]. To date, these systems assume that the external verifier has somehow obtained the TPM's authentic public key, thus ducking the bootstrapping problem. Whereas, the proposed SIMS uses DBMS to save the checksum measurements. The researchers and technical group of DBMS security have proved the database is sufficient secure because the DBMS environment offers built-in security tools for databases and besides other ad-hoc DBMS security tools can be adopted in this area.

In accordance to an access control, researchers have studied a related problem known as the Chess Grandmaster Problem, Mafia Fraud, or Terrorist Fraud [15], in which a criminal acts as a proved to one honest party and a verifier to another party in order to obtain access to a restricted area. Existing solutions rely on distance bounding [6], is ineffective for a TPM, or employ radio-frequency hopping [15] which is also infeasible for the TPM.

In Kiosk computing, Garriss *et al*. [1] have studied the problem of kiosk computing, a specific case of the problem considered in this work. They have noted the potential for a cuckoo attack (though not by that name).

Another related work is a CD-boot Linuxi [6]. It is a live Linux environment, which is easy to use because it is not installed in a hard disk, but simply boots directly from a CD. This helps secure the sensitive information because a clean environment can be prepared at boot time. For this, the CD-boot Linuxi is adapted to define a trustworthy environment. However, this work trusts the dynamic content because the authors in [6] assumed that if the static content has valid integrity then the dynamic data will not be effect by software attacks.

Recent work in software integrity verification provides techniques for measuring integrity at runtime, where a measurement agent observes the memory image of a running process and creates some meaningful description of the current state of a process [19]. Unlike in static and load time measurement architectures, the target of a runtime measurement is running and hence able to change its state. Therefore, an accurate measurement should reflect a coherent state of the target. A coherent measurement should satisfy two properties:

1. Atomicity ensures that a measurement corresponds to the state of the target at a particular point in time.
2. Quiescence ensures that the target data is in a consistent state, i.e., not a critical section. The authors in [19] address the former property, showing that they can obtain an atomic measurement using a memory copy-on-write strategy.

3. Design of Secure Integrity Measurement System (SIMS)

As discussed above, the problem to be addressed is that the data integrity of operating systems can be compromised. This means that, we cannot trust the program itself because it could be altered and then it produces incorrect results. We cannot also trust the kernel or the BIOS on which these programs are running since they may be tampered with too. In attempt to address this issue, we have developed a novel system called the SIMS for investigating survivability of running of executable content.

3.1. Assumptions

Before we describe the components of our framework architecture, we have established three assumptions because without such restrictions, there would always be adversaries (inside or outside organisation) that are able to trick remote clients or a remote server.

- Ensuring that the measurement list produced by the SIMS has not been tampered with.
- DBMS is created, maintained and operated in a secure manner for ensuring the correctness of the internal state of the DBMS. Therefore, the protection of the hash value database is assumed.
- Our SIMS is not targeted to detect and/or prevent hardware attacks against a system, so we assume that these attacks are not part of the threat model to this work.

3.2. Overview of SIMS Framework Architecture

An illustration of SIMS architecture is presented in Figure 1. This framework of SIMS consists of a number of components:

- *DBMS Table*: Is called original-fingerprint table. This table is used for mapping the hash values of executable content to their specific repositories on a remote client or a server. The DBMS can maintain details for all content in the background by rendering a specified relation as in the original-fingerprint table. The property of this table is append-only, modifications only add information with no information deleted. Hence, if old information is changed in any way then tampering is occurred.
- *Registry Component*: Manages the hashing measurements for all system content that have been developed for use (i.e., loading and running) in the secure environment. In this stage, the hashing measurements are done offline (i.e., the hashing measurements are done before loading). If any content undergoes any add-on content (new code that is added to the original content), it is released. The updated content must be regenerated by producing a new hash value (checksum). In addition, a registry component sends the list of hashing values to DBMS original-fingerprint table for storing and maintaining. Note that modifications to the DBMS original-fingerprint table are not permissible as that would enable an adversary to hide integrity-relevant actions.

