Middleware Support for Complex and Distributed Security Services in Multi-tier Web Applications

Philippe De Ryck, Lieven Desmet, and Wouter Joosen

IBBT-DistriNet
Katholieke Universiteit Leuven
3001 Leuven, Belgium
{philippe.deryck,lieven.desmet}@cs.kuleuven.be

Abstract. The security requirements of complex multi-tier web applications have shifted from simple localized needs, such as authentication or authorization, to physically distributed but actually aggregated services, such as end-to-end data protection, non-repudiation or patient consent management. Currently, there is no support for integrating complex security services in web architectures, nor are approaches from other architectural models easily portable. In this paper we present the architecture of a security middleware, aimed at providing a reusable solution bringing support for complex security requirements into the application architecture, while addressing typical web architecture challenges, such as the tiered model or the lack of sophisticated client-side logic. We both evaluate the security of the middleware and present a case study and prototype implementation, which show how the complexities of a web architecture can be dealt with while limiting the integration effort.

Keywords: middleware, multi-tier architecture, security, web application, non-repudiation.

1 Introduction

The online availability of more and more everyday services has heightened the complexity of web applications as well as the sensitivity of their assets, such as customer data, financial data or medical information. This evolution has also affected the security requirements, which have shifted from simple localized needs, such as authentication or authorization, to physically distributed but actually aggregated services, such as end-to-end data protection, non-repudiation or patient consent management.

The simple security requirements have been addressed by support for authentication schemes and access control models, both in distributed application architectures [9,10] as in web architectures [2,4,13]. Support for achieving complex security requirements is limitedly available for distributed application architectures, either as a middleware solution (e.g. [5,22]) or as a complementary software product (e.g. [18]). For web architectures however, there is no specific support

Ú. Erlingsson, R. Wieringa, and N. Zannone (Eds.): ESSoS 2011, LNCS 6542, pp. 114–127, 2011.

for achieving complex security requirements, nor are the previously mentioned approaches easily portable to such architectures.

In this paper we present the architecture of a security middleware, aimed at providing a reusable solution bringing support for complex security requirements into the application architecture. The middleware offers a generic support layer, which supports both the implementation and integration of specific complex and distributed security services into a web architecture. We will show that the security middleware provides complete protection and is tamper-proof. Additionally, we conduct an extensive case study focused on a concrete security requirement, fair mutual non-repudiation[14,23], which can informally be defined as "the inability to subsequently deny an action or event"[5]. This case study illustrates the practical applicability of the middleware as well as the feasibility to achieve complex security requirements in an existing application.

The remainder of this paper is structured as follows. Section 2 introduces a motivating example and identifies the main challenges for supporting complex security requirements in web architectures. Section 3 presents the security middleware architecture and discusses implementation details using the non-repudiation case study as an example. Section 4 presents the implemented prototype and the evaluation of the middleware solution. We conclude the paper by discussing potential improvements and some suggestions for future developments (Section 5) and a summary of the contributions of this paper (Section 6).

2 Motivation and Background

We demonstrate the need for complex security requirements by means of an online banking system. The security requirements of such a system have evolved from ensuring basic security, such as secure communication or preventing unauthorized access, to complex security properties, for instance *non-repudiation for financial transactions*. Looking at the current level of non-repudiation for transactions in such applications, it seems that the non-repudiation is based on the successful execution of an authentication scheme, where even the very robust offline two-factor authentication schemes used in European banking systems are unable to provide non-repudiation guarantees for specific transactions. Basing non-repudiation on an authentication scheme is a flawed approach, since non-repudiation is based on evidence generated from business data, transactions in this case, where an authentication scheme is clearly not. To achieve non-repudiation for financial transactions in an online banking system, the business operations leading to the execution of a transaction need to be augmented with a non-repudiation security service. This service ensures the correct execution of a non-repudiation protocol that achieves fair mutual non-repudiation [14,23] between both participating parties, before the transaction is executed.

