The Journal of Systems and Software 89 (2014) 87–98

Contents lists available at ScienceDirect

The Journal of Systems and Software

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

A formal methodology for integral security design and verification of network protocols

Jesus Diaz*, David Arroyo, Francisco B. Rodriguez

Grupo de Neurocomputación Biológica, Departamento de Ingeniería Informática, Escuela Politécnica Superior, Universidad Autónoma de Madrid, Spain

ARTICLE INFO

Article history:
Received 8 October 2012
Received in revised form 9 May 2013
Accepted 15 September 2013
Available online 8 October 2013

Keywords:
Secure development methodology
Formal model
Protocols security
Formal verification
WEP analysis

ABSTRACT

In this work we propose a methodology for incorporating the verification of the security properties of network protocols as a fundamental component of their design. This methodology can be separated in two main parts: context and requirements analysis along with its informal verification; and formal representation of protocols and the corresponding procedural verification. Although the procedural verification phase does not require any specific tool or approach, automated tools for model checking and/or theorem proving offer a good trade-off between effort and results. In general, any security protocol design methodology should be an iterative process addressing in each step critical contexts of increasing complexity as result of the considered protocol goals and the underlying threats. The effort required for detecting flaws is proportional to the complexity of the critical context under evaluation, and thus our methodology avoids wasting valuable system resources by analyzing simple flaws in the first stages of the design process. In this work we provide a methodology in coherence with the step-by-step goals definition and threat analysis using informal and formal procedures, being our main concern to highlight the adequacy of such a methodology for promoting trust in the accordingly implemented communication protocols. Our proposal is illustrated by its application to three communication protocols: MANA III, WEP's Shared Key Authentication and CHAT-SRP.

1. Introduction

The application of formal methods for the verification of security properties has received increased attention lately (see e.g. Weldemariam et al., 2011; Mohammad and Alagar, 2011). Additionally, there already exist plenty of automatic tools for the formal analysis of those security requirements (Blanchet, 2010; Andrew and Gordon, 2002). Several methodologies also endorse the application of these tools and theories in order to reach high security guarantees for critical systems and protocols (Hernan et al., 2006; Technical Report, 2009; Matsuo et al., 2010). However, a direct application of these tools may be very resource-consuming during the first stages of the design, since many flaws could be detected just by performing an informal analysis. We propose a methodology for the design of secure protocols that covers the whole design process and tackles the mentioned problems by following an iterative approach where the first steps are less resource consuming than the last ones. Consequently, the complexity of the detected flaws increases as we advance through the methodology steps. Also, special emphasis is made into creating a correct abstraction of the protocol environment and the attacker model. In this vein, the proposed methodology allows a finer control on the attacker modelization in order to fit it to the specific context in which the protocol will be executed. Like we will see, an active application of our methodology helps to reduce the impact of illegitimate actions over the resulting network protocols. In the examples shown here, we make use of ProVerif (Blanchet, 2010). In order to prove the queried security properties, ProVerif tries to automatically deduce them from the set of rules used to formalize the protocol. However, even though we have used ProVerif, any protocol verifier tool or method can be applied without affecting the other parts of our methodology.

The paper is structured as follows. We start in Section 2 with a brief introduction to the existing frameworks and procedures for the verification of security properties, along with a discussion on the advantages of applying the verification procedures at the different stages of the software life cycle. In Section 3 we introduce our methodology. In Section 4 we apply it to three different protocols (MANA III, WEP-SKA and CHAT-SRP). The three cases have been chosen because they are publicly available examples of protocols failing at some point of our methodology (for the first and second cases), allowing us to show its usefulness, while the third protocol has been explicitly created using our methodology, and successfully passes all its tests. Finally, we conclude this work in

* Corresponding author.
E-mail address: j.diaz@uam.es (J. Diaz).

0164-1212/$ – see front matter © 2013 Elsevier Inc. All rights reserved.
http://dx.doi.org/10.1016/j.jss.2013.09.020

Section 5, with a global perspective of the benefits provided by our methodology.

2. Related work

According to the standard ISO27001, the deployment of any information system cannot be interpreted as a product. Certainly, the real implantation of an Information Security Management System (ISMS) is a process following a *Plan-Do-Check-Act* methodology: (1) design of an ISMS, (2) consequent system implementation, (3) resulting product monitoring, (4) maintenance and improvement of the ISMS and (5) eventual re-design of the system to overcome problems not included in the original solution but detected during production. This adaptive procedure should be applied for the definition of any component of an information security system. Thus, in this work we apply it to the design of cryptographic protocols.

