CHALMERS

Achieving Authentication and Authorization in the Combine System

Master of Science Thesis in the Master Degree Programme,

Software Engineering and Technology

ANDREAS LÖWDELL

JOHAN SAKBAS

Department of Computer Science and Engineering
Division of Software Engineering and Technology
CHALMERS UNIVERSITY OF TECHNOLOGY
Göteborg, Sweden, May 2009

Achieving Authentication and Authorization in the Combine System

ANDREAS LÖWDELL
JOHAN SAKBAS

Examiner: JOACHIM VON HACHT

Department of Computer Science and Engineering
Chalmers University of Technology
SE-412 96 Göteborg
Sweden
Telephone + 46 (0)31-772 1000

Department of Computer Science and Engineering
Göteborg, Sweden, May 2009

Abstract

This thesis has aimed to design and implement a prototype for the authentication and authorization parts of the Pulsen Combine system. The system will be used by the social service agencies in several communes of Sweden. A study was carried out regarding the authentication part in Combine, whether it would be developed in-house or bought as a standalone product. The study resulted in the choice of a third party solution that will handle the authentication complexity. The authorization part was designed and further on implemented into several layers of the Combine system, adhering to well-known security principles.

Keywords: Authentication, Authorization, Security principles, Service Oriented Applications

Sammanfattning

Detta examensarbete har haft som mål att designa och implementera en prototyp av autentiserings- och auktoriseringsdelarna i Pulsen Combine. Systemet kommer att användas av socialtjänsten i ett flertal svenska kommuner. En studie, som behandlade huruvida autentiseringsdelen i Combine skulle utvecklas internt eller om den skulle köpas in i form av en fristående produkt, genomfördes. Studien resulterade i valet av en tredjepartslösning som kommer att hantera komplexiteten kring autentiseringen i Combine. Auktoriseringsdelen designades först och implementerades sedermera, i flera olika lager av Combine, med välkända säkerhetsprinciper i åtanke.

Nyckelord: Autentisering, Auktorisering, Säkerhetsprinciper, Serviceorienterade applikationer

Acknowledgements

We would not have been able to complete this thesis without the support and help we received from the people around us.

Yvonne Liberg and Ulf Lerge, among many others at Pulsen Application in Borås, have helped us in many ways during the lifetime of the thesis. They have supplied us with much of what we have needed, both practically and in terms of valuable knowledge.

Our supervisor and examiner at Chalmers University of Technology, Joachim von Hacht, has been a great help to us. He has shown us a big interest, and has committed to making sure that this thesis project was completed in a successful manner.

Last but not least, our families have supported us and been there for us during the whole thesis.

We would like to thank you all!

Table of Contents

1 Abbreviations

XML	eXtensible Markup Language
SSO	Single Sign-On
WCF	Windows Communication Foundation
SAML	Security Assertion Markup Language
LDAP	Lightweight Directory Access Protocol
E-ID	Electronic Identification
BIF	Bastjänster för InformationsFörsörjning (Swedish), Base services for information sustention

2 Introduction

Pulsen Combine[1] (Combine) [1] is the name of a yet to be finished web based social service system. The customers are several Swedish communes. Combine is going to support the complete work process for the employees in the communes. Citizens are also going to use the system to handle social service related errands and communicate with the employees in the communes. External performers, which accomplish tasks for communes, will use Combine too. The system hereby contains three separate parts – the *Public* application for citizens, the *Local* application for employees in communes and the *Performer* application for external performers.

A great number of scenarios concerning social service related errands will be supported in Combine. An example is when a citizen needs help with cleaning due to functional limitation. In the system, the citizen has the opportunity to apply for assistance. The letter of application further on appears in the employee application which in turn handles the request. The decision is afterwards presented back on the citizen's application. If the citizen was granted help the assistance is ordered. The order is sent to either employees in the communes or to external performers.

Combine will have many concurrent users and the information in the system will be very sensitive. Private personal data is stored in the system and it is necessary that this information is protected and inaccessible for anyone without authority. The requirements on how users will be granted access to Combine are therefore strong. Neither the development of authentication functionality, nor the work with how the authorization complexity should be solved had been initiated by the Combine team at the start of this thesis.

The authentication and authorization parts are non-trivial problems. The different applications require different authentication methods when users log on to Combine. Employees in the communes have differing roles and authorities which must be handled. Also, the organizational structures in the communes are not the same. The roles and authorities must therefore be handled separately from one municipality to another.

There is a desire to be able to log on to Combine using the existing internal authentication infrastructure in the communes. One reason is because it would be convenient to be excused from having one authentication procedure for every system that is used within the commune. This approach is called Single Sign-On (SSO) [2]. To achieve the SSO behavior with federated identities, the identity of an authenticated user has to be sent to, and further on trusted by, the Combine system. There are different technologies available to achieve this

[1] *The ANNA Project* was used as working title of Pulsen Combine.

functionality. Among all, the Security Assertion Markup Language (SAML) 2.0 [3] is the preferred technology for the Combine system. The reason is because there is a new Swedish standard, BIF [4, 5], under development which will implement SAML 2.0. BIF is meant to rule and guide when designing security such as authentication, authorization and auditing in social service and healthcare applications such as for instance Combine. To avoid making a solution that is incompatible with this new upcoming standard, it was a good idea to take BIF into consideration when designing the security parts of Combine.

3 Problem Definition

The aim of this thesis has been to design and implement a prototype for the authentication and authorization parts of the Combine system.

4 Method

This section describes the methods which were used within the different phases of the thesis.

4.1 Pre-study

The pre-study phase aimed to give an understanding of the problem domain as well as how the Combine system was built up. The reason for this was that the prototype and architectural recommendations that were to be given had to be adapted to how the involved communes wanted things to get solved, and also to how the Combine system was designed on an architectural level. The phase served as an essential base for making important design decisions and made sure that there was a good understanding of the requirements that had to be met.

The different available technologies that could be of use were given preliminary evaluations. Evaluating different authentication and authorization technologies and conducting some basic prototyping helped in getting an overview of how the problem could be solved. The study of available technologies was done using Internet searching and consulting with experienced personnel at Pulsen. Using the information that was found, analyses were done and decisions were made regarding the use of a certain technology or design pattern.

Interviews and meetings were held at an early stage, to get more information about how the Combine system was designed, and to find out how the team planned on continuing the development until the release of the product. The interviews were carried out in an informal way, and no particular interview methodology was judged to be necessary. Just as with the interviews, most meetings had the goal of giving us more information about the general domain and the architecture of the system.

4.2 Design

This phase aimed to result in a design of the prototype that was later to be implemented. Basic software principles had to be followed, such as ease of maintenance, low coupling and easy extendibility.