A different example are e-health systems, which are becoming increasingly popular and provide access to sensitive medical information, which needs to be shared among a patient, involved doctors, pharmacists, etc. As required by law [21], medical data can only be shared with a valid *patient consent*. Checking

patient consents when sharing data is an example of a complex security service, which requires extensive support in the application tier.

These examples already suggest some properties of different kinds of *complex security services*, such as achieving non-repudiation, patient consent management or end-to-end data protection. Compared to *simple security services*, such as an authentication or authorization service, a complex security service is tightly coupled to business functionality, such as the execution of financial transactions or accessing medical records. Additionally, most complex security services are also distributed, since there is an active involvement of the user and the server, which requires additional communication between the client and server. Examples of such interactive participation are the involvement in a non-repudiation protocol, the encryption of information or the creation of patient consents.

Support for simple security services has been integrated in distributed application architectures, such as CORBA [9,10], as well as in web architectures, such as JavaEE [4,13] or web services [17]. Complex security services on the other hand are not well supported, as will be discussed later in this section. Before discussing the existing support, we will first discuss several key challenges and requirements for integrating complex and distributed security services in web architectures.

2.1 Challenges for Complex Security Services in Web Architectures

When integrating complex security services in web architectures, the specific properties of this architectural model need to be taken into account. The web architecture is a tiered model, where the functionality is separated over four distinct tiers [20]. Figure 1 shows two different versions of the web architecture, with (a) being a full-fledged four tier web architecture such as JavaEE, mainly used in large enterprise applications, and (b) being a more lightweight version, where the web and application tier are located in one physical tier, but still remain logically separated from each other, as used by popular frameworks such as Spring and Struts.

Earlier work has already identified important issues with existing security technology, from which we can extract specific requirements for a security middleware. One of these requirements is compliance with the basic principles for

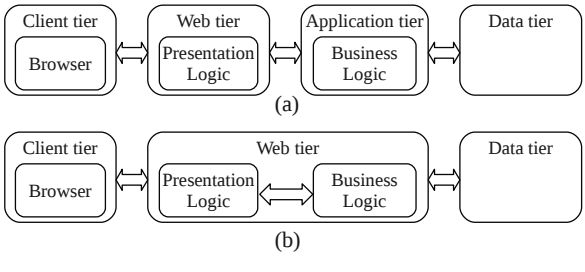


Fig. 1. (a) A typical enterprise web architecture with four distinct tiers. (b) A lightweight web architecture with business logic embedded in the web tier.

protection mechanisms [3,8], which state that (i) all paths to the functionality under protection need to be covered, (ii) the mechanism itself should be tamper-proof and (iii) the mechanism should be transparent to the application under protection, except for the specifically added security features. Apart from these security requirements, the middleware should be easy to integrate, without requiring intensive changes to the codebase of the application under protection, to ensure its practical applicability [2].

Next to these basic requirements, web architectures pose three interesting design challenges, that will need to be taken into account:

C1. Coupling with business functionality: to be able to augment the protection of business operations, the complex security service needs to be tightly coupled with the business functionality. For example to achieve non-repudiation on financial transactions, the security service will have to be coupled to the operation that executes a financial transaction.

C2. Request/response model: client/server communication in a web architecture is a uni-directional request/response model. This means that a message to the client can only be sent as a response to an earlier request. Additionally, a business transaction can consist of multiple request/response pairs.

C3. User involvement: a complex security service is typically distributed, where an active involvement of the user is required. This involvement requires both the ability to execute client-side code and to execute an out-of-band communication protocol, such as a non-repudiation protocol or the transfer of encrypted data.

Next to the middleware requirements and design challenges, the implementation of middleware for complex security services also poses specific challenges. One of them is the need for an interoperable representation of service-specific data, a lesson learned from earlier approaches [2]. Additionally, the correct implementation of a security protocol, if any is used, is truly important, since a vulnerability in such a protocol can render the entire security service useless.

