Journal of Engineering and Applied Sciences 12 (9): 2371-2376, 2017
ISSN: 1816-949X

Proposed System Architecture for Integrity Verification of Embedded Systems

Abdo Ali A. Al-Wosabi and Zarina Shukur
Research Center for Software Technology and Management (Softam),
Faculty of Information Science and Technology (FTSM),
University Kebangsaan Malaysia (UKM), 43600 Bangi, Malaysia

Abstract: Since, the digital devices play essential roles in our daily life, system integrity is important. Thus, there is a need to propose appropriate and effective techniques/tools to verify that the original/pure Embedded Systems (ESs) have been used in those devices. We present our proposed system architecture for ESs integrity verification which includes two main phases: fetching an ES code at a server site (i.e., data center) and examining the ES at a remote site (using a designed user application). The integrity of that ES could be verified by comparing the computed hash value, result could show whether that system has been altered or tampered with. We integrate hash function (SHA-2) with a random key to calculate a unique digest value for a targeted system. Also, we use timestamps and nonce values, two secure keys and public key algorithm to design a security protocol.

Key words: Embedded systems, software tampering, system integrity, integrity verification, computed hash value

INTRODUCTION

Security becomes as the top challenge and interesting research area and human lives are affected directly by the digital revolution (Alcaraz *et al*., 2012; Basile *et al*., 2012; Garcia and Jacobs, 2010). Precisely, our digital world relies on using Embedded Systems (ESs) in most of appliances and digital devices such as mobiles, digital cameras, smart cars and etc. Those systems full-fill a significant part in (almost) all aspects of our modern life. ESs usually accomplishes critical functions: monitoring and controlling real time objects and sensors and processing vital data and information. Defining the security aspects of ESs are related to the main principles of information security: confidentiality, integrity and availability.

However, those systems face substantial challenges, like limited processing power and memory resources and security threats because they usually operate in a physically unprotected environment (Nimgaonkar *et al*., 2013a, b). Attackers often exploit potential vulnerabilities in the system software in-order to gain unauthorized access for utilizing the system or fetching the system's data illegally. For instance, there are a number of real life cases have been reported lately in Malaysia and India.

In fact, the main idea behind our proposed framework is to scan the ES code and then calculate a unique hash value for that code. The digest value could be encrypted and stored into a dedicated database in-order to provide it for later integrity verification process.

Literature review: Based on the conducted literature reviews (Wosabi *et al*., 2015; Wosabi and Shukur, 2015; Ibrahim *et al*., 2015) our research is related to a number of research studies. The combination of numerous hashes for executable software which represents a signature of a whole executable software was recommended. Solution based on analyzing the real-time execution of code section has also been proposed. Zimmer *et al*. (2010) utilize Worst-Case Execution Time (WCET) bounds data to recognize code tampering in real-time Cyber-Physical Systems (CPS). On the other hand, Roger's outline applies a Parallel Message Authentication Code (PMAC) algorithm that consider utilizing a single hardware encryption module for both encryption and validation (Rogers and Milenkovic, 2009). The hardware monitor solution has also been applied by a number of researchers. Mao and Wolf (2010) the researchers presented a checking system to verify the proper software execution. Additionally, combination of hardware and software solutions can be exploited. For example,

Corresponding Author: Abdo Ali A. Al-Wosabi, Research Center for Software Technology and Management (Softam), Faculty of Information Science and Technology (FTSM), University Kebangsaan Malaysia (UKM), 43600 Bangi, Malaysia

Gelbart *et al.* (2009) proposed a system that combined compiler and hardware approach to protect the software. With regards to trusted computing (Garcia and Jacobs, 2010) outlined multiparty processing units (local substations) to compute the sum of their energy consumption without revealing user's information while Kumari *et al.* (2011) proposed the use of control mechanisms for data flow. A number of real world projects on data and code security in ES have been introduced and managed such as EVITA and INSIKA (2010) projects on European countries ("http://www.evita-project.org/", "http://www.insika.de/en/").

MATERIALS AND METHODS

Design overview: The proposed system architecture for ES integrity verification includes two main phases: fetching an ES code at server site (i.e., data center) and examining the ES at remote site, Fig. 1 shows the both phases. Moreover, section 4 explains the proposed protocol in-order to facilitate understanding the system architecture in term of sequential processes.

