

Incorporating security requirements into communication protocols in Multi-Agent Software Systems

Yuxiu Luo, Giannakis Antoniou, Leon Sterling
Department of Computer Science and software Engineering
University of Melbourne, VIC, 3010
{yxluo, gant, [leon](mailto:leon@csse.unimelb.edu.au)}@csse.unimelb.edu.au

Abstract

A communication protocol is a fundamental component of a multi-agent system. The security requirements for a communication protocol should be articulated during the early stages of software development. However, there is no formal way provided for software developers to find out what makes a communication protocol secure and what are secure designs. In this paper we propose a method that defines security requirements, bridges security requirement analysis with security design, and integrates the security techniques into a communication protocol to fulfill the security requirements.

1. Introduction

Security has become critical for robust software systems. Security requirements (SRs) should be considered in the early stage of software development life cycle (SDLC). However, security is managed in an ad-hoc fashion and as an afterthought. This leads to problems and serious design challenges [1]. Two of the reasons for poor security engineering are the lack of security knowledge of software developers and stakeholders [3], and a gap existing between SRs analysis and security design;

In software engineering, requirement analysis methods analyze SRs from several aspects. Design methods embed security related information into system design. Product-oriented approaches evaluate if the final product satisfies the SRs. Process-oriented approaches define development steps to deliver security critical software. Agent-oriented software engineering methodologies are extended to address SRs. But none of them advise stakeholders/inexperienced developers about basic SRs nor provide enough information for developers to build security. In security research, security techniques (STs) are developed to satisfy SRs. However, it is hard for software engineers to learn the techniques, understand their characteristics and choose a proper ST within

limited time. For communication protocol (CPs), development approaches, encryption algorithms and techniques that verify if a CP fulfills the SRs are proposed. They do not provide the solution to the problem defined. As show by above analysis, security can only be built at later stage of SDLC by experts.

This poster sketches a method that (1) allows inexperienced developers to define SRs; (2) bridges the gap between SR analysis and security design; and (3) secures CP by choosing proper security techniques. We do not analyze specific CPs or specific STs. Section 2 explains notations and security background. Section 3 shows our method. Section 4 presents an example. Section 5 concludes.

2. Notations and Background

The protocol where Alice sends a message to Bob is presented as: Alice→Bob: message;

The security services (SSs) that make a protocol secure are confidentiality, integrity, non-repudiation and authentication. In some cases, the freshness of a message should be guaranteed to prevent a replay attack. A random number, nonce, should be included as: Alice→Bob:S_{Alice}(msg, nonce).

STs support SSs. **Digital signature** (DS) achieves non-repudiation and integrity of the message and the authentication of the sender. Applying DS (with PKI) is represented as: Alice→Bob:S_{Alice}(msg); S_{Alice}(msg) means that message msg is signed by the private key of Alice. **Encryption** ensures the confidentiality of a message. Asymmetric encryption (AE) is represented as: Alice→Bob:{msg}K_{Bob}; {msg}K_{Bob} means that message msg is encrypted with Bob's public key; Symmetric encryption (SE) is represented as: Alice→Bob:{msg}K_{Key}; {msg}K_{Key} means that message msg is encrypted with the secret key Key;

3. Method

1. Establish CP: after system goals and roles are specified in analysis phase [4]. Protocols are established to model how roles are involved to fulfill system goals. In this step, security issues at enterprise

level (e.g. security policies, business rules/laws) are captured. In design phase, we define the agents that fulfill roles. Protocols are then refined by specifying the sequence of interactions among agents and the messages exchanged in each interaction.

2. Define SR: the security of a protocol is affected at two levels. First, at enterprise level, the protocol should be defined correctly. The sequence, the pre-condition, the post-condition of interaction and role/agent involved must comply with organization security concerns. Second, at system level, transactions and the messages exchanged should be secured. In this step, we define SRs for the protocol at system level. Security attributes (SAs), integrity, confidentiality, availability, authentication, non-repudiation and accountability, are the common features that a secure system should have [2] (and thus for a CP); therefore they imply the basic SRs for a protocol. Within the context of protocol, the first three are concerned with messages exchanged; the later three are concerned with transactions. General SRs for a protocol can be defined by attaching the first three SAs to each message exchanged and by attaching the three later SAs to each transaction of the protocol.