2.2 Support for Complex Security Services

Generic support for complex and distributed security services is very limited to non-existent, in all types of architectures, which is why a complex security service is often developed from scratch, as part of an application. One observation is that the existing support is mainly suited for a machine-to-machine environment, without much user-involvement. The remainder of this section will discuss these application-extrinsic approaches to support complex security services. These approaches are mainly limited to one specific security service, non-repudiation, but will be discussed with the goal of leveraging them to support complex and distributed security services in a web architecture.

One of the earliest approaches is the CORBAsec specification [9], which addresses complex security services in the third level of the specification, by proposing an API for non-repudiation. Several challenges with the CORBAsec

specification [2], such as transparency and a fixed evidence format, were addressed by Wichert et al. [22], who propose a generic transparent CORBA non-repudiation service, based on code annotations. Continued work by Cook [5] has led to a middleware solution to achieve mutual non-repudiation in a business-to-business context, on top of the JavaEE platform. The middleware supports transparent integration by using a container service of the platform, which is available both at the client and at the server side. The last approach, which is based on the approach of Wichert, supports non-repudiation in an effective and secure manner, and can even be extended to other complex security services. It is however not directly applicable in a web architecture, due to the dependence on container services at the client-side, but it is nonetheless inspiring.

Support for complex security services is also available in other contexts, albeit for machine-to-machine communication. One example in the field of web services is a transparent non-repudiation service [1], which executes a non-repudiation protocol in the background. This service can also be used for similar security services, which depend on a protocol execution. Another approach for web services is attaching a non-repudiation log to messages [19], a technique less suitable to support other complex and distributed security services, since it is not aimed at strong interactivity between client and server. A totally different approach is the use of a standardized security interface, as provided by GSS-API [15], which is aimed at achieving robust interoperability for implementations of a security service. Although the specification of GSS-API is rather limited to origin authentication, integrity and confidentiality, it could be extended to support more complex security services as well.

The discussed approaches towards supporting complex security services are not directly applicable in a web architecture, but provide sufficient inspiration. Especially the work of Cook, which provides non-repudiation in a business-to-business context, presents an interesting way to integrate a complex security service on the server-side. When trying to adapt the discussed approaches to fit in a web architecture, the previously identified problems and challenges reoccur.

3 Middleware Support

The goal of the security middleware is to augment the protection of the business logic of a web application, by providing an easy way to integrate a complex and distributed security service in the application. This can be achieved by means of a security middleware, which can be integrated with the architecture of the application under protection and addresses the specific web architecture challenges. Figure 2 provides a high-level overview of the tiered web architecture, where the white blocks represent the application under protection. The flow in such an application always starts from the client, based on the request/response model, which sends a request to the web tier. The web tier processes the request and will eventually invoke a business operation, located in the business tier. The business logic processes the actual request, potentially using persistent data from the data tier, and returns a response to the web tier, which sends a response to the client.

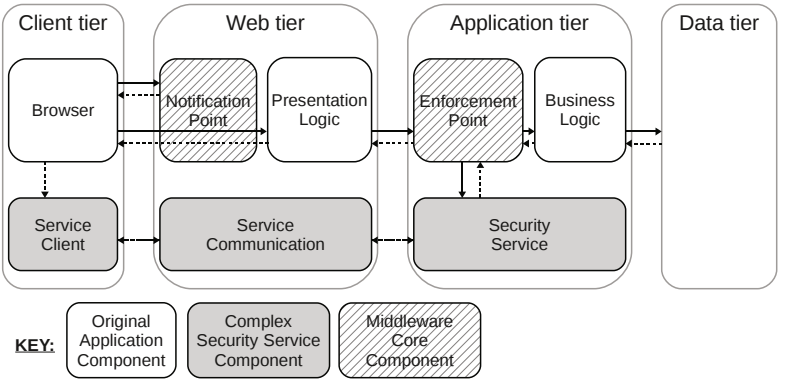


Fig. 2. The high-level architecture of the security middleware in a four tier web architecture

The first challenge for integrating a complex security service in a web architecture is achieving tight coupling with the business functionality of the application under protection. This can be achieved by integrating the security service into the application tier, similar to the approach of Cook [5]. Figure 2 shows how the *enforcement point* effectively couples the *security service* to the *business logic*.

