Verification and Trade-Off Analysis of Security Properties in UML System Models

Geri Georg, *Member*, *IEEE Computer Society*, Kyriakos Anastasakis, Behzad Bordbar, *Member*, *IEEE*, Siv Hilde Houmb, Indrakshi Ray, *Member*, *IEEE*, and Manachai Toahchoodee

Abstract—Designing secure systems is a nontrivial task. Incomplete or faulty designs can cause security mechanisms to be incorrectly incorporated in a system, allowing them to be bypassed and resulting in a security breach. We advocate the use of the Aspect-Oriented Risk-Driven Development (AORDD) methodology for developing secure systems. This methodology begins with designers defining system assets, identifying potential attacks against them, and evaluating system risks. When a risk is unacceptable, designers must mitigate the associated threat by incorporating security mechanisms methodically into the system design. Designers next formally evaluate the resulting design to ensure that the threat has been mitigated, while still allowing development to meet other project constraints. In this paper, we focus on the AORDD analysis, which consists of: 1) a formal security evaluation and 2) a trade-off analysis that enables system designers to position alternative security solutions against each other. The formal security evaluation uses the Alloy Analyzer to provide assurance that an incorporated security mechanism performs as expected and makes the system resilient to previously identified attacks. The trade-off analysis uses a Bayesian Belief Network topology to allow equally effective security mechanisms to be compared against system security requirements and other factors such as time-to-market and budget constraints.

Index Terms—Aspect-oriented modeling (AOM), Bayesian belief network (BBN), security analysis, trade-off analysis.

1 Introduction

Developing secure systems is a nontrivial task. We have identified four issues where consistent, comprehensive approaches can help designers create a system that meets its project goals. First, designers of secure systems need effective methodologies to systematically apply standards in the context of a particular development project. Security standards such as the ISO Common Criteria [1], [2] and risk management standards such as the Australian/New Zealand Risk Management standards [3], [4] exist to aid secure systems development; however, these standards generally address system security in the broad sense, often require extensive resources to use, and do not provide specific methods to help designers verify that particular resources are protected from specific kinds of attacks. They do not take into account project goals other than security, such as time-to-market, cost, effort, and compliance with governmental laws and regulations.

Second, designers must correctly incorporate security mechanisms into a system design, and provide assurance that the resulting system design is indeed secure. Often, breaches in security-critical systems pose unacceptable risks since they can cause irreparable damage to sensitive data or to a company's image. To mitigate such risks, designers incorporate security mechanisms into the system. Security mechanisms, designed to protect against certain attacks, are typically analyzed in isolation. However, the efficacy of an approach often lies in the manner in which it has been integrated with the application, so designers need to be able to analyze security mechanisms in the context of the whole system.

Third, system designers need techniques that allow them to compare alternative security mechanisms and decide which best meets the needs of the given application and other project development goals and constraints. Multiple mechanisms are often effective in protecting against a particular kind of attack. For instance, to protect against unauthorized access, a system design may incorporate access control lists or capability-based mechanisms. The optimal mechanism to use for a given application depends on how well the mechanism protects the given application, and how well it meets other project goals such as those mentioned above.

Finally, security mechanisms incorporated to protect against different attacks may actually interfere with each other, reducing their efficacy. For example, auditing may detect and deter malicious activities in the system, but may cause improper disclosure of sensitive data if the logs can be accessed by unauthorized entities. Designers need techniques that provide formal evaluation of these interactions, prior to system implementation and deployment.

We propose an Aspect-Oriented Risk-Driven Development Methodology (AORDD) [5], [6], [7] for designing secure applications. AORDD combines security development techniques, aspect-oriented techniques, and analysis techniques to address these four issues. Our previous papers provide high-level discussions of the techniques we use in AORDD, and proof-of-concept examples for its security and

• *G. Georg, I. Ray, and M. Toahchoodee are with the Computer Science Department, Colorado State University, 1100 Center Avenue Mall, Fort Collins, CO 80523-1873. E-mail: {georg, iray, toahchoo}@cs.colostate.edu.*
• *K. Anastasakis and B. Bordbar are with the School of Computer Science, University of Birmingham, Edgbaston, Birmingham, UK. E-mail: {k.anastasakis, b.bordbar}@cs.bham.ac.uk.*
• *S.H. Houmb is with the Services Platform Group, Telenor GBDR, Otto Nielsens vei 12, 7004 Trondheim, Norway. E-mail: siv-hilde.houmb@telenor.com.*

Manuscript received 9 Oct. 2008; revised 10 July 2009; accepted 18 Nov. 2009; published online 1 Mar. 2010.
Recommended for acceptance by K. Goseva and K. Kanoun.
For information on obtaining reprints of this article, please send e-mail to: tse@computer.org, and reference IEEECS Log Number TSESI-2008-10-0319.
Digital Object Identifier no. 10.1109/TSE.2010.36.

trade-off analysis elements. By contrast, in this paper, we discuss the analysis portion of the methodology in detail: systematic formal evaluation and trade-off analysis of alternative security solutions in complex systems design.

We recognize that in industrial systems development, a perfect system is almost never achievable in the time frame required and within the constraining budget, so we include techniques that take into account these and other project constraints. We also acknowledge that in an industrial setting, the size of a development team (and a system design team) may be quite large, and team members are often not experts in security issues or in formal mathematical-based evaluation techniques. We have, therefore, taken a pragmatic approach to secure systems development, utilizing existing techniques and developing a methodology that supports a more formal approach by nonexperts while realizing that the result may not be as precise and verifiable as an organization might ultimately desire. AORDD steps are, by consequence, quite flexible so that as new tools and techniques become generally available they can be used to add precision and verification to the methodology.

This paper makes two major contributions. The first is to demonstrate how designers can provide assurance, at the time of system design, that a security mechanism is effective in protecting against a given security breach. The second is to provide details of the trade-off analysis designers use to compare alternative mechanisms during systems design. We additionally discuss current and possible points of automation in the methodology analysis, and provide heuristics for activities that require human intervention.

The rest of the paper is organized as follows: Section 2 provides an introduction to our AORDD methodology and explains the analysis steps that are our focus. Additional background information regarding techniques we use for analysis is included in this section. Section 3 describes the e-commerce system that we use to illustrate our methodology, an attack example on that system, and two potential mitigating security mechanisms. Section 4 discusses formal security analysis using Alloy, a declarative first-order logic language with supporting tools designed for modeling and analyzing complex systems. We demonstrate an Alloy analysis on the example system and discuss its results. Section 5 presents our trade-off analysis that uses Bayesian Belief Network (BBN) technology, and the results of applying it on our example system. We also discuss the effect of changing trade-off priorities in the context of the example. Section 6 discusses related work, and Section 7 concludes the paper with our future research.

2 Background

2.1 AORDD Methodology

AORDD Methodology is targeted toward the development of complex systems where there are competing project and security goals. Under these conditions, it can be difficult for a designer to determine how different parts of the system, designed to meet different goals, interact with each other. Performing security analysis in the context of the whole system can help a designer understand these interactions better. Performing trade-off analysis can help a designer make informed choices when faced with multiple designs that mitigate security threats equally well. AORDD trade-off analysis allows designers to analyze various security design solutions against properties such as required security levels, and project constraints such as time-to-market, budget, laws, and regulations *at the same time*, in a single trade-off analysis.

We have been able to partially or fully automate portions of AORDD, for example, security analysis is automated by using the Alloy Analyzer and trade-off analysis by using our BBN topology. We have developed the methodology with an eye toward automating especially repetitive or tedious tasks while recognizing that very complex systems do require an investment of human expertise to develop. Our goal is to make onerous and error-prone tasks automated—leaving time for humans to apply their expertise where it is most needed.

The AORDD Methodology has two steps that must occur prior to any analysis. First, system architects and designers must create system functional models. Since the Unified Modeling Language (UML) is the de facto software specification language used in the industry, our tool chain requires that these models be specified using the UML 2.0 [8]. Second, designers must perform a *risk assessment* of the system. Risk assessment begins with system stakeholders (e.g., end users, designers, developers, and management) identifying sensitive system assets such as system information or services. Different stakeholders can place different values on an asset, so the stakeholder and the value they assign to a particular asset are both needed in our methodology. Designers must develop security requirements for these assets and identify possible threats against them, with the aid of security standards such as ISO 14508: Common Criteria [1] and ISO/IEC 13335-5: Guidelines for Management of IT Security [9]. Threats are attacks on the system with the goal of compromising assets. Designers and security experts must also rank the potential threats. Designers also identify potential security mechanisms that can mitigate specific risks, as part of the assessment process. Designers can be aided in a risk assessment by organizational experience or security experts. Note that security expertise may be in the form of international forum security postings.

In general, risk assessment is a very human-intensive task; however, some techniques such as CORAS [10], [11] are computer-assisted. Designers identify attacks associated with unacceptable risk as targets for analysis. The specifications of the attacks are referred to as *attack models*. Designers can model attacks separately or as a group, depending on their desired approach. This activity should not include every possible attack, but instead, it should focus effort on the attacks that designers, system architects, and security experts identify as being critical to the system.

AORDD supports an incremental approach to the design of secure systems, so many steps are iterative, allowing designers to create and analyze a more complete system design each time. For example, system designers often add security mechanisms into a system to mitigate a particular security threat only to discover that the addition has opened the way for a different threat, which may have already been considered. By incrementally adding security mechanisms as different threats are considered, and reanalyzing the design in light of previously considered attack models, designers can discover and address conflicts between multiple security solutions or between security goals and other project goals, prior to system implementation. Designers decide the amount of iteration needed based on project goals, for example, the desired security level or the time constraints of the project, and based on the number of risks that are identified.

We next present brief introductions to the major technologies we use in our analysis. These are Aspect-Oriented Modeling (AOM), Alloy, and BBN.

2.2 Aspects and Aspect Composition

We use aspect-oriented techniques to support analysis since attacks are not usually confined to one functional module of an application, but rather impact multiple modules. Similarly, security mechanism designs that prevent attacks affect multiple modules. Aspect-oriented methodologies were developed to allow a modular approach to development and reasoning of such crosscutting features. We use AOM techniques to specify attack models and security mechanism models. Aspects make it easier for designers to understand, manage, and change these models separately. Since models are developed separately, a library of reusable attack and security mechanism models is feasible. These models are all specified using the UML 2.0 [8]. We use tools to support integration, or composition, with system functional models. Although composition is largely automated, human designers must specify where and how aspects should be incorporated into a system design.

The aspects we describe in this paper are similar to those of other AOM or Aspect-Oriented Programming (AOP) approaches in that they represent a view of interest, e.g., security, and they are crosscutting. In AOM, crosscutting means that an aspect model must be integrated in several places across the main system functional modularity. In AOP, crosscutting means that aspect code must be inserted into multiple components of the implementation. In both AOM and AOP, it is necessary to define *what* an aspect will do, and *where* this action should be taken. Many AOP and AOM techniques use the term *advice* for the action an aspect will take and *join points* for where the action will be inserted in the main system functionality. *Point cuts* are used to specify more general rules of where to apply an aspect. Often, advice, joint points, and point cuts are specified as one entity, called an aspect.

We take a more general approach to AOM by defining a *generic* aspect that specifies not only *what* the aspect should do, but also *where* it can fit into a system design (by specifying parameters in the generic aspect template). A human designer must specify the generic aspect and its parameters. We create *context-specific* aspect models from generic aspect models by binding parameters in a generic aspect template to model elements in the main functionality design (which we term a *primary* model). Humans must specify the parameters they want bound between the generic aspect and the system design. Automated tools create defaults for parameters that are not specified, and instantiate the generic aspect template to create a context-specific aspect. This approach means that generic aspect models can be reused across multiple applications, whereas in the advice/join point/point cut view of aspects, only an

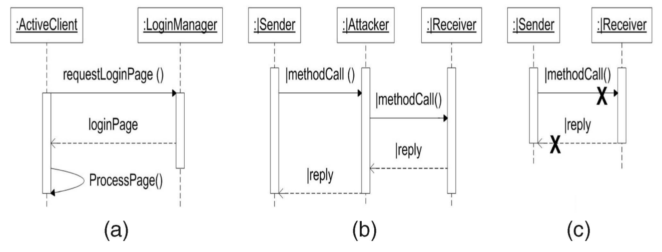


Fig. 1. Examples of portions of (a) a primary model, and (b) and (c) a generic MiM attack aspect. Parameterized model elements in (b) and (c) have names beginning with "|", while model elements indicated with "X" will be deleted if they exist in the primary model.

advice can be reused. Point cuts and join points are specific to the application in which the aspect will be inserted.

We compose context-specific aspect models and main functionality designs to create models in which the aspect has been integrated. In order for composition to produce a meaningful model, the models being composed must be specified at similar levels of abstraction. However, we do not require any particular level of abstraction in our techniques and tools. Therefore, designers can compose and analyze a set of models at different levels of abstraction to produce different kinds of information, depending on the amount of detail available at a particular point in the design cycle. Fig. 1 shows portions of a primary model and a generic aspect in the form of sequence diagrams.

