Specification and Formal verification of security requirements

Isaac Agudo, Javier Lopez

***Abstract:**. With the grown of internet and distributed applications, security requirements are going inherent to the software development process. Each time one communicates with some other one there are relevant security risk that must be taken in account. This is what is happening in the new soft-ware applications using client/server architecture. We propose including security requirements at the top level of development process, together with functional requirements because they are much related. With this information we are able to extract all communication protocols that are involved in our application and their associated security goals. This is the input to a verification phase in which we look for security flaws. The last step, and the more useful (and the not yet finished) is to use this information to modify our initial specification at the top level of the development process*

***Key words:** Computer Systems and Technologies, Security, formal verification.*

INTRODUCTION

The use of Internet, in particular in the form of world-wide-web and its wireless counterpart, has opened a whole new arena for electronic commerce and electronic business. New services and the existing ones moved on to Internet with an improved quality and have created large revenues. The potential growth in e-business can, however, be held back by concerns about the security of the software systems involved. Internet is notorious for lack of security. Online e-commerce applications are susceptible to failures and exposed to active attacks when not properly designed and tested. A common solution for securing electronic business is to employ crypto-graphic protocols (e.g., encryption, digital signature, authentication, identification, key management, etc.) at application levels. However, many cryptographic protocols are being and will be designed and/or implemented by "hackers", engineers oriented to application problems without appropriate methodologies at hand.

Systems complexity, in particular due to concurrency among systems, has been the main cause of design and/or implementation failures that are introduced into hardware/software systems. Cryptographic protocols are open to a further cause of failures: unlike normal hardware/software systems which are likely to interact with a friendly environment (for instance, a user will try care-fully to only input valid data to a program in order to avoid a run time crash) cryptographic primitives are assumed to interact with a hostile environment. They must be resilient to all imaginable misuses, not only by an attacker who may interact with the system without being invited, but also by a legitimate user (attacker from inside). Attackers make deliberate abnormal uses of the system, and if necessary, they may collude. That is why often cryptographic protocols, even designed and implemented by security experts, are vulnerable to failures. The long hidden failures in Needham-Schroeder protocols are a well-known lesson on unreliability of security experts **[8]**. In the area of design of cryptographic protocols there exists well-thoughts engineering guidelines for the design of cryptographic protocols (e.g., prudent engineering practice of Abadi-Needham **[1]** and robustness principles of Anderson-Needham **[2]**). However, these guidelines do not form a computational theory and therefore cannot lead to an automatic or even a computer-aided design method. Formal analysis, on the other hand, has been shown to be effective in identifying security flaws in many key distribution and authentication protocols. Several different approaches have been developed. Meadows developed a PROLOG based model checker (NRL Protocol Analyser)**[6, 7]**. The user supplies a description of an insecure state and the PROLOG searches backwards in an attempt to find an initial state. One serious problem is that the back-tracing algorithm is not guaranteed to terminate. Lowe **[5]** used the FDR model checker for CSP **[4]** to find successfully a previously unknown error in the Needham-Schroeder public-key authentication protocol. Schneider **[11]** uses CSP to model protocols

in a hostile environment and to express security properties. Verification proceeds by the discovery of a rank function.

SPECIFICATION OF SECURITY REQUIREMENTS

We aim to include the specification of security requirements into the design phase of software development. As in the framework of project CASENET[1], we need to integrate some modelling language, like UML **[13]**, and some security specification language, like SRSL **[14]**. This could we done by adding pre- and post-conditions to UML diagrams, as shown in the following figure:

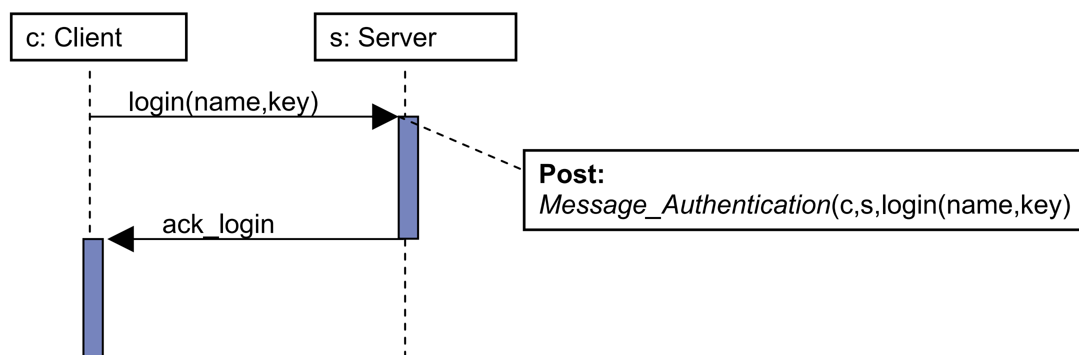


Figure 1: High level specification example

The above figure (Figure 1) shows an elementary example, where a client ***c*** is trying to log in a sever ***s***. Only to message are interchanged between ***c*** and ***s***, the first one consist on a login name and a password and the second one is only a confirmation message. The requirement means that after ***s*** receives the login message she must believe it is authentic, i.e. It comes form the same ***c*** that is specified in the message.