Involving the user in the complex and distributed security service poses a second major challenge. Solving this challenge means both the ability to execute client-side code and the ability to communicate with the security service at the application tier. The former is represented by the *service client* in Figure 2 and can be achieved using locally installed software, such as a browser plugin, or remotely downloaded code, such as JavaScript code, a Java Applet, Flash code, etc. Addressing the latter requires security service-specific out-of-band communication. Direct communication between the client and application tier is not possible, due to the architectural model as well as the underlying infrastructure, which contains perimeter firewalls and web application firewalls. Therefore, we have chosen to integrate the out-of-band communication with the normal communication channel through the web tier, as shown by the *service communication* component in Figure 2.

In the remainder of this section, we will discuss the design details of the security middleware, applied to the non-repudiation case study which has been performed. In this case study, the support for complex and distributed security services is used to integrate non-repudiation for financial transactions in an online banking application [6]. For the details of the concept of non-repudiation, which are not really relevant for this paper, we refer to the literature [5,14,23]. Even though the discussion is focused around non-repudiation to avoid an abstract explanation of how the middleware works, other complex security services, such as end-to-end data protection or patient consent management, are also supported.

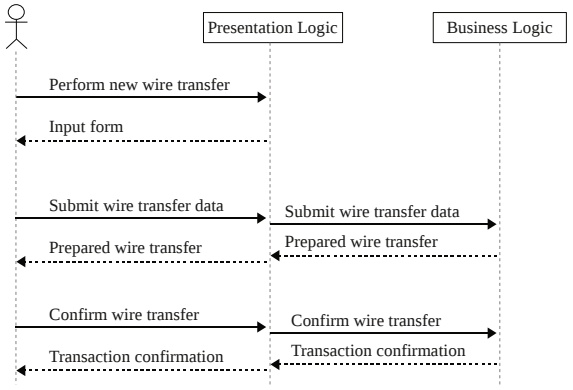
3.1 Detailed Design Based on a Non-repudiation Case Study

Figure 3 (a) shows the operations that lead to the execution of a financial transaction. Figure 3 (b) illustrates the changes introduced by the integration of a complex security service, a process which will be discussed next. To achieve non-repudiation for financial transactions, the non-repudiation service needs to be coupled with the "confirm wire transfer" operation. The coupling is achieved on the level of method invocations using interceptor techniques [7] in the enforcement point, similar to the approach taken by Cook [5]. The enforcement point knows which operations need to be protected by means of an application-specific configuration file. Apart from coupling the security service to a business operation, the enforcement point addresses the need of the security service to have access to extensive business information. For instance, the non-repudiation service needs details about the financial transaction and the user within the system, to construct a plaintext, which is the basis for the non-repudiation evidence.

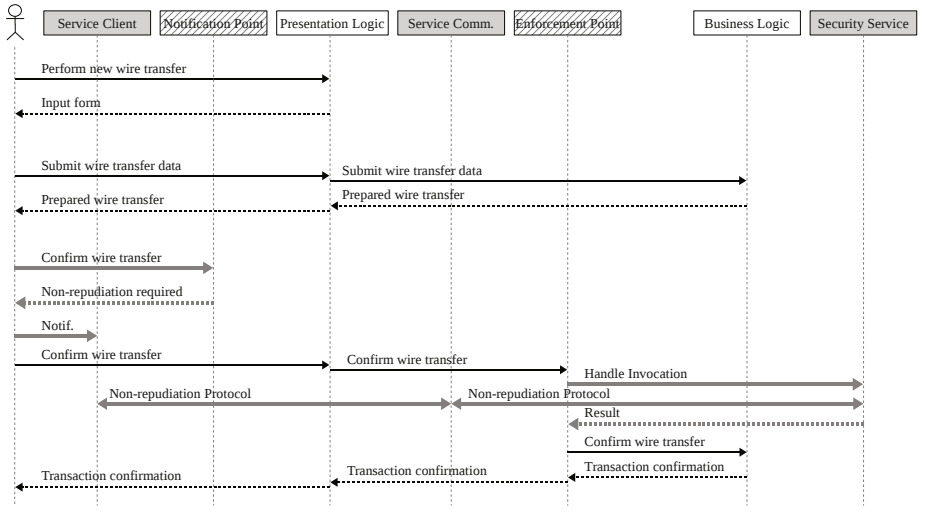
A complex and distributed security service requires client-side support and user participation, which is not available in a web browser. Providing this client-side support can be done using a locally installed piece of software, a very impractical approach due to deployment and management issues. A better alternative is the use of remotely downloaded code, of which multiple forms are widely supported (JavaScript, Java, Flash, etc.). The prototype implementation uses a Java Applet, since Java offers strong smartcard support, which the prototype uses for identity management and cryptographic operations.

