Aspect-Oriented Risk Driven Development of Secure Applications

Geri Georg[1], Siv Hilde Houmb[2], and Indrakshi Ray[1]

[1] Computer Science Department
Colorado State University
Fort Collins, Colorado, USA
{georg, iray}@cs.colostate.edu
[2] Computer Science Department
Norwegian University of Science and Technology
Trondheim, Norway
sivhoumb@idi.ntnu.no

Abstract. Security breaches seldom occur because of faulty security mechanisms. Often times, security mechanisms are incorrectly incorporated in an application which allows them to be bypassed resulting in a security breach. Methodologies are needed for incorporating security mechanisms in an application and assessing whether the resulting system is indeed secure. We propose one such methodology for designing secure applications. We begin by identifying the assets in the application that need protection. We then find the kinds of attacks that are typical for such applications. We show how to evaluate the application against such attacks. If the results are unacceptable, that is, they pose a high security risk, then some security mechanism must be incorporated into the application. We illustrate how this can be done and show how the resulting system can be evaluated to give assurance that it is resilient to the given attack.

1 Introduction

In the commercial world, designing secure applications is impacted by various parameters, such as time-to-market, cost and effort involved. The presence of these constraints often prevents the development of applications which are adequately secure. We propose a risk driven development approach for designing such applications. While designing an application one needs to understand the threats in the current design and the risks associated with those threats. If the risks are unacceptably high, the application must be redesigned. Redesigning the application means methodically incorporating security mechanisms into the application and evaluating whether the resulting application is adequately secure.

Security mechanisms are solutions to security problems in applications. For example, encrypting information is a solution to prevent malicious attackers from eavesdropping on sensitive information sent in clear-text. However, there might be several mechanisms to solve one problem. This implies that we need to evaluate to what extent the different mechanisms solve the problem for a given application and what is the cost associated with each. Security and risk management standards [1,2,6] were developed to aid secure systems development. Such standards often require extensive amount of work and

E. Damiani and P. Liu (Eds.): Data and Applications Security 2006, LNCS 4127, pp. 282–296, 2006.

also include other activities that are necessary for evaluating security mechanisms. In this paper, our goal is to complement the above mentioned work on standards and show how to assess whether an application is indeed secure when a particular security mechanism has been incorporated. This is important because often the security mechanisms designed to thwart attacks are adequate; yet security breaches still occur because the security mechanisms are often bypassed in an application.

Our approach begins by specifying the *primary model* which represents the application functionality. The items that need protection are identified as *assets*. The attacks on the application are then identified and modeled. The *attack model* is then composed with the primary model to produce the *misuse model*. The misuse model illustrates the degree to which the application can be compromised and the risk posed by the attack. If the risk is unacceptable, some security mechanism must be incorporated into the application. The model of the security mechanism is then methodically composed with the application. The result, which we refer to as the *security treated model*, represents the application in which the security has been incorporated. Finally, we show how the security treated model can be analyzed to give assurance that the application is indeed resilient to the given attack.

Our approach is based on aspect-oriented modeling techniques. Complex software is not developed as a monolithic unit but is decomposed into modules on the basis of functionality. An attack is not confined to one module of the application but impacts several of the modules. Similarly, a security mechanism will impact multiple modules of the application. Modeling security mechanisms and attack models as aspects have several benefits - it allows the attacks and the mechanisms to be understood in isolation, which makes it easier to manage and change these models. Once security mechanisms or attack models are represented as aspects, then techniques for composing aspects with the primary model can be used to understand the effect of the attack or the effect of security mechanism on the application.

The rest of the paper is organized as follows. Section 2 describes the e-commerce system which we use to illustrate our methodology. Section 3 gives an example attack and shows how the attack can be represented as an aspect. This section also describes how to generate the misuse model from which we we can identify the impact of the attack on our example application. Section 4 shows how the security mechanism designed to thwart the given attack can be represented as an aspect and how this mechanism can be integrated with the application. It also shows how the resulting system can be analyzed to give assurance that it is indeed resilient to the attack. Section 5 discusses some related work. Section 6 concludes the paper with some pointers to future directions.

2 Example E-Commerce System

We illustrate the reasoning about security risk mitigation with the login service of an e-commerce platform. The ACTIVE e-commerce platform provides services for electronic purchasing of goods over the Internet. The platform was developed initially for the purchase of medical equipment, although it is generalized to provide services for any kind of goods. (For details, please see T. Dimitrakos et. al [2]). ACTIVE is a general purchase platform that can host a variety of electronic stores for vendors. The infrastructure

consists of a web server running Microsoft Internet Information Server (IIS), a Java application server (Allaire JSP Engine) and a Microsoft SQL server running RDBMS. The communication between the application server and the database is handled using the JDBC protocol.

There are two types of consumer users in the ACTIVE system, *visitors* and *registered* users. Personalized shopping services are only available to registered users, but all users can browse and purchase items from ACTIVE. In addition, visitors cannot add any personal information to the system that will be retained for future shopping sessions.

