Zisman, A. (2007). A Static Verification Framework for Secure Peer-to-Peer Applications. Paper presented at the Internet and Web Applications and Services, 2007. ICIW '07. Second International Conference on, 13 - 19 May 2007, Mauritius.

A Static Verification Framework for Secure Peer-to-Peer Applications

Andrea Zisman
Department of Computing
City University – Northampton Square, EC1V 0HB
London, UK
a.zisman@soi.city.ac.uk

Abstract

In this paper we present a static verification framework to support the design and verification of secure peer-to-peer applications. The framework supports the specification, modeling, and analysis of security aspects together with the general characteristics of the system, during early stages of the development life-cycle. The approach avoids security issues to be taken into consideration as a separate layer that is added to the system as an afterthought by the use of security protocols. The main functionality supported by the framework are concerned with the modeling of the system together with its security aspects by using an extension of UML, modeling of abuse cases to represent scenarios of attackers and assist with the identification of properties to be verified, specification of properties to be verified in a graphical template language, verification of the models against the properties, and visualization of the results of the verification process.

1. Introduction

In the last years, various approaches have been proposed to address security issues during the development of software systems. A new area of research called *secure software engineering* has emerged to support the integration of security and software engineering [1][27]. This new area of research has been proposed to fulfill the lack (a) in existing approaches, techniques, and methodologies in the area of software engineering to provide support for the analysis and design of security requirements and properties, and (b) in existing approaches for security engineering which concentrate on security issues and consider limited aspects of the software system as a whole. The normal procedure is to add security characteristics as another layer in the system on an ad-hoc basis after the system is built, which generates conflicts [10][27]. Moreover, in such setting, an application is considered to be secure when it uses security protocols and cryptographic techniques.

The use of security protocols and cryptographic techniques gave rise to the development of formal verification techniques for security protocols [1][17][18][25][29]. Verification of software systems has been the subject of extensive research in which various approaches and tools have been proposed to support checking and analyzing that the system conforms to its requirements and specifications [8][12][19]. However, existing formal verification techniques that supports security are (i) limited to verify protocol designs, (ii) cannot guarantee security properties of protocol implementations, (iii) do not consider the system as a whole, and (iv) uses disjoint security models and system design models that are expressed in different ways [20]. As suggested in [3][10][11][15][24][26], security should be considered from the early stages and through all the stages of software development. Therefore, it is necessary to propose verification approaches that consider security aspects to be specified, modeled, and analyzed during early stages of the development life-cycle, as well as embedded with other characteristics of the system, and not as a separate layer that is added to the system as an afterthought in the form of security protocols.

The above situation is important to peer-to-peer applications. The high distribution, autonomy, and dynamic characteristics of the peers may cause security vulnerability to the system as a whole. Therefore, it is necessary to consider security since early stages of the system development and to consider the verification of security issues together with other aspects of the system.

The work presented in this paper focus on a static verification framework to support the analysis of abstract behavioral specifications of secure peer-to-peer systems. More specifically, the static verification framework requires capabilities to address some important challenges including the provision of support for:

- The construction of design models representing abstract behavioral of peer-to-peer system specifications that takes into consideration both security and general characteristics of the system;
- The construction of abuse cases representing scenarios of attackers in order to consider ways in which the system can be attacked, identify threats to the system, and elicit ways in which the security of the system can be invalidated;
- The specification of security and general application properties (requirements) to be verified against the abstract behavioral system specifications;
- The static verification of system specifications against security and general application properties (requirements);
- The visualization of the results of the static verification process.

The above challenges have been specified based on requirements and scenarios identified by industrial partners in the areas of media and security in an European project focusing on mobile peer-to-peer security (PEPERS) [28].

These challenges, scenarios, and requirements have been the main drivers to the framework described in this paper.

