Towards Integrating Trusted Execution Environment into Embedded Autonomic Systems

Mohamed Sabt, Mohammed Achemlal, Abdelmadjid Bouabdallah

Towards Integrating Trusted Execution Environment into Embedded Autonomic Systems

Mohamed Sabt*‡, Mohammed Achemlal*† and Abdelmadjid Bouabdallah‡
*Orange Labs, 42 rue des coutures, 14066 Caen, France
{mohamed.sabt, mohammed.achemlal}@orange.com
†Greyc ENSICAEN, 6 Bd Maréchal Juin, 14050 Caen, France
‡Sorbonne universités, Université de technologie de Compiègne,
Heudiasyc, Centre de recherche Royallieu, 60203 Compiègne, France
{mohamed.sabt, madjid.bouabdallah}@hds.utc.fr

***Abstract*—Nowadays, there is a trend to integrate trusted computing concepts into autonomic systems. In this context, the Trusted Execution Environment (TEE) was designed to enrich the previously defined trusted platforms. TEE is commonly known as an isolated processing environment in which applications can be securely executed irrespective of the rest of the system. In this work, we propose an architecture in which embedded autonomic systems rely on the properties of TEE to guarantee both their self-protection and self-healing.**

***Keywords*-TEE; self-protection; trusted autonomic system;**

I. Introduction

An embedded autonomic system is designed to be trustworthy in order to avoid failure despite the hostile conditions of their deployment environments. Trustworthiness is an important aspect of self-protection, which is one of the main properties of autonomic computing. It means that an autonomic system must protect itself from malicious attacks from both system user and malicious parties. However, self-protection alone does not ensure trustworthiness. This explains the recent trend to integrate trusted computing concepts into embedded autonomic systems. Cloud computing is also concerned by this trend.

Trusted Computing was defined to help systems to achieve security, privacy and data protection. Originally, trusted computing relies on a separate hardware module, such as the trusted platform module (TPM), that offers a functional interface for platform security. The main shortcoming of the TPM is that it does not provide an isolated execution environment for third-party, thereby reducing its functionality to a predefined set of APIs. A new approach to address trusted computing is to allow the execution of arbitrary code within a confined environment that provides tamper-resistant execution to its applications. We are going to refer to this environment using the term coined by GlobalPlatform in [1], that is *trusted execution environment* (TEE).

Very often, embedded autonomic systems include modules that provide the properties of self-protection and self-healing. Such modules are themselves vulnerable, and hence need to be protected. In this work, we propose to shield these modules inside a trusted execution environment. The protection is achieved by securing the embedded system kernel using two independent mechanisms: introspection, and "trap and emulate". An outline of the architecture is presented in Section III, preceded by a concise definition of TEE in Section II.

II. Trusted Execution Environment

Trusted Execution Environment (TEE) is a tamper-resistant processing environment that runs on a separation kernel following the dual-execution-environment approach [2]. It guarantees the authenticity of the executed code, the integrity of the runtime states (e.g. CPU registers, memory and sensitive I/O), and the confidentiality of its code and data stored on a persistent memory. In addition, it shall be able to provide remote attestation that proves its trustworthiness for third-parties. The content of TEE is not static; it can be securely updated. The TEE shall resist against all software attacks as well as the physical attacks performed on the main memory of the system.

The building blocks are the following: secure boot, secure scheduling, inter-environment communication, secure storage, trusted I/O, secure provisioning and secure attestation.

III. Architecture

Some embedded systems are designed to be reliable, in the sense that they should keep on working without human intervention. With the advent of autonomic computing, reliability is achieved by self-protection and self-healing. Self-healing allows systems to recover from an internal problem, while self-protection prevents systems from being corrupted.

A conceptual architecture of autonomic systems is proposed in [3]. Rather than being dispersed throughout the system software, authors propose to collect all self-management components in one single entity. This entity is called *autonomic manager*. Autonomic manager accomplishes four tasks: (1) monitor: collecting information about the running system; (2) analysis: studying the collected data in order to diagnose the actual state of the system; (3) decision:

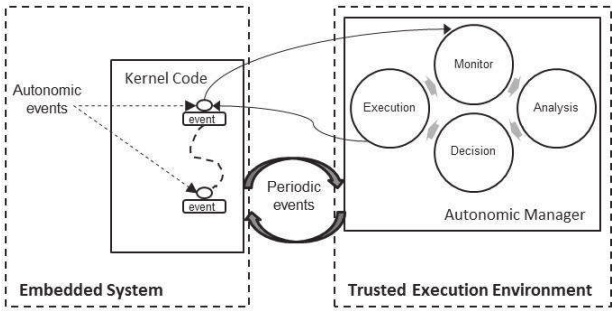

Figure 1. An Overview of our Architecture

adopting the strategies to be taken according to the reached conclusions of the data analysis; and (4) execution: carrying out the decisions that were taken during the previous task. The autonomic manager runs a permanent loop of monitoring-analysis-deciding-execution. This loop allows embedded systems to protect and to fix themselves.