The IST EU-project CORAS (for details see CORAS project report [7]) performed three risk assessments of ACTIVE in the period 2000-2003. The project looked into security risks of the user authentication mechanism, secure payment mechanism, and the agent negotiation mechanisms of ACTIVE. The example in this paper concentrates on the result from the risk assessment of the user authentication mechanism, and its impact on login services.

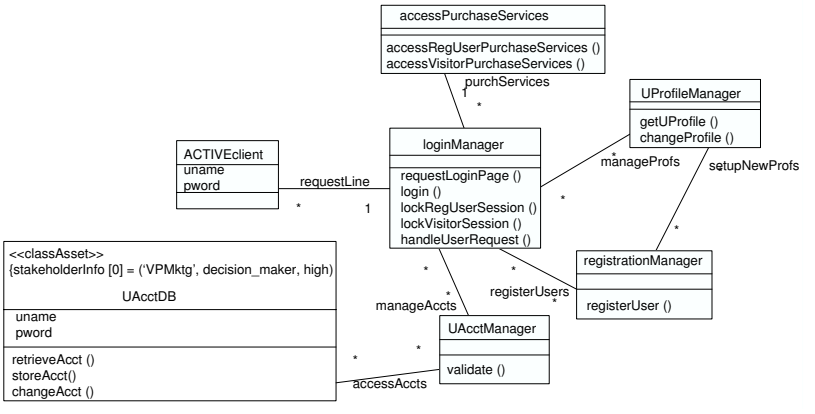

Fig. 1. Primary model (E-commerce login service) static diagram

We begin by creating a static diagram of the login service components. This diagram is shown in Figure 1. (We have simplified the diagram to only include model elements directly affected by the attack and its treatment.) There are several classes that play a part in the login process. A user wishing to login to the e-commerce system runs an *ACTIVEclient* in a web browser on their local machine. The browser communicates with a login manager (*loginManager*) which is located on a server across the Internet. The login manager has several related classes. An account manager (*UAcctManager*) and the associated database (*UAcctDB*) are used to authenticate users using a simple user name and password provided by the client web browser. A profile manager (*UProfileManager*) is used to keep track of personalized shopping information. A registration manager (*registrationManager*) is used to allow a visitor to become a registered user and a purchase service class (*accessPurchaseServices*) is used to access the different shopping services.

Risk-driven development (RDD) UML profile elements are also shown in Figure 1. These profile elements are used to annotate UML diagrams with additional information

useful in risk treatment trade-off analysis. For example, the <<classAsset>> stereotype is used to indicate that the *UAcctDB* class is an asset in the system. A RDD profile tag (stakeholder information) is associated with the asset. Stakeholder information is an array containing the name of the stakeholder ("VPMktg" in this case), the role of the stakeholder (decision maker), and the value the stakeholder places on the asset (in this case, extremely high value).

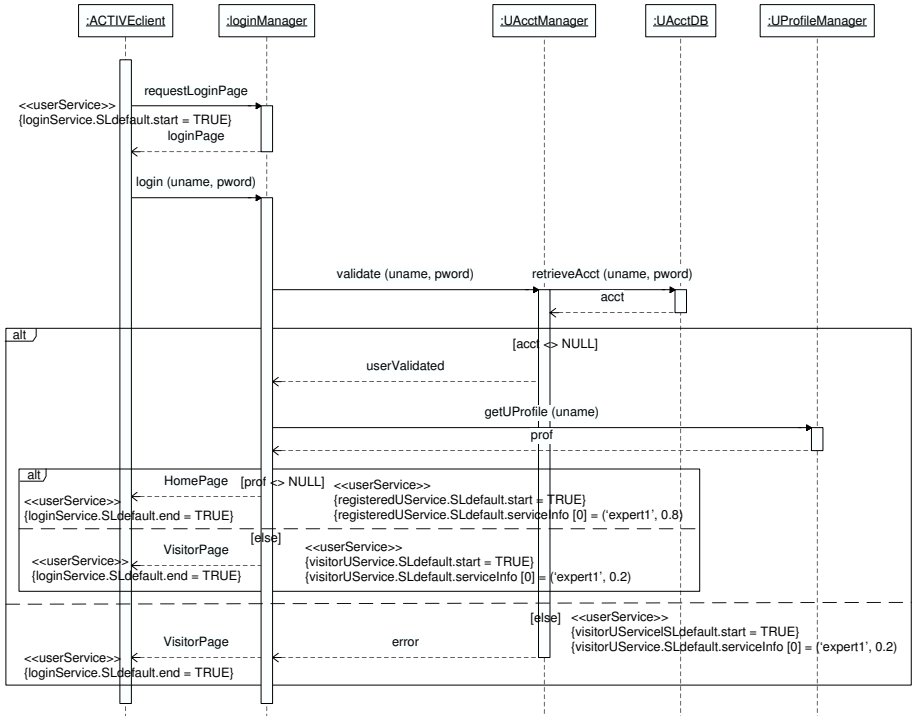

Fig. 2. Primary model (E-commerce login service) sequence diagram

The login sequence is shown in Figure 2. The *registrationManager* and *accessPurchaseServices* classes are not shown in this figure since they are classes whose services are used after a user has entered the system as a registered user or a visitor.