The portion of a primary model in Fig. 1a shows two classes: *ActiveClient* and *LoginManager*. A message is sent from *ActiveClient* to execute the *requestLoginPage* method in *LoginManager*. The result of this operation returns a *loginPage* message to *ActiveClient*. *ActiveClient* then executes an internal method *ProcessPage*. A portion of a generic man-in-the-middle (MiM) attack aspect model is shown in Figs. 1b and 1c. There are three classes: $|Sender$, $|Attacker$, and $|Receiver$. The " | " symbol at the beginning of any name in the generic aspect model serves as an indicator that this element is a parameter that can be bound to elements in the primary model that are of the same UML type, prior to model composition. The generic aspect Fig. 1b shows a message to execute a method called $|methodCall$ to be sent to $|Attacker$, and from $|Attacker$ to $|Receiver$. There is a response, $|reply$, that is sent back. Fig. 1c shows behavior that is not allowed (indicated by an X mark), that is, some message or reply going directly between $|Sender$ and $|Receiver$. Our composition techniques allow us to specify such elements that will be removed prior to composition if they exist.

We specify bindings of generic aspect parameters to primary model elements of the same type, then instantiate the aspect to create a context-specific aspect model, which we compose with the primary model. For example, using the models in Fig. 1, we can specify that $|Sender$ should be bound to *ActiveClient*, $|Receiver$ should be bound to *LoginManager*, $|methodCall$ should be bound to *requestLoginPage*, and $|reply$ to *loginPage*. There is no corresponding primary model element to $|Attacker$, so our tools automatically create a binding from $|Attacker$ to *Attacker*.

Figs. 2a and 2b show the context-specific MiM aspect model fragments with element names as specified in the bindings discussed above. Fig. 2c shows the result of

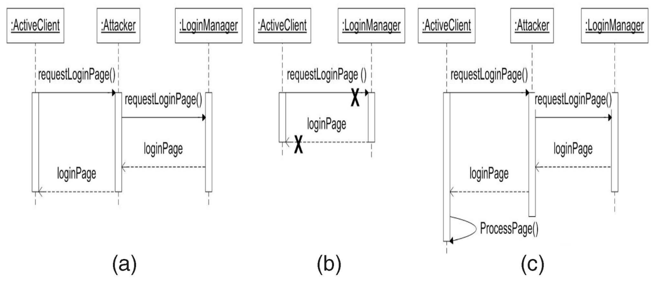


Fig. 2. (a) and (b) Portion of context-specific MiM aspect model and (c) composed model fragment. The generic aspect from Figs. 1b and 1c is instantiated using user-specified bindings for generic parameter names to create Figs. 2a and 2b. A default composition algorithm is used to compose these with primary model from Fig. 1a to create Fig. 2c.

composing the context-specific model with the primary model, in this case producing a sequence diagram. Composition includes removing the direct messages between *ActiveClient* and *LoginManager*, and adding the messages from *ActiveClient* to *Attacker* and from *Attacker* to *LoginManager*. Composition also adds the final execution message of the primary model, *ProcessPage*, to the composed model. Our general composition algorithm is to identify elements in a context-specific aspect model that have the same name as elements in the primary model, and that are of the same type and add these to the composed model if they do not have the deletion mark X, e.g., Fig. 2b. Model elements in either the context-specific or primary model that do not have corresponding elements in the other model are also added to the composition. This composition approach allows addition of, for example, the *Attacker* object lifeline and the *ProcessPage* message. Users can specify additional composition directives if this default behavior is not desired. Default behavior and directives are discussed in France et al. [12], [13], Georg et al. [14], [15], and Straw et al. [16].

We use AOM to specify generic attack models and generic security mitigation mechanism models. We compose an attack model with a system primary model to create what we term a *misuse* model, that is, a model of the system whereby some asset may be compromised through application of a successful attack. Composing a security mechanism with a primary model yields a *security-treated system* model. In a similar fashion, composing an attack model with a security-treated system model yields a *security-treated misuse* model. We perform security analysis on misuse models and trade-off analysis on security-treated system models along with the results of security analysis. Misuse models typically consist of both static class diagrams and dynamic behavior diagrams. In this paper, we demonstrate dynamic behavior specified as sequence diagrams, but these diagrams are not a requirement of the techniques we use. We find that sequence diagrams are especially convenient when dealing with behavior such as security protocols, and since our examples use such protocols we have chosen to utilize sequences in our modeling and analyses.

2.3 Alloy

We approach analysis in AORDD in two steps. First, we perform a formal security analysis to give assurance that the system, created by integrating a security mechanism model, is indeed resilient to the targeted attack. We transform a UML misuse model into Alloy and use the Alloy Analyzer [17], [18] to reason about its security properties. The results of the analysis either give assurance that the security properties exist, or show that they do not.

Alloy is a fully declarative first-order logic language designed for modeling and analyzing complex systems. An Alloy model consists of a number of *signature* and *relation* declarations. A signature specifies entities used to model the system, and relation declarations specify the dependencies between such entities, allowing the designer to specify complex structures. Alloy is supported by a fully automated constraint solver, called the Alloy Analyzer, which analyzes system properties by performing an exhaustive search for model instances that violate properties of the system. Users must express the properties to be checked as Alloy logical statements that are called *assertions*. The Alloy Analyzer translates a model into a Boolean expression and analyzes it using embedded SAT-solvers. The user needs to specify a *scope* to the tool. A scope is an integer number used to create a maximum bound of the domain of model elements to make the analysis tractable. Bounding enables the tool to create finite Boolean formulas for evaluation by the SAT-solver.

Considerations regarding the technique. There are several considerations that users need to take into account when using Alloy. In particular, if the Alloy Analyzer produces an instance that violates an assertion (a counterexample), we can infer that the specified property is not satisfied. However, for a chosen scope, if no counterexample emerges, it is possible that the property is violated in a larger scope. The underlying idea of the Alloy Analyzer is to deploy automated analysis to inspire confidence in the correctness of a design. The larger the scope, the more confidence is warranted, but the longer the analysis will take. Experience has shown that design flaws are often discovered in small scopes [18].

Choosing the right scope, and the degree of confidence a given scope provides, depends on the problem and the security property being analyzed. Currently, there are no generic guidelines on how to choose the scope for a given problem. However, when developing security-critical systems, where a higher degree of confidence is required, a designer can use the Alloy Analyzer as a first line of defense to discover flaws in the design of a system. We approach the issue of scope by starting with a small scope, and then, while the Alloy Analyzer does not provide a counterexample, increasing the scope until the time required for the analysis is not tractable or becomes burdensome. Table 2 in Section 4.1.3 shows analysis times for one of the security mechanism analyses included in this paper: No counterexample was found up to a scope of 20. By contrast, the analysis of the other security mechanism discussed in this paper produced a counterexample using a scope of six.

If the analyzer does not produce a counterexample, other techniques such as Model Checking and Theorem Proving can also be used to ensure that the security property is not violated. Such techniques are more time-consuming and require human intervention and expertise. Our approach can therefore save time and resources by using the Alloy Analyzer to rapidly discover a number of flaws that would otherwise require much more time and resources to

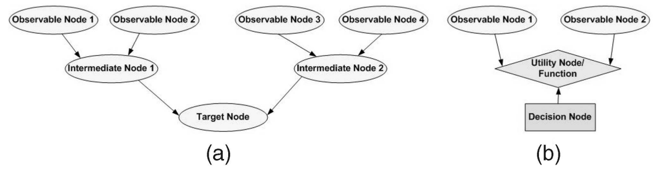


Fig. 3. Hierarchy and node types in BBN: (a) belief network with observable nodes and a target computation node and (b) decision network with a target decision node supported by a utility function that holds rules/decision preferences.

uncover. For more details on Alloy and its comparison with other formal methods, refer to Jackson [18].

Sometimes a user may incorrectly assume that a security property is satisfied when no counterexample is produced by the analyzer, when, in fact, the model is overconstrained or the scope is not large enough to produce an instance of the model. Therefore, before checking any assertion, it is imperative that the user ask the Alloy Analyzer to generate an instance of the model. If the analyzer produces an instance, then the user can check for counterexamples to the assertions.

The Alloy Analyzer allows us to check for the possibility of a security property violation in the system model. This is achieved by checking if a specific attack, such as the man-in-the-middle attack, can occur. UML models for each attack scenario are created and integrated into the security-treated system. The models are then transformed into Alloy via UML2Alloy and the security property is checked by the Alloy Analyzer. Therefore, our approach does not allow general, a priori security evaluation of a system design. However, our method enables the user to work with UML models and benefit from the capabilities provided by Alloy.

2.4 Bayesian Belief Networks (BBNs)

The second step in AORDD analysis is to compute a BBN trade-off analysis network. BBN is a powerful technique for reasoning under uncertainty, using disparate information [22], [23]. Input to the BBN consists of the evidence from the security analysis, risk information from other AORDD steps, and trade-off parameters. The trade-off parameters consist of project-specific goals, including required security level, time-to-market, budget, laws and regulations, and business goals. The trade-off analysis computes a fitness score, showing how well the proposed security mechanism meets the project goals. However, project-specific goals are rarely static over the course of system development, so our BBN topology allows designers to easily change parameters and priorities in real time as they explore candidate security mechanisms.

A BBN is a connected and directed acyclic graph that consists of a set of nodes (also called variables) and directed arcs (also called links). Nodes correspond to events or concepts and are structured into either: 1) a belief network (Fig. 3a) or 2) a decision network (Fig. 3b). A belief network consists of a hierarchy of stochastic nodes (shown as ovals in Fig. 3). Stochastic nodes can be observable, in which case they model information that can be directly observed or obtained. They can also be intermediate, in which case they represent information connecting observations with a target node. A target node holds the result of a BBN computation and represents the outcome of the network. Decision networks include at least one decision node (shown as a rectangle in Fig. 3b), in addition to any number of stochastic nodes. A decision node is supported by a utility node/function (represented by a diamond shape in Fig. 3b), which holds the rules or decision preferences of the decision variable. A decision variable is the target node of a decision network.

Nodes denote the variables and the directed arcs represent dependencies between the variables, i.e., a BBN expresses the conditional probability relations between variables. Possible outcomes of a node are specified using a set of states, and multiple nodes may be used to determine the state of a node. States of a node are expressed using probability density functions (pdfs), which express the confidence that a node will be in a particular state. The value of a pdf at a node depends on the states of the nodes connected to it. The pdf for a node is referred to as the Node Probability Table (NPT). The status of a topology is continually updated as different types of information are entered and propagated backward and forward along the edges of the network. Several networks can be connected into a topology consisting of one top-level network and a number of connected subnetworks. In this case, the target node of each subnet holds its outcome. We use this type of structure to increase the flexibility and evolution of our trade-off analysis.

To compute a topology, a user inserts information into observable nodes, which is propagated to the target node via any intermediate nodes. Propagation uses causal relationships defined by the topology and the NPT or utility function associated with each node. There are two types of evidence that can be entered at the observable nodes in a BBN. These are hard and soft evidences. Hard evidence comes from actual observations that a particular node is in a particular state. Soft evidence is information that cannot be verified or observed, such as expert opinions or experience from similar systems. Soft evidence is expressed using probability distributions. BBN is based on the Bayesian interpretation of probability (see Fenitti [19]). For more information on BBNs, the reader is referred to Jensen [20] and Pearl [21]. For more information regarding the highly relevant work on application of BBNs in software safety assessment, the reader is referred to Gran [22] and the SERENE project [23].

Considerations regarding the technique. There are three considerations that a designer should take into account when using BBN technology: the nature of BBN computations, the relations of hard and soft evidence to those computations, and the affect of network topology on the computations.

BBN is a technique used to reason in the face of uncertainty. The computation of a BBN topology provides the probability of a variable state, but not verification of its truth. The more confidence we have in the data input to the BBN, the more confidence we can have in the resulting computation, but the computation still represents a probability. Approaches such as trust-based information aggregation schemas can help users assess trustworthiness of information input into a topology [5].

Hard data can be obtained from relevant prior observations (e.g., from honeypot installations where a clone system is deployed and actual attacks are measured), while both hard and soft data can be obtained from expert

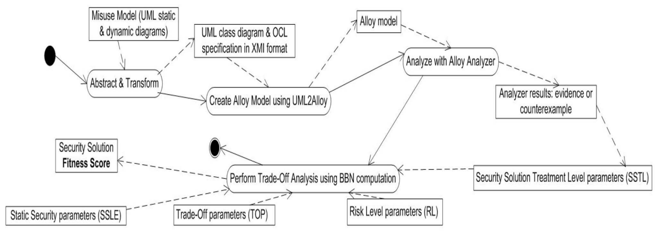

Misuse Model (UML static & dynamic diagrams)
UML class diagram & OCL specification in XMI format
Alloy model
Abstract & Transform
Create Alloy Model using UML2Alloy
Analyze with Alloy Analyzer
Analyzer results: evidence or counterexample
Security Solution Fitness Score
Perform Trade-Off Analysis using BBN computation
Security Solution Treatment Level parameters (SSTL)
Static Security parameters (SSLE)
Trade-Off parameters (TOP)
Risk Level parameters (RL)

Fig. 4. UML activity diagram showing steps in AORDD security (first three activities) and trade-off (fourth activity) analyses. Activities are represented as ovals and objects that they use or create as rectangles. Solid arrows indicate control flow and dashed arrows indicate object information flow.

opinions, experience with similar systems, and national and international forums on security. In general, all kinds of information can be used as input to a BBN, since the technique is designed for handling disparate information.