The static verification framework assumes the use of an extension of UML (viz. UMLSec [21][22]) to describe the design models of the system to be verified and the use of UML to specify the abuse cases. The framework also includes a property editor to support the specification of the properties to be verified and a property selector to assist the designers to choose the properties to be verified from a library of properties. The verification of the properties against the design models is executed by the model checker SPIN [31] after translating the properties and design models into LTL [34] and SPIN Promela language, respectively. The verification of security protocols is executed by using AVISPA tool [4]. The results of the verification process is presented to the designer by highlighting the parts in the design models that have violated a certain property, by indicating that the property has not been violated, or by indicating that a conclusion cannot be drawn from the verification process for a certain property.

The remainder of this paper is structured as follows. In Section 2 we present an example that will be used throughout the paper to illustrate our approach. In Section 3 we describe the static verification framework. In Section 4 we give an account of related work. Finally, in Section 5 we summarize the work and present directions for future work.

2. Example

In order to illustrate, consider an example of a peer-to-peer system to exchange job tasks and associated data between different journalists and photographers working in a media company. This example has been extracted from the scenarios specified by the industrial partners of the PEPERS project. In the example the system is composed of several peers with different roles such as managers, chief journalists, journalists, photographers, editors, legal and advertisement staff. The access to the system by the peers is either by the use of desktop computers or mobile devices (PDAs).

Consider the scenario with six peers participating in a new assignment. These peers have the roles of one manager, one chief journalist, two journalists, one photographer, and one editor. Suppose that an important international event is about to happen and the manager of the company identifies this potential event and contacts one of the chief journalists to discuss a plan to cover the event. The chief journalist selects a team of people that will be able to cover the event. This team of people is composed of two journalists, one photographer that works together with one of the journalists, and one editor. The two journalists and the photographer are already on the site of the event and will be responsible for covering, writing notes, and taking pictures of the event. The editor is located in the main branch of the company and should dedicate all his time to give support for preparing and editing the reports of this event during the next three days.

Given the importance of the event, the manager and chief journalist would like their company to have exclusivity of the story and, therefore, all information and requests exchanged between any of the peers should not be intercepted by anyone else inside and outside the company. In addition, each of the two journalists covering the event should not exchange notes and reports about the event, since the chief journalist is interested in receiving independent information about the event from each journalist. Moreover, it is necessary to make sure that the exchanged information between the different peers is always from the correct party so that false information is not sent and received from unauthorized and malicious parties. Furthermore, it is also necessary to guarantee that the information sent from one peer to another is not modified when in transit.

The journalists and photographers communicate with the chief journalist by mobile devices. The chief journalist, manager, and editor communicate with each other and with the other members of the team by using their respective desktop computers. Table 1 shows a summary of the tasks executed by the different peers in the scenario.

Table 1: Tasks executed by the peers in the scenario

Manager becomes aware of a new event and contacts chief journalist about the event
Chief journalist selects team of people to cover the event and contacts them about the event
Chief journalist sends messages to journalists and photographer about the coverage of the event and any relevant information for them to execute the task
Journalists and photographer attend the event and prepare notes and take pictures about the event
Journalists and photographer send material about the event to chief journalist
Chief journalist receives the material, approves them, and sends the material to editor
Editor receives the material, combines the notes into a single report, and makes all necessary amendments and adjustments
Editor sends the report to chief journalist for approval
Chief journalist approves the report or suggest any further amendments to editor
After final approval from chief journalist, editor sends report to be printed

The described scenario includes some security properties such as authentication, confidentiality, integrity, and role based access control. A definition of these properties based on definitions given in the literature is presented below:

• *Authentication*: It is concerned with the property of guaranteeing the identity of a person or entity. An example of authentication in the scenario is "*the editor can only send the report for final approval to the chief journalist*". In this

example, the chief journalist needs to be authenticated before the editor sends the report.

• *Confidentiality*: Also known as *secrecy*, is concerned with the property of guaranteeing that data is not made available to unauthorized individuals. An example of confidentiality in the scenario is "*only the chief journalist can receive notes about the events from the journalists covering the event*".