Figure 2 shows the sequence for a login operation. First, a user, through a web browser (*ACTIVEclient*), requests a login page from the e-commerce system by sending *requestLoginPage* to *loginManager*. *loginManager* responds with *loginPage*. The user enters his unique user name (*uname*) and password (*pword*), and this information is sent to *loginManager*. The server then sends *validate* message to *UAcctManager*. The *UAcctManager* sends an *error* message to the *loginManager* if the user account does not exist or cannot be validated. Otherwise *userValidated* message is returned to the server. If the user login information is valid, the *loginManager* sends *getUProfile* message to *UProfileManager*. The *UProfileManager* retrieves the user's *profile* and sends it to the

loginManager. Using this information the *loginManager* creates an appropriate home page which is returned to the user's web browser. If the user's login information could not be validated, or the user's profile could not be obtained, a visitor page is returned to the browser. Although this is the end of the login sequence, the user can either continue as a visitor, or register as a new user in the e-commerce system.

In this example, our requirements are that users should have access to the e-commerce system and non-users should be allowed access as visitors. Consequently, we add several RDD profile elements in the login service sequence. The first defines the start of the login service with the stereotype <<userService>>, and the associated tag {*loginService.SLdefault.start = TRUE* }. This service begins when the *ACTIVEclient* sends a *requestLoginPage* message to the *loginManager*. Another set of user service beginning tags occurs when *loginManager* returns either a *HomePage* (*registeredUService*) or a *VisitorPage* (*visitorUService*) to the *ACTIVEclient*. These other user services have service information associated with them, namely the probability that they will be achieved, and the source of that information. In this example, "expert1" supplied the information, but there can be multiple sources of such information, including running system data from a honeypot. In this example, the probability of a user being registered is higher than that of a user being a visitor. Note that the visitor user service is achieved under two circumstances: 1) if the user name and password cannot be validated, and 2) if the user profile information cannot be obtained. Finally, RDD profile elements identify the points in the sequence when the login service has been completed, namely, at the end of the message when the login manager returns *HomePage* or *VisitorPage* to the *ACTIVEclient*. Note that the messages in the sequence diagram all have an explicit return message. This is required to be able to compose sequence diagrams according to our composition mechanism.

3 The Man-in-the-Middle Attack

The risk assessments performed as part of the CORAS project identified the login process as being vulnerable to man-in-the-middle attacks. During this kind of attack user names and passwords can be intercepted by an attacker, and used at later times to impersonate a valid user.

Each attack in our model is an aspect because an attack is not confined to one specific module of the application but impacts the entire application. We propose to represent those attacks that are not confined to one specific application as a generic aspect. Generic aspects are represented as patterns which are described using UML templates. These templates must be instantiated for each application to obtain a *context-specific attack model*.

In this section, we show how the man-in-the-middle attack can be represented as a *generic aspect*. Messages between a requestor and authenticator are intercepted by an attacker. This can only occur if all messages flow through the attacker and not through a direct association between the requestor and authenticator. The attacker either intercepts the message intended for the server, or the attacker eavesdrops on the communication medium between the browser and the server. In the first case, the attacker must pose as the server so that the message intended for the server really gets sent to the attacker.

The attacker then relays messages between the client browser and the server until the private information has been obtained by the attacker. In the second case the attacker does not impersonate the server, but rather just eavesdrops on the message flow. The attacker may not obtain all of the messages flowing between the client and server, but simply sample messages in the hopes of obtaining information. We use the first type of man-in-the-middle attack in this paper since the attacker can actually participate in complex protocols, and change messages if desired before passing them on to the client or server.

Due to space constraints, we not not show the attack model, but rather describe it. The attack model is shown as part of the misuse model described in the next section.

The generic attack model describes two service levels for the user service. The first is the default service level (indicated by *SL0*) which is the "best" level of service. This level of service is based on the physical connection. The second level of service described is level X (denoted by *SLX*), which means that either a non-user has gained access to the system, or that users have lost access to system services (that is, the system has gone down). In short, the service level X means that the system has been compromised. In addition, a requestor, authenticator, and an attacker must all be connected to the same network to enable a man-in-the-middle attack.

The annotations for the misuse service include service information such as that included for the user services in the primary login sequence, but they also contain other information. Misuse service information consists of the source of the information, the probability that the service level will be achieved, the average time it takes to achieve the service level (MTTM), the average effort it takes (METM), and the impact on asset value (IV). Probability information can either be supplied by an expert, based on experience with similar systems, or by a honeypot system that logs actual events. Different generic diagram with different probabilities, values of MTTM, METM, and IV can be created for cases where the connection is an Internet, LAN, or some other type of connection.

3.1 Generating the Misuse Model

In order to understand the impact the man-in-the-middle attack has on the e-commerce application, we need to generate the misuse model. The misuse model will indicate how much the primary model can be compromised by the attack. Two steps are needed to generate the misuse model:

1. Instantiate the generic attack aspect to obtain the context-specific attack aspect.
2. Compose the context-specific attack aspect with the primary model to obtain the misuse model.