3. Enhance SRs into CP: this step bridges between SR analysis and security design bidirectionally. First, we apply corresponding STs to support the SRs defined for the protocol (see 1st and 2nd column in table 1¹) Several STs can be available for one SR; developers/stakeholders decide based on advantages and disadvantages of a ST (see table 2) and available project resources. Second, we analyze whether the protocol can be attacked and apply STs to avoid it when possible attack is found (see 3rd and 2nd column in table 1).

Table 1. Security Repository Example

SR	ST	Prevention /attack
Confidentiality	SE, AE	Interception
Integrity	DS	Message modification
Non-repudiation	DS	Deny of an action
Authentication	DS	Masquerade

Table 2. Security Technique Selection Example

ST	Advantage	Disadvantage
AE	Simple key management	Computationally intensive algorithms
SE	Less computationally intensive algorithms	Complex key management, must have secure key sharing channel

¹ Availability (readiness of a correct service or message) is not applicable in protocols because all the exchanged messages are available. Accountability (availability and integrity of the identity of the person who performed an operation) is out of protocol context, because once we ensure the integrity, the accountability is automatically satisfied. Thus they are not in table 1.

4. Example

We demonstrate only steps 2 and 3 due to space limit. A correct CP with security issues captured at enterprise level is assumed to be available (see right part of Figure 1). Alice submits advertisement (ads), auction due date, credit card information and minimum acceptable offer (limit) to auction web site (AWS). Bob requests ads (Request Ads), bids (Bid) for product and gives credit card information to AWS. AWS shows ads (Ads Info) and informs bid results (Bid accepted).

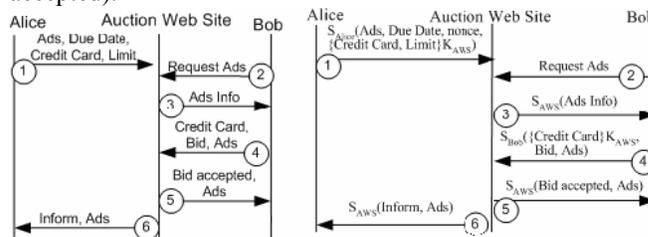


Figure 1. Fulfill SRs for Communication Protocol

We take the first transaction as an example. First, we define SRs by attaching SAs needed for the exchanged message and for the transition. Then we have SRs: Ads (*integrity*); DueDate (*integrity*); Credit Card (*confidentiality, integrity*); limit (*confidentiality, integrity*); Alice → AWS: Ads, Due Date, Credit Card, Limit (Non-repudiation), Second, we integrate security by applying proper ST. We use DS for non-repudiation, integrity and AE for confidentiality. Besides, the freshness of the message should be guaranteed to prevent malicious intermediate users from re-submitting a message copy to AWS. Therefore we have: Alice → AWS: $S_{Alice}(Ads, Due Date, nonce, \{Credit Card, Limit\}; K_{AWS})$. For transaction: Bob → AWS: Request Ads, no SR is required because both the message and the transaction are public. Left part of the figure 1 shows the secured CP produced by our method.

5. Conclusion

We present a simple method for software engineers to enhance SRs into CP in multi-agent system. We can improve it by considering relationships among different roles/agents.

6. References

- [1] R. Anderson, *Security Engineering: A Guide to Building Dependable Distributed Systems*, 2001.
- [2] A. Avizienis, J. C. Laprie, B. Randell, and C. Landwehr, "Basic concepts and taxonomy of dependable and secure computing", *IEEE Trans. Dependable and Secure Computing*, 2004
- [3] B. W. Lampson, "Computer security in the real world", in *Annual Computer Security Applications Conference*, 2000.
- [4] Y. Luo, L. Sterling, K. Taveter, "Modeling a Smart Music Player with a Hybrid Agent-Oriented Methodology", in *Proc. of International Requirements Engineering Conference*, 2007