• *Integrity*: It is concerned with the property of guaranteeing that data is not modified when in transit and that sent and received data are the same. An example of integrity in the scenario is given by the requirement that "*the information sent from one peer to another cannot be modified when in transit*".

• *Role based access control*: It is concerned with the access rights of an entity over resources in the system based on individual roles. An example of role based access control is the scenario is: "*the journalists prepare notes while the photographer takes pictures of the event*".

3. Static Verification Framework

Figure 1 presents an overview of the architecture of the static verification framework. As shown in the figure, the framework is composed of (a) Design Model Constructor (DMC), (b) Abuse Case Constructor (ACC), (c) Property Editor (PE), (d) Property Selector (PS), (e) Static Verification Tools (SVT), (f) Results Visualization Tool (RVT), and (g) Translators. We explain below each of the components of the framework.

3.1. Design model constructor

This component is responsible to support the design of structural and behavioral models of the system being developed together with its security aspects. In addition, if a specific security protocol is used, this protocol can also be modeled using this component. We propose the design models and security protocols to be specified in an extension of UMLSec [21]. As outlined in [11], the adoption and extension of standards like UML to include modeling of security features is quite attractive. Moreover, as affirmed by [14], an object-oriented approach to support security should be preferred over a procedural approach due to the value of information hiding and encapsulation in the design of secure systems. The rationale and advantages of using a UML-based approach to support the modeling of the system and security protocols specifications are many: (i) UML is the de facto standard to support design of software systems, (ii) UML has been largely used in industrial settings and there are a large number of developers familiar and trained in UML, (iii) there are many tools to support the use of UML, (iv) UML offers a relative precise definition as well as notions of modularity, compactness, and reuse, which comply to policy representation, and (v) UML allows for the possibility of unifying design of systems and security policies.

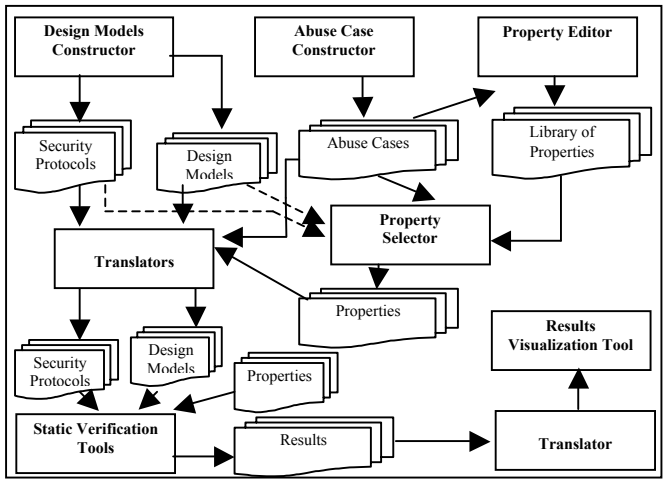

Figure 1. Architecture of static verification framework

The design models and security protocols in our work are based on an extension of UMLSec to allow modeling of security requirements such as confidentiality, integrity, authentication, and access control, and modeling of the functionality of peer-to-peer applications such as roles, resources, operation, and events.

UMLSec [21] provides stereotypes, tags, and constraints described in a profile that can be used to represent security requirements. In order to illustrate, consider part of the scenario described in Section 2 modeled in a class and sequence diagram with UMLSec extensions for journalist and chief journalist classes shown in Figure 2. In this example the classes are stereotyped as <<critical>>, denoting that these classes have sensitive data. The stereotype <<critical>> has associated tags {authenticity}, {secrecy}, and {integrity} to represent the security requirements on the respective data. As shown in the figure, the representation of these requirements does not guarantee that the system really conforms to them and need to be further verified (see Subsection 3.4). Due to space limitations we do not represent here the whole scenario in UMLSec and do not discuss all the different stereotypes tags, and constraints of the profile.