There is no rule as to how much hard or soft evidence is required for the trade-off analysis BBN topology to aid in security decisions. Security decisions are very subjective by nature and performed by humans interpreting the information given to them. Our goal is to promote the use of sources of hard evidence to enable more informed decisions that are less biased on the trust between the decision maker and the information source. We support this goal with our trade-off analysis. The trade-off analysis is a best effort approach that represents a step forward from the current subjective and often highly biased security decision processes.

BBN is very flexible, and a topology can be quite sensitive to some of its variables, so care must be taken when using the technique. The topology we present in this paper is stable and has been used for a number of trade-off cases. However, it is possible for a designer to change the network to tailor it to a new situation, including changing the specification of affected NPTs to capture the effects that the changes should have on target nodes. In this case, the designer must also perform a sensitivity analysis to ensure that the updated topology and associated NPTs achieve the desired affects. Section 5.6 includes a brief discussion of topology sensitivity, and ways to address it.

2.5 AORDD Analysis Process Steps

Fig. 4 shows a UML activity diagram that describes the steps in an iteration of AORDD analysis. The solid circle and outlined solid circle shown in Fig. 4 represent the initial and final states (respectively) of an AORDD analysis. Ovals are activities (four in this diagram), and rectangles are objects produced or consumed during the activity. Solid arrows show control flow, while dashed arrows show flow of objects among activities. The dashed arrow into the *Security Solution Treatment Level parameters* object indicates that information from *Analyzer results* is needed by it. The first three activities produce an evaluation of the security provided by a security solution to protect against a successful attack. The fourth activity results in a fitness score for the security solution with respect to security and other trade-off parameters. The abbreviations shown in parenthesis are used in the trade-off analysis discussion in Section 5.

Based on the results of the security evaluation, a designer might decide to iterate the security analysis steps with a different security solution prior to performing any trade-off analysis. Similarly, based on the fitness score, a designer might decide to iterate the trade-off analysis, changing the priority of trade-off parameters or relaxing some of them. In practice, security acceptance criteria are often relaxed in the face of budget and/or time-to-market constraints. Relaxing constraints can have a great effect on fitness score. We discuss flexibility in trade-off analysis in Section 5.5.

2.5.1 Security Analysis

We use the UML2Alloy tool to transform a UML model into Alloy. Its input consists of a UML class diagram in XML Metadata Interchange (XMI) format [24], and an accompanying OCL [25] specification of behavior. We therefore begin with the *Abstract & Transform* activity as the first activity in AORDD analysis. This activity takes as input a UML misuse model that a user creates by composing an attack model with either a system model or a security-treated system model, as discussed in Section 2.2. A designer must abstract the misuse model to only include elements associated with testing the security properties of interest. We have created a set of heuristics to aid in this process, which we describe in Section 4. Details can be found in our previous work [15]. We use a UML CASE tool, ArgoUML [26], to create the UML class diagram and OCL specification. ArgoUML, like most UML tools, allows us to export the model in XMI format.

The next activity, *Create Alloy Model using UML2Alloy*, applies UML2Alloy to the XMI representation. UML2Alloy implements transformation rules to create an Alloy model [27], [28]. This model is input to the next activity *Analyze with Alloy Analyzer*. The Alloy Analyzer searches the state space exhaustively on all possible valid instances of the model, up to the user-specified scope, for a counterexample. The output from the analyzer must be interpreted by a human, and be input into the BBN topology via computer assistance. If a counterexample is produced, the input to the BBN should reflect that the security mechanism does not preserve security. Otherwise, the input represents the analysis assurance that the security mechanism included in the misuse model provides protection against the attack.

Considerations regarding the technique. There are a number of OCL constraints that cannot be directly expressed in Alloy and are thus not supported by UML2Alloy (for example, the OCL "iterate" construct). Another issue is that OCL lacks inherent support to capture temporal properties. As a result, different methods have been proposed to extend OCL with the ability to express temporal constraints [29]. It is, however, possible to depict simple but crucial constraints related to time if a designer models time explicitly and uses conventional OCL to express constraints. Details on exactly which OCL statements are supported by UML2Alloy are presented in our previous work [27].

2.5.2 Trade-Off Analysis

Our trade-off analysis BBN topology consists of multiple subnetworks that relate to a security solution and security analysis. This is because a simple security analysis output is not sufficient for a designer to determine whether a security solution is adequate. Analysis either proves a particular

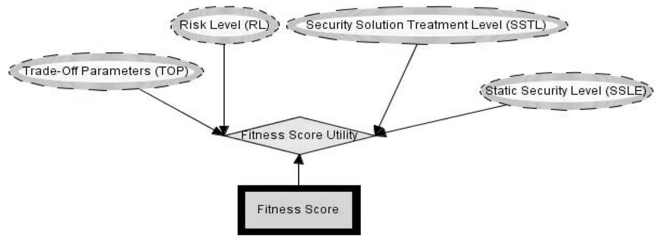

Risk Level (RL)
Security Solution Treatment Level (SSTL)
Trade-Off Parameters (TOP)
Static Security Level (SSLE)
Fitness Score Utility
Fitness Score

Fig. 5. Top-level topology of the trade-off analysis BBN. The target of the computation is the decision variable Fitness Score (rectangle with thick border). Four subnets (ovals with dotted and thick outlines) affect the computation.

successful attack path (misuse) to be executable, or provides evidence that it is not executable. However, the existence of an attack path does not imply that the attack will actually happen. It means that there exists a possibility of an attack. A successful attack depends on other factors, such as the likelihood or frequency of the attack and the mean time and effort needed to launch a successful attack. These latter characteristics in turn depend on the skills, motivation, and resources of the attacker [2]. Our trade-off analysis takes these characteristics into consideration, along with the impact of a successful attack on the value of system assets. We also include the project-specific consequence of incorporating a security mechanism to prevent the attack, in the form of variables related to the development effort in terms of cost and time.

Our trade-off topology consists of four subnetworks, which are discussed in detail in Section 5. The subnets are shown as object inputs to *Perform Trade-Off Analysis using BBN Computation* activity in Fig. 4. The information categories represented by the trade-off subnets are the static security level variables (SSLE), risk level variables (RL), the security solution treatment level variables (SSTL), and the trade-off parameters (TOPs).

The SSLE variables represent information regarding the criticality of the system assets, along with stakeholder asset value information, that system designers obtain from the risk assessment process. The RL variables represent information regarding identified security risks. Designers obtain part of this information during risk assessment and part through security analysis of an initial system misuse model. Recall that risk assessment is a required step of the AORDD methodology that must be performed prior to any analysis.

SSTL variables represent information relevant for measuring the abilities of a security solution to prevent the attack, along with development and maintenance costs. Again, designers obtain part of this information through security analysis, and part from the risk assessment process. The TOP variables consist of relevant project goals and their relative priorities. This information comes from various project stakeholders and decision makers. The trade-off parameters are used to compute a fitness score that reflects the ability of the security solution to meet the set of trade-off goals.

We use the Hugin tool [30] to specify and compute the trade-off topology. We present BBN diagrams and computations in this paper using output from this tool. Fig. 5 shows a Hugin representation of the top-level portion of the

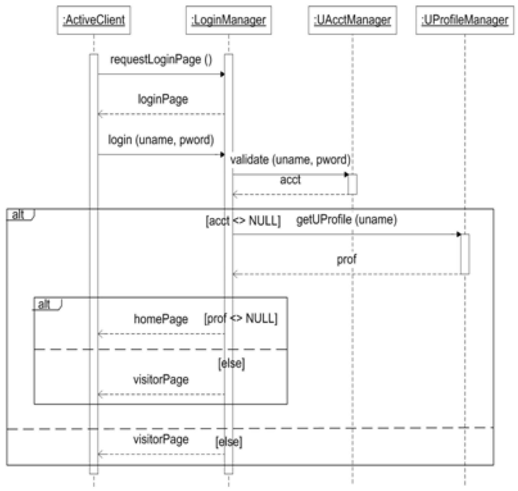


Fig. 6. Original ACTIVE login sequence. A user runs *ActiveClient* on a local system to request a login page from a remote ACTIVE system (*LoginManager*) and to provide credentials. A tailored *homePage* is returned if the credentials are accepted and the user has an ACTIVE profile; otherwise, a *visitorPage* is returned.

topology. Each oval section of the topology is a subnetwork. The topology computes a decision variable *Fitness Score* (rectangle with thick border) for a security solution using four subnets (ovals with dotted and thick outlines). Subnet values and a decision variable utility (diamond) are used to compute the score.

A user must populate the BBN topology parameters, with computer assistance, then computing the BBN topology is fully automated. However, the results must be interpreted by a human designer.

3 Example E-Commerce System

We next present the example we will use to illustrate AORDD analysis. This section also presents a possible attack against the example system and two potential security solutions to prevent a successful attack.

Our example is an e-commerce platform called ACTIVE. ACTIVE provides services for electronic purchasing of goods over the Internet [31]. The IST EU-project CORAS [10], [11] performed three risk assessments of ACTIVE in the period 2000-2003. The project identified several security risks, including attacks against user authentication in the login service.

Fig. 6 shows the primary model sequence diagram associated with logging into the system. There are four design classes involved, one running on a user's local machine (*ActiveClient*) and three running on a machine over the Internet. The *ActiveClient* on the user's local machine communicates with an ACTIVE system login manager (*LoginManager*), which uses two other classes to: 1) authenticate the user (*UAcctManager*) and 2) provide user-specific information (*UProfileManager*). A user runs *ActiveClient* to request a login page from *LoginManager*. When the *loginPage* is returned, the user enters a user name (*uname*) and password (*pword*) that are transmitted to *LoginManager*. *UAcctManager* authenticates the user, and *LoginManager* obtains user-specific information, a profile, from *UProfileManager*. It uses

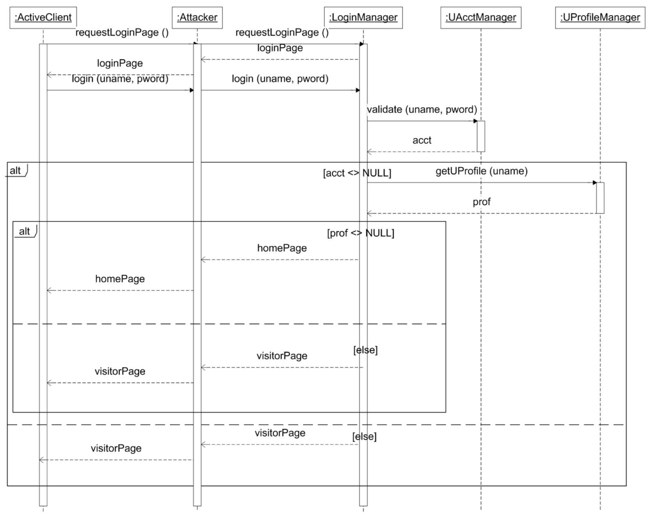


Fig. 7. Misuse model of original ACTIVE login sequence and MiM attack, created by composing primary login sequence model with context-specific MiM passive attack models. All communication between *ActiveClient* and *LoginManager* goes through *Attacker*. The attack is successful if *Attacker* obtains *homePage*, or *uname* and *pword*.

this information to build a *homePage* that is returned to the user. If the profile does not exist or the user cannot be authenticated, a *visitorPage* is returned that does not contain any user-specific information. Alternative actions are shown in Fig. 6 using an *alt* box; the dotted line across the box separates the actions, and guards such as *[acct <> NULL]* specify the conditions under which each alternative action should be taken.

3.1 The Man-in-the-Middle Attack

The risk assessment performed by the IST EU-project CORAS demonstrated that the ACTIVE login service is vulnerable to an MiM attack, which allows an attacker to intercept information that may be confidential. This type of attack has two variations: passive and active. During a passive attack, the attacker eavesdrops on the message flow between a requestor and authenticator. By contrast, an attacker participates in the communication during an active attack: changing, deleting, or inserting messages between the requestor and authenticator. The generic MiM model in Figs. 1b and 1c specifies a passive attack; messages pass through the attacker, but are not changed prior to forwarding.

In order to understand the impact, a man-in-the-middle attack has on the e-commerce login service, we need to generate a misuse model and analyze it using the Alloy Analyzer. The results will indicate if the primary model can be compromised by the attack, in which case the analyzer will produce a counterexample.

The misuse model of the ACTIVE login service and a passive MiM attack is presented in Fig. 7. We create the misuse model by instantiating a generic man-in-the-middle aspect model, as shown in Figs. 1b and 1c, in the context of the ACTIVE login service, and compose the aspect with the login sequence. A partial explanation of this process is as follows: We must instantiate the generic aspect twice, to handle the two messages passing back and forth from *ActiveClient* to *LoginManager*. For example, in the first instantiation, $|methodCall()$ is bound to *requestLoginPage()* and $|reply$ is bound to *loginPage*. $|Sender$ is bound to *ActiveClient* and $|Receiver$ is bound to *LoginManager*. We use these bindings for the generic aspects, as shown in Figs. 1b and 1c. We use our default composition algorithm to compose the context-specific version of the generic aspect from Fig. 1c. This composition removes the direct messages passing between *ActiveClient* and *LoginManager* from the composed model. When we compose the context-specific aspect derived from Fig. 1b, the default algorithm adds the *Attacker* lifeline and the messages from *ActiveClient* to *Attacker*, and then, to *LoginManager*, and back. The second message is handled in a similar fashion, with two bindings of the reply message, once for the *homePage* and once for the *visitorPage*. The remaining portions of the primary model are left intact in the composed model since they are present in the primary model.