The initiation of a complex security service is not trivial. Following the request/response model of a web architecture, the client should initiate the communication process, but does not know when non-repudiation is required. The enforcement point does know this, but there is no way for the enforcement point or security service to send a message to the client tier, without potentially causing unwanted effects in the presentation logic (e.g. triggering generic exception handling mechanisms). This problem has been addressed by introducing the *notification point*, which is configured to know which incoming requests will lead to the invocation of a protected business operation. It intercepts such requests and responds to the client with a security notification. Receiving such a notification informs the client of the non-repudiation requirement, ensures that the client-side applet is loaded and automatically re-sends the original web request through the web tier. The notification point recognizes this repeated request and allows it to be handled by the presentation logic, from where it will result in the invocation of a protected business operation, which is handled by the enforcement point. In the mean time at the client-side, the security notification has been passed on to the applet, which starts executing the complex security service independently from the containing page.

The final part of the design is the realization of an out-of-band communication channel, which has to pass through the web tier, in order not to violate the tiered model of the web architecture. Therefore a communication component, which converts incoming web requests to method invocations for the security service and does the reverse with the responses, is provided by the web tier. The service



(a) Original application



(b) Application with an integrated non-repudiation service

Fig. 3. Sequence diagram showing the execution of a financial transaction

client can communicate directly with the web tier using Java Sockets if desired, but can also re-use the already established communication channel between the browser and the web tier, using a JavaScript communication mechanism provided by the page. The latter approach has the advantage that the security features of the existing communication channel, such as SSL protection, are also available for the security service-specific communication.

The discussion of the middleware in the non-repudiation case study shows how specific challenges within a web architecture can be addressed, but does not explain how other complex and distributed security services can be supported. Figure 4 shows the internal design of the security middleware, which clearly shows the generic middleware layer, aimed at supporting multiple complex security services, as well as the non-repudiation-specific layers.