Instantiating the Generic Aspect: The generic aspect is application-independent. It is specified using UML templates. These templates must be instantiated for a given application. This instantiation is done by binding names in the generic aspect to those in the primary model. Elements present in the generic aspect that do not have a counterpart in the primary model must also be instantiated. The instantiation of the generic aspect will be referred to as a *context-specific aspect*. For the e-commerce example, a context-spceific aspect is obtained by making the *ACTIVEclient* the requestor of an

authentication, the *loginManager* the authenticator, and the login message the authentication request. The user service of interest is the login service.

Obtaining the misuse model: The context-specific aspect must be composed with the primary model to obtain the *misuse model*. The first step is to compose the class diagrams of the attack and primary models. For lack of space, we do not show the class diagram of the attack model or the composition process. The result from this composition is the class diagram of the misuse model shown in Figure 3. The misuse class diagram differs from the primary model class diagram in the following ways: (i) an *attacker* class is added, (ii) an association between *attacker* and *ACTIVEclient* is added, (iii) an association between *attacker* and *loginManager* is added, and (iv) direct association between the *ACTIVEclient* and *loginManager* is deleted because all communications now go through the attacker class.

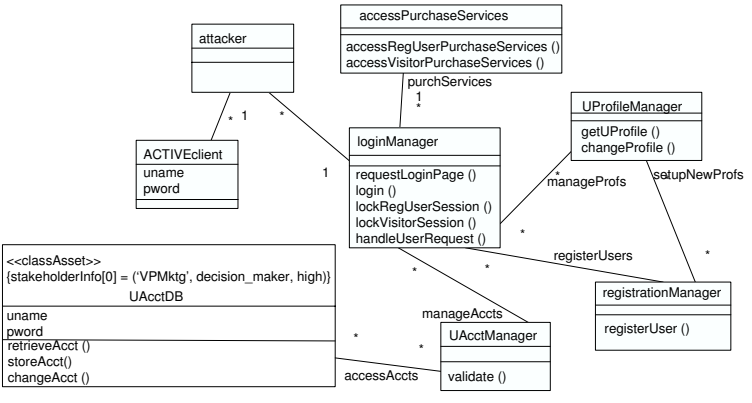

Fig. 3. Misuse model (primary model + man-in-the-middle attack) static diagram

The sequence diagrams describe the behavior of the primary model and the man-in-the-middle attack. The sequence diagrams must also be composed. The composition must be performed such that important properties of each model are preserved. Here again, we do not describe the mechanics of the composition process. The composed sequence diagram will serve to illustrate how much the primary model can be compromised.

The properties identified for the login service sequence that need to be preserved in this composition process are: (1) an application session is created, (2) users must be validated, (3) registered users receive a home page with profile information, and (4) unregistered users receive a visitor page. The properties that need to be preserved from the man-in-the-middle misuse are: (1) all messages from the client to the server through the duration of the session must pass through the attacker, (2) an authenticated session returned to the attacker indicates that the SLX service level has been achieved (3), no session returned to the attacker indicates that the SLX service level has not been achieved. The resulting composed sequence diagram is shown in Figure 4.

The main change in this sequence diagram from that given in Figure 2 is that an attacker lifeline has been inserted and all communication between the *ACTIVEclient* and

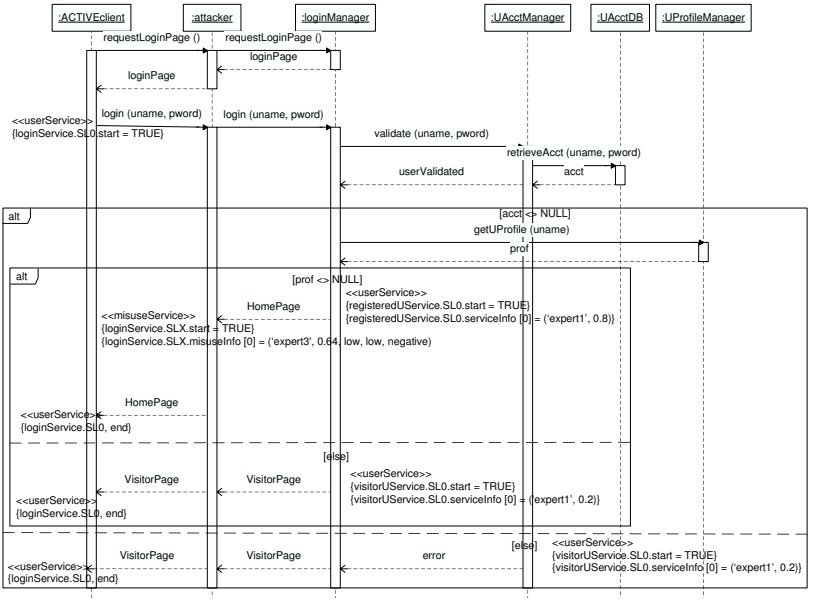

Fig. 4. Misuse model (primary + man-in-the-middle) sequence diagram

the *loginManager* go through *attacker*. The other change is in the probability associated with the SLX service level of the login service. This probability has been changed to reflect that the original probability is now included in an alternative sequence whose probability is 0.8 (the probability that the user profile exists). The value is calculated as part of the composition, by multiplying the outermost probability by the inner probability to obtain the new value of the inner probability.