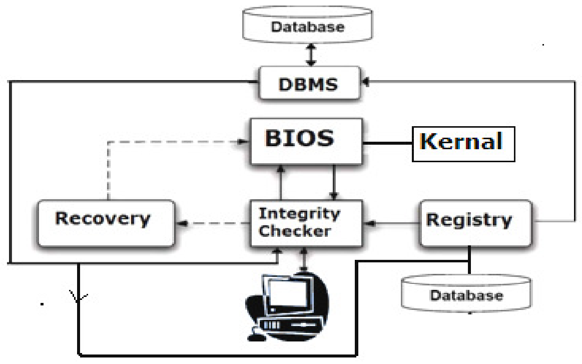


Figure 1: Schematic view of SIMS architecture.

The hashing measurement can take place where one of three conditions is satisfied:

1. When a new executable content is developed for use.
2. When a used executable content is legally changed.
3. When the digital certificate of a used executable content has expired. Some executable content have a digital certificate, so it is important to take this case into our considerations.

On each modification, the DBMS obtains a checksum and computes a cryptographically strong one-way hash SHA-1 function of the (new) data of executable content. However, some developers may not take into account the digital certificate expiration of (used) executable content because they rely on web browsers, web servers and operating system settings. Therefore, an adversary can replace an original element with an expired original element to evade the executable content in a remote machine. To counteract this risk, we compute the checksum using SHA-1 hashing function in concatenation with private key, executable content id, issue date, expiration time and IP address (we use IP address at the application layer).

It should be noted that the secret key is used to sign the executable content element, can be distributed from secure certificate authority or it results from some mathematical equation. In this paper, the secret key relies on the content element and on other factors such as the time factor, which specifies the expiry of a secret key to identify tampering attacks.

We have used a Hash Message Authentication Code (HMAC) in our secure the system environment to ensure the data integrity. MAC is used to compute the checksum of a executable content in the repositories of remote clients. We compute a HMAC via one-way hash functions (SHA-1) and a changeable secret key to create unique fingerprint for each executable content.

In the registry component, the backup process has been established. This process makes a backup of the data files to a directory, disk or computer across the network. A registry component then monitors the executable content and keeps the backup updated with new or changed files. It runs in the background with no user interaction.

- *Integrity Checker*: The integrity checking process aims to detect the alterations in the verification data, which should be significantly much smaller than the multimedia data, can be stored in secure DBMS tables.

When DBMS sends the hash value (checksum) of a loaded executable content to the verification process, the verification process checks to see if the content has been modified since it was used. The cryptographically original checksum is obtainable through the DBMS original-fingerprint table. The checksum of the loaded content is calculated and compared with the one retrieved from the DBMS table. Any tampering causes the content integrity check to fail. If there is a mismatch between the calculated checksum and the original checksum, the content integrity check will fail. Based on whether the test passes or fails, the integrity checker makes the decision about the next step in the process. If the integrity check passes, the executable content is sent to the running process straight away. If it fails, it is sent to the recovery component.

- *Recovery Component*: Recovers the tampered loaded content if the integrity check fails. In other words, if the integrity verification process fails, it is sent to the recovery component, which tries to extract the original content that is known to be safe. Once it is determined that the executable content has been modified in an unauthorised manner, the proposed SIMS will try to recover the original executable content through restoring it from the secure backup, put it in a new assembly and discard the tampered content. When the new assembly is generated, the recovered assembly is sent to its direction and execution continues as normal.

It is important to back up of the data. The backing up of data is the ability to recover that same data and recover it in a timely state to keep a service up and running. It should be noted that each backup copy has unique name for searching purposes. The registry component makes a backup copy for every executable content that has been registered (i.e., register a backup copy in the original-fingerprint table by generating a new assembly of hash value). This backup copy stores in the secure backup for recovery purposes.

The steps of recovery process for web content are as follow:

- When the detection of altered executable content happens at the loading level.
  1. Get a backup Copy (extracting executable content details, getting the executable content name, searching using the unique name in the secure backup).
  2. Verify backup Copy from tampering by the integrity verification process.
  3. Run the backup Copy.
- Discard the tampered content.
- Generate a log file (contains details such as the content information, the original checksum, the re-calculated checksum, the state of database, the state of verification process, and the state of the recovery process).