As this is a high level specification we need to translate it to a low level one in order to analyse the security requirements. We then establish two different levels, which we name application and protocol levels respectively. These levels have to be related in both directions in order to allow a feedback from the analysis level.

Into the protocol level we can also distinguish two different sublevels, a middle level in which we specify the protocol as a sequence of message and a very low level in which we use some formalism that allow us to express the protocols steps together with security requirements. In this case we opt for using APA, a formalism based on Automata theory that will be described later.

The following figure (Figure 2) describes the whole process of specification and analysis and the main technologies we use at each step.

[1] http://www.casenet-eu.org

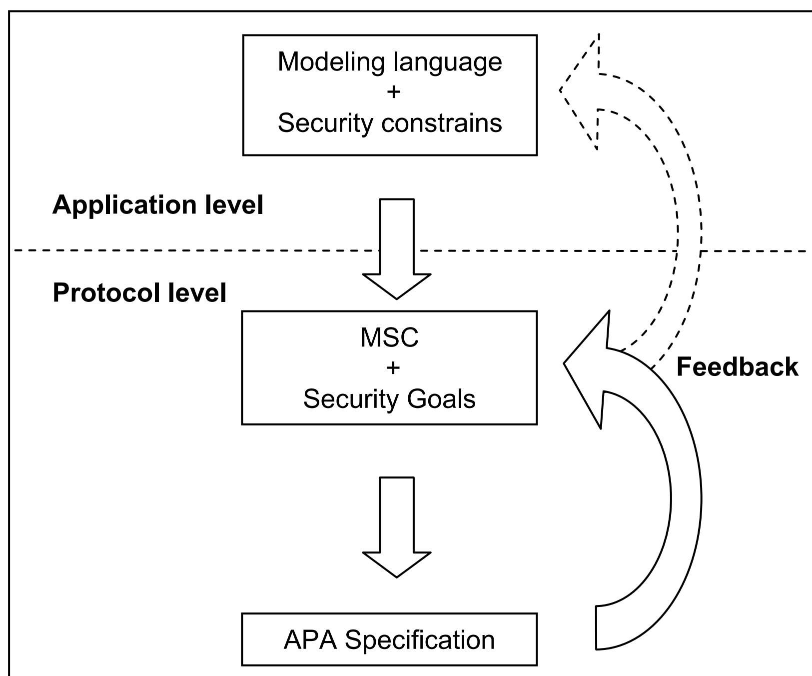


Figure 2: Full cycle

PROTOCOL LEVEL

At the protocol level, we expect a description of the protocol as a Message Sequence Chart (MSC) **[14]** and, additionally, the security goals inherited from the application level.

These security goals could be required only at a specific protocol step. In that case we consider it an active requirement because it depends on a specific action (authenticity). On the other hand, it may depend on a global one (confidentiality). In that case we consider it a passive requirement because it does no depend on a specific action.

We translate this specification into an Asynchronous Product Automata and then we analyze it using SHVT (Simple Homomorphism Verification Tool) **[15]**.

If a flaw is found, this means that some state of the APA violates any of the "security goals". A report is obtained that consists of a path from the "initial state" of the protocol to the state where the flaw has been found. The path of state transitions can be translated to an analogous path consisting of protocols steps (MSC). That information helps to understand how to modify the initial protocol specification. Further work has to be done in order to transport this information to the application level **[14]**.

MESSAGE SEQUENCE DIAGRAM (MSC)

A MSC specification of a security protocol consist mainly on:

- *Header*: Describes the scenario, actors, keys, and cryptographic functions we use, as well as the initial knowledge of each actor.
- *Protocol Steps*: Describe how actors exchange messages using the following notation:

  A: **out** (**in**) m1, 1(**{**M**}**K) **to** (**from**) B;

  This example specify that *A* sends to *B* the message *M*, encrypted with key *K*

(B is the intended recipient). We use *m1* and *1* as labels to reference the protocol step at the security analysis.