The misuse model in Fig. 7 differs from the original login service sequence diagram of Fig. 6 in that all communication between *ActiveClient* and *LoginManager* goes through *Attacker*. In order for *Attacker* to obtain registered user information, it can either eavesdrop until this information passes by, in the form of a *homePage*, or until *uname* and *pword* pass by, which will allow *Attacker* to impersonate the registered user at a later time. Either of these situations indicates a successful attack.

The misuse model shown in Fig. 7 has been transformed into an Alloy model and analyzed using the Alloy Analyzer, as described in Section 4. The result of this analysis is a counterexample, which indicates that the attack can succeed. These results are used in the Risk Level (RL) subnet of the BBN trade-off analysis that is discussed in Section 5.2. We next discuss two security mechanisms that can be used to protect a system against an MiM attack.

3.2 Security Mechanisms to Counter Man-in-the-Middle Attacks

System designers must identify security properties relevant to mitigating a risk to system assets. We identify properties according to the ISO/IEC TR 13335:2001 Information Technology—Guidelines for Management of IT Security [9]. This standard defines seven security properties: accountability, authenticity, availability, confidentiality, integrity, nonrepudiation, and reliability. The security properties relevant to protecting the ACTIVE login service assets from an MiM attack are authenticity, confidentiality, and integrity.

We have incorporated and analyzed two mechanisms that provide these properties. The first mechanism is Secure Remote Password (SRP) [32], [33] and the second is Secure Sockets Layer (SSL) [34]. SSL is an authentication mechanism often used in Web applications and is part of commonly available Web clients. It operates just above a reliable transport layer (e.g., TCP). SRP is an alternative mechanism that is not generally available at lower levels of communication and must be added at the application level. Both mechanisms provide user authentication, data confidentiality, and data integrity. SSL is often used to authenticate a server to a client and can also be used to authenticate a client to a server, while SRP always authenticates both parties to each other. Confidentiality is provided through symmetric key encryption in both mechanisms. SSL

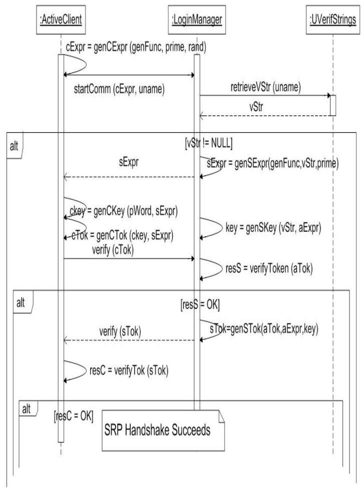

:ActiveClient
:LoginManager
:UVerifStrings
cExpr = genCExpr (genFunc, prime, rand)
startComm (cExpr, uname)
retrieveVStr (uname)
vStr
alt
[vStr != NULL]
sExpr
sExpr = genSExpr(genFunc,vStr,prime)
ckey = genCKey (pWord, sExpr)
cTok = genCTok (ckey, sExpr)
verify (cTok)
key = genSKey (vStr, aExpr)
resS = verifyToken (aTok)
alt
[resS = OK]
verify (sTok)
sTok=genSTok(aTok,aExpr,key)
resC = verifyTok (sTok)
alt
[resC = OK]
SRP Handshake Succeeds

Fig. 8. Portion of SRP security-treated system model, showing composition of SRP authentication and key generation with primary model of Fig. 6. Original authentication using *uname* and *pword* is removed, and SRP is included.

provides additional integrity through the use of hashed message digests, while SRP relies on encryption to provide integrity.

3.3 Incorporating SRP Authentication in the Application

SRP is based on generating symmetric encryption/decryption keys in a client and server without passing secret information between them. Instead, tokens are passed back and forth, and each side verifies the other's token, to provide both client and server authentication.

SRP requires that the client and server have each received portions of a shared secret from a trusted source, usually based on a password. The client has the password, and the server has a verifier string created from the password. Both items must be kept secret. The client and server must also agree upon a generator function that is used to create a key, and a large prime number. SRP begins when a client creates an expression using the generator function, a random number, and the large prime number, and sends these to the server, along with its identity, as part of an initiation message (i.e., *startComm()* in Fig. 8). The server retrieves the verifier string associated with the client identity. If no verifier string is found, the server aborts the communication. Otherwise, the server creates an expression from the verifier string, the generator function, and the large prime number, and sends it back to the client. The client generates a key from this expression and the password, while the server generates a key from the expression it received from the client and the verifier string. The client creates a token using its key and the server expression and sends it to the server for verification. If the server cannot verify the token, it aborts the communication. Otherwise, the server next creates a token from its key, the client token, and the client expression, and sends it to the client for verification. If the client verifies this token, authentication has taken place between both parties and the client can use its key to encrypt a message to be sent to the server. The server uses its key to decrypt the message and act on it.

We have defined a generic SRP aspect that consists of the following three subaspects. Part A specifies authentication and key generation between a client and server. Part B specifies deletion of any previously existing authentication messages between the client and server and any helper classes the server might have used to authenticate the server (along with associated messages). Part C specifies message transfer between a sender and receiver including message encryption prior to being sent and decryption upon receipt.

We create an SRP security-treated login service model in several steps. Here, we discuss the steps needed to incorporate SRP authentication and key generation into the primary model. First, we create a context-specific subaspect of Part A by binding the aspect client to *ActiveClient* and the aspect server to *LoginManager*. We also bind the client identity to uname. Second, we create a context-specific model of Part B, specifying that the *requestLoginPage()* and *loginPage* messages are previously existing authentication messages, *UAcctManager* is a helper class for authentication, and *validate()* and *acct* are also messages associated with this previous authentication. We compose these context-specific aspects with the primary model from Fig. 6 to incorporate SRP authentication and key generation. Fig. 8 shows a portion of the resulting model. Note that: 1) Message and helper classes from the previous authentication are removed and 2) SRP authentication and key generation occur with *ActiveClient* serving as a client and *LoginManager* serving as a server. A misuse model must then be created by composing this security-treated model with a generic MiM aspect model, as we discuss next.

3.4 Creating the Misuse Model of the Security-Treated Primary Model

Once security mechanisms have been incorporated into the login service primary model, we need to verify whether the security properties of authenticity, confidentiality, and integrity are preserved in the presence of the man-in-the-middle attack. We therefore create a security-treated misuse model to analyze these properties using the Alloy Analyzer. The man-in-the-middle attack we compose with the security-treated system is an active version of the attack, where the Attacker can change and insert messages flowing between the client and server. Instantiating this generic attack aspect is more complex than instantiating a generic passive attack because some security domain knowledge must be used to decide what messages need to be changed, inserted, or deleted to promote a successful attack. For user authentication, personal information, such as name, a personal expression, a personal certificate, or the like, can often be substituted to fool a server or client. Since communication between a client and server is encrypted as part of the SRP mechanism, a successful attack occurs if the attacker can obtain a decryption key.

We show a portion of the SRP security-treated misuse model in Fig. 9. It is similar to the misuse model of Fig. 7 in that all communication between *ActiveClient* and *LoginManager* pass through *Attacker*. However, the active attack differs in three ways: 1) *Attacker* substitutes its own expression and name in the *startComm* message (*aExpr* and *aname*); 2) *Attacker* generates its own key and token (*key* and *aTok*); and 3) *Attacker* substitutes its token for that of the

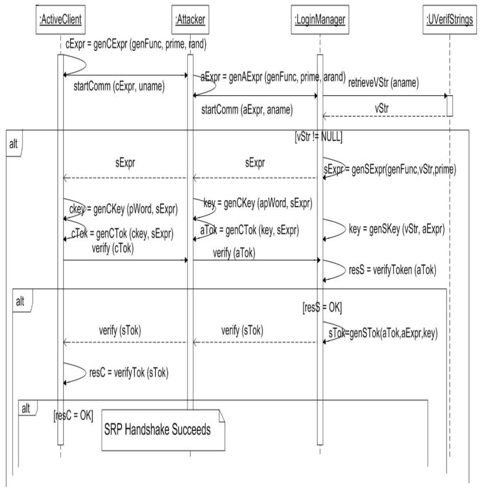

:ActiveClient
:Attacker
:LoginManager
:UVerifStrings
cExpr = genCExpr (genFunc, prime, rand)
startComm (cExpr, uname)
aExpr = genAExpr (genFunc, prime, arand)
retrieveVStr (aname)
startComm (aExpr, aname)
vStr
alt
[vStr != NULL]
sExpr
sExpr
sExpr = genSExpr(genFunc,vStr,prime)
ckey = genCKey (pWord, sExpr)
key = genCKey (apWord, sExpr)
cTok = genCTok (ckey, sExpr)
aTok = genCTok (key, sExpr)
key = genSKey (vStr, aExpr)
verify (cTok)
verify (aTok)
resS = verifyToken (aTok)
alt
[resS = OK]
verify (sTok)
verify (sTok)
sTok=genSTok(aTok,aExpr,key)
resC = verifyTok (sTok)
alt
[resC = OK]
SRP Handshake Succeeds

Fig. 9. Portion of SRP security-treated misuse model including active MiM attack. If successful, *Attacker* substitutes its expression (*aExpr*), identifier (*aname*), and token (*aTok*) in the SRP authentication, allowing it to generate the same key as *LoginManager*. Subsequent messages can then be decrypted by *Attacker*.

ActiveClient in the *verify* message (*aTok*). All other messages are simply passed through *Attacker* to the intended recipient.

The result of substitutions by *Attacker*, as part of a successful attack, allows *Attacker* to generate the same key as *LoginManager*. *Attacker* is thus able to decrypt messages from *LoginManager* meant for *ActiveClient*, such as the homepage.

3.5 Creating a Misuse Model for the SSL Mechanism

SSL provides authentication through certificate checking, and confidentiality and integrity using secret keys generated by the client and server. Although authentication is normally only used to authenticate a server, client authentication can also be requested. A certificate check may be minimal, in which case only a few items in the certificate are checked (e.g., whether it has been signed by a trusted authority), or it may be more thorough and include checking the certificate IP address against the desired server address, and checking that the certificate is not on a revocation list. In the discussions that follow, we assume minimal certificate checks.

The misuse model used to analyze SSL is created by first composing the primary login sequence model with a context-specific SSL aspect model to create a security-treated model. Then, the security-treated model is composed with an active man-in-the-middle attack. The SSL security-treated system and misuse models may be found in our related technical report [35].

4 Formally Verifying Security Properties in the Misuse Model

We use the Alloy Analyzer [17], [18] to help reason about the SRP and SSL security-treated misuse models and the ability of these mechanisms to protect the ACTIVE login service from an MiM attack. We first discuss the SRP security-treated misuse model. We abstract the UML misuse model to a format suitable for the UML2Alloy tool, which transforms it into an Alloy model. We then run the Alloy Analyzer on the model to search for counterexamples to assertions about the security properties we wish to verify. In order to abstract the security-treated misuse model properly, we identify the security properties in detail and ensure that relevant elements remain in the abstracted model. Recall that the threat posed by a man-in-the-middle attack is to obtain access to system assets. In the original ACTIVE login sequence, this threat is realized when the attacker obtains a user name and password or a homepage (*uname*, *pword*, and *homePage*, respectively).

The system assets are a bit different in the SRP security-treated misuse model shown in Fig. 9 since the *uname*, *pword* pair no longer exists. The *homePage* asset still exists, but an attacker can no longer use eavesdropping to gain access to it since it is encrypted using the key generated during SRP authentication. The attacker therefore needs to obtain enough information to generate the key used by *LoginManager* to encrypt the homepage.

We define two security properties for our example as follows: 1) If the SRP authentication succeeds and a *homePage* is sent to *ActiveClient*, *Attacker* does not have the secret key used by *LoginManager* to encrypt the page, and 2) if *Attacker* does gain access to this key, the SRP authentication must fail and no *homePage* or *visitorPage* can be sent to *ActiveClient*. In the following sections, we explain how the Alloy Analyzer can be used to verify that these properties hold.

4.1 Analyzing the Misuse Model for Security Properties

Recall from Fig. 4 that the first step in formal security analysis is to abstract and transform a misuse sequence model into a UML class diagram and accompanying OCL behavioral specification. The class diagram depicts the entities that take part in the sequence diagram, and defines their methods and messages. OCL statements specify method behavior in the form of pre and postconditions, and the order of execution represented in the original sequence diagram. We use UML2Alloy [28] to automatically transform the class diagram and OCL statements into an Alloy model, which we subsequently analyze using the Alloy Analyzer.

4.1.1 Stage 1: Abstracting a Sequence Diagram

We first simplify the original misuse model (e.g., Fig. 9) by removing elements so that UML2Alloy produces a model that only contains items necessary to reason about its security properties, e.g., 1) and 2) above. We recognize that identifying which security properties are relevant to a system is a design task that falls in the purview of security engineering. Such design tasks require deep insight into the system and cannot be automated. However, it is possible to support security engineering tasks as we briefly discuss in Section 7.

We discuss abstraction to test security property 1), which depends on a successful SRP authentication. The protocol is successful if the sequence reaches the true portion of the $[resC = OK]$ guard (in the third nested *alt* box of Fig. 9). We need to test whether *Attacker* has the same key as *LoginManager*.

