The Journal of Systems and Software 84 (2011) 1114–1129

Contents lists available at ScienceDirect

The Journal of Systems and Software

journal homepage: www.elsevier.com/locate/jss

Procedural security analysis: A methodological approach

Komminist Weldemariam*, Adolfo Villafiorita

Center for Scientific and Technological Research, Fondazione Bruno Kessler, via Sommarive 18, Trento 38123, Italy

ARTICLE INFO

Article history:
Received 22 May 2010
Received in revised form
15 December 2010
Accepted 31 January 2011
Available online 1 March 2011

Keywords:
Security assessment
Formal specification and verification
Electronic voting

ABSTRACT

This article introduces what we call procedural security analysis, an approach that allows for a systematic security assessment of (business) processes. The approach is based on explicit reasoning on asset flows and is implemented by building formal models to describe the nominal procedures under analysis, by injecting possible threat-actions of such models, and by assuming that any combination of threats can be possible in all steps into such models. We use the NuSMV input language to encode the asset flows, which are amenable for formal analysis. This allows us to understand how the switch to a new technological solution changes the requirements of an organization, with the ultimate goal of defining the new processes that ensure a sufficient level of security.

We have applied the technique to a real-world electronic voting system named ProVotE to analyze the procedures used during and after elections. Such analyses are essential to identify the limits of the current procedures (i.e., conditions under which attacks are undetectable) and to identify the hypotheses that can guarantee reasonably secure electronic elections. Additionally, the results of the analyses can be a step forward to devise a set of requirements, to be applied both at the organizational level and on the (software) systems to make them more secure.

1. Introduction

Often risks and attacks not only depend upon the security levels software and hardware systems offer, but also on the procedures and controls regulating the way in which they are operated. In such cases, introducing technical security mechanisms—such as Adida (2006), Sastry et al. (2006) and Yee (2007)—is not sufficient. To fill this gap, we focus on the definition of methodologies and techniques to model and formally verify procedures and systems processes. This requires not only the definition of adequate modeling convention, but also the definition of general techniques for the injection of attacks, and for the transformation of business processes into representations which can be given as input to model checkers. For this purpose, we have defined a methodology that allows one to perform security assessment on the procedures. The underlying approach that we follow lies on the intersection of three areas, namely, Business Process Engineering and Re-engineering (BPR), security, and formal methods.

1.1. Motivation

We all take actions to avoid security risks in our daily life. They may be as simple as locking the office door when leaving for a day. For our home computer, maybe it is sufficient to activate the firewall and keep updated on relevant security patches. The situation becomes more complex and difficult if the system we need to protect is a major information system that performs critical steps of complex business processes, whose execution might also require several manual actions to be carried out.[1] This is exactly the case of (voting and) e-voting: even in those countries that have adopted a high level of automation, procedures and controls performed by people on physical assets (e.g., printouts of the digital votes) remain an integral and unavoidable part.

In order to ensure a sufficient level of security, therefore, there is a need for a thorough security risk analysis methodology that considers procedures as part of the modeling and analysis process. The approaches discussed so far, e.g., Fovino and Masera (2006), Basin et al. (2003) and Hogganvik (2007), say little or are otherwise ineffective on these *procedurally rich* scenarios. With "procedurally rich" we denote situations in which software systems are just part of a complex organizational setting, in which procedures have to be executed on security-critical assets that belong both to the digital and physical realms.

We address (some of) the issues above by dealing with the identification, modeling, establishment, and enforcement of security policies about the procedures that regulate access and manipula-

* Corresponding author.
E-mail addresses: sisai@fbk.eu, komminist@gmail.com (K. Weldemariam).

[1] In the rest of the paper we use the terms business processes, processes, procedures, and workflows as synonyms.

0164-1212/$ – see front matter © 2011 Elsevier Inc. All rights reserved.
doi:10.1016/j.jss.2011.01.064

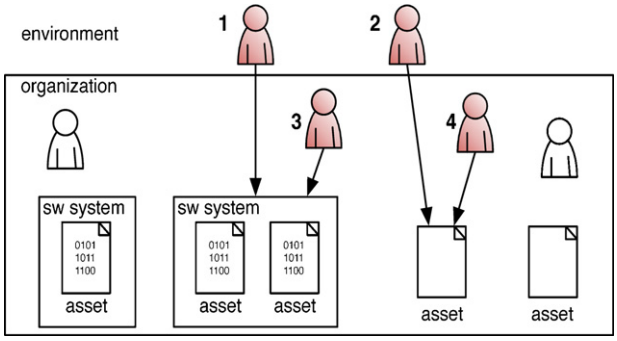

Fig. 1. A reference scenario for procedural security analysis.

tion of critical assets. The breach of security objectives during the execution of the procedures or usage of systems is known as *threat to the procedures* (or *procedural threats*). We call *procedural security analysis* the process of understanding the impact and effects of procedural threats, namely courses of actions that can take place during the execution of the procedures, and which are meant to alter, in an unlawful way, the assets manipulated by procedures.