3.2 Evaluating the Impact of Attack on the Application

The misuse model must be analyzed to determine the impact the attack can have on the primary model. The login service composed with the man-in-the-middle attack thus contains some properties that are undesirable. Paramount is the achievement of the SLX service level. The presence of the SLX service level means that some user service has been made available to persons not authorized to use it. Specifically in this example, an attacker gains knowledge of the user account login information, *uname* and *pword*. The class containing these items has been tagged as an asset in the primary static diagram, with a value that is "extremely_high". Once the *HomePage* message is returned to the attacker, the value of this asset has been decreased, as is indicated by the RDD tag stating that the impact on asset value is "negative". The ability of the attacker to extract these secrets can be formally analysed using tool support of the formal security analysis techniques developed by Jürjens [17]. To counter this attack, some security mechanisms must be incorporated with the application. The mechanism that we choose is TLS Authentication that is described next. We chose to use TLS since it is a follow-on to SSL (Secure Sockets Layer), which is a commonly available authentication mechanism used

in web applications. Other mechanisms could also be used to provide a stronger authentication mechanim for the application.

4 Incorporating TLS Authentication in the Application

The security properties of integrity and confidentiality are compromised with the man-in-the-middle attack, so mechanisms that address integrity and confidentiality are potential risk treatments. We demonstrate the use of transport layer security (TLS) [10] to mitigate the man-in-the-middle attack risk. TLS is based on passing certificates between a client and server for authentication purposes, and to establish secret session keys for the encryption of all subsequent messages. In this paper, we use the version of TLS proposed by Jürjens [17]. The sequence of the TLS mechanism is shown as a generic aspect diagram in Figure 5.

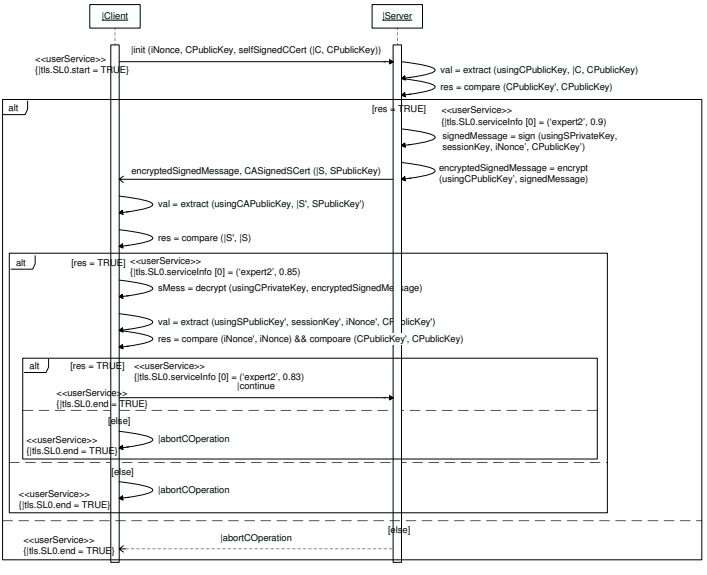

Fig. 5. Generic aspect of TLS mechanism sequence diagram

The TLS generic aspect contains two main classes: |*Client* and |*Server*. For this example, certificate creation and certificate authority public keys are assumed to be obtained in a secure manner. The client must have the certificate authority's public key, and the server must have a certificate, signed by the certificate authority (CA), of its name and public key. The notation in Figure 5 includes the concept of sent and received values, using a primed (') sent value name to indicate a value that has been received. Other assumptions include the fact that both nonces (unique identifier numbers) and session keys must change each time the protocol is initiated.

A TLS sequence begins with |*Client* sending an *init* message that contains a nonce (*iNonce*), its public key (*CPublicKey*), and a self-signed certificate containing its name

and its public key (*selfSignedCCert(|C, CPublicKey)*). When *|Server* receives this message, it extracts the client name and public key using the client public key sent in the message (shown as *extract(usingCPublicKey, |C, CPublicKey)* in Figure 5). It checks to make sure that the public key in the signed portion of the message is the same as the public key sent in the unsigned portion of the message. If not, the entire operation is aborted.

If the client public keys match, the server creates a message containing the session key that needs to be used for encryption once the connection is complete, the nonce received in the original client message, and the client public key. This message is then signed using the server's private key. This signed message is then encrypted using the client's public key. The result, along with the server's certificate (signed by a trusted certificate authority) is sent to the client in a respond message. This is message labeled *encryptedSignedMessage, CASignedSCert*(*S*,*SPublicKey*) in Figure 5.

The client first extracts the server name and public key using the certificate authority's public key. If the name of the server in the message (*|S′*) matches the name of the server (*|S*) to which the original init message was sent, the protocol proceeds. Otherwise the client aborts the operation. The encrypted portion of the message is decrypted using the client private key (*CPrivateKey*), and the items in the resulting signed message are extracted using the server's public key. The received nonce value (*iNonce*) in the signed message is compared to the nonce originally sent by the client (*iNonce′*), and the client public key (*CPublicKey)* in the signed message is compared to the client's public key (*CPublicKey'*). If either of these items does not match, this indicates that an attack on the communication has occurred, and the client aborts the operation. If the items match, then the communication path is secure, and the client can encrypt its secrets using the session key and transmit them to the server.