Fetching system code: Fetching code, at server site, stores an encoded form of the fetched code into a dedicated database and facilitates verification processes whenever requested. This phase Encompasses Scanning ES code, encrypting it and storing the ciphered code into a database. Hence, a necessary application could be designed to perform those functions. Note that, a symmetric encryption algorithm with a secure key could be applied to encrypt the extracted code before saving it into the database. For example, AES has been suggested according to its speed, key size and its immunity against breakable.

A secure cryptographic hash algorithm (e.g., SHA-2) would be implemented to calculate the HMAC. In fact, the integrity information of an ES is extracted by computing a hash value of the software code stored in the system's ROM. An identical secure key would be used as a seed value to calculate the hash values of the remote system and the previously saved code at the server site. Note that, only hash function would be used and there is no need for calculating a digital signature because a secure protocol with challenge-response mechanism is used to validate that the transferred data comes from a trusted parties (i.e., server and user).

Requesting ES code-integrity verification: Requesting the code-integrity verification, at remote site, would be facilitated by downloading a dedicated application on the user's device. To request for system integrity verification, the user must: register online at the server and provide his user id and provide a digital certificate contains his public key. Hence, user data could be stored into a dedicated database and it would be available to be retrieved during the integrity verification of a remote system. After the user completes the registration process successfully, the downloaded application could facilitate capturing the ES Id such as scanning the quick response code of a targeted system.

Once the server receives the user pre-request for integrity check, it would generate a Nonce value (N) and a Time Stamp (TS1). Those values would be encrypted using an earlier saved user's public key and then the encrypted values would be sent to that user. When the user application receives and decrypts those values with its private key, it then verifies the received timestamp value. Once the timestamp verified, the application prepares a request message containing: two secure keys (Key 1 and 2) a hash value of the targeted system code (calculated by key 1) a current timestamp and the received nonce value.

Certainty, those secure keys (i.e., Key 1 and 2) must never send as plaintext, so public key encryption algorithm would be used for key encryption. The server public key would be used to ensure secure communication between the user application and the server. Encrypting the secure keys with the public key of the server would ensure that those keys would be decrypted only by specific entity who does own the corresponding private key (i.e., the server).

Moreover, hash value of the ES code would be calculated to ensure code-integrity of a targeted system. This value would be generated based on the stored code on the targeted system's ROM. The nonce value and the timestamps included in the request and response validation to avoid replay of previous valid remote code-integrity and to avoid that remote-verification requests could be replayed to perform DoS attack (Garcia and Jacobs, 2010; Basile *et al.*, 2012; Alcaraz *et al.*, 2012).

Verifying remote code integrity: When integrity verification request is received from the user application, the code-integrity of the targeted ES needs to be verified. Secure keys (Key 1 and 2) and nonce value would be extracted with the server's private key. Hence, the received HMAC and the Time Stamp (TS2) values would be decrypted using Key 2.