4. Implementation and Evaluation of SIMS

The SIMS prototype consists of three mechanisms: Registry, integrity checker, and recovery. The SIMS is implemented in Java and MS Access Database. The web servers used are Apache 1.3.20 running on MS Windows Server 2003, and Apache Tomcat 5.01 on MS Windows Server 2003. We have carried out the following two experiments to test the security objective. How does SIMS provide tamper detection and recovery on Apache Tomcat and Microsoft IIS web servers of the proposed SIMS:

- *Experiment 1*: To investigate the tamper detection and recovery on Apache Tomcat web server.
- *Experiment 2*: To investigate the tamper detection and recovery on Microsoft IIS web server.

We have picked over five hundred web requests from the:

1. UK Hillside Primary school web site.
2. Borland JBuilder JSP shopping cart.

We have also identified a potential victim list of target web resources and manually confirmed exploitable flaws in the identified web resources. Over 45 tampering attacks were launched against the designated directories of the suggested web sites hosted on Tomcat and IIS web servers. The results of this experimental study have shown that the proposed system may provide a high coverage of detection and recovery.

Table 1 shows a partial list of measurements for the executable content and Table 2 shows the corresponding list of the same executable content that has been compromised by the tampering attacks. The entries in Table 2 illustrates that after the attack, the checksum of the requested resource *"/cyberventues/st_proj.html"* is different, indicating that the malicious software replaced the original version. The SIMS uses the difference in the hash value to detect whether malicious software has replaced the original resource.

Table 1. A partial list of original hashing measurement.

N	Hash Value (Signature)	Requested Resource
1	D35C729762B3FC8795FB9063 1BCF64DEA061E33B	/cyberventues/st proj.html
2	8AEEED714BACD7AECB74A 054A5BE54A185CC9C52	/Hachett trees07/trees.htm
3	D90FB8C0A13CEA7C87CD51 5E2277E299195A1436	/.../amandahtml/kiana.htm
4	1E7707F952FCBF9627076FAE 387C1E6685FA6192	/.../.../natalie.htm
5	32CE36ED9039D3C2951350C FC5843BC145998CDA	/.../.../nf/nat 1.GIF
…	. . .	…
2941	56236652C0E32364638C5294B 501E786BD0F4B91	/StartHere.jsp
2942	0709306BA800150FB58C6520 F3310AEBD759AA16	/Store.jsp
…	. . .	…
2961	8C967D27FB403E39848F4656 3070046EDA501529	/travel-styles.css
2962	DF9F21A89CB51BDFCADAF D6A0DE728AA2E152808	/tree2/backblue.gif
2963	14D5F457E32810A1D95D21F FE398AB39D110D177	/tree2/fade.gif
2964	B532E95DA8926D183002CA5 6BF26245EF49BD7E2	/.../beginning.html

Table 2. A partial list of hashing measurement after running malicious content manipulation software.

1	9D354FF4B08DDB0FE8BD065 6692AE895C6F36887	/cyberventues/st proj.html
2	83AC2737D5DBAEBA408E35 13AF402158ACE0A4BE7	/Hachett trees07/trees.htm
3	45A9CF81C355F5383E9CD18 C3F7BDDFDB1062003	/.../amandahtml/kiana.htm
4	0887CBC41B81A9C418EF1ED 91C5AC4D4FA93D14A	/.../.../natalie.htm
5	F688117B8752340851B89ECA 5ECC853915815FB8	/.../.../nf/nat 1.GIF
…	…	…

The verification process checks to see if the executable content has been modified since it was used. Based on whether the test passes or fails, the integrity checker mechanism makes the decision about the next step in the process. If the integrity check passes, the executable content is sent to the running process straight away. If it fails, it is sent to the recovery component. The original hash values of *"/cyberventues/st_proj.html"* are shown in Table 1.