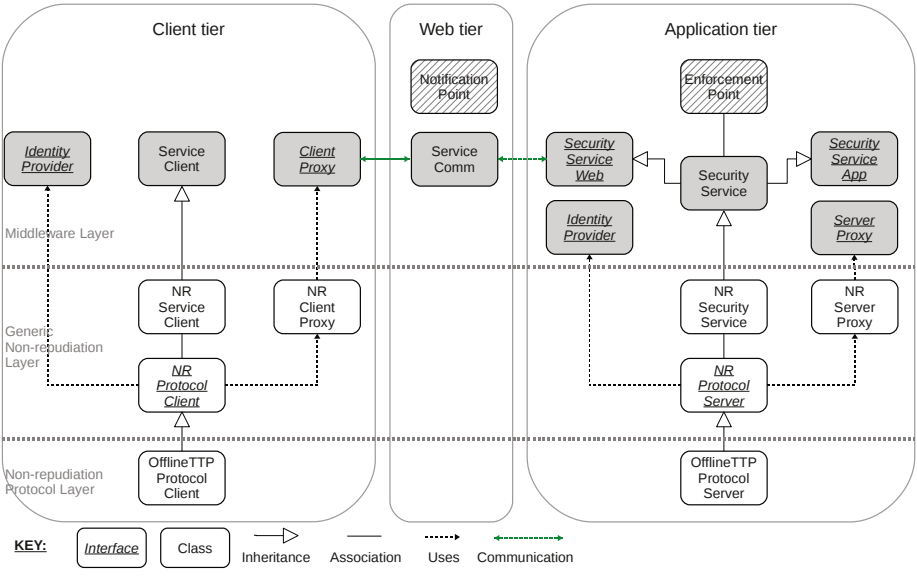


Fig. 4. The detailed design of the security middleware and a specific complex security service (non-repudiation)

The concrete implementation of complex and distributed security services on top of the middleware layer can be achieved by providing customized operations in the *service client* and *security service*. Using security service-specific configuration files, the service can be configured to fit the application requirements. The middleware layer provides abstractions for the concrete details of integration in the application, which allows the easy and rapid implementation of a complex security service, such as patient consent management or end-to-end data protection. An example of this are the generic and configurable enforcement point and notification point. Both the client and server-side offer an *identity provider*, which simplifies the management of identity information, such as certificates, smartcards, etc. The *client proxy* at the client side provides abstractions for communication with the server or with external parties. The server-side equivalent is the *server proxy*, which simplifies communication with external parties.

4 Prototype and Evaluation

This section presents the evaluation of the middleware. The actual integration of a prototype of the middleware with a prototype of a banking application shows the practical applicability as well as the configuration effort. Additionally, we discuss how we have addressed the basic requirements and design or implementation challenges. Finally, we also present the results of a security analysis, indicating the security of the security middleware.

4.1 Prototype Implementation and Configuration

We have implemented the case study of the online banking application, with the requirement of achieving non-repudiation for financial transactions. The banking system is designed and implemented independently from the security middleware, to assess the practicality of the integration process. Although the implemented prototype offers a subset of the designed functionality, it does include important security features such as strong authentication and authorization. The business logic of the partial implementation of the banking application counts 3514 lines of code. Next to the banking application, we have also developed a prototype of the security middleware and the non-repudiation security service. The generic middleware layer counts 3219 lines of code, the non-repudiation support layer counts 1407 lines and the specific protocol implementation, in the form of state machines, counts 4294 lines. Detailed design information can be found in [6] and the source code is available for download[1].

Using the developed prototypes, we have configured the non-repudiation service to be integrated with the banking application, as previously shown in Figure 3. The configuration and integration process is purely declarative, by means of XML configuration files, which allows the middleware to be integrated, without changing the original application code. The amount of configuration instructions to integrate the middleware is dependent on the number of operations that need to be protected. In the non-repudiation case study, we protected two sensitive business operations (executing financial transactions and creating new customers), which resulted in the following configuration cost: 57 XML elements[2] have been created and 131 source lines of code (Java) have been written. This last cost is to be attributed to the retrieval of business information, which forms the plaintext of the non-repudiation protocol.

4.2 Analysis of the Security Middleware

Next to the practical applicability, addressing the design challenges is an important aspect of the security middleware. The principal design challenges, more specifically (a) coupling with business functionality, (b) the request/response model and (c) user involvement have been addressed in the design of the middleware, as discussed in the previous section. Service-specific data is represented using an XML format, which is open and easily readable. A detailed protocol analysis is out of scope for this paper, but the implementation of the non-repudiation protocol has been achieved using a careful conversion to state machines, which have literally been implemented.

The security properties of the middleware, as well as the effectiveness of the provided protection are verified using a detailed security analysis using STRIDE [11]. The security analysis examines the claim that *protected business operations will only be executed if the security service has been contacted and has approved*

[1] `http://people.cs.kuleuven.be/philippe.deryck/papers/essos2011/src.zip`

[2] The XML element count does not include structural elements, only elements with application-specific values.

the pending execution. This means that if the security service is not satisfied and signals a problem, the business operation will not be executed. The details of a specific security service are regarded as a black box in the security analysis, because the middleware supports multiple different security services.