During the design phase, brainstorming sessions were carried out. These sessions aimed to find some initial scenarios, and bring some preliminary solutions to the table. They were interchanged with meetings together with the Combine team, to discuss the progress and harmonize the results with new information about the development of the system.

Furthermore, simple prototyping was carried out in this phase. When making design choices for a system that uses technology that one is not accustomed to, simple programming helped in confirming new ideas and design choices.

4.3 Implementation

The aim of the prototype, which was implemented in this phase, was to show that the proposed design was feasible, and compatible with the Combine system.

During the implementation of the prototype, the agile process Scrum was utilized. This meant joining the daily Scrum meetings with the team, where development issues were discussed. [6]

No clear iterations (or sprints) were used, as it did not seem necessary for the successful completion of the project. The size of the project was simply not big enough to have any meaningful division into time slots.

4.4 Evaluation

The aim of the evaluation was to see how the prototype correlated to the problem definition, and how well it fitted with the rest of the Combine system. Conclusions were drawn and recommendations for future work were given.

Also, in this last phase of the project, code reviews were done. This was to make sure that the prototype followed software conventions, such as maintainability and transparency.

The prototype was presented to the Combine team, and feedback was received. The feedback resulted in the definition of future work.

5 Security Principles

When designing the authentication and authorization parts of the Combine system, a variety of security principles were adhered to. These principles are generally supported within the software security community, and therefore serve as a sensible base for the work in this project [7]. The principles that were followed during the thesis are described in the next paragraphs.

5.1 Security in Depth

This principle stresses the importance of having more than one security layer. Having only one layer of security in any type of application should be considered to be unsecure. This is because a person with ill intent, in many cases, is able to breach at least one layer. [8]

5.2 Positive Security Model (Whitelist)

When dealing with security, policies have to be made. In many cases lists are maintained of identities/protocols that are allowed/disallowed to access a certain resource. These policies and lists should in many cases be based on a Whitelist principle. This means that, for instance, the list should contain all the identities that are allowed access, instead of containing all identities that are disallowed. This might seem as a small detail in how to design the security of a software application, but it is actually of significant importance. This is because following the Blacklist principle (which is the opposite of the Whitelist principle) means that there is a constant need to be in the lookout for new threats to add to the Blacklist. Also, there is usually no guarantee that one can find all the possible threats, since new threats always are being discovered. Keeping a list of allowed instances is therefore much easier administratively as well as more secure; giving full control of who/what has access to ones application. This does, of course, not mean that using the Blacklist principle is negative in all situations. When building the architecture of an application, though, it is usually a good idea to base it on the Positive Security Model. [9]

5.3 Fail Safely

This principle says that when a program fails, it should fail safely. This means for instance that the original exception that was thrown never should be shown to the user, but general information should be given instead. If original errors and exceptions are shown to the end users, the information they provide could reveal information about the system that makes it more open to an attack.

Also, this principle says that one should be careful of how access rights are given to users in the system. It is important to make sure that a thrown exception never makes the program bypass important access code. This is illustrated in the example in Listning 1.

```
isAdmin = true;
try {
    codeWhichMayFail();
    isAdmin = isUserInRole( “Administrator” );
}
catch (Exception ex){
    log.write(ex.toString());
}
```

Listning 1 – Fail Safely example

What the example illustrates is that a user gains administrational rights if an exception is thrown at “codeWhichMayFail()”. It would therefore be a better idea to begin with denying the user administrational rights, and then granting the right if “isUserInRole” returns true. [10]

5.4 Keep Security Simple

This principle simply says that the more complex a solution is, the more difficult it will be to maintain it. It is also more difficult to make sure that the system gets full security coverage. Thus, keeping it simple is a key part in making a successful implementation. There is of course always a balance in the complexity of a system versus the level of protection. This principle should therefore not be read in isolation, all factors should be taken into consideration when designing security parts in systems. [11]

6 Authentication

Authentication can be summarized as the act of confirming that something that has been claimed is true. The claim can be that a computer program is from a trusted source, the origin of something is what it claims to be, or that a person is who she claims to be. In this report, whenever referring to authentication, the latter one is meant. [12]

6.1 Authentication Methods

There are many different ways of authenticating a person. Different methods are of different security levels and are based on different paradigms. One of the most basic kinds of authentication methods is the one-phased. This method is based on the idea that the person who wants to authenticate herself knows something that no one else knows, i.e. a password. In most environments that are not very security critical, this method is used. A negative aspect about the one-phased method is that knowledge is the only authenticating factor. If someone finds out what the password is, the security is in effect totally breached.

There is also a two-phased authentication method. Not only is it required that the person knows something, but also that she possesses something, for instance a smart card or some other security token. This way, if the password is leaked, an intruder still can not authenticate herself since she does not have access to the security token. Two-phased authentication is used in situations where security is of higher importance, one example being many office environments where employees log in to their computers using smart cards.

Biometric authentication is a rather common and secure way of authenticating people. It is based on the idea that a person uses a part of the body which is unique to that person to identify and authenticate herself. Parts of the body that can be used are fingerprints, iris of the eye among others. [13]

6.2 Single Sign-On and SAML 2.0

Single Sign-On (SSO) is when a user can utilize several different applications at once, by only logging in to one of them. This is based on trust between applications. The applications trust that when a user has logged in at one application, the user is authenticated in a secure manner and therefore does not have to login again when trying to access one of the other applications.

Achieving cooperation between different applications to get the Single Sign-On behaviour can be done by implementing the SAML[2] standard. SAML is an XML-based standard that has the primary aim to enable the exchange of security information between so called security

[2] Further on in this report, whenever using the term SAML, version 2.0 is referred.

domains. Security domains could for instance be different applications. The security information concerns authentication and authorization.

The SAML standard is built up by assertions, protocols, bindings and profiles. Assertions are the basic building blocks of the SAML message. The protocols tell in what order the messages are sent. Bindings specify how the messages are sent technically, i.e. how they are transported over the Internet. Finally, profiles are groupings of the previous components, putting them in relevant groups that are useful for most users of the standard. One example of a profile is the Web Browser SSO Profile, which specifies all needed components (assertions, protocols and bindings) to make a successful SSO federation using web browsers.

6.3 BIF

BIF is a new and upcoming Swedish national standard and infrastructure for information security within the public healthcare sector. It will, when completed, ensure that the integrity of all patients in Sweden is kept. It will also guarantee that only authorized users, for instance doctors and other healthcare personnel, can access certain features and view certain information.

Some important reasons for this new security infrastructure are:

- Making it easier for healthcare personnel to access information about patients that live in another administrative region of the country and/or patients from another kind of institution (hospital, clinic, health centre)
- Easing the authorization process of all healthcare (for instance social service) systems in the country by providing one infrastructure
- Securing the information and integrity of the Swedish patients

The implementation of this infrastructure will be in the form of web services that can be called and utilized by authorized systems.