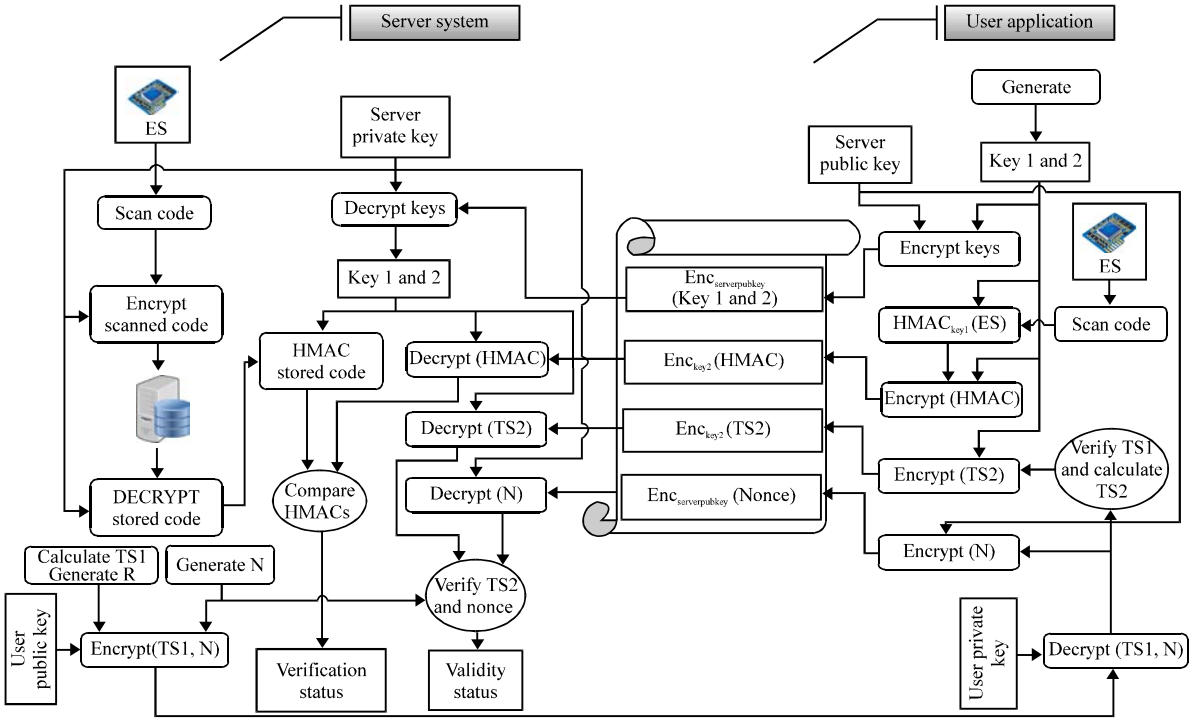

Fig. 1: The proposed system architecture for ES integrity verification

At first, the server validates the received request according to the extracted values of timestamp and nonce (i.e., TS2 and N in Fig. 1). The request is valid only if: the extracted timestamp is within acceptable time range and the decrypted nonce value is equal to the nonce value that previously generated at the server. If this validation fails, server informs the user and ends the verification processes. Otherwise, server retrieves the encrypted code of the targeted ES from the database, decrypts it with the server's secure key, generates HMAC value from the stored code with the extracted Key 1 and decrypts the received HMAC value with the extracted Key 2. It then compares these two hash values in-order to verify the code-integrity of the remote ES. Finally, it notifies the user with the result (either verified code or tampered code) and logs the verification status into the database to be used for further requests (Fig. 1).

Remote code integrity verification protocol: This section outlines the proposed protocol for conducting a code-integrity verification of remote ES. It ensures that messages are authentic, recent and confidential. Additionally, this protocol demonstrates how basic cryptographic primitives could secure the exchanged/transported data and provide validity.

Verification request: The verification applicatio on the user's device initiates the protocol generating a request. User request includes User Identification (User-ID) and the targeted ES identifier (ES-ID). The request would be received by the server and then the server fetches the related data of that user and ES from the database in-order to validate the request (Steps 1-7). Once the request is valid, the server calculates the timestamp and generates the nonce value. Then, it encrypts those two values and sends the ciphered values along with the request confirmation to the targeted user (Steps 8 and 9).

RESULTS AND DISCUSSION

Data delivery for integrity verification: When the user application receives the request confirmation, it uses its private key to extract the received timestamp and nonce values and it then validates the decrypted timestamp. If it is within acceptable range, it scans the ES code (Fig. 2: Step 10 and 11). Then, the application generates the two secure keys (Key 1 and 2). Key 1 would be used to calculate HMAC of the ES code and Key 2 would be used to encrypt the generated hash value. Also, it calculates the current Time Stamp (TS2) and encrypts it with Key 2. Besides, it then uses the server public key to encrypt the