3.2 Abuse case constructor

This component allows for the specification of scenarios from the perspective of attackers or malicious users of the system. More specifically, abuse cases can be used to assist engineers of the system to consider ways in which the system can be attacked, identify potential threats to the system, and elicit ways in which the security aspects of the system can be invalidated. We propose to define abuse cases as UML use cases and state machine diagrams due to their popularity and use to describe system scenarios. Moreover, the use of state machine diagrams allow for a more detailed definition of the behavior of the attackers.

In our framework, the abuse cases can be used to support two activities: (a) identification or selection of properties and

(b) description of properties. In the first activity, the abuse cases are represented as use cases and support the identification or selection of properties to be verified by illustrating situations that the designers may not be necessarily aware. In the second activity, the abuse cases are represented as state machines describing properties to be verified which can be composed with the design models and verified for their termination. In the case of termination, the property (state machine) violates the design model.

As an example of using use cases as abuse case, consider the abuse case shown in Figure 3 in which journalist J1 exchanges notes about the event with journalist J2 (bolded arrows), invalidating one of the requirements in the system. This situation helps the designer identify the following property to be verified in the system: "*there should be no communication between journalists J1 and J2 during the coverage of the event*".

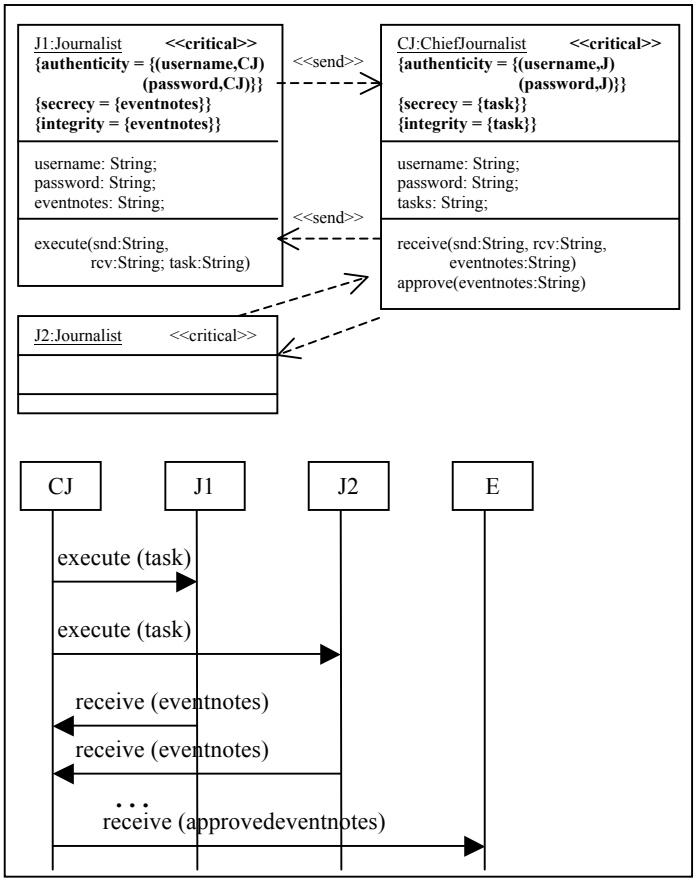

Figure 2. Example of part of the scenario in UMLSec

3.3. Property editor and property selector

We propose to use a graphical property editor to specify the various properties to be verified by the static verification tool by using a template language. The idea of using such an editor is to avoid having designers to express the properties directly in formal languages such as temporal logic formulas (CTL[9] and LTL[34]) or regular expressions, as required by the majority of verification tools. All the specified properties are stored in a Library of Properties.