4.1 Generating the Security Treated Primary Model

The sequence diagram in Figure 5 can be composed with the e-commerce sequence diagram in Figure 2 in order to add TLS capabilities to the e-commerce system. Similarly, the static portion of the aspect model can be composed with the static diagram of the login service, although the result of this composition is not discussed in this paper.

To compose the sequence diagrams, we use the same method as we used to compose the primary sequence with the man-in-the-middle attack sequence. The TLS aspect, specified in template form, must be instantiated for the e-commerce application. This instantiation is done with the following bindings: (i) *|Client* in TLS is bound to *ACTIVEclient* in the e-commerce application, and (ii) *|Server* is bound to *loginManager*.

Properties in the login service sequence and in the TLS sequence are identified, and the properties that need to be preserved in the composed sequence are also identified. The resulting composed sequence diagram is shown in Figure 6.

The sequence shown in Figure 6 begins as the sequence did in Figure 2, with the *ACTIVEclient* requesting a login page from the *loginManager*. The *loginManager* responds with *loginPage*. Now the TLS sequence is inserted; instead of *ACTIVEclient* sending a login message with a user name (*uname*) and password (*pword*), a different login message is sent. This new login message contains a nonce (*iNonce*), the user's public key (*CPublicKey*), and a self-signed certificate containing the user name and user's public key (*selfSignedCCert(uname, CPublicKey)*). The logic for the TLS handshake

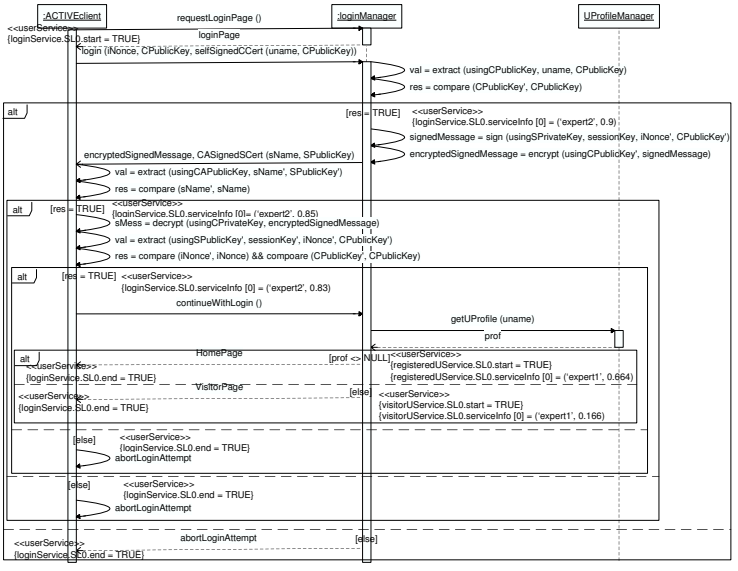

Fig. 6. Security treated model (primary model + TLS) sequence diagram

continues as in the TLS aspect model, with model element name changes per the bindings discussed above. Once the TLS handshake completes successfully, the *ACTIVEclient* sends a continue message to *loginManager*, which in turn causes the *loginManager* to get personal profile information (if it exists), and a *HomePage* is sent back to the user via *ACTIVEclient*. If the profile information does not exist, a *VisitorPage* is sent back to the user. Note that the probabilities of the *registeredUService* and *visitorUService* have been changed as was discussed in the previous composition section to reflect the probability that the third test is successful (0.83 multiplied by 0.8 and 0.2 respectively). We can informally argue that the properties identified for each model have been preserved in the composed model.

4.2 Analyzing the Security Treated Primary Model

Once the security mechanisms have been incorporated into the primary model, we need to verify whether the given attack is prevented in this new model. That is, we need to determine whether the TLS authentication adequately protects the application from the man-in-the-middle attack. We can reason about the effective security by composing the man-in-the-middle aspect with the security treated primary model.

The models are composed as before and the properties that need to be preserved in the security treated model are identified and used to create the composed sequence diagram.

Figure 7 shows the sequence when the man-in-the-middle attack is composed with the system protected by the TLS mechanism. We can reason informally about the composed sequence as follows. First, the properties identified as part of the composition are preserved in the composed sequence. Next, consider the *login* message parameters