The security of the autonomic manager is of crucial importance. It is must be particularly protected and strongly isolated from the rest of the system. The reliability depends on the proper functioning of the autonomic manager. For instance, if the collected data are tampered with or the analysis module is corrupted, then the decided strategies will not be correctly taken.

We propose to shield the autonomic manager inside a trusted execution environment. The proposed architecture is depicted in figure 1. In our architecture, autonomic manager, which runs inside TEE, provides two security mechanisms: introspection, and trap and emulate. These mechanisms are triggered by two types of exceptions:

- **Software Interrupt Exceptions** that are inserted inside the kernel code and triggered before the execution of certain system instructions;
- **Periodic Exceptions** that are triggered by a secure timer that is managed by the TEE to guarantee the execution of autonomic manager after a certain amount of time.

Security mechanisms based on **trap and emulate** aim at depriving the kernel from executing privileged functions, such as memory management. They force the kernel to route through the autonomic manager for inspection and approval before executing certain instructions. This is achieved using software interrupt exceptions, that we call *autonomic events*. Self-protection is guaranteed by such security mechanisms. Indeed, even a malicious kernel cannot harm the rest of the system, since the TEE must approve the execution of the critical functions. For instance, malicious attacks cannot inject code to the kernel because any write instruction to the memory must be inspected by the autonomic manager whose protection of kernel integrity is one of its roles. Regarding the second security mechanism, namely **introspection**, it aims at constantly monitoring the integrity state of the loaded kernel and react if problems are detected. For instance, the TEE might force the system to halt running if the integrity of the kernel code is not valid. Introspection is achieved using periodic exceptions, and it allows embedded systems to perform self-healing. In this work, we do not define the exact mechanisms for introspection or trap and emulate.

Architectures with similar goals have been proposed. Azab et.al perform real-time protection for kernels of mobile devices using TrustZone-based TEE [4]. It is worth noting that self-protection of systems using TEE is not limited to embedded systems. Lacoste et al. are working on using TEE in order to protect their framework of mobile cloud security management [5].

IV. Conclusion

In this work, we gave a brief definition of TEE and demonstrated how TEE can be used to design better and more trustworthy autonomic systems.

Our main goal is to show that TEE has practical interest and can be used to construct complex systems. However, the lack of formal definition for TEE hinders the adoption of such technology. For instance, some might question the trustworthiness of TEE because of the absence of proper definition and precise architecture of TEE. Future work will focus on defining a framework that computes the 'trust level' for different TEE platforms. We believe that such framework should increase the trust given to TEE, and therefore encourage its integration in more systems.

References

[1] GlobalPlatform, "TEE System Architecture," 2011. [Online]. Available: www.globalplatform.org/specificationsdevice.asp

[2] M. Sabt, M. Achemlal, and A. Bouabdallah, "The Dual-Execution-Environment Approach: Analysis and Comparative Evaluation," in *ICT Systems Security and Privacy Protection*, ser. IFIP Advances in Information and Communication Technology. Springer Berlin Heidelberg, 2015, to be published.

[3] F. Duan, X. Li, Y. Liu, and Y. Fang, "Towards Autonomic Computing: A New Self-Management Method," in *Artificial Intelligence and Computational Intelligence*, ser. Lecture Notes in Computer Science, H. Deng, D. Miao, J. Lei, and F. Wang, Eds. Springer Berlin Heidelberg, 2011, vol. 7002, pp. 292–299.

[4] A. M. Azab, P. Ning, J. Shah, Q. Chen, R. Bhutkar, G. Ganesh, J. Ma, and W. Shen, "Hypervision Across Worlds: Real-time Kernel Protection from the ARM TrustZone Secure World," in *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '14. New York, NY, USA: ACM, 2014, pp. 90–102.

[5] M. Lacoste, A. Wailly, A. Tabourin, L. Habermacher, X. L. Guillou, and J.-P. Wary, "Flying over Mobile Clouds with Security Planes: Select Your Class of SLA for End-to-End Security," in *Proceedings of the 2013 IEEE/ACM 6th International Conference on Utility and Cloud Computing*, ser. UCC '13. Washington, DC, USA: IEEE Computer Society, 2013, pp. 50–59.