On the grounds of the above introduced *Plan-Do-Check-Act* methodology, when creating secure cryptographic protocols we require a framework for assessing the fulfillment of the assumptions made at the design stage, a procedure for evaluating the goodness of the implemented product, and a model for identifying possible problems or security threats. Regarding the evaluation tasks, there are two main types of frameworks aimed to the task of creating secure systems and protocols: frameworks applied at the design phase (Hernan et al., 2006; Technical Report, 2009; Jurjens, 2003; Matsuo et al., 2010), and frameworks applied at the development stage (Swigart and Campbell, 2008; Bhargavan et al., 2010). In turn, the former sometimes rely on tools for automated reasoning specialized in the verification of security properties (Blanchet, 2010; Blanchet and Cadé, 2012; Barthe et al., 2009; Andrew and Gordon, 2002; di Genova et al., 2006; Paulson et al., 2012). Similarly, the latter are usually based on tools crafted for specific programming languages (Goubault-Larrecq and Parrennes, 2005; Chaki and Datta, 2009; Bhargavan et al., 2010; Aizatulin et al., 2011b,a; Bengtson et al., 2011). Although in some proposals only one type of framework is used, it should be noted that both of them should be applied if the goal is the *complete* identification and analysis of security requirements and assumptions. Imagine that we choose to apply a verification tool only to the obtained system implementation, and we indeed find security flaws. It may happen that, while trying to fix them, we realize that one (or several) of them is not just a coding flaw, but a design flaw. That means that we must go back to the design phase and fix the flaw at that stage. Moreover, if the flaw is serious enough, the existing implementation may need extensive changes, which will incur in unacceptable costs and delays. Yet another disadvantage is that code verification obviously relies on the existence of a tool for verifying the specific programming language that has been used. Although these tools are gaining popularity, the huge number of existing programming languages, along with the complexity of creating such a tool, suggests that in the real world we should not trust in having always a code verification tool suitable to our needs. On the other hand, applying a verification tool only to the design does not guarantee that the implementation of that design will also be secure even though the design seems to be so. However, it is technology independent. Thus, the verification of the security properties of both the design and implementation products should be applied when possible.

One major issue when creating security systems is the definition of the security requirements that are expected to be fulfilled (rather than what actions should and should not be executed Brown, 2013). In turn, the *attacker model* has direct influence when it comes to prove if the final system is compliant with those security requirements. This model defines the attacker capabilities that are necessary to conduct the *threat analysis* (Anderson, 2008, Chapter 11). Indeed this is of the utmost importance, since depending on what the attacker can do, a designer/developer may need to protect different resources or take one approach or another. There exist several classifications. For instance, an attacker can be said to be *internal* or *external*, depending on whether it is one of the entities which take part in the protocol/system, or a third party that is not included in it. An attacker can also be categorized as *passive* if the related attacks consist in observing the messages and a subsequent information inference, or as *active* if he/she actually inserts and/or modifies information on any communication link. On the other hand, there are *local adversaries* only threatening a subset of an information/communication system elements, versus *global adversaries* that can access every component of a system (Díaz et al., 2002). Major threats are determined by the so-called *Byzantine adversary*, most commonly used as reference when designing fault tolerant systems, and in the context of anonymous communication systems (Digital Privacy: Theory, Technologies, and Practices, 2007, p. 78). This kind of adversary is an internal attacker that behaves randomly in order to corrupt the system output (Lamport et al., 1982).

Additionally, the attacker model can be further refined by considering the attacker computing capabilities. One main approach is the Dolev-Yao attacker model (Dolev and Yao, 1983), which has proven to be a very powerful abstraction when used in conjunction with methodologies for the formal verification of protocols (Abadi and Rogaway, 2002; Kemmerer, 1987; Paulson, 1998; Houmani et al., 2009). It assumes an *omniscient* attacker who monitors and can modify the messages sent through all communications channels, but cannot break cryptographic primitives. In contrast with this attacker model, there is the computational attacker model (Bellare et al., 2000), which assumes that attackers are Probabilistic Polynomial Time Turing Machines (PPTTM) and hence their computing capabilities are consequently determined. This implies that the cryptographic primitives are not assumed to be perfect and thus any *breakable* primitive in a PPTTM scenario is vulnerable.

As main conclusion of this section we should recall the needs for an adaptive and feedback process when designing any information security protocol. Any possible methodology should be built upon evaluation tools for the goals and assumptions involved both in the design and the implementation stages, and any threat identified during implementation or production should be handled as feedback to further improve the system design. In fact, the more detailed attacker capabilities that are applied within the development level frameworks are not considered in their design level counterparts, and the more general models are ignored during development. This makes sense up to a certain point, since during the design phase the knowledge of the final system is not as deep as it is during its development. Hence, threats that are clear in the implementation stage can be quite obscure in the design process. However, there may be exceptions that could and should be taken into account during the design. For instance, the typical attacker models do not take into account pieces of information that can be easily accessible to an attacker by alternative means, like Facebook pages, and public databases. Although these are very specific facts, the effects that they could have on the ISMS performance can already be detected and considered at the design stage. Consequently, our proposal includes a finer control on the attacker capabilities during the design of the protocol, and sets explicit control points that provide feedback when security errors are found, easing their correction in subsequent iterations. Also, in order to reach the highest assurance levels of other methodologies, it includes the application of procedural (either formal or computational) verification methods.

3. The methodology

Our methodology consists of two main parts, like shown in Fig. 1. The first part includes the specification of the security aims/requirements of the protocol, and the informal evaluation of