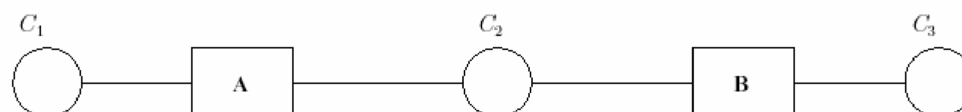
- *Security Goals*: Describe the security goals that must hold after each protocol run or a specific protocol step.

ASYNCHRONOUS PRODUCT AUTOMATA

Asynchronous Product Automata (APA) are a universal and very flexible operational description concept for cooperating systems. It naturally emerges from formal language theory.

An APA can be seen as a family of elementary automata. The set of all possible states of the whole APA is structured as a product set, and each state is divided into state components.

In the following, the set of all possible states is called *state set*. The state sets of elementary automata consist of components of the state set of the APA. Different elementary automata are put together by shared components of their state sets. Elementary automata can communicate by changing shared state components.



The previous figure shows a graphical representation of a small asynchronous product automaton consisting of two elementary automata, *A* and *B*, and state components C_1, C_2 and C_3. State sets are Z_{C1}, Z_{C2} and Z_{C3}, which are domains of state components). The state set of the APA, as well as the state set of *A*, is the product of Z_{C1} and Z_{C2}. The state set of *B* is the product of Z_{C2} and Z_{C3}. The full specification of the automaton includes the transition relations of the elementary automata and the initial state. The neighbourhood relation *N* (represented as lines) indicates which state components are included in the state of an elementary automaton and may be changed by a state transition of the elementary automaton. A state transition of automaton *A* may change the content of C_1 and C_2 while *B* may only change C_2 and C_3.

In this example, C_2 represents the network communication channel, because it is the only state component that can be accessed by both *A* and *B*. C_1 and C_3 represent the internal memory of *A* and *B*, respectively. For a detailed definition of Asynchronous Product Automata see **[12]**.

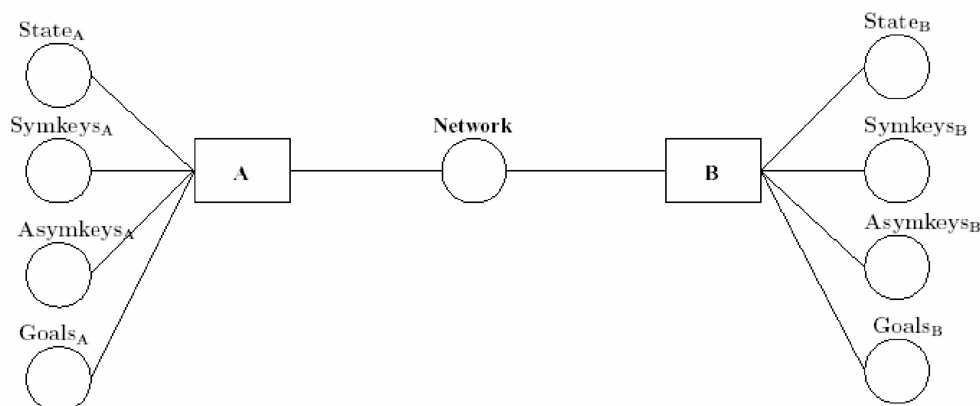
Together with the previous structure it is necessary to specify the state transitions, which define the APA behaviour.

SPECIFYING SECURITY PROTOCOLS WITH APA

In order to specify a security protocol we need to firstly define the domain of the State components (Z_C), and the initial state of the automata. Secondly, we need to define the state transition patterns, what mainly corresponds to the protocol steps, and also to internal computation steps like generation of nonce.

As for modelling nonces and generated keys we use global State components named Nonces and Keys. Each time an Agent needs to generate a nonce or a key, she reads it form the corresponding State component, and removes it to prevent non-freshness of this item. Thus, we allow exhaustive model checking by defining all the possible nonces and keys, but also propose the use of formal language methods to avoid state explosion problems.

The general model corresponds to this figure:



where:

- *State*: Stores information about the status of the protocol, like which is the next thing the Agent has to do.
- *Symkeys*: Stores all the symmetric keys the Agent uses.
- *Asymkeys*: Stores all the asymmetric keys the Agent uses.
- *Goals*: Specifies the security goals we expect for each step of the protocol. We use a logic language for describing the security requirements in which we define a few predefined predicates. One example is:

$$\forall P \in \mathcal{A}gents \setminus \{A, B\} : \textit{not_knows}(P, K)$$

TRANSLATING MSC TO APA

Firstly, the header of the MSC specification provides the information for defining the initial State of the APA. It includes number and name of agents, number of simultaneous protocol runs, possible nonces and generated keys. It also includes initial knowledge of participants, like messages to be sent or private, public and shared keys.