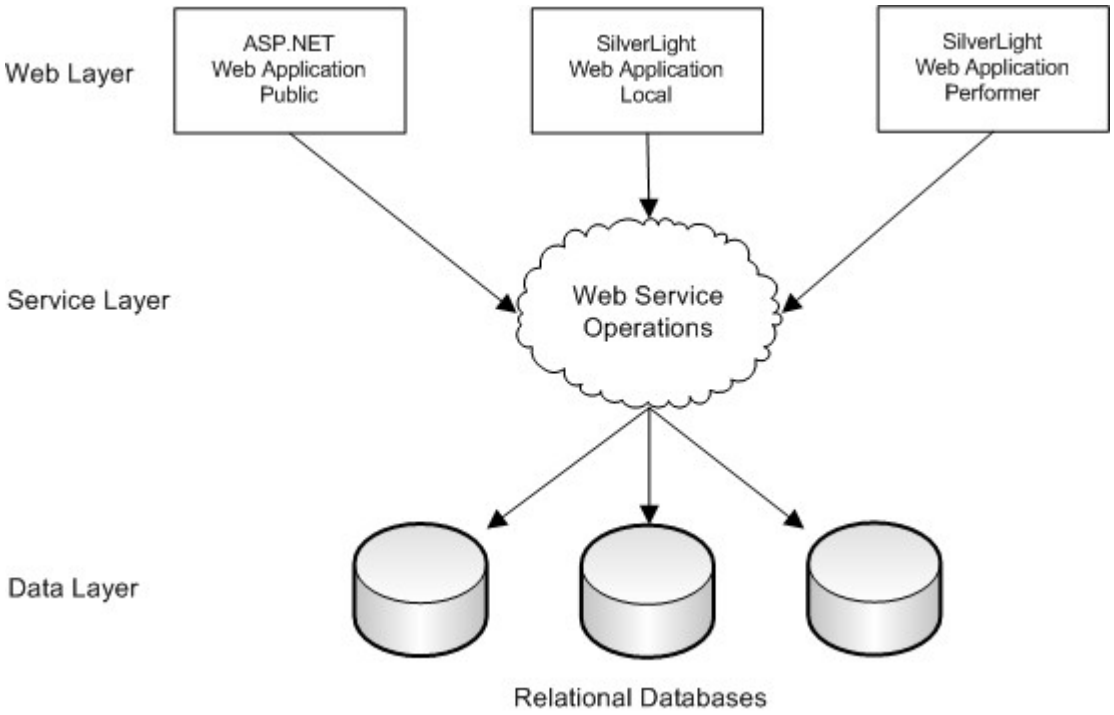
7 Authorization

Authorization, in contrast to authentication, is not about confirming the identity of a person, but rather that the person has enough credentials to perform a specific action. However, authorization can not be done without first having to authenticate. There has to be a guarantee that the user is who the user claims to be before confirming that enough authority exists to access some data or perform an action. [14]

8 The Combine System

This section describes the Combine system; the architecture of the system, technologies that are used and the organizational aspects from the communes that were of importance to this thesis.

8.1 System Architecture



Listning 2 - System Architecture

Service orientation is a central term when looking at how Combine has been built. This means that most of the business logic in the system is put inside web services. These services are then called by the web layer when data needs to be fetched, calculated or persisted.

Combine is devised of several layers, the most bottom one being the data layer. The data layer has a number of databases that contain all the persisted data the system needs to store. There will be one database instance per commune and private performer. The different applications will also use some common databases.

The service layer is located above the data layer. This layer handles all the communications with the data layer. Requests from the layer above therefore always goes through the service layer when fetching data from storage. The service layer also performs computations on the data that is fetched from the data layer. It therefore serves as a business layer for the system.

The three web applications reside above the service layer. There, all the presentation logic is located.

8.2 Platform

Combine is built using C# and the .NET (3.5) Framework from Microsoft. The data layer is built using Microsoft SQL Server (MSSQL 2008), and the service layer uses the Windows Communication Foundation (WCF). Finally, the web applications are constructed using ASP.NET and SilverLight. Two of the three web applications that are included in Combine are using SilverLight and one is a standard web application. The SilverLight applications are for the employees at the communes and the external performers respectively, while the web application is meant to be utilized by citizens of the communes.

SilverLight is, at the writing of this report, a quite recent addition to the web browser plug-in arena. It enables features that are quite similar to that of Adobe's Flash, which are vector graphics, animations and media playback. SilverLight applications are implemented using Microsoft's .NET platform. Therefore, languages like C# and Visual basic can be used to quite easily create rich internet applications. [15]

8.3 Organizational Architecture

A requirement has been made on Combine; it should enable the communes to represent their organizational structures in the system, in the form of an organizational tree. This tree should be built through an administrative and graphical user interface by the commune administrator, and should serve as the base for all authorization functionality within the system. This means that an organizational structure is built by an administrator, followed by the adding of rights to the nodes in the tree. Every user in the system belongs somewhere in the tree, depending on what role the user has in the organization.

The tree is central to all authorization functionality since being a member of a certain node gives the user access to certain functionalities in the system. A user gets connected to one node in the tree (for instance representing the role of a nurse), and so attains access to all components and functionality that is connected to that node. Also, specific for the Combine communes, being a member of one node in the tree automatically also grants the same rights as all children of that node. This is because the tree is built to reflect the hierarchical organization of the communes, i.e. management positions can be found at the top nodes.
Henceforth in this thesis, whenever referring to a user having a right, it refers to the user being a member of a certain node. No difference is thus made between membership of a node and having a right.

Ideally, an administrator in the commune should be able to dynamically bind and unbind users to nodes in the tree, and also easily change the rights for different nodes.

9 Pre-study

In the pre-study phase the goal was to get a clear picture of how the authentication and the authorization parts could be designed and integrated into Combine. To be able to get this picture the Combine system was analyzed.

9.1 Authentication

Initially, there were different views and opinions on how the authentication part could be realized in the Combine system. Either the authentication part could be developed in house or it could be handled by using a, for Pulsen, well known security product on the market – the PortWise security platform [16].

To be able to make a correct decision, the market was searched for frameworks and toolkits that could correspond to the same functionality as the PortWise product, i.e. be able to perform the same tasks as PortWise. These were then compared to each other, and finally compared to the PortWise alternative.

When the thesis work began there was a focus on BIF and therefore also on Single Sign-On. To achieve the SSO behaviour the protocol SAML was the choice of preference. One advantage with PortWise is that it supports the SAML standard. If PortWise would be chosen the customers could thereby be able to authenticate their users in their own environments using their own authentication methods/products. When the market was searched, therefore only frameworks and toolkits with SAML compatibleness were of interest, to achieve the same possibility for the customers.