The claim will be investigated for a specific attacker profile: the attacker can carry out active attacks on information that passes the trust boundaries of the system. This includes messages sent between client and server, but also messages sent between two client components, or messages between the server and another party. Internal attacks on the server infrastructure or components will not be considered (e.g. spoofing attacks between the web and application tier). The analysis also makes certain environment assumptions, such as absence of business logic flaws in the application under protection, secure communication channels, a trusted codebase for client and server platform and reliable certificate validation using the Online Certificate Status Protocol (OCSP) [16].

After analyzing the middleware solution and carefully applying STRIDE to all elements, we have concluded that the only way to have a protected business operation executed is by getting a positive answer from the security service. This positive answer can only be obtained if the security service has fulfilled the required security properties. Concretely for the non-repudiation security service, this can only happen if the non-repudiation protocol ends in the *SUCCESS* state, which ensures the possession of the required evidence. Any attempt to tamper with the security service-specific data, such as the communication messages or identity information, will be detected before the security service gives its answer. This includes the invocation of the servlet using potentially compromised[3] JavaScript operations. Therefore, we can conclude that a protected business operation can only be executed if the security service has been executed successfully, for instance if both parties possess the appropriate non-repudiation evidence.

5 Discussion

In this section we discuss potential improvements and some suggestions for future developments. These include a discussion of lack of support for transactions, a description of how the security middleware can be implemented for lightweight web architectures, potential automation using tool support and a future approach by integrating the support in the application platforms.

Transactional Support. One problem with invoking a security service before the method invocation takes place, is that it might provide certain security properties about the invocation, but says nothing about the result. A simple approach is to make the security service provide protection for the response too [5]. The problem with this approach lies in dealing with malicious clients or

[3] Using XSS attacks in the application under protection, the JavaScript code of the security middleware could be compromised, but can not be used to subvert the server-side security service using falsified evidence.

communication failures. What if the client refuses to execute a non-repudiation protocol for the response? Can the operation be undone?

The appropriate solution for this problem involves the support for transactions, using a transaction service, which supports a rollback operation in case the security service fails on the response of the invocation. Sensitive operations desiring this behavior should be implemented with transaction support, or should be modified to do so.

Lightweight Web Architectures. As discussed in the introduction, not all web architectures follow the four tier architecture. For web architectures with only three tiers, the use of the Spring framework is very popular, even for more complex business systems. The conversion of the EJB implementation to a Spring implementation is straightforward. The only difference is the implementation of the enforcement point, which is no longer located in the application tier. The enforcement point can be implemented as a bean in the web tier, and will operate on business operations, which still are methods of Java objects. The interception mechanism is changed from EJB Interceptors to Spring Method Interceptors [12].

Tool Support. The integration process can be greatly simplified using automated tool support. The automation of the application-independent integration is straightforward, and the application-dependent integration can be partially supported as well. By means of a tool, the integrator can select the business operations that need to be protected and automatically configure the servlets requiring a notification message. The integration process is further supported by aiding with the configuration of the business data required by the security service.

Native Support. The current implementation has been developed to be compatible with current browser and server technology, as is the de facto standard for web application solutions. In that light, some of the proposed solutions can be simplified by evolving towards built-in support for complex security services. An important improvement to modern browser technology, would be the possibility to execute complex security protocols, such as non-repudiation, directly from the browser, much like authentication now. A second improvement is possible at the server-side, where application servers can provide context-spanning services, that allow the application tier to simply notify a client of a specific security requirement, without the presentation logic of the web tier having to deal with it. These two functionality improvements can simplify the design of the middleware to the *enforcement point* and the *security service*.

6 Conclusion

Based on the observation that there is currently no support for integrating complex and distributed security services into a multi-tier web application and that approaches in other architectures are not readily portable to a web architecture, we have analyzed what challenges need to be overcome to provide such support.