We have developed a set of heuristics to help designers abstract a misuse sequence diagram [15]; however, abstraction is essentially a user-driven process. We proceed as

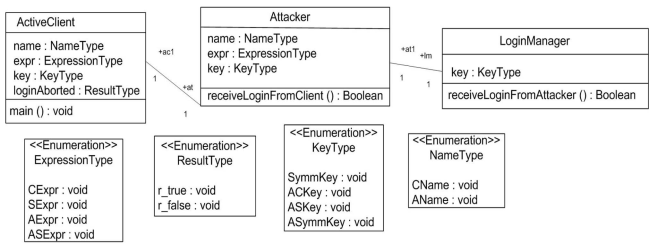


Fig. 10. Portion of misuse model class diagram used for UML2Alloy tool translation. A user creates the diagram by abstracting the SRP security-treated misuse model shown in Fig. 9 to retain messages and variables needed to test security properties. Class method names correspond to the retained messages.

follows, referring to Fig. 9: First, we analyze the tests included in each *alt* box, the variables used in these tests, and the dependencies of each variable—that is, the variables and messages that affect their value at the time of the test. For example, the outermost *alt* box in Fig. 9 contains the test $[vStr\,!{=}NULL]$, which utilizes the variable *vStr*. This variable is returned as a result of the *retrieveVstr(aname)* message from *LoginManager* to *UVerifString*. Thus, *vStr* has a dependency on *aname*. The value for *aname* was obtained from the message *startComm(aExpr, aname)* sent from *Attacker* to *LoginManager*. We derive a similar dependency set of variables and messages for the second *alt* box (with the test $[resS = OK]$), and finally, the third *alt* box (with the test $[resC = OK]$). The final set of dependencies gives us the model elements needed to decide whether the authentication is successful. We perform a similar analysis to determine the messages and variables that go into generating the key, then remove all model elements that are not in these dependencies.

The next part of the abstraction consists of constructing the class diagram and accompanying OCL that will be used as input to UML2Alloy. We first specify classes for the lifelines still left in the abstracted sequence diagram. We create methods for these classes that are named for the messages received by the object lifeline in the sequence diagram. For example, looking back at Fig. 9, we will create a class named *LoginManager*. In the examples below, we have created a method called *receiveStartCommFromAttacker*, named for the *startComm* message sent to *LoginManager* from *Attacker*. We must also add a method named *main*, essentially to start the sequence. This method is placed in the *ActiveClient* class, and its purpose is to invoke the *receiveStartCommFromClient* method in *Attacker*. We add variables that are used in methods as attributes to the class diagram, and we add relations between classes that send and receive messages.

Fig. 10 depicts a portion of the class diagram we created after abstracting the sequence diagram in Fig. 9. It shows classes corresponding to lifelines in the sequence diagram, i.e., *ActiveClient*. Fig. 10 also shows attributes, their corresponding types, and methods in the classes. Attribute types denote enumerations we define since Alloy does not have built-in data types such as integers. Each enumeration type defines a set of literals, which represent the distinct values the type can have. We also abstract internal method calls shown in Fig. 9 to class attributes. Thus, for example, we abstract $cExpr = genCExpr(genFunc, prime, rand)$ to

```
context ActiveClient::main ()
post main:
ActiveClient.allInstances() ->
  forAll (ac:ActiveClient | ac.name =
       NameType::CName and
 ac.expr = ExpressionType::CExpr and
 ac.at.receiveStartCommFromClient ())
```

Fig. 11. OCL specification of the *main()* method of the *ActiveClient* class. The specification sets class attributes *name* and *expr*, and invokes the *receiveStartCommFromClient* method in the *Attacker* class.

the value of the *expr* attribute of the *ActiveClient*. To show that when the parameters are changed, a different value of *expr* is obtained, we use an enumerated type (*ExpressionType* in Fig. 10) that defines different literals.

We must next create textual OCL specifications of the methods in the class diagram that specify system invariants and behavior. We specify these methods using OCL pre and postconditions. Preconditions are OCL statements that must be satisfied before the invocation of a method. Postconditions are true after the execution of the method. For the purpose of analysis, the specification needs to have an entry point, which in this case is the *main()* operation of the *ActiveClient* class.

We name the relationship ends between classes for navigation purposes in the OCL pre and postconditions. For example, to navigate from *ActiveClient* to *Attacker*, we use the *at* role on the relationship between these classes. Thus, the OCL fragment to specify the invocation of the *receiveStartCommFromClient* method in *Attacker*, from *ActiveClient*, uses *at.receiveStartCommFromClient()*. Fig. 11 shows the OCL definition of the *main()* method of the *ActiveClient* class. The OCL statement specifies that the values of the name and expr attributes of every *ActiveClient* in the model are the Enumeration literals *cName* and *cExpr*, respectively. It also specifies that the *receiveStartCommFromClient()* method of the *Attacker* should be invoked.

The *receiveStartCommFromClient()* method of the *Attacker* corresponds to the first message passed by the *ActiveClient* to the *Attacker* in the abstracted version of the sequence, as shown in Fig. 9. Its OCL specification is shown in Fig. 12. The *Attacker* replaces the *name* and *expr* it receives with its own and invokes the *receiveStartCommFromAttacker()* method.

The specifications of the rest of the methods defined in the abstracted class diagram are similar: The methods set the values of local attributes depending on the messages received and then invoke a method that simulates the passing of messages between the lifelines of the sequence diagram.

```
context Attacker::receiveStartCommFromClient ():
Boolean
post receiveStartCommFromClient :
self.name = NameType::AName and
self.expr = ExpressionType::AExpr and
self.lm.receiveStartCommFromAttacker ()
```

Fig. 12. OCL specification of the *receiveStartCommFromClient()* operation of the *Attacker* class. The specification sets *Attacker* attributes *name* and *expr*, and invokes the *receiveStartCommFromAttacker* method in the *LoginManager* class.

TABLE 1
Overview of Important Transformation Rules in UML2Alloy

UML Model Element	Corresponding Alloy Model Element
Package	Module
Class	Signature
Property	Field
Association	Multiplicity Constraints
Operation	Predicate or Function

4.1.2 Stage 2: Using UML2Alloy to Generate the Alloy System Model

As depicted in Fig. 4, the UML2Alloy tool is used to create an Alloy model from the class diagram and associated OCL specification. The OCL specification defines system behavior, and users must also develop OCL expressions for the security properties they want to check, as discussed below. The user must create an XMI format of the class diagram and OCL specification, using a UML design tool.

UML2Alloy reads the UML model data from the XMI file and generates an equivalent Alloy model. UML2Alloy then uses the Alloy Analyzer API to carry out fully automated analysis on the generated Alloy model. (Note that in this paper, we used UML2Alloy to automatically generate the Alloy model from the UML model and then used the Alloy Analyzer as a stand-alone tool to carry out the security analysis.) If the Alloy Analyzer finds that the security property under investigation is violated, it provides a counterexample, an instance of the model that violates the security property. If a counterexample is found, it is displayed in UML2Alloy as a UML object diagram. If the analysis performed by the Alloy Analyzer does not find a counterexample, the user is informed that the security property holds for the user-specified scope.

Table 1 provides the most important transformation rules from UML class diagram model elements to Alloy model elements. In particular, a UML package maps to the notion of an Alloy module. A UML class maps to an Alloy signature. In UML, a class denotes a set of object identifiers (object ids). In Alloy, a signature denotes a set of atoms. An object id in UML uniquely identifies an instance of a class, in the same way that an atom in Alloy identifies an instance of a signature. A property in UML denotes either an attribute or an association end of a class and is translated to an Alloy field of a signature. In UML, an association connects two (or more) association ends. Multiplicity annotations on association ends specify how many instances of the classes participating in an association can exist. UML2Alloy maps these multiplicity markings of the association ends to Alloy multiplicity constraints. Finally, if a UML operation is of type void, UML2Alloy maps it to an Alloy predicate, otherwise to an Alloy function.

UML2Alloy uses similar transformation rules for OCL to Alloy. Both OCL and Alloy are based on first-order logic. They are therefore quite similar, and the mapping from OCL to Alloy is straightforward when dealing with first-order logic statements. For example, the *forAll* OCL construct is translated to *all* in Alloy, and the *exists* OCL construct to *some* in Alloy. For an extended study of the similarities and differences of OCL and Alloy, refer to Vaziri and Jackson [36].

```
pred main (){all ac :  ActiveClient | ac.name =
cName && ac.expr = cExpr
&& receiveStartCommFromClient[ac.at]}
```

Fig. 13. Alloy code produced by UML2Alloy from the translation of the OCL statement of *main()*. The *name* and *expr* variables are set and the *receiveStartCommFromClient* operation is invoked in the attacker.

The Alloy language has notions such as the scope and assertion commands which allow the Alloy Analyzer to perform fully automated analysis of Alloy models. Since our work aims to make UML class diagrams fully analyzable using Alloy, we need to extend the UML class diagram notation to introduce concepts such as the scope and assertion commands. We achieve this extension with the help of a UML profile for Alloy, which defines a number of stereotypes that a designer can use in UML class diagrams. We explain the transformation rules from UML to Alloy, the UML profile for Alloy, and the implementation of UML2Alloy in detail in our previous work [27].

When we apply UML2Alloy to the OCL specification in Fig. 11, it produces the Alloy code, as shown in Fig. 13. The UML operation is translated to an Alloy predicate, denoted by the *pred* keyword. The *forAll* OCL statement is translated to *all* in Alloy, and navigation expressions remain unchanged (i.e., *ac.name*). The *and* OCL statement is transformed to the Alloy conjunction symbol (`&&`).

It is important to note that OCL pre and postconditions are automatically transformed into an Alloy model evaluated over a single state. In particular, in this example, we are interested in the information exchanged between the *ActiveClient*, the *Attacker*, and the *LoginManager*. This approach to transformation is also followed by Torlak et al. [37] when analyzing the Needham and Schroeder protocol [38].

4.1.3 Stage 3: Using the Alloy Analyzer

Assertions are statements that capture properties we wish to verify. The Alloy Analyzer automatically checks such assertions and if they fail, it produces a counterexample. We have specified assertions corresponding to the security properties discussed at the beginning of Section 4. The first property is that the *Attacker* cannot generate the same key that is created independently by the *LoginManager* and the *ActiveClient* if the protocol is successful, which we call *SymmKey* in Fig. 10. We create an assertion that if the protocol does not abort, the *Attacker* key attribute does not have the value *SymmKey*. Fig. 14 depicts the OCL specification of this property. It indicates that for every *ActiveClient* in the system, if the login has not been aborted, the value of the key of the *Attacker* is not *symmKey*. It also states that the

```
context ActiveClient
ActiveClient.allInstances() -> fo-
rAll(ac:ActiveClient |
ac.loginAborted = ResultType::r_false implies
 (ac.at.key <> KeyType::SymmKey and
  ac.at.lm.key = KeyType::SymmKey and
   ac.key = KeyType::SymmKey))
```

Fig. 14. OCL assertion that *Attacker* has not generated the same key as *ActiveClient* and *LoginManager* if the SRP protocol is successful.

```
assert ok{all ac:ActiveClient |
      ac.loginAborted = r_false =>
{ac.at.key != symmKey &&  ac.at.lm.key = symmKey
      && ac.key = symmKey } }
```

Fig. 15. Alloy translation of OCL assertion, as shown in Fig. 14. If the protocol does not abort, the *Attacker* has not generated the same key as that generated by *ActiveClient* and *LoginManager*.

values of the keys of the *LoginManager* and the *ActiveClient* are both *symmKey*. This last statement is used to check that if the protocol does not abort, both the *ActiveClient* and the *LoginManager* have generated the symmetric key and can thus encrypt and decrypt messages. UML2Alloy automatically transforms the OCL statement shown in Fig. 14 to the Alloy assertion shown in Fig. 15.

We chose a value of 20 for the scope of analysis, and used the Alloy Analyzer to check the assertion in Fig. 14. A scope of 20 means that the Alloy Analyzer will attempt to find an instance that violates the assertion, using up to 20 instances for each of the entities defined in the class diagram of Fig. 10 (e.g., *ActiveClient*, *Attacker*, and *LoginManager*). The assertion produced no counterexample, meaning that it is valid for the given scope. Table 2 shows how much time the Alloy Analyzer required to provide results on a 2 GHz Core 2 Duo processor.

TABLE 2
SRP Model Analysis Time for a Given Scope

Scope	Time Required
10	2 seconds
14	5 seconds
20	27 seconds

4.2 Analysis of the SSL-Treated Login Sequence

We have created a complete SSL-treated system misuse model, similar to the SRP-treated misuse model (see our related technical report [35]). We abstracted this misuse model into a class diagram and OCL specification. The model specifies an active man-in-the-middle attack on a system that uses minimal certificate checking. We used the Alloy Analyzer to check the Alloy version of this model against the assertion of Fig. 14, and it produced a counterexample, which means that the assertion was violated. The counterexample shows an instance where *Attacker* gains access to the symmetric key and is therefore able to decrypt the messages exchanged between the *LoginManager* and the *ActiveClient*. In this case, the attacker substitutes its own certificate for that of the server, and since the certificate check is minimal, *ActiveClient* does not recognize the fact that a substitution has occurred. In another experiment, we generated a model where *Attacker* only uses passive attacks (i.e., does not change any of the messages). The Alloy Analyzer did not produce a counterexample for the assertion using this model. We use these results in trade-off analysis, as presented in the next section.