9.1.1 Market analysis

One framework and one toolkit of interest were found during the market analysis.

A candidate was the Geneva Framework [17] which is a Microsoft product[3]. Initially, it seemed to have full support for the SAML standard. Unfortunately, when investigating further, it was found that there did not seem to exist full support for the SAML protocols which was believed to be a requirement for the established scenarios. Also, the product had not yet been released as a final version, which created further uncertainties.

The toolkit that was found with SAML compatibleness was the SAML v2.0 .NET Component from ComponentSpace [18]. This component implements the full SAML standard. No clear incompatibilities were found during the analysis and testing, although there were no references to be found of its' successful use in a real project.

[3] The first release of this framework reached the market in October 2008. The aim of the Geneva framework is to make the implementation of claims-based applications and services easier.

The SAML v2.0 .NET Component was the strongest challenger of the two candidates when considering developing the authentication part in house. Concerning the Geneva Framework, there was not time to wait for all the required functionality to be implemented, and for a final working version to be released.

Even though the SAML v2.0 .NET Component seemed to fit the requirements, it was found that the level of complexity of developing a SAML solution was much greater than initially expected. This lead to the establishment of the fact that there was simply not enough time or resources to develop the authentication part of Combine in-house. Also, the authentication part is one of the most important and sensitive parts of the system. With the choice of PortWise the Combine system can rely on, and be protected by, a product with denary of years of experience in the security discipline.

A choice was therefore finally made, to use PortWise for all authentication purposes in Combine.

9.2 Authorization

As pointed out, it is important to have several layers of security. A primary goal during the pre-study phase was to find several layers in the application in which authorization checks would be appropriate.

9.2.1 Authorization in the web layer

It is important to the security of Combine to have the possibility to hide or show alternatives to the system's different users. The rights of a user determine if the user is authorized to access buttons, menus, pages and/or other graphical components. For example a user with administrational rights will have access to pages that an ordinary employee does not.

It was decided to use a pre-processing method when authorizing if a user would have access to, for instance, the functionality of a button, i.e. if it is possible to click on it. This pre-processing method would thus be executed before the page is shown to the user. Also, a web filter was decided to be implemented, serving as an authorization check for every requested page in Combine. A web filter serves as an intercepting method that all incoming requests pass through before being dispatched to the target resource.

9.2.2 Authorization in the service layer

It must be made sure that a user only can call the web services that the user is authorized to utilize. Role-based Authorization [19] was evaluated briefly as a candidate for serving as a tool when solving the service authorization problem. Role-based Authorization is built on the premise that each method or service accepts certain roles to invoke it. Furthermore, every caller has a role that is accessible for reading. In Role-based authorization, the role is placed inside the calling thread, which is not suitable in a service-oriented setting. The service will in no case have direct access to the calling thread. When examining the

possibility to customize this aspect, it was found that there did not seem to be a way of achieving this, so this method of authorization was discarded.

In opposite to Role-based Authorization, a kind of filter, Parameter Inspector[4], seamed to fit the purpose. When using the Parameter Inspector on a service operation one has to add an attribute[5] to the operation. This leads to the implementation of the attribute being executed before the service operation itself. It is hereby possible to perform custom checks before letting the call pass to the operation.

When experimenting with the Parameter Inspector one problem was found. The Parameter Inspector was not compatible with the fault handling functionality in the service layer of Combine. All service operations in Combine are tagged with an attribute that adds fault handling to the service layer. The problem occurred when a service operation was tagged with both the fault handling attribute and the Parameter Inspector attribute. Those two attributes could not be used at one service operation simultaneously.

Therefore, it was perceived that it would be a solution to combine the implementation of those two attributes. Only one custom attribute on the service operation would be needed and the conflict would not arise. This seemed to work properly and will be discussed in more detail in the design part of this thesis.

[4]Parameter Inspector is an extensibility object used to inspect and alter messages after they are received or before they are sent [20].

[5] *"The common language runtime allows you to add keyword-like descriptive declarations, called attributes, to annotate programming elements such as types, fields, methods, and properties. Attributes are saved with the metadata of a Microsoft .NET Framework file and can be used to describe your code to the runtime or to affect application behaviour at run time."* [21]