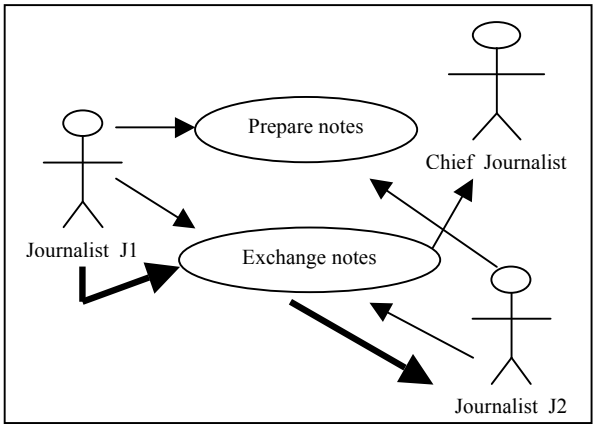

Figure 3. Example of an abuse case

The template language of the property editor is based on the work in [13] and specifies properties that will be translated into LTL formulae as required by the verification tool (see Subsection 3.4). The language allows for the specification of expressions separated by operators.

As an example, consider the secrecy tag represented in Figure 2 for *eventnote* data. More specifically, the tag, the operations and dependability in the classes represent the requirement that "*during the coverage of the event Chief Journalist should be the only member in the team that receives notes of the event from the Journalists*". Or in other words, "*no other peer different from Chief Journalist should receive notes of the event from Journalists*". The above requirement could be expressed in LTL as:

<> receive(J1, CJ, eventnotes) or
[] !receive (J1, !CJ, eventnodes)

Figure 4 shows part of the property editor with the specification of the above property. As shown in the figure, the template provides predefined values for some of the main elements in an expression from which the user can choose, or allows the user to input a different value.

It should be noted that in some cases it might not be possible to represent all properties to be verified in the template language. In this case, the designer should be able to specify the property directly in LTL and the property editor supports this functionality.

In order for the designer to choose the properties to be verified by the verification tool for a certain situation, the static verification framework contains a *Property Selector* component that allows the designer to browse and choose the relevant properties from a library of properties. The selection of properties is based on the design models, or the security protocols to be verified, and the abuse cases.

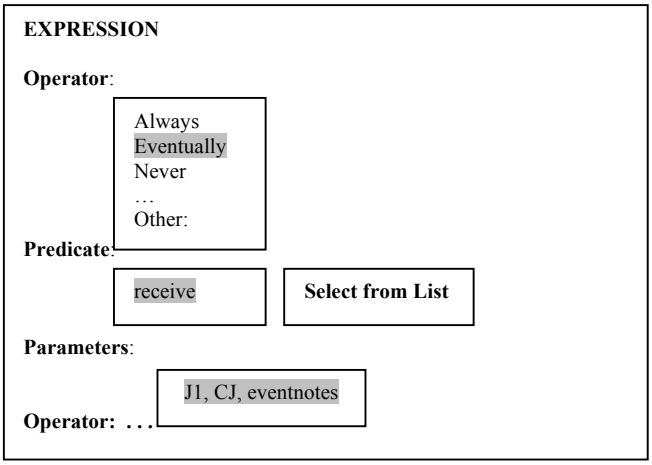

Figure 4. Example of the property editor

3.4. Static verification and visualization tools

In the framework, we propose to use SPIN [31] as the static verification tool to support the verification of design models of the system, and AVISPA [4] to support the verification of security protocols.

The decision to use model checkers as verification tools instead of theorem provers in the framework are due to the facts that: (a) theorem provers require human intervention during the verification process with knowledge in logic while model checkers execute the verification process automatically, (b) theorem provers generate a large number of lemmas that are not easy to be understood by designers while model checkers generate counter examples when a property is violated, and (c) the proofs in theorem provers are normally complex and intellectually challenging.