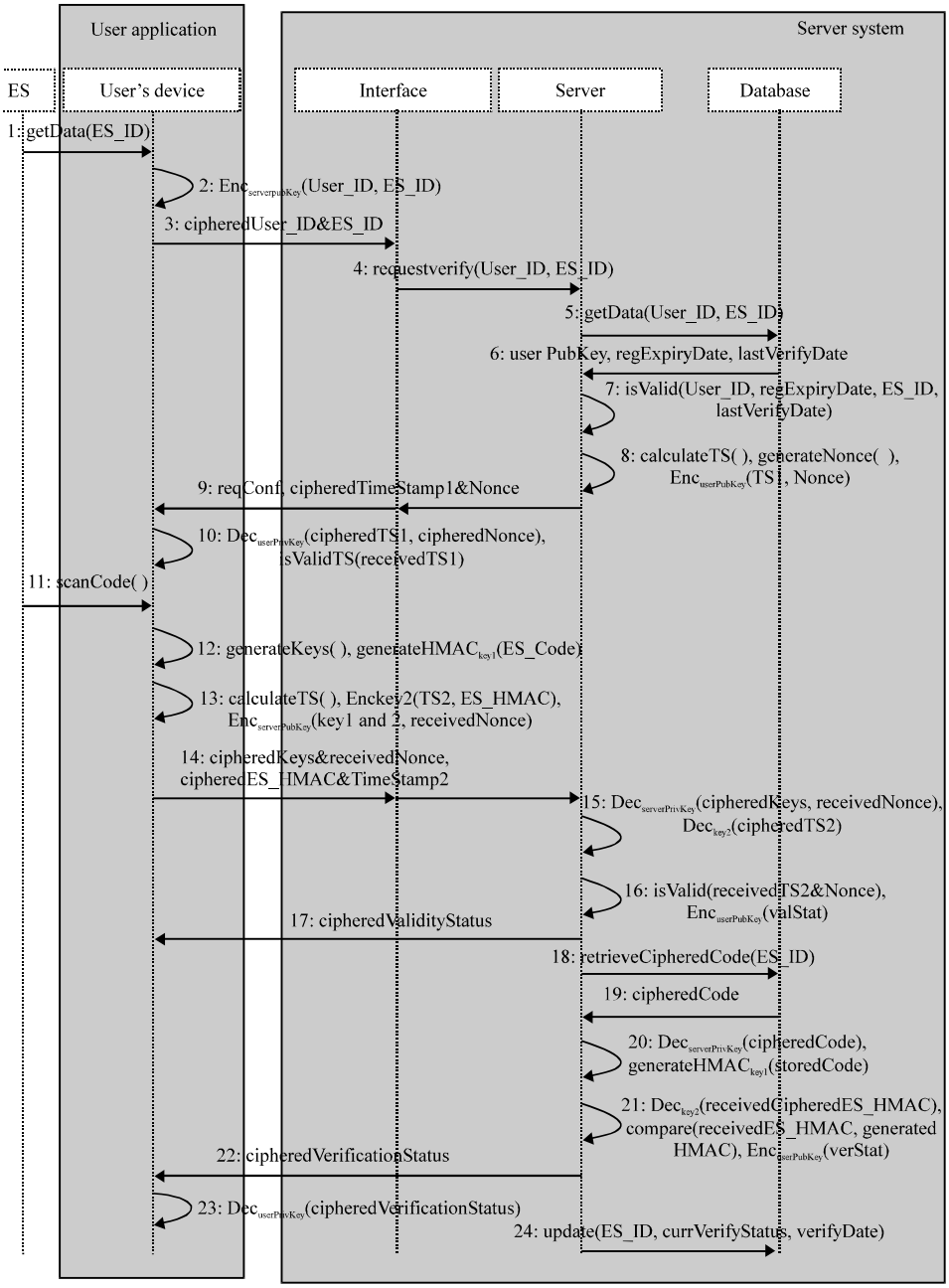

Fig. 2: Remote code integrity verification protocol

secure keys (i.e., Key 1 and 2) and the received nonce value. Hence, all the encrypted values are forwarded to the server via its interface (Fig. 2: Steps 12-14).

Verifying code integrity: Whenever the server receives the ciphered values, it decrypts the secure keys (i.e., Key 1 and 2) and the ciphered nonce by using its private key, decrypts the timestamp by using the decrypted Key 2 and then validates the received request based-on the extracted timestamp and nonce values. If this request is not valid, a notification message would be replied (Fig. 2: Steps 15-17).

Otherwise, the server retrieves an encrypted code of that ES from the database and decrypts it with the private key. It then uses the decrypted Key 1 to generate the hash value of the retrieved code. Also, it uses the decrypted Key 2 to decrypt the received hash value of the ES code and compares it with the generated hash value, result could show whether the ES has been altered or tampered with (Fig. 2: Steps 18-24).

CONCLUSION

Embedded devices are extensively used most of the time in our digital era. However, most of those devices have to function under unconfident settings where attackers could gain physical access. To provide system integrity detection, hash function (such as SHA-2) and basic cryptographic primitives (such as symmetric and asymmetric encryptions) could be used.

In this study, the focus is specifically on verifying ES integrity. The study contains two main contributions. First, it introduces a system architecture for code integrity verification of ES by verifying the digest values of a targeted system. Then, it presents a security protocol for integrity verification using timestamps and nonce values, two secure keys and public key algorithm.