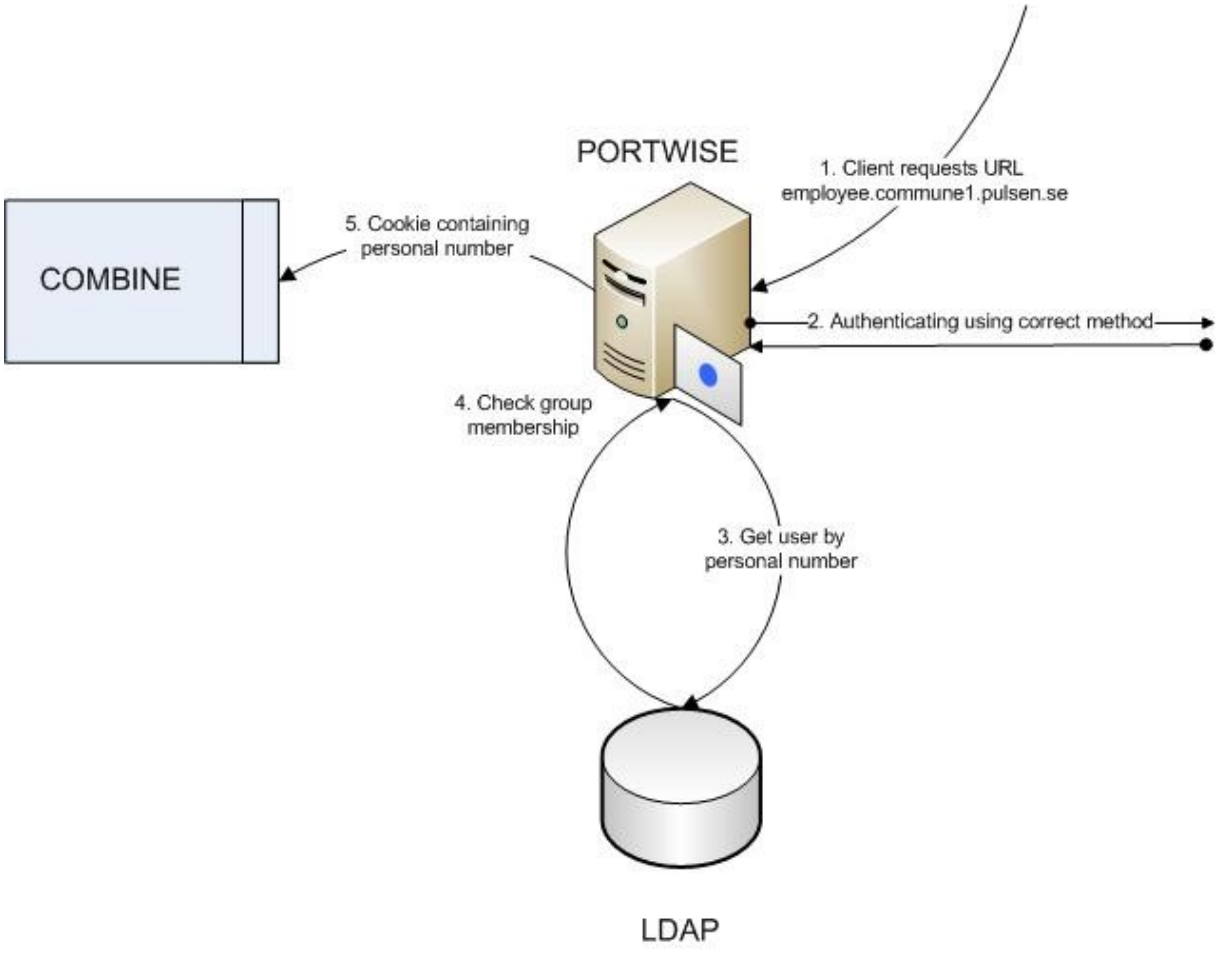
10 Design

10.1 The PortWise Security Platform

When using PortWise one has the opportunity to configure the product and use parts of the functionality so that it adheres to the working environment. In this description only parts necessary for the actual thesis are concerned and discussed.

There was a strong focus on Single Sign-On authentication early in the pre-study. As the time passed by, the customers realized that they will not be ready to facilitate SSO authentication. Therefore, in the first release of Combine, the authentication part will be carried out without any SSO and SAML functionality involved. Anyhow, the choice of PortWise makes it possible for customers to choose these technologies in the future. They will not be discussed in more detail in this thesis.

In Listning 3, a brief graphical explanation of the final authentication and authorization process in PortWise is given.



Listning 3 – The PortWise authentication and authorization process

When a user wants to enter the Combine system the user has to request the correct URL to the part of the system (i.e. the application) to which he or she wants to be granted access. The URL consists of both a role (employee, citizen or performer) and a customer (commune). Examples of how the URL's will look like are shown in Listning 4. All URL's points to the server(s) where PortWise is located.

```
employee.customer1.pulsen.se
citizen.customer1.pulsen.se
performer1.customer1.pulsen.se
```

Listning 4 - Addresses to the Combine system

In PortWise, the URL's are further mapped to the different Combine applications. This means that, for instance, employee.customer1.pulsen.se is mapped to the location of the application which the employees at the specific commune (customer1) will use.

Before entering the Combine system all users have to be authenticated correctly using authentication methods that are configured inside the PortWise product. PortWise supports different authentication methods. It is possible to add new ones or to use the built-in ones. For example it is possible to use the built in PortWise Password method or add an E-ID or smart card method. For instance, when an employee requests the *Local* application, the user has to be authenticated using a smart card authentication method. When a citizen requests the *Public* application, he or she has to be authenticated using E-ID. Those different authentication methods are set up as different access rules.

Just because a user has been authenticated correctly it is not sure that the user will be granted access to the requested application. For instance, a citizen should not be able to get access to the employee application, just because the identity of the citizen has been authenticated. There is a need for an authorization check that makes sure that the authenticated user belongs to the requested application. This is achieved by setting up access rules which check whether a user is member of a certain group in a directory. Consequently, a user requesting the URL employee.customer1.pulsen.se must exist in an EmployeeCustomer1 group in the directory.

A short summary of which access rules that are required for different users in Combine is given in Listning 5.

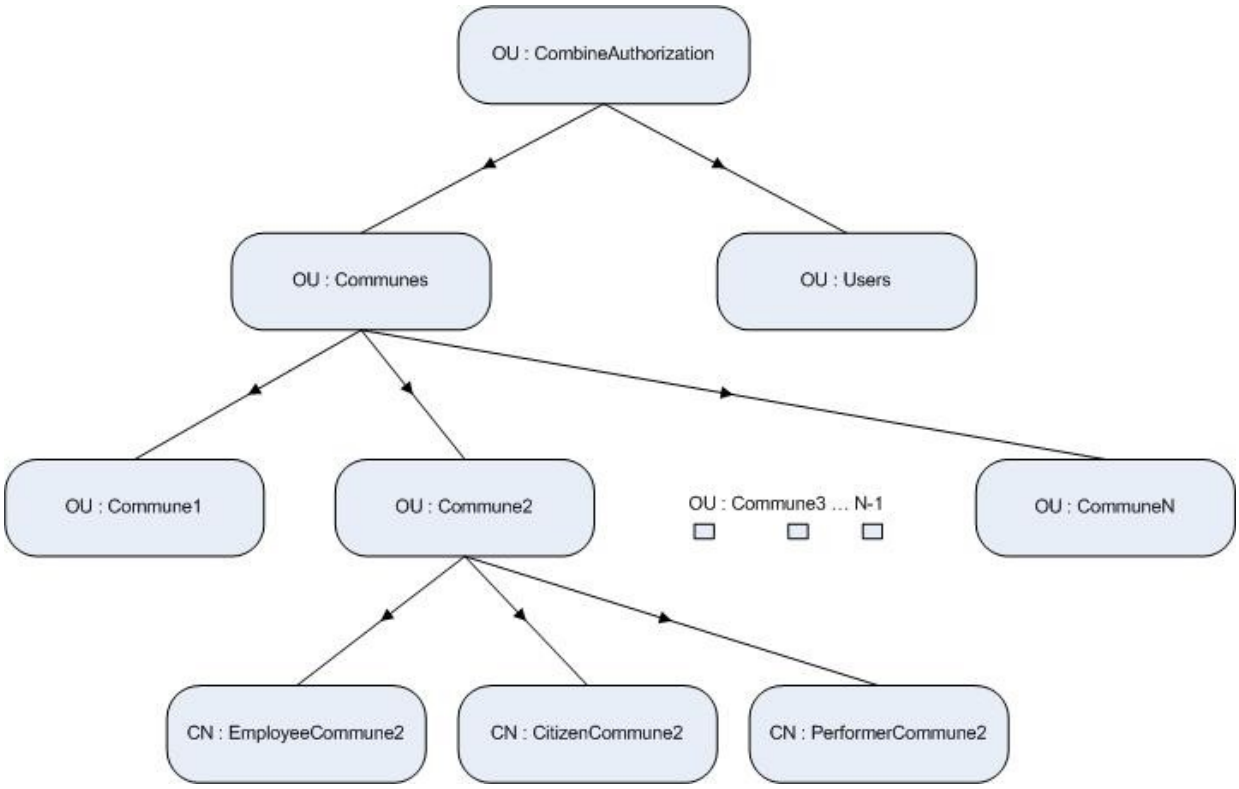
	Local users	Public users	Performers
Group Membership	X	X	X
E-ID		X	
Smart card	X		
Other hard certificate			X

Listning 5 - Access rules for different user roles

The outcome of the authentication process is an identification number (personal number) that is passed to the actual application in a cookie. When the user finally has reached the requested application, all following requests to/from the application will continue to pass through the PortWise system. This makes it possible to log user actions and handle timeouts.