5 AORDD Trade-Off Analysis

We now discuss each of the four subnetworks shown in Fig. 5, in the context of our example ACTIVE system. We briefly discuss two subnets: SSLE and RL subnets. (See our technical report [35] for details on these subnets, how they are configured, and an example utility function definition.) We discuss the SSTL subnet and TOP subnet in detail, along with results of the top-level network computations of fitness for SRP and SSL. We use a qualitative scale for all of the variables in the SSLE, RL, and SSTL subnets, *{low, medium, high}*. The TOP subnet and the top-level fitness network use these and additional qualitative scales that are described in the network discussion. We finish this section with observations regarding network topology sensitivity.

5.1 Static Security Level Subnet (SSLE)

The SSLE subnet represents stakeholder assessment of the value of system assets. Asset value is an integral part of the trade-off analysis, because successful attacks lower asset value, while security mechanisms are intended to mitigate risk and thereby increase asset value (hopefully at least back to its original value). There are always multiple stakeholders viewpoints regarding system asset value, so the SSLE subnet topology includes variables that apply relative weight to a stakeholder's assessment. The stakeholders' assessment of asset value and the stakeholder weight are the observable nodes in the subnet. A decision node that represents the computation and its accompanying utility node determine the influence of each stakeholder on the outcome of the subnet. For our example, the subnet computation leads to an SSLE value of *high*.

5.2 Risk Level Subnet (RL)

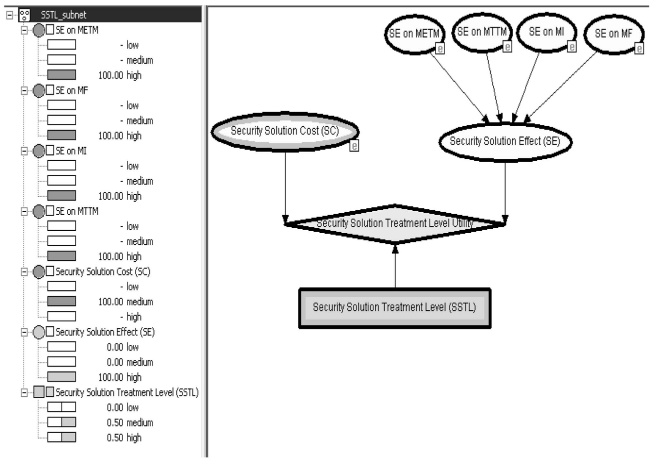
We use results from the initial system design security analysis as input to the variables in the RL subnet. All nodes in the subnet are stochastic and are:

1. the average effort an attacker must use to launch a successful attack, also known as a misuse of the system (METM),
2. the mean time it takes for an attacker to launch a misuse (MTTM),
3. how often a misuse will occur (MF), and
4. the impact of a misuse (MI).

We can specify effort in many ways, for example, in terms of skills, experience, and resources required. Similarly, we can specify frequency in different ways. We can use numeric (e.g., *1 hour*) or subjective (e.g., *often*) scales to describe attack characteristics. The only requirement is that the same scales be used for the same variables when performing trade-off analysis on alternative security mechanisms. We derive the value for the risk variables MTTM, METM, and MF directly from the result of the Alloy security analysis based on an initial misuse model (e.g., Fig. 9). The security analysis produced a counterexample for our example of the passive man-in-the-middle attack, which is a simple attack, and one that requires little time or effort on the part of the attacker. The variable values we use based on these results lead to an RL subnet computation distribution of $RL.low = 0.1$, $RL.medium = 0.7$, and $RL.high = 0.2$.

5.3 Security Solution Treatment Level (SSTL)

The SSTL subnet contains variables relating to an alternative security solution and how well it protects target assets. The



```
SSTL_subnet
  SE on METM
           - low
           - medium
      100.00 high
  SE on MF
           - low
           - medium
      100.00 high
  SE on MI
           - low
           - medium
      100.00 high
  SE on MTTM
           - low
           - medium
      100.00 high
  Security Solution Cost (SC)
           - low
      100.00 medium
           - high
  Security Solution Effect (SE)
        0.00 low
        0.00 medium
      100.00 high
  Security Solution Treatment Level (SSTL)
        0.00 low
        0.50 medium
        0.50 high
```

SE on METM
SE on MTTM
SE on MI
SE on MF
Security Solution Cost (SC)
Security Solution Effect (SE)
Security Solution Treatment Level Utility
Security Solution Treatment Level (SSTL)

Fig. 16. Hugin tool SSTL subnet for SRP with evidence inserted and propagated. An "e" next to a node denotes hard evidence, defined as a result of the Alloy security analysis. The computed solution treatment level is equally likely to be *medium* or *high*, and it will never be *low*.

SSTL subnet variables include the extent to which the mechanism provides security properties, its effect on RL subnet variables METM, MTTM, MF, and MI, and its cost. We model the cost as a subnet that combines implementation cost, maintenance cost, and time to implement.

The security analysis described in Section 4 did not produce a counterexample for the SRP security-treated misuse model. In the analysis, we used a scope of 20, which, for our example, gives strong evidence that SRP protects the ACTIVE login sequence against active man-in-the-middle attacks. These results mean that the security effect (SE) is verified as being *high*, and hence, the variables "SE on METM," "SE on MTTM," "SE on MI," and "SE on MF" are all *high*. Fig. 16 shows the Hugin tool calculation of the SSTL subnet with this evidence entered and propagated. The right frame shows the network (SSTL in this case) and nodes marked with an "e" show where evidence has been entered. The left frame shows evidence as dark solid bars.

For purposes of demonstration, we define the cost of SRP as *medium* because the code is not shipped with Web clients as part of browsers, and thus, it must be added to both clients and servers at the application level. The resulting computation is the following pdf for the target variable SSTL treatment level: "$\text{SSTL.low} = 0.0$," "$\text{SSTL.medium} = 0.50$," and "$\text{SSTL.high} = 0.50$." The interpretation of this pdf is that it is just as likely that the treatment level is *medium* as *high* and the treatment level will never be *low*.

The Alloy Analyzer security analysis of the SSL security-treated misuse model did produce a counterexample for an active man-in-the-middle attack. The counterexample demonstrates an attack that does not require more than medium attacker skill, little resources, and time, so the solution effect is modeled as *low*. These results mean that all variables related to the effect of the security solution on an active MiM attack are set to *low* ("SE on METM," "SE on MTTM," "SE on MF," and "SE on MI"). For purposes of demonstration, we define the cost of SSL as *low* since the code is shipped with Web clients as part of browsers.

For the passive version of the attack, the Alloy Analyzer did not produce a counterexample, so we infer that the SSL protocol preserves the security properties under this particular attack. Its effect on the risk variables MI, MF, METM, and MTTM is therefore *high*. Table 3 shows the resulting SSTL calculations in the context of both passive and active man-in-the-middle attacks. There is a significant difference in the resulting treatment levels. For an active attack, the security analysis showed that all security effect variables are in the *low* state. The SSTL subnet is configured such that if all security effect variables are in the *low* state, both the solution effect and the resulting treatment level are in the *low* state, independent of the cost.

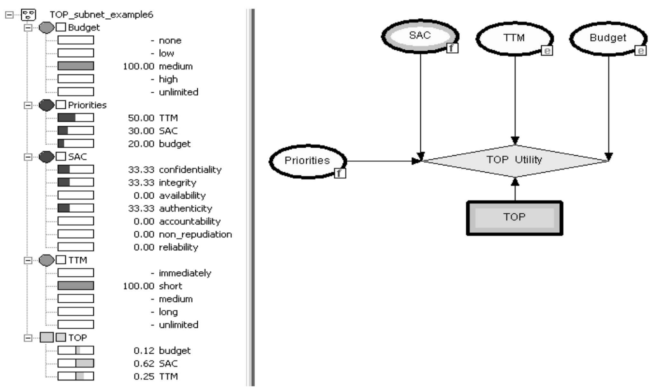
TABLE 3
SSL Treatment Level for Active and Passive Attacks

Variable name	Active attack value	Passive Attack Value
SSTL	1.00 low	0.00 low
	0.00 medium	0.25 medium
	0.00 high	0.75 high

However, in all other cases, cost is given a high importance in the calculation of the resulting treatment level. This bias can be seen in the resulting treatment values for the passive attack, where it is three times more likely that the treatment level is *high* rather than *medium* and it is never *low*.

5.4 Trade-Off Parameters Subnet (TOP)

Fig. 17 shows the TOP subnet with relevant information and results for our example. We first discuss the security-relevant node, the SAC variable. To simplify the demonstration of the trade-off tool, we use the qualitative scale *{N/A, low, medium, high}* for this node, which we represent as states of the security acceptance criteria. Here, *N/A* means not applicable, *low* means low value or importance, *medium* means medium value or importance, and *high* means high value or importance. The SAC variable is actually an input node that receives input from an associated subnet, as shown by the gray line inside the SAC oval. This subnet contains one node for each of the seven possible security



```
TOP_subnet_example6
  Budget
           - none
           - low
      100.00 medium
           - high
           - unlimited
  Priorities
       50.00 TTM
       30.00 SAC
       20.00 budget
  SAC
       33.33 confidentiality
       33.33 integrity
        0.00 availability
       33.33 authenticity
        0.00 accountability
        0.00 non_repudiation
        0.00 reliability
  TTM
           - immediately
      100.00 short
           - medium
           - long
           - unlimited
  TOP
        0.12 budget
        0.62 SAC
        0.25 TTM
```

Fig. 17. Hugin output of the TOP subnet. Budget and TTM represent hard evidence (denoted by an "e" by the node) of medium and short, respectively. SAC attributes (confidentiality, integrity, and authenticity) and priority (TTM, then SAC, and then budget) represent soft evidence, or probability distributions (denoted by an "f" by the node). The resulting TOP variable gives the derived priorities between the trade-off parameters, taking the related weights into consideration. The result is a priority of first SAC, then TTM, and finally budget.

properties [9]. In our example, the security properties confidentiality, authenticity, and integrity are relevant, and the corresponding nodes are in the *high* state. The accountability, availability, nonrepudiation, and reliability nodes are not relevant and the corresponding nodes are in the *N/A* state. Since we consider the three properties to be equally important, the result of the evidence propagation from the SAC subnet to the SAC variable in the TOP subnet is an equal probability between the confidentiality, integrity, and authenticity states. In addition to the security relevant information, there are time-to-market (TTM) and budget constraints. Since we need to produce a product in a small time, we define a value *short* for TTM. Our limited economic resource for incorporating a security solution to prevent MiM attacks necessitates that we define a value of *medium* for budget.

Finally, we include an additional variable called *priorities* in the TOP subnet that a designer can adjust when it is not possible to meet all the initial constraints. For example, the budget constraints may not allow satisfying all the security requirements and we need to prioritize the security requirements.

We model the variable "TOP" as a utility and decision node construct, and it produces a prioritized list of trade-off parameters. The utility node (with its associated utility function) specifies the dependencies between the states of the incoming nodes, and the decision node holds the result of the utility function executed on the input information. For example, consider the following configuration of the TOP utility function that we have implemented in our example.

We identify three trade-off parameters of interest—budget, security acceptance criteria (SAC), and TTM. These are the states of the *priorities* variable in Fig. 17. The priorities used in our example are the following: First priority is given to TTM, second priority is given to security requirements, and third priority is given to budget. We use a simple score-based schema to define the values of the *priorities* states (all scales are [1, 100]):

- the first priority trade-off parameter is assigned the score value 50,
- the second priority trade-off parameter is assigned the score value 30, and
- the third priority trade-off parameter is assigned the score value 20.

The utility function cannot use these priorities directly since each of the variables corresponding to the related parameters in the TOP subnet has several states associated with it. The utility function must take these states into consideration when deriving the distribution of the utility value of each trade-off parameter. We can use logic statements to handle this problem, as follows:

We first define a weight for each trade-off parameter, using the following schema. There are five states (see Fig. 17, left frame) associated with the TTM variable, and for simplicity, we assign the weight of 0.2 to each state. We assign a parameter weight by assuming that the states represent an enumerated ordering from *immediately* through *unlimited*. We then sum the weights of each state up to and including the targeted state to derive the weight of TTM. A target state of *short* TTM results in a weight of 0.4. We use the same method to derive a weight for the *budget* parameter: The five states associated with budget are each assigned a weight of 0.2, and weights of each state are again summed, up to and including the targeted state. The target budget of *medium*, therefore, results in a weight of 0.6. For SAC, we keep the value assigned to it, as discussed previously, scaled between 0 and 1 (i.e., 0.3).