The altered hash values after running tampering are shown in Table 2. It is suggested that the SIMS system has the capability to detect and recover executable content that has been tampered with, and hence, the SIMS system satisfies the security objective as defined above.

4.1. Case Study for Micro-Benchmarking Performance

We measured the runtime performance of the Registry mechanism with a set of micro-benchmarks. We measured the latencies of registry mechanism in two different cases, namely, SHA-1 (10 digits) and SHA1-extended (16 digits). In the SHA-1 case, we calculated the hash value of executable content using SHA-1 function (10 digits). The SHA1-extended represented the case when we calculated the hash value of a executable content by SHA-1 function where number of digits was 16. Since the goal is to measure the latency, we ran the registry mechanism 15 times over 200 entries of different sizes for every case (SHA-1 and SHA1-extended) using MS Windows XP Professional. The implementation of the micro-benchmarks is based on the HBench framework industrial standard [5].

An illustration of results is presented in Table 3. It should be noted from this table that the Registry overhead in the case of SHA-1 (10 digits) is low -- the average running time was 1.4274 seconds (representing the average of time taken to run 15 trails), which is less time when using SHA1-extended (2.2176 seconds). These cases do not only measure the overhead of the hash value itself, it also measured all functions in a web register mechanism for both cases (SHA-1 and SHA1-extended).

We have concluded that the SHA1-extended is the most costly in performance terms. This is understandable, because the SHA1-extended contains 16 digits instead of 10 digits. Readers should note that this work as detailed in this thesis is more concerned with security than performance.

It is anticipated that performance gains can be expected from an industrial standard web server. In the case of content, the registry is able to hash the executable content before a request is responded to, however in the case of content, the hashing is required at the time of delivery and hence requires more computerised effort.

We have also presented the registry performance of a executable content as a function of file sizes. We measured the Registry mechanism running time for both: SHA-1 and SHA1-extended, varying the input file sizes. The results are shown in Table 4. When the file size is large, the difference in the hashing overhead can be significant. For example, a 64 Kilobytes file when using SHA1-extended takes about 12.47 milliseconds, where it takes about 3.2 milliseconds for SHA-1. Furthermore, when using SHA1-extended, a 13 Megabytes file records 1531 milliseconds performance overhead, but when using SHA-1, the same file records 620.067 milliseconds performances overhead.

In Figure 2, the performance overhead (ms) with the SHA1-extended case is represented by an unbroken curve, while the performance overhead (ms) with the SHA1 case is represented by a dashed line. The horizontal axis represents the file sizes in byte, and the left vertical axis represents the overhead running time in milliseconds. As expected, the performance overhead has a direct correspondence to the file size, i.e., the larger file size is the greater performance overhead.

As shown in Figure 2, longer keys take much more computing resource to decrypt, and hence make them less vulnerable to attack. However, the SHA1-extended is also more costly in performance terms, but this is the cost that legitimate users pay for higher levels of security. The impact of hashing and encryption are issues that increase the overhead and they are rarely considered in the area of web engineering and design. In this experimental study, we measured the performance for two cases: SHA-1 (10 digits), and SHA1-extended (16 digits).

Table 3. Overhead of a registry mechanism.

Registry Call	*Overhead (ms)*
SHA1-extended (16 digits)	2217.6 (2.2176s)
SHA-1 (10 digits)	1427.4 (1.4274s)

Table 4. Registry performance for both SHA1-extended and SHA-1 as compared with file sizes.

File Size (Byte)	*Overhead (ms) with SHA1-extended*	*Overhead (ms) with SHA1*
1KB	0.64	0.627
16KB	5.13	2.13
64KB	12.47	3.2
2MB	163.73	118.73
5MB	348	251.97
13MB	1531	620.067

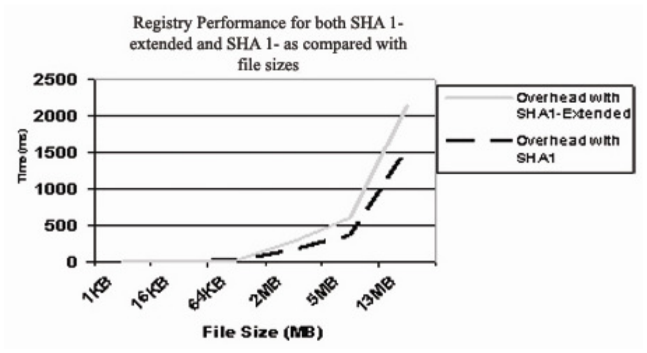


Figure 2. The linear chart of a registry performance for both SHA1-extended and SHA-1 as compared with file sizes.