The choice to use SPIN and AVISPA has been based on our study of existing verification tools in which we analyzed eleven verification model checker tools based on different criteria namely: (a) handling of state space explosion problem, (b) representation of different types of data structures by the modeling languages, (c) representation and manipulation of built-in-intruder models, (d) support for dynamic creation of processes, (e) availability and extensibility of the tool, (f) applicability of the tool, and (g) usability of the tool. These criteria have been identified based on the requirements and scenarios elicited by the partners in the PEPERS project [28]. The results of the study have demonstrated that SPIN can satisfy the majority of the analysis criteria and that AVISPA has been successfully in verifying a large number of security protocols.

In order to present the results of the verification process, the framework contains a graphical *Vizualisation Tool* that shows the parts of the design models that may violate a certain property together with the property. In the cases where a property is not violated or cannot be processed by the tool (i.e. a conclusion cannot be drawn), the visualization tool also indicates these situations to the designer.

3.5. Translators

As shown in Figure 1, the static verification framework contains various translators to support the mapping between (a) design models represented in UMLSec into Promela [31] specification language, (b) security protocols represented in UMLSec into HLPLS AVISPA language [4], (c) properties expressed in the template language into the property specification language of the verification tools, (d) state machine diagrams representing the abuse cases that can be used as properties, and (e) results of the verification process into the visualization tool.

3.6. Discussion

The static verification framework addresses the challenges described in Section 1. In addition, it offers the following advantages: (a) modeling of peer-to-peer applications together with security aspects by using a de facto standard for modeling distributed systems; (b) verification of general and security requirements of the system; (c) verification of security protocols independent of their applicability in the system by using specific verification tool; (d) freedom of system developers to design their systems independent of the static verification process; (e) use of scenarios that consider ways in which the system can be attacked or identify potential threats to the system; (f) use of a graphical template language to specify properties to be verified avoiding system designers to be familiar with formal languages; (g) selection of properties to be verified based on the design models, security protocols, and abuse cases; (h) visualization of the results of the verification process in a user friendly way.

Our approach is different from the work in [21][22], since it supports the verification of both security properties and general peer-to-peer applications. In addition, although in [22] the authors propose the use of SPIN to support model checking, this has been restricted to the verification of cryptographic protocols. Moreover, the static verification framework includes the notion of (a) abuse cases and their constructors, (b) property editor, property library, and property selection based on abuse cases and design models, and (c) a visualization tool to allow the representation of the results of the static verification process with respect to the original design models and properties.

4. Related Work

Approaches to support secure software engineering can be categorized in three main groups based on different software development life-cycle phases: (a) security requirements engineering and analysis, (b) security modeling and development, and (c) secure software code analysis and testing.

Some approaches for security requirements engineering and analysis propose the use of misuse cases [2] and abuse cases [24] in which a scenario is described from the point of view of an attacker to the system. However, although they are

effective to analyze security threats, they are inappropriate for specifying security requirements. In our framework, we use abuse cases to assist with the identification of properties to be verified and propose ways of using abuse cases to represent properties to be verified.

Other approaches for assisting requirements elicitation, specification, and analysis are the common criteria [32] and attack trees [35]. The work in [10] defends the idea of using roles for defining goals and policies and introduces the notion of anti-requirements to represent the requirements of malicious users. In [33] the authors propose to use security goals and anti-goals in which anti-goals represent malicious obstacles set by attackers to threaten the security goals. As outlined in [27], in all these approaches security is considered vaguely and there is a lack of precise definition for security properties. In [36] the authors propose to use i* to support security requirements.

The approaches for security modeling and development make use of patterns [13][14], agent oriented methodologies [16], and extensions of UML [21][23]. Patterns can be used to document common solutions to recurring problems by describing security problems that occur in a context and presenting solutions accepted among security experts. However, the representation and selection of security patterns are still empirical tasks and it is not easy to identify the patterns to a specific situation. The work in [16] (Secure Tropos), combines two software engineering approaches for the development of a methodology that takes into account security and trust issues as part of the development process. In UMLsec [21] the authors proposed a special UML profile to represent security requirements. SecureUML [23] integrates the specification of access control policies into UML. The approach uses UML class diagrams, object diagrams, additional stereotypes, and OCL constraints. Our framework uses UMLSec to model the system and provides other advantages as discussed in Subsection 3.6.