Currently, research is in progress to evaluate the proposed verification protocol. While formal methods are very precise and accurate for presenting system specifications, they are not widely used (Shukur *et al*., 2006; Sullabi and Shukur, 2008). Thus, a number of researchers use the Compiler for the Analysis of Security Protocols (Casper) to translate protocols descriptions into the corresponding process algebra Communicating Sequential Processes (CSP) model. Also, they use the Failure Divergences Refinement (FDR) in-order to describe and analyze those protocols (Shaikh and Devane, 2010; Ryan and Schneider, 2001; Lowe, 1997). So, our future plans include verifying the proposed protocol using those tools. Indeed, this would give a reliable verification measurement in-order to figure-out potential flaws and correct them.

ACKNOWLEDGEMENT

This research is partially supported by grant no. PRGS/1/2015/ICT01/UKM/01/1.

REFERENCES

Alcaraz, C., J. Lopez, R. Roman and H.H. Chen, 2012. Selecting key management schemes for WSN applications. Comput. Secur., 31: 956-956.

Basile, C., D.S. Carlo and A. Scionti, 2012. FPGA-based remote-code integrity verification of programs in distributed embedded systems. IEEE. Trans. Syst. Man Cybern. C. Appl. Rev., 42: 187-200.

Garcia, F.D. and B. Jacobs, 2010. Privacy-Friendly Energy-Metering Via Homomorphic Encryption. In: Security and Trust Management, Cuellar, J., L. Javier, B. Gilles and P. Alexander (Eds.). Springer, Berlin, Germany, ISBN:978-3-642-22443-0, pp: 226-238.

Gelbart, O., E. Leontie, B. Narahari and R. Simha, 2009. A compiler-hardware approach to software protection for embedded systems. Comput, Electr. Eng., 35: 315-328.

Ibrahim, M.A., Z. Shukur, N. Zainal and A.A.A. Wosabi, 2015. Software manipulative techniques of protection and detection. ARPN. J. Eng. Appl. Sci., 10: 17953-17961.

Kumari, P., F. Kelbert and A. Pretschner, 2011. Data protection in heterogeneous distributed systems: A smart meter example. Karlsruhe Institute of Technology, Karlsruhe, Germany.

Lowe, G., 1997. Casper: A compiler for the analysis of security protocols. Proceedings of 10th IEEE Computer Security Foundations Workshop, June 10-12, IEEE, Los Alamitos, CA., United States, pp: 18-30.

Mao, S. and T. Wolf, 2010. Hardware support for secure processing in embedded systems. IEEE Trans. Comput., 59: 847-854.

Nimgaonkar, S., M. Gomathisankaran and S.P. Mohanty, 2013a. MEM-DnP a novel energy efficient approach for memory integrity detection and protection in embedded systems. Circuits Syst. Signal Process., 32: 2581-2604.

Nimgaonkar, S., M. Gomathisankaran and S.P. Mohanty, 2013b. TSV: A novel energy efficient memory integrity verification scheme for embedded systems. J. Syst. Archit., 59: 400-411.

Rogers, A. and A. Milenkovic, 2009. Security extensions for integrity and confidentiality in embedded processors. Microprocess. Microsyst., 33: 398-414.

Ryan, P. and S.A. Schneider, 2001. The Modelling and Analysis of Security Protocols: The Csp Approach. Addison-Wesley, Boston, USA., ISBN:0-201-674718, Pages: 299.

Shaikh, R. and S. Devane, 2010. Formal verification of payment protocol using AVISPA. Int. J. Inf., 3: 326-337.

Shukur, Z., N. Alias, M.H.M. Halip and B. Idrus, 2006. Formal specification and validation of selective acknowledgement protocol using Z/EVES theorem prover. J. Applied Sci., 6: 1712-1719.

Sullabi, M.A. and Z. Shukur, 2008. SNL2Z: Tool for translating an informal structured software specification into formal specification. Am. J. Applied Sci., 5: 378-384.

Wosabi, A.A.A.A. and Z. Shukur, 2015. Software tampering detection in embedded systems a systematic literature review. J. Theor. Appl. Inf. Technol., 76: 211-216.

Wosabi, A.A.A.A., Z. Shukur and M.A. Ibrahim, 2015. Framework for software tampering detection in embedded systems. Proceeding of the 2015 International Conference on Electrical Engineering and Informatics, August 10-11, 2015, IEEE, Sepang District, Malaysia, ISBN:978-1-4673-7319-7, pp: 259-264.

Zimmer, C., B. Bhat, F. Mueller and S. Mohan, 2010. Time-based intrusion detection in cyber-physical systems. Proceedings of the 1st ACM IEEE International Conference on Cyber-Physical Systems, April 13-15, 2010, ACM, New York, USA., ISBN:978-1-4503-0066-7, pp: 109-118.