The utility function value is derived by multiplying the parameter weight by the value of its associated state in the *priorities* variable and then normalizing the result over the combined variables' weight and priority. Since information is propagated back and forth through the network, the resulting TOP utility function is as follows:

$$\begin{aligned}
init.budget &= priority.budget \times budgetWeight,\\
init.TTM &= priority.TTM \times TTMWeight,\\
init.SAC &= priority.SAC \times SACWeight,\\
U_N &= \frac{1}{init.budget + init.SAC + init.TTM},\\
Top.budget &= init.budget \times U_N,\\
Top.TTM &= init.TTM \times U_N,\\
Top.SAC &= init.SAC \times U_N,
\end{aligned}$$

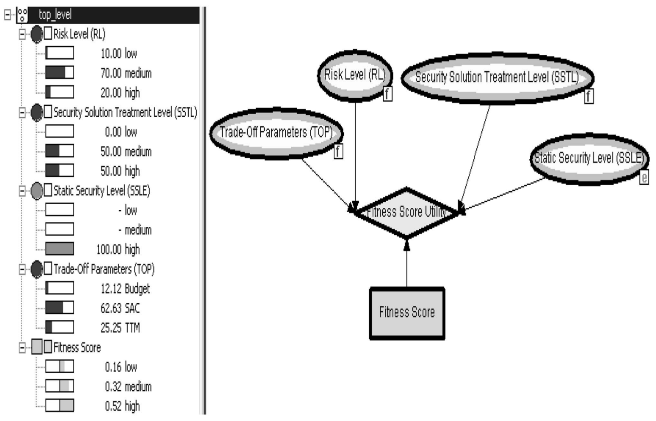
where U_N is the utility normalization factor. Each *init.* variable refers to the initial priority weight of the variable and is obtained by multiplying the weight (determined by the above schema) with the variable priority (e.g., 50 for TTM). Each *Top.* variable holds the updated priority weight after normalization.

5.5 Comparing SRP against SSL—The Fitness Network

The Hugin tool computes each subnet, using all evidence entered into the topology, and propagates the results into the respective observable nodes in the top-level fitness score network, as shown in Fig. 5. The fitness score utility function uses a ranked-weight schema. Higher priority trade-off parameters are ranked with higher fitness scores so that factors other than security can be taken into account when deciding between alternative security solution designs. This schema also gives us the ability to easily change the importance of a trade-off parameter if project circumstances change and we need to put more emphasis on meeting a different project goal. The fitness score is thus a measure of the degree that a particular security solution meets the security, development, project, and financial constraints of the project (specified in the TOP subnet).

Fig. 18 shows the resulting fitness score for the SRP security solution, which tells us that, when the priorities are TTM, SAC, and budget, the fitness of SRP in mitigating an active MiM attack is more than three times more likely to be *high* than *low* and 1.6 times more likely to be *high* than *medium* (16 percent for *low*, 32 percent for *medium*, and 52 percent for *high*). Note that these results do not mean that the fitness score is high 52 times out of 100, but that our belief is that it will be *high* more than half the time within a particular time frame.

We compute the fitness score for SSL by changing the SSTL variables in the top-level network in the BBN topology, based on the values discussed in Section 5.4.



```
top_level
  Risk Level (RL)
      10.00 low
      70.00 medium
      20.00 high
  Security Solution Treatment Level (SSTL)
       0.00 low
      50.00 medium
      50.00 high
  Static Security Level (SSLE)
          - low
          - medium
     100.00 high
  Trade-Off Parameters (TOP)
      12.12 Budget
      62.63 SAC
      25.25 TTM
  Fitness Score
       0.16 low
       0.32 medium
       0.52 high
```

Risk Level (RL)
f
Security Solution Treatment Level (SSTL)
f
Trade-Off Parameters (TOP)
f
Static Security Level (SSLE)
e
Fitness Score Utility
Fitness Score

Fig. 18. The resulting security solution fitness score for SRP. When the priorities are TTM, SAC, and then budget, the fitness of SRP to mitigate an active MiM attack is more than three times more likely to be *high* than *low* and 1.6 times more likely to be *high* than *medium*. An "*e*" next to a node denotes hard evidence, and an "*f*" denotes soft evidence (a probability distribution).

The computation produces a fitness score of 23 percent for *low*, 23 percent for *medium*, and 54 percent for *high* for SSL in the presence of an active MiM attack. (For a passive MiM attack, the result changes slightly and becomes 12 percent for *low*, 35 percent for *medium*, and 53 percent for *high*).

The above results imply that the fitness scores for SRP and SSL with the current trade-off parameter priorities differ by a small measure, so either one can be chosen. However, the situation changes if the priority of the trade-off parameters changes. If all emphasis is put on security requirements, meaning that the trade-off parameter SAC is given a priority of 100 percent, and both the active and passive MiM attack are taken into consideration, the fitness score changes to 20 percent for *low*, 75 percent for *medium*, and 5 percent for *high* for SRP and 0 percent for *low*, 90 percent for *medium*, and 10 percent for *high* for SSL. These results make sense as the treatment level of SRP is higher than for SSL in the context of the active MiM attack. The fitness score is, however, still not completely in favor of SRP for two reasons. First, SRP involves a higher cost and time to market than SSL. Second, the risk level of the MiM attack is most likely medium. SSL has a low treatment level for active attacks, but never a low treatment level in the context of passive MiM attacks, as shown in Table 3. For this reason, the fitness score of SSL turns out to be heavily ranked toward medium when both attack types are taken into consideration.

5.6 Sensitivity of the Topology

The outcome of a BBN computation is sensitive to the configuration of the BBN topology with its composite subnets and associated pdfs, and it must be constructed carefully to ensure representative results. Also, different BBN topologies might interpret the same evidence differently depending on which nodes in the network are sensitive, meaning that they are given high priority by the computation engine. These nodes are called neighboring networks. The reason they are important is that the Hugin evidence propagation algorithm first reduces the topology according to its rule set, and then, transfers the BBN topology into a set of trees that are computed separately. It is, therefore, important to understand how the evidence propagation works in order to construct a healthy BBN topology. In practice, topology designers can perform uncertainty and sensitivity analyses to identify whether the nodes and/or evidence intended to be of high importance are treated properly. The Hugin tool provides these analyses, and we have constructed the trade-off topology using their results.

6 Related Work

A great deal of research is motivated toward making real-world systems secure. Since this is a very complex problem, most of the literature addresses only parts of it. We aim to provide a more comprehensive solution by showing how such a complex system can be developed using our AORDD framework, how the trade-off decisions can be made using probabilistic analysis, and the resulting system formally analyzed to give assurance of security properties. Our work either complements or builds upon existing work, some of which we describe briefly in this section.

Standards such as the ISO 15408 Common Criteria for Information Technology Security Evaluation [1] can help developers focus on processes and development activities that lead to more secure systems. However, these standards only address the development activities of the system, not its operational security, which is observed during system operation at a particular point in time, as described by Littlewood [39]. These standards also assume assessment by certified assessors. Experienced assessors must apply trade-off techniques such as Architecture Trade-off Analysis Method (ATAM) [40] and Cost Benefit Analysis Method (CBAM) [41] at an architectural level. Any of these assessments requires a strong resource commitment on the part of the organization that uses them. Our trade-off analysis incorporates ideas from both ATAM and CBAM, and provides methodologies and tools that lighten the resource requirements. We also incorporate security-specific parameters in the form of both static (during development) and dynamic (operational) security levels and other project-specific parameters as input to trade-off analysis.

We have chosen not to incorporate financial approaches such as Net Present Value (NPV) or Real Option Value (ROA), described by Daneva [42] and Benaroch [43], or existing return on security investment (RoSI) approaches, such as that of Sonnenreich et al. [44], as we believe that these approaches take too narrow a scope for the trade-off analysis of security mechanisms. These approaches often depend on the ability to measure all variables in terms of economical value. The problem arises when designers wish to compare alternative security solutions, as there is little hard evidence available to assess the true cost of a successful attack or the amount of money saved by implementing a mitigating solution versus no mitigation. The only hard evidence that may be available is the procurement, implementation, and maintenance cost of the security solution. Furthermore, a security solution decision process includes factors to which cost cannot be attached, such as legislation, time-to-market, etc. Our BBN trade-off topology can incorporate any or all of these disparate factors, and is, therefore, easy to evolve and adapt according to what the designer considers to be important and what information is available.

Risk identification, assessment, and management are the targets of the CCTA Risk Analysis and Management Methodology (CRAMM) [45] and CORAS [10], [11] frameworks. CORAS makes use of multiple standards, including the Australian/New Zealand Standard for Risk Management, ISO/IEC 17799-1: Code of Practice for Information Security Management [46] and ISO/IEC 13335: Information technology—Guidelines for Management of IT security [9]. CORAS adapts the asset-driven structured approach in CRAMM and uses model-based risk assessment in integrated system development processes. Our AORDD framework [6], [7] makes use of the CORAS processes.

One suitable choice of formalism for UML models is Alloy, which is specifically designed for Object-Oriented design. We base our security analysis on Alloy, by transforming UML class diagrams and OCL into Alloy models. Massoni et al. [47] also transform UML Class diagrams to Alloy in order to analyze structural properties of UML models. However, their work covers a small subset of UML and OCL and is not automated. Our methods encompass larger subsets of UML and OCL, and we provide automation through the UML2Alloy tool [27].

Torlak et al. [37] have used Alloy to analyze man-in-the-middle attacks. They introduce the idea of *knowledge flow*, which models how knowledge (information) is exchanged between the participants of a protocol handshake. In particular, they focus on the information that the man-in-the-middle can possess, irrespective of the order the messages are exchanged between the participants. They used their method on the Needham and Schroeder protocol [38] and verified the existence of a flaw already discovered by Lowe [48]. This work is similar to the approach presented in this paper with the exception that while they model the system directly in Alloy, our work utilizes more popular notations such as class diagrams and OCL, and provides transformations to Alloy. Our work also provides a method to interpret the Alloy Analyzer results in the context of other project goals through the BBN trade-off analysis.

There are also various tools to support the analysis of UML models. For example, the UML Specification Environment (USE) tool [49] is a Java implementation, which provides simulation and validation functionality. More specifically, designers can use the USE tool to generate snapshots that conform to the model. They can check if a specific instance of the model (generated via a script) conforms to the invariants. In contrast, our approach uses the Alloy Analyzer, which automatically searches the state space exhaustively (up to the user specified scope), without the requirement to learn and use a scripting language to generate the instances.

Basin et al. [50] describe an approach to creating security design models that integrate security policies into UML design models. Their approach is language-based and uses languages for a particular security policy and system design, along with a dialect (that specifies how they are related), to specify system designs that contain the security policy. Their early work describes an example of these concepts, in the form of the SecureUML language that can be used to specify access control policies for role-based access. The dialect Basin et al. develop constrains how the access control policies are used to augment the system design. The authors demonstrate generation rules to transform the model into control infrastructures such as those provided by EJB or .NET. In recent work, Basin et al. [51] also provide an OCL query methodology and tool that allows directed inquiry into specific access control in potential runtime instances, as specified by scenarios, or system states.

Our work differs in several areas. For example, we approach the modeling of security policies through aspect composition rather than through specific languages. We feel that representing security policies as models allows a designer great flexibility and reduces any dependence on a language developer. In addition, the combination of security policies specified as aspect models and composition allows a designer to freely explore many alternatives in search of a satisfactory design. Different languages for each prospective policy would be required in a language-based approach. Our methodology is focused on early design analysis, whereas Basin et al. have emphasized development and implementation issues. Their newer OCL query tool can be used in design to explore particular state access control, whereas our analysis methods are based on dynamic behavior models that result in a set of system states where we can test assertions, the possibility of overconstraining the system, etc. We use an established tool, the Alloy Analyzer to perform these analyses. Finally, by using aspect models and composition, we can analyze the effectiveness of particular security policies, as realized by security mechanism designs, against particular security threats to a system design.

7 Conclusions and Future Work

Ad hoc approaches for developing secure systems may result in security breaches. We propose an AORDD methodology for designing secure systems. Designers perform a risk assessment to identify the attacks on the system and evaluate how the assets of the system can be compromised. Designers then methodically identify and incorporate security mechanisms into the design that protect against these attacks. They evaluate the resulting system to give assurance that it is indeed secure. Multiple security mechanisms are often effective in protecting against a given attack, so designers must identify and integrate the mechanism most suitable for the application.

This paper makes two major contributions. First, we show how to formally verify that a security mechanism incorporated into a system is effective in protecting against a given security breach. Toward this end, we show how a system modeled using UML is converted to a form that can be automatically verified using the Alloy Analyzer. Second, we show how to compare the suitability of different security mechanisms for a given application on the basis of factors such as the level of protection offered, cost and effort involved, and the time to market. We demonstrate how a BBN topology can be used to perform such trade-off analysis and provide evidence to make a suitable choice between competing designs.

In this paper, we illustrated the case for a single attack. However, in reality, there are multiple attacks and multiple security mechanisms must be incorporated. Moreover, incorporating a security mechanism should not open up new vulnerabilities. The existing approach addresses these issues through AOM techniques. Designers can continually augment system designs by composing additional security mechanisms to mitigate additional attacks. They can then compose multiple attack models with these system models, and analyze them with the Alloy Analyzer. However, this approach can be cumbersome for designers, so we hope to provide an easier approach for handling multiple attacks. In this respect, we are currently investigating techniques that formalize the dependencies between the different types of attacks and security solutions. Such formalization will allow us to group attacks and security solutions. This, in turn, will facilitate minimizing the time required for security analysis.

A main focus of our on-going research relates to analyzing the model in which a security mechanism design has been incorporated. The analysis, shown in this work, consists of three parts: 1) converting UML sequence diagrams to OCL, 2) using the UML2Alloy automated conversion tool to transform OCL specifications into those of Alloy, and 3) verifying security properties in the resulting Alloy model using the Alloy Analyzer. Parts 2 and 3 are completely automated. To increase the automation of part 1, we are working on algorithms that will automatically convert UML sequence diagrams into OCL. This work must include a method to create abstractions of the model that do not alter the security property being verified. This work will make the analysis more automated.