Techniques for secure software code analysis and testing have been advocated in [6][7][30]. These approaches make use of static analysis tools to detect security flaw in source code [7], theorem prover [6], and testing firewalls [30]. However, they cannot solve all security problems and are normally used to look for a fixed set of patterns or rules in a source code.

Although many approaches have been proposed to support secure software engineering, to the best of our knowledge, these approaches have tackled individual aspects of secure software engineering. However, a framework that incorporates all the functionality of the static verification framework described in this paper has not yet been proposed.

5. Conclusion and Future Work

The work presented in this paper is part of a large program of research to support verification of secure peer-to-peer applications. We presented a framework for static verification of peer-to-peer applications that allows for various functionalities including (a) design of peer-to-peer systems that considers general and secure characteristics of the system based on UML, (b) use of abuse cases to support the identification and specification of properties to be verified, (c) specification of security and general application properties to be verified in a graphical template language, (d) verification of design models against properties, and (e) visualization of the verification results embedded in the design models of the system. The work presented in this paper contributes to the area of secure software engineering by integrating security into early stages of software engineering and not as an afterthought layer when the system is developed. Currently, we are implementing the main components of the framework and evaluating the work with the industrial partners in the PEPERS project.

ACKNOWLEDGMENT

The work reported in this paper has been funded by the European Commission under the Information Society Technologies Programme as part of the project PEPERS (contract IST-26901).

6. References

[1] Abadi M., Blanchet B., and Fournet C. Just Fast Keying in the Pi Calculus. *Proceedings of the 13th European Symposium on Programming (ESOP'04)*, volume 2986 of *Lecture Notes on Computer Science*, Spain, 2004.

[2] Alexander, I., Misuse cases: Use cases with hostile intent. IEEE Software, 20, 58-66, 2003.

[3] Anderson, R., Security Engineering: A guide to Building Dependable Distributed Systems. Wiley Computer Publishing.

[4] AVISPA. http://www.avispa-project.org.

[5] Chang, C. Symbolic Logic and Mechanical Theorem Proving. Academic Press, 1973.

[6] Chess B., Improving Computer Security Using Extended Static Checking, Proc. IEEE Symp. Security and Privacy, IEEE CS Press, 2002.

[7] Chess B. and McGraw G., Software Security, IEEE security and Privacy Magazine, 2(2):53-84, 2004.

[8] Clarke, E., and Grumberg, O., Model Checking. MIT Press, 1999.

[9] Clarke, E.M., Emerson, E.A, and Sistla, A.P. Automatic Verification of Finite-State Concurrent Systems Using temporal Logic Specifications. ACM Transactions on Programming Languages and Systems, 8(2): 244-263, 1986.

[10] Crook, R., Ince, D., and Nuseibeh, B., Modelling Access Policies Using Roles in Requirements Engineering. Information and Software Technology, 45(14), 979-991, 2003.

[11] Devambu, P. and Stubblebine, S., Software Engineering for Security: A Roadmap, Proceedings of the 22nd International

Conference on Software Engineering. Track on the Future of Software Engineering, Limerick, Ireland.

[12] Duffy, D. A. Principles of automated theorem proving. Wiley, 1991.

[13] Dwyer, M.B., Avrunim, G.S., and Corbett, J.C., Patterns in Property Specifications for Finite State Verification, in Proc. of the 21st International Conference on Software engineering (ICSE'99), May 1999.

[14] Fernandez, E.B., Larrondo-Petrie, M.M., Sorgente, T., and Vanhilst, M., A Methodology to Develop secure Systems Using Patterns, in Integrating Security and Software Engineering Advances and Future Visions, editors H. Mouratidis and P. Giorgini, ISBN 1-59904-147-2, 2007.