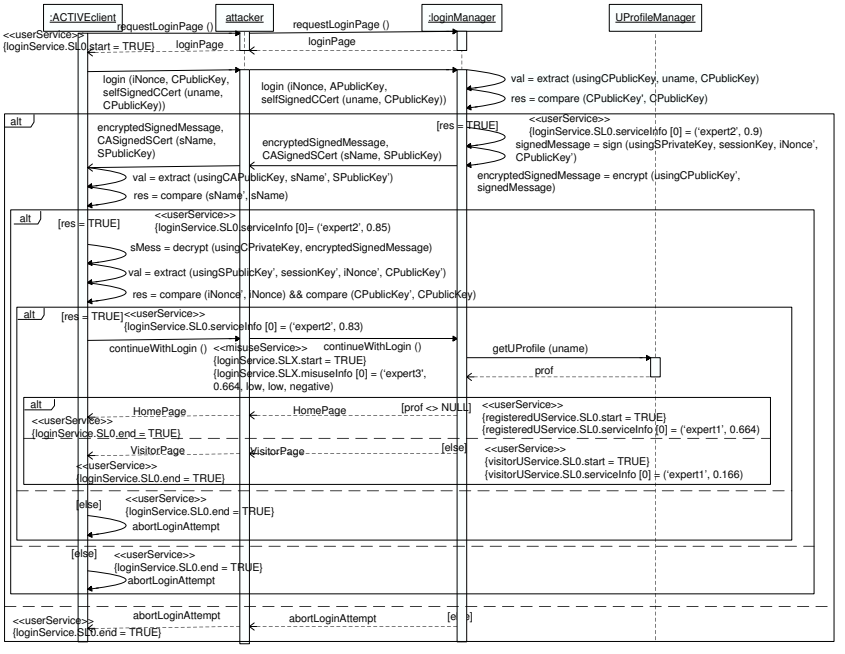

Fig. 7. Misuse model (security treated model + man-in-the-middle) sequence diagram

between the *ACTIVEclient* and *attacker* and between the *attacker* and *loginManager*. The *attacker* must replace the *ACTIVEclient* public key (*CPublicKey*) with the attacker's own public key. This must be done so that any messages from the *loginManager* that have been encrypted using the "client" public key are encrypted with the attacker's public key. This encryption means that the attacker can decrypt them. Since the attacker is posing as the *ACTIVEclient*, the client's certificate must be changed to include the client user name and the attacker public key. The result is that the login message parameters change to replace the client's public key with the attacker's public key, *APublicKey*. Once the *loginManager* receives this message, it uses the public key in the message to extract the name and public key in the certificate.

Since the public key in the message is the one used to encrypt the certificate, the first test comparison will work. Next the *loginManager* creates a signed message containing the attacker's public key, and encrypts it using that same public key. This message and the *loginManager's* CA-signed certificate is sent to the attacker, which decrypts the signed message with its private key. The signed message from the server is then encrypted with the *ACTIVEclient*'s public key, and is sent to the *ACTIVEclient*, along with the server's official certificate from the CA.

The *ACTIVEclient* first extracts the server name and public key from the CA certificate using the CA public key. A comparison is made between the server name the *ACTIVEclient* has and the server name in the certificate. This test will work. Next, the *ACTIVEclient* decrypts the signed message from the *loginManager* using its private key. It then compares the message nonce included in that message with the one it originally

sent, and the client public key included in that message with its own public key. This test will fail because the client key included in the signed message from the *loginManager* is that of the attacker. Therefore the sequence will always move to the third test failure alternative where the *abortLoginAttempt* message will be returned to the user of *ACTIVEclient* and the sequence ends. Thus, the treatment prevents the attack, and consequently the undesirable properties it allows, from occurring.

5 Related Work

Standards such as the ISO 15408 Common Criteria for Information Technology Security Evaluation [6] 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. They are also based on assessment by certified assessors. Trade-off techniques such as Architecture Trade-off Analysis Method (ATAM) [19] and Cost Benefit Analysis Method (CBAM) [18] operate at an architectural level, and also require experienced assessors. Any of these assessments require a strong resource commitment on the part of the organization that uses them. Risk identification, assessment, and management are the targets of the CCTA Risk Analysis and Management Methodology (CRAMM) [3] and CORAS [7,21] frameworks. CORAS makes use of multiple standards, including the Australian/New Zealand Standard for Risk Management [1], ISO/IEC 17799-1: Code of Practice for Information Security Management [13] and ISO/IEC 13335: Information technology – guidelines for management of IT security [14]. CORAS adapts the asset-driven structured approach in CRAMM, and uses model-based risk assessment in integrated system development processes. Our Aspect-Oriented Risk Driven Development (AORDD) framework [12,11] makes use of the CORAS processes and the asset-driven approach of CRAMM. The analysis described in this paper is a part of the AORDD framework. It is lightweight in that there is no need for a certified assessor, and it also provides information that is directed to a single risk treatment, rather than to an overall system. Unlike the process-targeted frameworks and standards, it deals with system operation.

The aspect-oriented techniques we use are part of our on-going AOM research, where aspects are UML templates that are instantiated in the context of a system prior to composition (see France et al. [8,9] and Straw et al. [22] for details on the AOM notations and composition). Jacobson [15,16] and Kiczales [20] describe AOM techniques that require that an aspect contains information regarding where and how it will be composed with a system model. Our generic aspects are free of this information and thus can be reused in multiple systems by instantiating them in different contexts. Clarke et al. [4,5] describe AOM composition techniques that augment or replace model elements and behavior. Our composition also allows elements and behavior to be deleted from a composition, or to be interleaved with other behavior and elements. The latter capability has been particularly useful in our AORDD work with security risk treatments.