We discussed the architectural and detailed design of a security middleware which provides this support, while addressing the aforementioned challenges. We have implemented the security middleware and used a practical evaluation approach, based on a case study of an online banking system. In this evaluation, we have shown that the middleware can be used to integrate complex security services into a multi-tier web application, while providing strong security guarantees.

Acknowledgements. This research is partially funded by the Interuniversity Attraction Poles Programme Belgian State, Belgian Science Policy, IBBT, IWT and the Research Fund K.U. Leuven.

References

1. Agreiter, B., Hafner, M., Breu, R.: A fair non-repudiation service in a web services peer-to-peer environment. Computer Standards & Interfaces 30(6), 372–378 (2008)
2. Alireza, A., Lang, U., Padelis, M., Schreiner, R., Schumacher, M.: The challenges of corba security, pp. 61–72 (2000)
3. Anderson, J.P.: Computer security technology planning study volume ii. Technical report, Electronic Systems Division, Air Force Systems Command, Hanscom Field, Bredford, MA (October 1972)
4. Ball, J., Carson, D.B., Evans, I., Haase, K., Jendrock, E.: The java ee 5 tutorial. Sun Microsystems, Santa Clara (2006)
5. Cook, N.: Middleware Support for Non-repudiable Business-to-Business Interactions. PhD thesis, School of Computing Science, Newcastle University (2006)
6. Debie, E., De Ryck, P.: Non-repudiation middleware for web-based architectures. MSc thesis, Katholieke Universiteit Leuven (2009)
7. DeMichiel, L., Keith, M.: Enterprise javabeans, version 3.0. Sun Microsystems (2006)
8. Erlingsson, U., Schneider, F.: Irm enforcement of java stack inspection. In: IEEE Symposium on Security and Privacy, pp. 246–255 (2000)
9. Object Management Group. Security service specification v1.8 (March 2002)
10. Object Management Group. Corba specification (January 2008), `http://www.omg.org/spec/CORBA/3.1/`
11. Howard, M., Lipner, S.: The Security Development Lifecycle. Microsoft Press, Redmond (2006)
12. Johnson, R., et al.: Spring java application framework - reference documentation (2009)
13. Koved, L., Nadalin, A., Nagaratnam, N., Pistoia, M., Shrader, T.: Security challenges for enterprise java in an e-business environment. IBM Systems Journal 40(1), 130–152 (2001)
14. Kremer, S., Markowitch, O., Zhou, J.: An intensive survey of fair non-repudiation protocols. Computer Communications 25(17), 1606–1621 (2002)
15. Linn, J.: Rfc2743: Generic security service application program interface version 2, update 1. RFC Editor United States (2000)
16. Myers, M., Ankney, R., Malpani, A., Galperin, S., Adams, C.: Rfc2560: X. 509 internet public key infrastructure online certificate status protocol-ocsp (1999)
17. Nadalin, A., Kaler, C., Hallam-Baker, P., Monzillo, R., et al.: Web services security: Soap message security 1.0 (ws-security 2004). OASIS Standard, 200401 (2004)

18. Nenadic, A., Zhang, N., Barton, S.: Fides–a middleware e-commerce security solution. In: Proceedings of the 3rd European Conference on Information Warfare and Security, pp. 295–304 (2004)
19. Parkin, S., Ingham, D., Morgan, G.: A message oriented middleware solution enabling non-repudiation evidence generation for reliable web services. In: Malek, M., Reitenspieß, M., van Moorsel, A. (eds.) ISAS 2007. LNCS, vol. 4526, pp. 9–19. Springer, Heidelberg (2007)
20. Singh, I., Johnson, M., Stearns, B.: Designing enterprise applications with the J2EE platform. Addison-Wesley Professional, Reading (2002)
21. Tribble, D.A.: The health insurance portability and accountability act: security and privacy requirements. American Journal of Health-System Pharmacy 58(9), 763 (2001)
22. Wichert, M., Ingham, D., Caughey, S.: Non-repudiation evidence generation for corba using xml (1999)
23. Zhou, J., Gollmann, D.: Evidence and non-repudiation. Journal of Network and Computer Applications (1997)