[15] Giorgini, P., Massaci, F., and Mylopoulos, J., Requirements Engineering Meets Security: A Case Study on Modelling Secure Electronic Transactions by VISA and Mastercard, Proceedings on the International Conference on Conceptual Modelling (ER), LNCS 2813, Springer-Verlag, 2003.

[16] Giorgini, P., Mouratidis, H., and Zannone, N., Modelling Security and Trust with Secure Tropos, in Integrating Security and Software Engineering Advances and Future Visions, editors H. Mouratidis and P. Giorgini, ISBN 1-59904-147-2, 2007.

[17] Gritzalis, S., Spinellis, S., Georgiadis P., Security Protocols Over Open Networks and Distributed Systems: Formal Methods for their Analysis, Design, and Verification. Computer Communications, 22(8): 695-707, 1999.

[18] Heitmeyer, C., Applying Practical Formal Methods to the Specification and Analysis of Security Porperties.

[19] Holzmann, G., Design and Validation of Computer Protocols. Prentice Hall, 1991.

[20] Jayaram K.R. and Mathur A.P., Software Engineering for Secure Software – State of the Art: A Survey. Technical report of CERIAS (Purdue University) and SERC, 2005.

[21] Jurjens J. Secure System Development with UML. Springer-Verlag 2005

[22] Jurjens, J. and Shabalin P. Automated Verification of UMLsec Models for Security Requirements, Proceedings of the Unified Modelling Language Conference (UML 2004), Lisbon, Portugal, . LNCS, Springer-Verlag. 2004.

[23] Lodderstedt, T., Basin, D., and Dorser, J., SecureUML: A UML-based Modelling Language for Model Driven Security, Proceedings on the UML'02 Conference, LNCS 2460, Springer-Verlag.

[24] McDermott, J. And Fox, C., Using Abuse Case Models for Security Requirements Analysis, In ACSAC'99: Proceedings of the 15th Annual Computer Security Applications Conferences, USA, 1999.

[25] Meadows, C., Formal Verification of Cryptographic Protocols: A Survey. International Conference on the Theory and Application of Cryptology (ASIACRYPT), 1995.

[26] Mouratidis, H., Giorgini, P., and Manson, G., When Security Meets Software Engineering: A Case of Modelling Secure Information Systems. Information Systems, 30(8), 609-629, 2005.

[27] Mouratidis H. and Giorgini P., Integrating Security and Software Engineering: An Introduction, in Integrating Security and Software Engineering Advances and Future Visions, editors H. Mouratidis and P. Giorgini, ISBN 1-59904-147-2, 2007.

[28] PEPERS. http://www.pepers.org/.

[29] Roscoe, A.W., Modelling and Verifying Key-Exchange Protocols Using CSP and FDR. The 8th IEEE Computer Security Foundations Workshop (CSFW '95), Ireland, 1995.

[30] Senn D., Basin D. and Caronni G., Firewall Conformance Testing, in TestCom, pp. 226-241, 2005.

[31] SPIN. http://spinroot.com/spin/whatispin.html

[32] Stoneburner, G., Hayden, C., and Feringa, A., Engineering principles for Information Technology Security (A Baseline for Achieving Security), Computer Security Division, Information Technology laboratory National Institute of Standards and Technology, 2001.

[33] Van Lamsweerde, A., and Letelier E., Handling Obstacles in Goal-Oriented Requirements Engineering, TSE, 26(10), 2000.

[34] Vardi, M.Y.. An automata-theoretic approach to linear temporal logic. In Banff Higher Order Workshop, pages 238-266, 1995.

[35] Viega, J., McGraw, G., Building Secure Software: How to Avoid Security Problems the Right Way, 1st edition, Addison-Wesley, 2001.

[36] Yu E., Liu L., Mylopoulos, J., A Social Ontology for Integrating Security and Software Engineering, in in Integrating Security and Software Engineering Advances and Future Visions, editors H. Mouratidis and P. Giorgini, ISBN 1-59904-147-2, 2007.