However, while we can transform security property assertions, it is not possible to fully automate the process of identifying which security properties are relevant to a system. We aim to support this security engineering task by providing a checklist of security properties, with associated design solutions, as a repository of templates in the form of aspects. A designer or security engineer can use such a repository to instantiate a design solution that provides a relevant property and compose it into the model, speeding the application of our approach. Creation of such a repository involves substantial implementation, and therefore, remains an area for future work. Another automation opportunity related to part 2 is to develop a scheme to make the attack models specified in Alloy reusable across different applications. The scheme would then be incorporated into parts 1 and 2 as applicable.

Another research thread consists of formally defining and modeling factors that will affect the choice of the security mechanisms, such as the performance or fault tolerance. We are currently modeling the performance requirements of security mechanisms using Layered Queuing Networks. Our intent is to include the performance analysis results as another input to the trade-off analysis tool as part of the decision-making criteria for choosing a security solution design.

ACKNOWLEDGMENTS

This work was partially supported by the US Air Force Office of Scientific Research under Award No. FA9550-07-1-0042.

REFERENCES

[1] ISO 14508, *Common Criteria for Information Technology Security Evaluation, in Version 3.1, Revision 2,* 2007.

[2] ISO 14508-4, *Common Methodology for Information Technology Security Evaluation: Evaluation Methodology, in Version 3.1, Revision 2,* 2007.

[3] AS/NZS, *Australian/New Zealand Standard for Risk Management AS/NZS 4360:2004,* ANZ Standard, ed., 2004.

[4] AS/NZS, *Australian/New Zealand Standard HB 436:2004 Risk Management Guidelines—Companion to AS/NZS 4360:2004,* ANZ Standard, ed., 2004.

[5] S.H. Houmb, "Decision Support for Choice of Security Solution: The Aspect-Oriented Risk Driven Development (AORDD) Framework," Dept. of Math. Sciences, Norwegian Univ. of Science and Technology, 2007.

[6] S.H. Houmb et al., "Cost-Benefit Trade-Off Analysis Using BBN for Aspect-Oriented Risk-Driven Development," *Proc. IEEE Int'l Conf. Eng. Complex Computer Systems,* pp. 195-204, 2005.

[7] S.H. Houmb et al., "An Integrated Security Verification and Security Solution Design Trade-Off Analysis Approach," *Integrating Security and Software Eng.: Advances and Future Vision,* H. Mouratidis and P. Giorgini, eds., IGI Global, 2007.

[8] OMG, *Unified Modeling Language: Superstructure Version 2.1.2 Formal/07/11/02,* 2002.

[9] ISO/IEC 13335-5, *Information Technology—Guidelines for Management of IT Security,* 2001.

[10] 1CORAS, *IST-2000-25031,* 2003.

[11] K. Stølen et al., "Model Based Risk Assessment in a Component-Based Software Engineering Process: The CORAS Approach to Identify Security Risks," *Business Component-Based Software Eng.,* F. Barbier, ed., pp. 189-207, Kluwer, 2002.

[12] R. France et al., "A UML-Based Pattern Specification Technique," *IEEE Trans. Software Eng.,* vol. 30, no. 3, pp. 193-206, Mar. 2004.

[13] R. France et al., "Aspect-Oriented Approach to Design Modeling," *IEE Proc. Software,* vol. 151, no. 4, pp. 173-186, 2004.

[14] G. Georg, J. Bieman, and R. France, "Using Alloy and UML/OCL to Specify Run-Time Configuration Management: A Case Study," *Proc. Workshop pUML-Group Held Together with the UML,* A. Evans et al., eds., pp. 128-141, 2001.

[15] G. Georg et al., "An Aspect-Oriented Methodology for Designing Secure Applications," *Information and Software Technology,* vol. 51, no. 5, pp. 846-864, 2009.

[16] G. Straw et al., "Model Composition Directives," *The Unified Modelling Language: Modelling Languages and Applications (UML),* T. Baar et al., eds., pp. 84-97, Springer, 2004.

[17] Alloy, http://alloy.mit.edu, 2009.

[18] D. Jackson, *Software Abstractions: Logic, Lanaguage, and Analysis.* MIT Press, 2006.

[19] B.D. Finetti, *Theory of Probability,* vols. 1 and 2. John Wiley and Sons, 1973.

[20] F. Jensen, *An Introduction to Bayesian Network.* UCL Press, 1996.

[21] J. Pearl, *Probabilistic Reasoning in Intelligent Systems.* Cambridge Univ. Press, 1998.

[22] B.A. Gran, "The Use of Bayesian Belief Networks for Combining Disparate Sources of Information in the Safety Assessment of Software Based System," Dept. of Math. Science, Norwegian Univ. of Science and Technology, 2002.

[23] SERENE, SERENE: Safety and Risk Evaluation Using Bayesian Nets, p. ESPIRIT Framework IV nr. 22187, 1999.

[24] OMG, *XML Metadata Interchange (XMI) V2.0 Formal/05-05-01,* 2005.

[25] OMG, *Object Constraint Language Version 2.0 Formal/06/05/01,* 2006.

[26] ArgoUML, http://argouml.tigris.org, 2009.

[27] K. Anastasakis et al., "UML2Alloy: A Challenging Model Transformation," *Proc. 10th Int'l Conf. Model Driven Eng. Languages and Systems,* G. Engels et al., eds., pp. 436-450, 2007.

[28] B. Bordbar and K. Anastasakis, "UML2ALLOY: A Tool for Light-Weight Modelling of Discrete Event System," *Proc. Int'l Conf. Applied Computing,* N. Guimarães and P.T. Isaías, eds., pp. 209-216, 2005.

[29] P. Ziemann and M. Gogolla, "An Extension of OCL with Temporal Logic," *Proc. Workshop Critical Systems Development with UML,* J. Jürjens, ed., pp. 53-62, 2002.

[30] HUGIN, *Hugin Expert A/S,* 2007.

[31] T. Dimitrakos et al., "Integrating Model-Based Security Risk Management into Ebusiness Systems Development: The CORAS Approach," *Proc. IFIP Conf. E-Commerce, E-Business, E-Government,* J. Monteiro, P. Swatman, and L. Tavares, eds., pp. 159-175, 2002.

[32] T. Wu, "The Secure Remote Password Protocol," *Proc Internet Soc. Network and Distributed System Security Symp.,* pp. 97-111, 1998.

[33] T. Wu, *The SRP Authentication and Key Exchange Systems,* N.W. Group, ed., 2000.
[34] TLSWG, *SSL 3.0 Specification,* 1996.
[35] G. Georg et al., *Security Property Verification and Trade-Off Analysis Using UML.* Colorado State Univ., 2008.
[36] M. Vaziri and D. Jackson, "Some Shortcomings of OCL, the Object Constraint Language of UML," *Proc. Int'l Conf. Technology of Object-Oriented Languages and Systems,* Q. Li et al., eds., pp. 555-562, 2000.
[37] E. Torlak et al., *Knowledge Flow Analysis for Security Protocols.* Computer Science and Artificial Intelligence Laboratory, MIT, 2005.
[38] R.M. Needham and M.D. Schroeder, "Using Encryption for Authentication in Large Networks of Computers," *Comm. ACM,* vol. 21, no. 12, pp. 993-999, 1978.
[39] B. Littlewood et al., "Towards Operational Measures of Computer Security," *J. Computer Security,* vol. 2, nos. 2/3, pp. 211-230, 1993.
[40] R. Kazman, M. Klein, and P. Clements, "ATAM: Method for Architecture Evaluation," Technical Report CMU/SEI-2000-TR-004, Carnegie Mellon Univ./Software Eng. Inst., 2000.
[41] R. Kazman, J. Asundi, and M. Klein, "Making Architecture Design Decisions: An Economic Approach," Technical Report CMU/SEI-2002-TR-035, Carnegie Mellon Univ./Software Eng. Inst., 2002.
[42] M. Daneva, "Applying Real Options Thinking to Information Security in Networked Organizations," CTIT Report TR-CTIT-06-11, Univ. of Twente, 2006.
[43] M. Benaroch, "Managing Information Technology Investment Risk: A Real Options Perspective," *J. Management Information Systems,* vol. 19, no. 2, pp. 43-84, 2002.
[44] W. Sonnenreich, J. Albanese, and B. Stout, "Return on Security Investment (ROSI)—A Practical Quantitative Model," *J. Research and Practice in Information Technology,* vol. 38, no. 1, pp. 45-56, 2006.
[45] B. Barber and J. Davey, "The Use of the CCTA Risk Analysis and Management Methodology CRAMM in Health Information Systems," *Proc. Medical Informatics Conf.,* K.C. Lun et al., eds., pp. 1589-1593, 1992.
[46] ISO/IEC, *Information Technology—Code of Practice for Information Security Management,* 2000.
[47] T. Massoni, R. Gheyi, and P. Borba, "A UML Class Diagram Analyzer," *Proc. Int'l Workshop Critical Systems Development with UML at UML,* pp. 100-114, 2004.
[48] G. Lowe, "Breaking and Fixing the Needham-Schröeder Public-Key Protocol Using FDR," *Proc. Int'l Conf. Tools and Algorithms for Construction and Analysis of Systems,* T. Margaria and B. Steffen, eds., pp. 147-166, 1996.
[49] M. Gogolla, J. Bohling, and M. Richters, "Validating UML and OCL Models in USE by Automatic Snapshot Generation," *Software and System Modeling,* vol. 4, no. 4, pp. 386-398, 2005.
[50] D. Basin, J. Doser, and T. Lodderstedt, "Model Driven Security: From UML Models to Access Control Infrastructures," *ACM Trans. Software Eng. and Methodology,* vol. 15, no. 1, pp. 39-91, 2006.
[51] D. Basin et al., "Automated Analysis of Security-Design Models," *Information and Software Technology,* vol. 51, no. 5, pp. 815-831, 2009.

Geri Georg received the MS and PhD degrees in computer science from Colorado State University, Fort Collins, where she is a research associate in computer science. Her research interests include modeling and analysis of crosscutting system properties in complex distributed systems. She is also interested in aspect-oriented modeling and model visualization of these properties. Prior to joining Colorado State University, she worked at the corporate research laboratories of Hewlett Packard Company and Agilent Technologies. She is a member of the ACM and the IEEE Computer Society.

Kyriakos Anastasakis received the MSc degree in advanced computer science and the PhD degree in computer science from the University of Birmingham, United Kingdom, in 2004 and 2009, respectively. His doctoral thesis is entitled "A Model Driven Approach for the Automated Analysis of UML Class Diagrams." His research is focused on software engineering, model-based approaches for systems development, and formal methods. He worked as an independent software consultant in Birmingham and as a software developer for Ulysses Systems prior to his postgraduate studies.

Behzad Bordbar received the PhD degree in pure mathematics from the University of Sheffield, United Kingdom. He is currently a lecturer in the School of Computer Science, University of Birmingham, United Kingdom. He was a research fellow in discrete event systems at the University of Ghent Belgium, in massively distributed manufacturing systems at Aston University, Birmingham, and in distributed multimedia systems at the University of Kent, United Kingdom, which he joined as a lecturer. His research interests are in software tools and techniques for design, analysis, and implementation of large distributed systems. In particular, he is currently interested in model-driven architecture, domain-specific languages, performance modeling, and fault tolerance in service-oriented architectures. He is a member of the IEEE and the IEEE Computer Society.

Siv Hilde Houmb received the PhD degree in computer science (security) from the Norwegian University of Science and Technology, Trondheim, Norway. She is a researcher and security expert at Telenor R&I and a Telenor delegate on three standardization technical committees: ETSI TC TISPAN, ETSI TC IST, and ETSI TC M2M. She also works as a Special Task Force security expert at the Telecommunication Standardization Organization ETSI (for Telenor). Her research interests include decision support methodologies and techniques that allow architects to choose among sets of security solutions in security critical information systems taking all of security, development, project, and financial constraints into consideration.

Indrakshi Ray received the PhD degree from George Mason University. She is an associate professor in the Computer Science Department at Colorado State University. Prior to joining Colorado State, she was a faculty member at the University of Michigan-Dearborn. Her research interests include security and privacy, database systems, e-commerce, and formal methods in software engineering. She has published several refereed journal and conference papers in these areas. She served as the general chair for SACMAT '08, the program chair for SACMAT '06, and the program cochair for the IEEE/IFIP TSP '08 and IFIP WG 11.3 DBSEC 2003. She has also been a member of several program committees such as EDBT, SACMAT, ACM CCS, and EC-Web. She is a member of the ACM, the IEEE, and the IEEE Computer Society.

Manachai Toahchoodee received the BS degree in mathematics and the MS degree in information technology from King Mongkut's University of Technology Thonburi, Thailand. He is a doctoral candidate in the Department of Computer Science at Colorado State University. His research interests include spatiotemporal access control models, modeling and analysis of access control models, and aspect-oriented secure software development. He has served as an instructor at the University of the Thai Chamber of Commerce, and as a system analyst for IBM Solution Delivery and Sanofi-Aventis Co., Ltd., both in Thailand.