Next, for each step of the protocol we translate it into a proper automata transition according to the items used in the message. As stated before, we use the state component *State* to store information about the protocol flow. When translating a protocol step, all the previous steps that were stored in the state component *State* comprise the context of this transition and the results of this transition are added to *State* to be used in followings transitions. This process sorts the transitions as in the MSC specification.

CONCLUSIONS AND FUTURE WORK

Using formal methods for design of software is a well known way to produce reliable software. It also prevents the necessity for error correction and allows a better understanding of the application's functionality.

We suggest the use of this well know software engineering techniques together with the specification of security requirements. For this purpose we propose some tips for the integration of the analysis of security protocols with software design techniques.

Further work has to be done in order to close the gap between specification of security requirements and formal analysis. Although it is relatively easy to translate these high level security requirements to a proper formalism and analyse it, there is not a unified formalism that allows analysis of all security formalisms. Moreover, there is not an easy

way to translate the report of analysis to the upper level in order to modify the specification.

REFERENCES

[1] Martin Abadi and Roger Needham. Prudent Engineering Practice for Cryptographic Protocols. In Proceedings of the 1994 IEEE Computer Society Symposium on Security and Privacy, pages 122–136, Los Alamitos, California, 1994. IEEE Computer Society Press.

[2] Ross Anderson and Roger Needham. Robustness principles for public key protocols. In D. Coppersmith, editor, Advances in Cryptology – CRYPTO '95, volume 963 of Lecture Notes in Computer Science, Berlin, 1995. Springer Verlag.

[3] G. Bella and L.C. Paulson. Kerberos version iv: Inductive analysis of the secrecy goals. In 5th European Symposium on Research in Computer Security, Lecture Notes in Computer Science, pages 361–375. Springer-Verlag, 1998.

[4] C. Hoare. Communicating Sequential Processes. Prentice-Hall, 1985.

[5] G. Lowe. Breaking and fixing the Needham-Schroeder public-key protocol using CSP and FDR. In Second International Workshop, TACAS '96, volume 1055 of LNCS, pages 147–166. SV, 1996.

[6] C. Meadows. Applying formal methods to the analysis of a key management protocol. Journal of Computer Security, 1992. [7] C. Meadows. The NRL protocol analyzer: An overview. In Proceedings of the Second International Conference on the practical Applications of PROLOG, LNCS. SV, 1995.

[8] R. Needham and M. Schroeder. Using encryption for authentication in large networks of computers. Communications of the ACM, pages 993–999, 1978.

[9] L. C. Paulson. The inductive approach to verifying cryptographic protocols. Journal of Computer Security, 6:85–128, 1998.

[10] L. C. Paulson. Inductive Analysis of the Internet Protocol TLS. ACM Trans. on Information and System Security, 2(3):332–351, 1999.

[11] S. Schneider. Verifying authentication protocols with CSP. In IEEE Computer Security Foundations Workshop. IEEE, 1997.

[12] S. Gürgens, P. Ochsenschläger, C. Rudolph. Role Based Specification and Security Analysis of Cryptographic Protocols Using Asynchronous Product Automata, DEXA Workshops, 2002.

[13] Javier Lopez, Jose A. Montenegro and Jose L. Vivas. Towards a UML-based framework for security requirements engineering, International Workshop on Requirements for High Assurance Systems IEEE, 2003

[14] Javier Lopez, Juan J. Ortega, Jose M. Troya. Protocol Engineering Applied to Formal Analysis of Security Systems, In Proceedings of the International Conference on Infrastructure Security, InfraSec 2002 Bristol, UK, October 1-3, 2002.

[15] P. Ochsenschläger, J. Repp, and R. Rieke.The SH-Verification Tool, In Proceedings of 13th International Florida Artificial Intelligence Research Society Conference (FLAIRS-2000), pages 346-350, Orlando, FL, USA, May 2000.

ABOUT THE AUTHORS

Isaac Agudo, PhD Student, Department of Languages and Computer Science, University of Malaga, Phone: +34 952 13 3371, E-mail: isaac@lcc.uma.es.

Professor Dr. Javier Lopez, Department of Languages and Computer Science, University of Malaga, Phone: +34 952 13 3371, E-mail: jlm@lcc.uma.es