The reference scenario is shown in Fig. 1. Our target of evaluation, a term borrowed from Common Criteria (Common Criteria, 2007), is a (complex) organizational setting in which procedures transform and elaborate assets. Assets might be made of bits (such as, for instance, an electronic vote stored in a voting machine) or live in the physical world (like, for instance, the paper trail of a voting machine). The procedures and organization are meant to add value to the assets and protect valuable assets from attacks. Think, for instance, of paper ballots: before an election a ballot is a replaceable piece of paper.[2] After a vote has been cast, the same ballot represents the decision of a voter, that needs to be appropriately accounted for and protected.

Attacks in our model might either come from external sources or from insiders, as shown in the figure, where the shaded actors represent a set of adversaries, whereas the non-shaded represent trusted actors.

We distinguish, in particular, the following kinds of attacks:

1. *Attacks on digital assets* (item 1 and item 3 in Fig. 1). These attacks are meant to alter one or more digital assets of an organization. Attacks can either be carried out by external sources (the environment), by internal sources, or a combination of both. Opportunities for attacks are determined by the organizational setting and by the security provided by the digital systems. At attacker, in fact, might need to get access to a digital resource (possibly by circumventing existing control procedures) and overcome the software/hardware protections put in place to protect the digital assets.
2. *Attacks on other kind of assets* (item 2 and item 4 in Fig. 1). These attacks are meant to alter one or more non-digital assets of an organization. Attacks can either be carried out by external sources (the environment), by internal sources, or by a combination of both. Opportunities for attacks are determined by the organizational settings only. The attacks may lead to compromise digital assets as well (e.g., stealing a password provides access to digital assets).

Existing security assessment methodologies, like Fovino and Masera (2006) and Hogganvik (2007), usually focus on understanding items 1 and 3, namely, types and effects of attacks on (software) systems. We propose a tool-supported methodology to tackle also points 2 and 4, namely attacks on assets that are not necessarily digital and that derive from the way in which procedures are implemented and carried out.

1.2. Technical elements of the approach

In order to achieve the goal stated above, we approach the problem by reasoning about the procedures and controls that regulate the usage of systems:

- *Provide models of the procedures under evaluation.* During which we provide models that describe the procedure or procedures to be analyzed. In order to ease the task of translating the models into executable assets flows, we stick to a subset of the Unified Modeling Language (UML) notations (Booch et al., 2005).
- *Extend model.* During which we generate an *extended model* from the models defined in the previous step. The extended model is generated by injecting[3] threat actions into the nominal flow of the procedures. Thus, in the extended model, not only assets are modified according to what the procedures define, but they can also be transformed by the (random) execution of one or more threat actions.
- *Encode the asset flows.* During which we transform the model obtained at the previous step into asset-flows—namely executable specification written in the NuSMV input language (Cimatti et al., 2002), that describe the evolution of assets. The NuSMV model of the asset flows is based on the definition of program counters that ensure that procedures are executed according to the specifications, and by defining one module per asset with one state variable per asset feature. The state variables encode how features change during the execution of the procedures. Accessory information, such as actors responsible for the different activities can be used, e.g., to enrich the language used to express security properties. The necessity of modeling actors roles in NuSMV depends upon the target of the security analysis.
- *Specify security properties to model check.* During which we specify the (un-)desired (procedural) security properties—namely, the security goals that have to be satisfied or violated are then encoded using Linear Temporal Logic (LTL) or Computational Tree Logic (CTL) (Pnueli, 1977) formulas. These, together with the model, are given as input to NuSMV.
- *Perform analysis and results analysis.* During which we run the model checker to perform analyses. If a property is proved to be false, NuSMV generates a counterexample which opens up further discussion. Counterexamples of security properties encode the sequence of actions that have to be executed in order to carry out an attack on an asset.

With this approach analyzing attacks is reduced to a model checking problem in which the required final state of some key assets is expressed using LTL/CTL and the counterexample generated by executing the *extended model* contains the sequence of threat-actions causing the final state not to be reached. The model checker takes care of pruning useless threats, namely threats which do not lead to any successful attack. Analogously to what happens in safety analysis when analyzing, e.g., the loss of critical functions, enhancing the procedures results in reducing the probability of an attack or making the attack more complex, rather than eliminating

[2] Simplifying quite a bit. In fact, by stealing a blank ballot it is possible to implement an attack that allows to control voters.

[3] Note that by fault injection we mean the extension of the assets-flow model with a specification of the possible threat-actions. We adopt this terminology, which is standard in safety analysis, even though it may not be fully appropriate in our domain. See, e.g., Bozzano and Villafiorita (2007).