The designed system SIMS has a number of advantages over other existing solutions as follows:

- The SIMS would also verify the dynamic content as well as the static content.
- The SIMS supports a recovery component to recover only the altered content so it does not need to recover the whole generated static and dynamic content.
- The measurements of content are protected by a secure DBMS rather than using TPM that is part of the TCG standards.
- The proposed system can measure the executable content from the BIOS and the content that is generated at the application layer.
- Unlike some existing solutions, the SIMS architecture is not complex. The SIMS architecture does not involve redundant servers running on diverse operating systems and various operating systems.
- The SIMS increases the end-of-end performance by utilizing a hashing strategy that could minimize the latency for integrity verification.
- The SIMS does not require modifications to existing web application and operating systems architectures.
- The SIMS would not be a blind system that is unable to understand the request and response across a network or on BIOS.
- The SIMS does not rely on a database of known tampering attacks.

5. Conclusions and Future Work

A novel system has been designed and implemented, called a SIMS to provide a practical integrity for flexible and traditional remote systems. The SIMS can be used for various operating systems such as Windows, Linux, and UNIX. All and executable content that are loaded onto any operating system is measured before execution. These measurements are protected by a secure DBMS rather than using TPM that is part of the TCG standards. The proposed system can measure the executable content from the BIOS and the content that is generated at the application layer. In attempt to ensure the survivability of all and executable content, we enable a integrity checker mechanism to detect malicious codes in a content and recover the original content of the compromised one.

Our experimental results indicate that the proposed SIMS has high coverage of detection and prevention against software attacks. In the next part of research, we are interested to evaluate our proposed system on Linux OS. Therefore, we will compare our results with the Linux Kernel Integrity Monitor (LKIM) [10]. LKIM sets up contextual inspection as a means to more completely characterise the operational integrity of a running kernel on Linux.

References

[1] Alexander D., Arbaugh W., Keromytis A., and Smith J., "Safety and Security of Programmable Network Infrastructures," *IEEE Communications Magazine*, vol. 36, no. 10, pp. 84-92, 1998.

[2] Alkassar A., Stuble C., and Sadeghi A., "Secure Object Identification or Solving the Chess Grandmaster Problem," *in Proceedings of the Workshop on New Security Paradigms*, USA, pp. 77-85, 2003.

[3] Arbaugh W., Farber D., and Smith J., "A Secure and Reliable Bootstrap Architecture," *in Proceedings of Security and Privacy, IEEE Symposium*, USA, pp. 0-65, 1997.

[4] Bond M., "Attacks on Crypto Processor Transaction Sets," *in Proceedings of the 3rd International Workshop on Cryptographic Hardware and Embedded Systems*, Springer-Verlag, pp. 220-234, 2001.

[5] Brown A. and Seltzer M., "Operating System Benchmarking in the Wake of Lmbench: A Case Study of the Performance of NetBSD on the Intel x86 Architecture," *in Proceedings of the ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, USA, pp. 214-224, 1997.

[6] Chaum D., "Distance-Bounding Protocols Extended Abstract," *in Proceedings of EUROCRYPT, Lecture Notes in Computer Science*, Springer-Verlag, pp. 344-359, 1993.

[7] Dyer J., Lindemann M., Perez R., Sailer R., VanDoorn L., Smith S., and Weingart S., "Building the IBM 4758 Secure Coprocessor," *Journal of IEEE Computer Society*, vol. 34, no. 10, pp. 57-66, 2001.