10.1.1 PortWise user directory

The access rule that checks the group membership of a user requires, as mentioned before, a directory containing users and groups. This directory will contain all the users that can access the different Combine applications. The directory will only contain a basic identification number of the user, and no other information. All other information about the users will be kept inside the system, in the data layer. Thus, in short, the point of having such a directory is to make sure that only people that have the right to enter the requested application is granted access by PortWise. Note, however, that the directory does not have anything to do with the organizational structure within the communes. The only thing the directory establishes is a users' basic belonging, i.e. employee, citizen or performer in a certain commune. PortWise requires this directory to exchange information through the LDAP-protocol [22]. The design of the directory is shown in Listning 6.



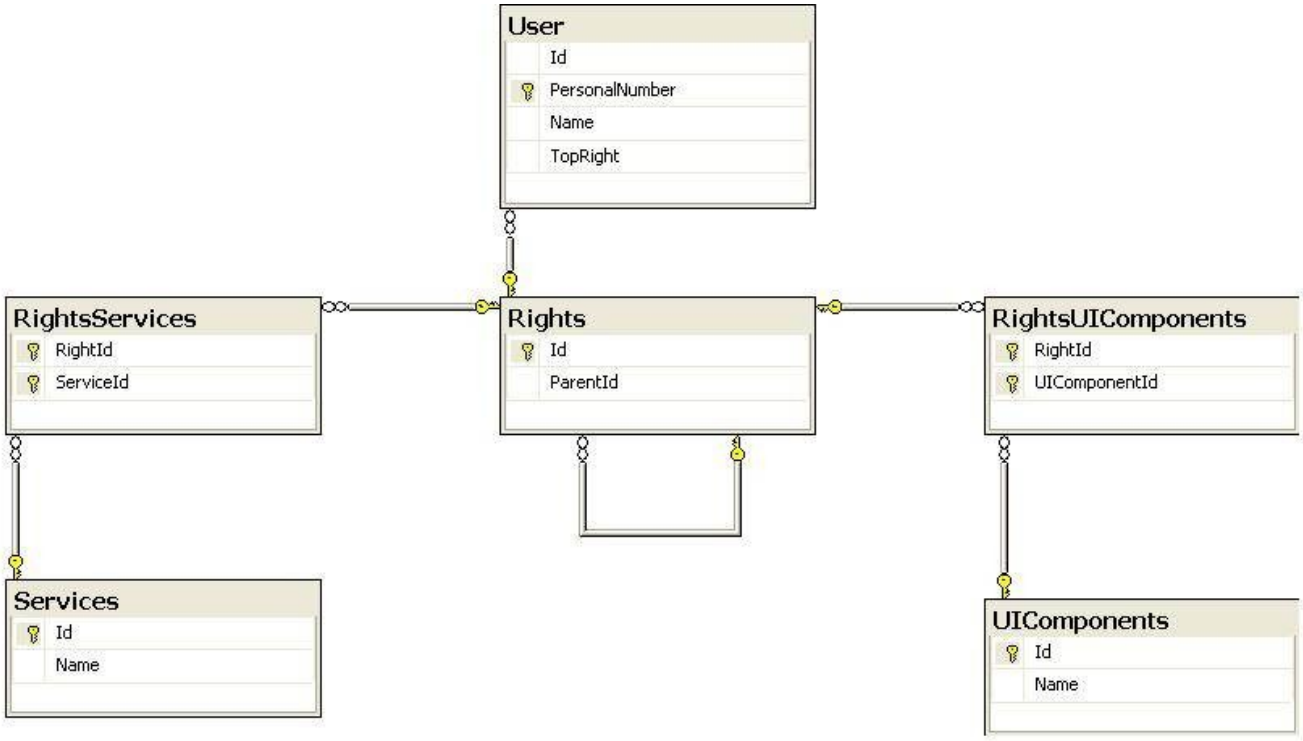
Listning 6 - LDAP structure for the authorization part in PortWise

The hierarchical structure needed for checking group memberships has a root organizational unit[6] (OU: CombineAuthorization) that serves as the base of the information needed for the authorization part in PortWise. The tree is then divided into two sub branches. One branch consists of an organizational unit (OU:Users) that contains the users of the system, independent of role and geography. The other branch consists of an organizational unit (OU:Communes) that handles the different communes (customers) in the system.

The organizational unit called Communes contains one organizational unit for each customer (commune) in the system. Furthermore each single customer organizational unit contains three LDAP groups[7]. The first group contains the employees of the commune, the second group contains citizens that in some way are associated[8] to the commune and the third group is for performers.

[6] An organizational unit is a way of organizing and categorizing data entries in an LDAP directory.
[7] An LDAP group is an LDAP entry that has the ability to associate with other entries as members.
[8] A user in the citizen group can be either a resident or a relative to a resident in that commune.

10.2 Database design



Listning 7 - Database design

The central part of the database design is the Rights table. This table contains all the rights that exist in the system. It is possible to derive a list of rights for each user in the system from this table.

In the Combine system, an organizational tree is built up for each commune. In the database, this is realized by implementing the Rights table as a tree. Each entry in the table has a parent node, and so a hierarchical system is built up (see the Combine system part for more information about the organizational tree). Every user of the system has a location in this tree. Also, when being the member of a certain node in the tree, the user automatically gets the rights of all the child nodes. In the design of the database, a column called TopRight has been added to the User table. This column is meant to represent one node in an organizational tree that the user is a member of. How to traverse the tree to get all the child nodes of the users' main member node will not be discussed in this report.

The table Services contains the names of all the service operations that should be under the control of authorization checking. If an operation[9] only can be accessed by some users of the system, it has to be recorded in this table. The Rights table is coupled with the Services table through a coupling table, as can be seen in Listning 7. This coupling is meant to connect each Service operation with all the rights in the Rights table that are authorized to call the operation.

[9] An example of an operation might be "GetUserAddress"

The UIComponents table is based on the same idea as the Services table, only this table contains user interface components like buttons, panels and pages. Also, like the Services table, the UIComponents table is connected with the rights table.