6 Conclusion

In this paper, we propose a methodology for designing secure applications. We identify the assets in the application that need protection. We then find the kinds of attacks

that are typical for such applications, based on risk assessments that are beyond the scope of this paper. We show how to evaluate the application against such attacks. If the results of this evaluation indicate that the assets may be compromised, then some security mechanism must be incorporated into the application. Our focus is therefore on evaluating the ability of security mechanisms to protect against previously identified risks rather than on detecting new vulnerabilities. We illustrate how this can be done and show how the resulting system can be evaluated to give assurance that it is resilient to the given attack. A lot of work remains to be done. In this paper, all our analysis was done manually without any tool support. In future, we plan to investigate how this analysis can be automated to some extent. Specifically, we will look at how existing theorem-provers and model-checkers can aid this process.

Acknowledgements

This work was partially supported by AFOSR under Award No. FA9550-04-1-0102.

References

1. Australian/New Zealand Standards. AS/NZS 4360:2004 Risk Management, 2004.
2. Australian/New Zealand Standards. HB 436:2004 Risk Management Guidelines – Companion to AS/NZS 4360:2004, 2004.
3. B. Barber and J. Davey. The Use of the CCTA Risk Analysis and Management Methodology CRAMM in Health Information Systems. In K.C. Lun, P. Degoulet, T.E. Piemme, and O. Rienhoff, editors, *Proceedings of MEDINFO'92*, pages 1589–1593. North Holland Publishing Co, Amsterdam, 1992.
4. S. Clarke. Extending standard uml with model composition semantics. *Science of Computer Programming*, 44(1):71–100, 2002.
5. S. Clarke and E. Banaissad. *Aspect-oriented analysis and design*. Addison-Wesley Professional, Boston, 2005.
6. ISO 15408:1999 Common Criteria for Information Technology Security Evaluation. Version 2.1, CCIMB–99–031, CCIMB-99-032, CCIMB-99-033, August 1999.
7. CORAS (2000–2003). IST-2000-25031 CORAS: A Platform for risk analysis of security critical systems. http://sourceforge.net/projects/coras, Accessed 18 February 2006.
8. R. France, D.-K. Dim, S. Ghosh, and E. Song. A UML-based pattern specification technique. *IEEE Transactions on Software Engineering*, 3(30):193–206, 2004.
9. R. France, I. Ray, G. Georg, and S. Ghosh. Aspect–oriented approach to design modeling. *IEE Proceedings on Software*, 4(151):173–185, 2004.
10. TLS: Network Working Group. The TLS Protocol Version 1.0, RFC 2246, January 1999.
11. S. H. Houmb, G. Georg, R. France, J. Bieman, and J. Jürjens. Cost-benefit trade-off analysis using bbn for aspect-oriented risk-driven development. In *Proceedings of Tenth IEEE International Conference on Engineering of Complex Computer Systems (ICECCS 2005)*, pages 195–204, June 2005.
12. S.H. Houmb and G. Georg. The Aspect-Oriented Risk-Driven Development (AORDD) Framework. In O. Benediktsson et al., editor, *Proceedings of the International Conference on Software Development (SWDC–REX)*, volume SWDC–REX Conference Proceedings, pages 81–91, Reykjavik, Iceland, 2005. Gutenberg.
13. International Organization for Standardization (ISO/IEC). ISO/IEC 17799:2000 Information technology – Code of Practice for information security management, 2000.

14. International Organization for Standardization (ISO/IEC). ISO/IEC TR 13335:2001 Information technology – Guidelines for management of IT Security, 2001.
15. I. Jacobson. Case for aspects – Part I. *Software Development Magazine*, pages 32–37, October 2003.
16. I. Jacobson. Case for aspects – Part II. *Software Development Magazine*, pages 42–48, November 2003.
17. J. Jürjens. *Secure Systems Development with UML.* Springer, Berlin Heidelberg, New York, 2005.
18. R. Kasman, J. Asundi, and M. Klein. Making architecture design decisions: an economic approachn. Technical report CMU/SEI-2002-TR-035, CMU/SEI, http://www.sei.cmu.edu/pub/documents/02.reorts/pdf/02tr03.pdf, 2002.
19. R. Kazman, M. Klein, and P. Clements. Atam: method for architecture evaluation. Technical report CMU/SEI-2000-TR-004, CMU/SEI, http://www.sei.cmu.edu/pub/document/00.reports/pdf/00tr004.pdf, 2000.
20. G. Kiczales, E. Hilsdale, J. Hugunin, M. Kersten, J. Palm, and W. Griswold. Getting stared with aspectj. *Communications of the ACM*, 10(44):59–65, 2001.
21. K. Stølen, F. den Braber, T. Dimitrakos, R. Fredriksen, B. A. Gran, S. H. Houmb, Y. C. Stamatiou, and J. Ø. Aagedal. Model–based risk assessment in a component-based software engineering process: The CORAS approach to identify security risks. In Franck Barbier, editor, *Business Component-Based Software Engineering*, pages 189–207. Kluwer, 2002. ISBN: 1–4020–7207–4.
22. G. Straw, G. Georg, E. Song, S. Ghosh, R. France, and J. Bieman. Model composition directives. In T. Baar, A. Strohmeier, A. Moreira, and S Mellor, editors, *UML*, volume 3273 of *Lecture Notes in Computer Science*, pages 84–97. Springer, 2004.