[8] Garriss S., C´aceres R., Berger S., Sailer R., VanDoorn L., and Zhang X., "Trustworthy and Personalized Computing on Public Kiosks," *in Proceedings of the 6th International Conference on Mobile Systems, Applications and Services*, USA, pp. 199-210, 2008.

[9] Iliev A. and Smith S., "Protecting Client Privacy with Trusted Computing at the Server," *Computer Journal of IEEE Security and Privacy*, vol. 3, no. 2, pp. 20-28, 2005.

[10] Loscocco P., Wilson P., Pendergrass J., and McDonell C., "Linux Kernel Integrity Measurement using Contextual Inspection," *in Proceedings of the ACM Workshop on Scalable Trusted Computing*, USA, pp. 21-29, 2007.

[11] McCune J., Parno B., Perrig A., Reiter M., and Isozaki H., "Flicker: An Execution Infrastructure for TCB Minimization," *in Proceedings of the 3rd ACM SIGOPS/EuroSys European Conference on Computer Systems*, USA, pp. 315-328, 2008.

[12] Nakamura M. and Munetoh S., "Designing a Trust Chain for a Thin Client on a Live Linux Cd," *in Proceedings of the ACM Symposium on Applied Computing*, USA, pp. 1605-1606, 2007.

[13] Parno B., "Bootstrapping Trust in a Trusted Platform," *in Proceedings of the 3rd Conference on Hot Topics in Security*, USA, pp. 1-6, 2008.

[14] Sailer R., Zhang X., Jaeger T., and Vandoorn L., "Design and Implementation of a TCG-Based Integrity Measurement Architecture," *in Proceedings of USENIX Security Symposium*, USA, pp. 223-238, 2004.

[15] Smith S. and Weingart S., "Building a High-Performance Programmable Secure Coprocessor," *Journal of Computer Network*, vol. 31, no. 9, pp. 831-860, 1999.

[16] Smith S., Gollmann D., Karjoth G., and Waidner M., "Outbound Authentication for Programmable Secure Coprocessors," *Journal of Lecture Notes in Computer Science*, vol. 2502, no. 1, pp. 72-89, 2002.

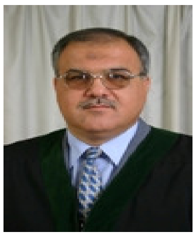
[17] Smith S., "Outbound Authentication for Programmable Secure Coprocessors," *in Proceedings of the 7th European Symposium on Research in Computer Security*, Springer-Verlag, pp. 72-89, 2002.

[18] Thober M., Pendergrass J., and McDonell C., "Improving Coherency of Runtime Integrity Measurement," *in Proceedings of the 3rd ACM Workshop on Scalable Trusted Computing*, USA, pp. 51-60, 2008.

[19] Trusted Computing Group, "Trusted Platform Module Main Specification, Part 1: Design Principles, Part 2: TPM Structures, Part 3: Commands," Version 1.2, Revision 62, 2003.

[20] Trusted Computing Group, available at: http://www.trustedcomputinggroup.org, last visited 2010.

Shadi Aljawarneh holds a BSc degree in computer science from Yarmouk University in Jordan, a MSc degree in information technology from Western Sydney University and a PhD in software engineering from Northumbria University-England. He is currently assistant prof. in faculty of IT in Isra University, Jordan where he has worked since 2008. His research is centered in web and network security, e-learning, bioinformatics, and ICT fields. Aljawarneh has presented at and been on the organizing committees for a number of international conferences and is a board member of the International Community for ACM, ACS, and others.



Abdullah Alhaj he awarded the BSc and MSc degree in computer engineering from Lvov Polytechnic Institute Lvov, USSR in 1988. Between 1991 and 1996 he worked for the Ministry of Education and Altahaddi University, Libya. Later, in 1997 to 2007 he worked as a lecturer for the Ministry of Higher Education (Colleges of education and applied sciences) in Sultanate of Oman. In November 2007 he got his PhD in computer network security from the University of Bradford, UK. His main expertise and areas of interest are in computer networks, network security and computer architecture. He is currently an assistant professor in computer science department, faculty of science and IT, University of Jordan, Aqaba, Jordan.