10.3 Authorization in the service layer

When a call reaches a web service, the service can not be sure that the request comes from an authorized user. To avoid calls from unauthorized users an authorization mechanism has been implemented. Every time, before a call reaches the destination service, a check is made to see if the user has adequate rights for entering the service implementation.

As discussed in the pre-study, several technologies for authorizing the service calls were evaluated. The outcome of the evaluation resulted in that it was applicable to integrate the service authorization with the fault handling functionality of the service operations.

In the implementation of the fault handling attribute, as described in the pre-study section, the authorization code is run. The service operation rights are compared to the user rights. If they coincide, access is granted and the call is passed through to the service operation. Otherwise, the user is not granted access and the call is sent back with an error description.

All service operation calls are logged. Information about a call is logged if a user is granted access to a service operation. It is also logged if a user does not have adequate rights to execute the service implementation. It is important to log calls to the services. If someone tries to call a service without adequate rights it may be an intrusion attempt. The logged data can be used to analyse suspicious activities and can also serve as evidence in possible disputes.

10.4 Authorization in the presentation layer

10.4.1 Graphical Rendering

Using authorization when rendering graphical components serves as a first line of defense when entering the different applications. The authorization on this level is on a superficial level, since the application is service oriented. What this means is that even though a user is unable to see a specific button or menu in the graphical interface, there is not much that stops the user from trying to call operations in the service layer by other means. Having the PortWise solution as an intercepting instance helps in many ways, making it very unlikely that someone can access the service layer directly from the outside. Despite this, authorization checks are important in the presentation layer.

As explained above, all the user interface components are put into the UIComponents table in the database. Using this backend architecture, authorization checks are made quite easily. When using the .NET framework, whenever a graphical component is loaded by the

server for a user request, it is possible to inject code that runs before the component is shown. For instance, when a user requests a page, there is a Page_Load method that is run before sending the page to the client browser. These kinds of methods were utilized to render the menus and buttons in such a way that users could only see the components that they were authorized for.

Before each menu is rendered and shown to the user, code is run that make this authorization check. The code calls an authorization service operation that takes the component name as a parameter. The operation then makes a lookup in the database to see if the given user id has the required rights to see the specified component. The operation then returns a boolean value, true or false, depending on if access was granted or not.

10.4.2 Page Loading

Before a user can access a page in the *Public* web application, an authorization check is made. There are more ways to try to access a page in a web application than by clicking on a link or a button. The user might for instance try to write the URL for the page in the browser URL field. This leads to one more security mechanism being needed to protect the web application from malicious use.

A well known concept and design pattern for web applications is the web filter. The web filter is based on the idea that every request that comes into the application server is processed with some generic[10] code. Technically speaking, the web filter is a method that is run for each new request, before the requested page or resource is fetched and sent back to the client. Such a filter is therefore ideal to use when authorizing a user to access a certain page.

For the filter to host a suitable solution, it is required to:

- Know which page the client is trying to access.
- Access the current user session, to be able to fetch information about who the user is.
- The ability to redirect the user to somewhere else than the requested page, should the authorization fail.

The ASP.NET framework has support for such a mechanism, namely the IHttpModule interface [23]. Implementing this interface made it possible to insert custom code into a point in the request lifecycle, and thus achieving the web filter functionality.

The page loading authorization was thus designed as follows. For every request, the method OnAquireRequestState is run. The method first checks to see if the session is new. If it is, the

[10] For instance, clean URLs are often implemented in web filters.

session is populated with useful information. If the session is not new, the authorization check is made. It is made by making a service call to the same authorization service as for the graphical rendering, and works in the same way. One difference is, though, that the user is redirected to an error page if the authorization fails. Recall that in the graphical rendering case, the user simply could not view the button if the authorization for the button failed. Also, in the OnAquireRequestState method, the session is validated for general security purposes, but this falls outside the scope of this thesis project.

11 Implementation

The design was implemented, and resulted in a prototype. The prototype was a simulation of the real Combine system, with the authentication and authorization functionalities added. Implementing the solution in this way helped to keep it in a controlled environment and made it easy to verify that it worked as expected.

11.1 Problems

Some issues arose during the implementation of the prototype.

11.1.1 Parameter Inspectors

The problem concerning the Parameter Inspector, being incompatible with the fault handling mechanism of Combine, was solved by excluding the Parameter Inspector from the solution, and putting the authorization logic inside the fault handling code instead.

11.1.2 SilverLight and HTTP 302

When making web service calls from within the SilverLight application, there was a problem with receiving HTTP 302 (redirect) messages that came from the PortWise platform when there had been a timeout. When a timeout had occurred, the PortWise platform redirected to a login page, so that the user could be authenticated again, before being allowed to access the system.

The problem was solved as follows. Whenever an exception is reached in the SilverLight application that is related to a web service call, a manual redirect is made to the PortWise log in page.

12 Conclusion

The goal with this thesis was to ensure that users of Combine are authenticated and authorized in a secure and proper manner.

12.1 Authentication in the Combine system

The developing of the authentication part in-house was compared against integrating the PortWise security platform with the Combine system. It was decided to make use of the existing knowledge and functionality within the area and therefore the PortWise platform was chosen.

With this choice, the security aspects concerning the authentication were assigned to a product whose main purpose is security and login procedures. The Combine team doesn't need to put a lot of resources and time on a complex authentication solution, but can instead focus on the development of the core system.

The PortWise product furthermore supports authentication using SAML which may be useful in the future. If the customers of Pulsen want the login procedure to take place in their own environments, it will be possible.

12.2 Authorization in the Combine system

12.2.1 Implemented Principles

A layered security model has been developed. If an intruder breaks one layer, there are more security layers to break until the secret information can be retrieved. This model follows the behavior of the *defense in depth* principle.

When authorization checks are made in the system, user rights are compared against rights necessary to access the requested resource. The rights of a user symbolize information and functionality that the user has access to in the system and not the opposite - what a user is disallowed to. This structure follows the *Whitelist* principle.

When exceptions are thrown, caused by for instance denied access to a resource, the original exceptions are not shown to the user of the system. This follows the *Fail Safely* principle. The exception messages are translated into pre-decided error messages so that they do not show the user any critical information about the system. Also, in the application code, authorization logic flows make sure that a user can not attain rights illicitly by making the application throw exceptions.

During the project, the *Keep Security Simple* principle has pervaded the whole thinking. This has been manifested in, for instance, the web filter and the authorization in the service layer being transparent to the rest of the system. This means the code for those respective

parts were put in only one place each, instead of having to spread it throughout the whole system.

12.3 Database design

Designing the database in the described way brings some benefits to the solution as a whole. Decoupling rights from components which are under authorization checking makes it easy to keep from being too dependent on a single organizational structure. There could just as well have been a totally different structure in the rights table, and the overall database design would not have been majorly affected by this.

The implementation of the organizational requirements is placed mainly on the software tier, instead of on the database tier, i.e. parsing of the Rights table is done in the code which leaves the Rights table simple and generic.

The database design makes it easy to restructure the tables in such a way as to add more tables containing authorization controlled units. For instance, it is not a problem to split the UIComponents table into two tables, one that keep all the graphical elements and one that keeps all the pages. All that need to be done, except adding the new table, is to add a coupling table that connects the new component table with the Rights table.

Finally, the design makes it possible to easily build a graphical administration interface to administer the rights, add components and change user authorization levels. This is due to the straight forward connections of all tables.

12.4 Presentation layer

The authorization service operation, which takes a name of a graphical component as input, could be written in more ways than the explained way. One other possibility is, for instance, that the service operation takes a group of buttons as a parameter, and returns a dictionary (Hash structure) with button–boolean pairs. This way, there will not be unnecessarily many service calls in the application. One thing is certain though, and that is that it is not a good idea to have to make one service call per button in the system. This would obviously mean a too large amount of service calls and database lookups.

Implementing the authorization functionality, within the presentation layer, in the described way bring some benefits. First of all, this kind of solution makes it easy to write a stable and easily verifiable implementation, seeing as how the only place, in the *Local* application, that needs authorization code is in the controls where the menus are rendered. Also, putting all the authorization related code on the server side definitely adds to the security.

12.5 Future work

After conducting an evaluation of the implementation, some issues and remaining work were established.

12.5.1 Administration

It is essential that a system with as many different users and authorities as Combine is easy to administer. The implemented security model is designed to be easy to administer. To make the administrational work even easier, it may be a good idea to introduce access categories. An example of an access category can be “health-care personnel”. Being a member of this category involves having access to a number of different functionalities in the system.

The purpose of the access categories would be to serve as an intermediate layer between the nodes in the tree and the services and/or user interface components in the system. More specifically, a node (role) in the tree will be connected to one or several access categories which in turn are connected to services and user interface components. This would be in contrast to having a set of rights connected directly to each single service or user interface component in the system.

The administrator of the system does hereby not connect each single service or component in the system to each single node in the tree but instead connects more abstract access categories with nodes in the tree. This relieves the administrator from having to deal directly with web service operation names and names of user interface components.

12.5.2 Security in the data layer

Having authorization checks in the data layer adds to the security of a system and makes sure that only authorized instances can make database lookups and editing. Even though the data layer most probably will have hardware based protection (for instance ip filtering) there is a need for protection against the authenticated users of the system, to prevent illicit use.

References

1. ”Pulsen utvecklar nytt IT-stöd för socialtjänsten”, Pulsen Application 2009, http://www.pulsen.se/aktuellt/aktuellt/pulsenutvecklarnyttitstodforsocialtjansten.5.4be8140311f12eea0f880008085.html
2. ”Security Analysis of the SAML Single Sign-on Browser/Artifact Profile”, Thomas Groß(IBM Zurich Research Laboratory) 2003, http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?arnumber=1254334
3. “SAML V2.0, Executive Overview”, OASIS ,http://www.oasis-open.org/committees/download.php/13525/sstc-saml-exec-overview-2.0-cd-01-2col.pdf
4. ”BIF, Bastjänster för Informationsförsörjning”, PortWise, http://www.portwise.com/index.php?option=com_content&view=article&id=246:bif&catid=52:losningar&Itemid=500
5. ”BIF – Bastjänster för informationsförsörjning”, BIF-projektet, http://www.bifprojektet.se/
6. "Requirements Engineering and Agile Software Development", Frauke Paetsch (Fachhochschule Mannheim), Dr. Armin Eberlein(University of Calgary), Dr. Frank Maurer(University of Calgary), http://ase.cpsc.ucalgary.ca/ase/uploads/Publications/PaetschEberleinMaurer.pdf
7. "Software Security", Gary McGraw, IEEE 2004, http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?arnumber=1281254
8. SP800-27, "Engineering Principles for Information Technology Security (A Baseline for Achieving Security), Revision A", National Institute of Standards and Technology, U.S. Department of Commerce 2004, http://csrc.nist.gov/publications/nistpubs/800-27A/SP800-27-RevA.pdf
9. ”On the Automated Creation of Understandable Positive Security Models for Web Applications”, Christian Bockermann, Ingo Mierswa, Katharina Morik, Artificial Intelligence Unit, Department of Computer Science, University of Dortmund 2008, http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=04517455
10. "Security Design Patterns Part 1", Sasha Romanosky, Morgan Stanley Online 2001, http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.21.4117&rep=rep1&type=pdf
11. “Keep security simple”, OWASP, http://www.owasp.org/index.php/Keep_security_simple
12. "Fourth-Factor Authentication: Somebody You Know", RSA Security 2006, http://www.rsa.com/rsalabs/staff/bios/ajuels/publications/fourth-factor/ccs084-juels.pdf
13. "Biometric Technology Application Manual, Volume 1, Biometrics Basics", National Biometric Security Project 2005, http://biometricsinternational.org/downloads/documents/2007BTAMMasterVolume2.pdf

14. ”Authorization-Based Access Control for the Services Oriented Architecture”, Alan H. Karp Hewlett-Packard Laboratories 2009, http://www.hpl.hp.com/techreports/2006/HPL-2006-3.pdf
15. SilverLight FAQs, Microsoft Corporation, http://www.microsoft.com/silverlight/overview/faq.aspx
16. “About PortWise”, PortWise, http://portwise.com/index.php?option=com_content&view=article&id=150&Itemid=59
17. ”Microsoft Code Name "Geneva" Framework Whitepaper for Developers”, Microsoft Corporation 2008, http://download.microsoft.com/download/7/d/0/7d0b5166-6a8a-418a-addd-95ee9b046994/GenevaFrameworkWhitepaperForDevelopers.pdf
18. ”ComponentSpace SAML v2.0 Component - Summary”, ComponentSource, http://www.componentsource.com/products/componentspace-saml2-component/summary.html
19. “Role-Based Authorization”, Microsoft Corporation, http://www.asp.net/learn/security/tutorial-11-vb.aspx
20. “How to: Inspect or Modify Parameters”, Microsoft Corporation,http://msdn.microsoft.com/en-us/library/ms733747.aspx
21. ”Extending Metadata Using Attributes”, Microsoft Corporation, http://msdn.microsoft.com/en-us/library/5x6cd29c(VS.71).aspx
22. “Understanding LDAP”, IBM, http://keinzerberus2000.aldebaran.de/LDAPIBMUnderstandingsg244986.pdf
23. “Walkthrough: Creating and Registering a Custom HTTP Module”, Microsoft Corporation, http://msdn.microsoft.com/en-us/library/ms227673.aspx