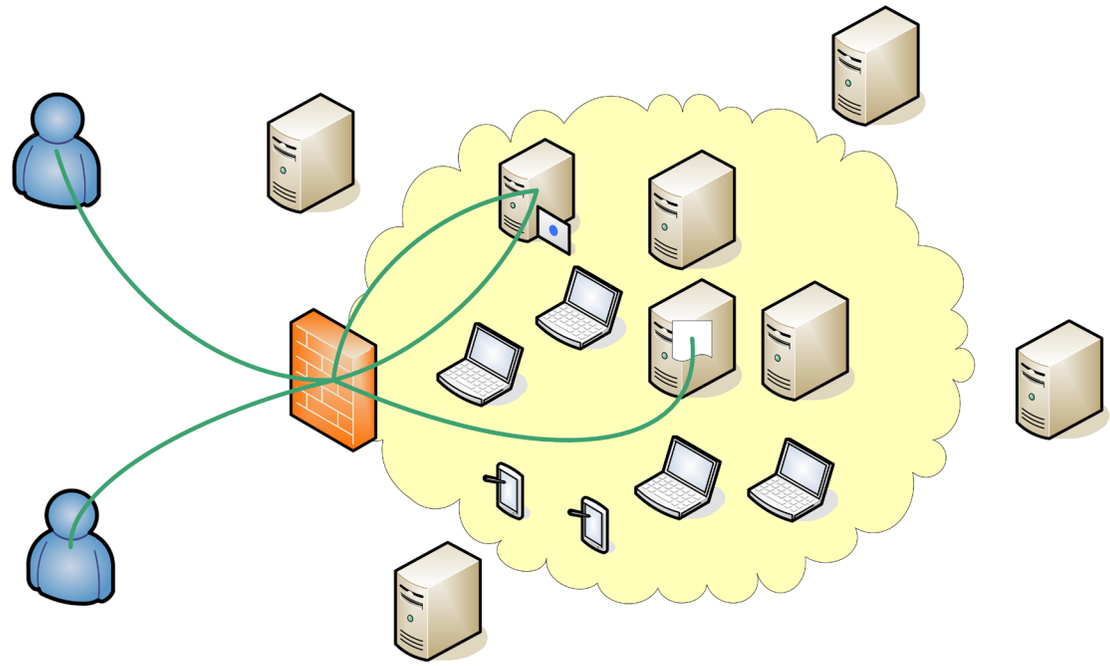
Access control in a coalition system

ALF BENGTSSON, LARS WESTERDAHL



FOI, Swedish Defence Research Agency, is a mainly assignment-funded agency under the Ministry of Defence. The core activities are research, method and technology development, as well as studies conducted in the interests of Swedish defence and the safety and security of society. The organisation employs approximately 1000 personnel of whom about 800 are scientists. This makes FOI Sweden's largest research institute. FOI gives its customers access to leading-edge expertise in a large number of fields such as security policy studies, defence and security related analyses, the assessment of various types of threat, systems for control and management of crises, protection against and management of hazardous substances, IT security and the potential offered by new sensors.

FOI
Swedish Defence Research Agency
Command and Control Systems
P.O. Box 1165
SE-581 11 Linköping

Phone: +46 13 37 80 00
Fax: +46 13 37 81 00

www.foi.se

FOI-R--2393--SE
ISSN 1650-1942

User report
December 2007

Command and Control Systems

Alf Bengtsson, Lars Westerdahl

Access control in a coalition system

Titel	Accesskontroll i koalitionssystem
Title	Access control in a coalition system
Rapportnr/Report no	FOI-R--2393--SE
Rapporttyp Report Type	Användarrapport User report
Månad/Month	December/December
Utgivningsår/Year	2007
Antal sidor/Pages	42 p
ISSN	ISSN 1650-1942
Kund/Customer	Försvarsmakten
Forskningsområde Programme area	7. Ledning med MSI 7. C4I
Delområde Subcategory	71 Ledning 71 Command, Control, Communications, Computers, Intelligence
Projektnr/Project no	E7153
Godkänd av/Approved by	Martin Rantzer

FOI, Totalförsvarets Forskningsinstitut
Avdelningen för Ledningssystem
Box 1165
581 11 Linköping

FOI, Swedish Defence Research Agency
Command and Control Systems
SE-581 11 Linköping

Sammanfattning

Den här rapporten beskriver hur en webbaserad koalition kan byggas upp med informationssäkerhet i åtanke. Tekniker för att skapa ett grovkornigt accesskontrollsystem för webbaserade applikationer presenteras.

Samarbete mellan organisationer i koalition med varandra kräver någon form av kommunikationsverktyg. För att snabbt kunna upprätta kommunikation och ledning krävs det att ingen speciell utrustning behövs utan att deltagarna skall klara sig med vad som kan uppfattas som normalt tillgängligt. Av den anledningen är ett genomgående krav att accesskontroll systemet skall kunna fungera till en webbaserad applikation.

Nyckelord: Roll-baserad accesskontroll, webbapplikationer, webbtjänster

Summary

This report describes how a web-based coalition can be formed with information security in mind. Technologies for creating a coarse-grained access control system for web-based applications are presented.

Cooperation between organizations within a coalition requires some kind of communication system. To quickly establish command and communication, it is necessary that no special equipment is needed. Members of the coalition should be able to make do with what is assumed to be commonly used. Hence, a general requirement is that the access control system should work on a web-based application.

Keywords: Role-based access control, web applications, web services

Innehållsförteckning

List of figures

1 Introduction

1.1 Problem statement

Most missions, whether they are military operations or support to the community, are executed in coalition with other forces or agencies. On peace keeping missions collaboration with foreign military forces is most common but as incidents occurs, collaboration with local authorities and non-government organisations may be necessary. Incidents may very well occur within our own borders as well, in which the armed forces will support the community.

When the armed forces collaborate with another organisation, be they foreign or domestic, it can be called a coalition. The coalition can be well planned and prepared, with letters of agreement and understanding, or it can be a quickly assembled force to meet a specific problem. This report focuses on a coalition which is formed quickly and thus do not have the luxury of long-time planning and adaptation.

Coalitions that forms after something have occurred are difficult to plan for. First of all, no-one knows what is going to be done. Second, no-one knows with whom the problem is going to be solved. With these prerequisites it is easy to realise that the command and control system have to be a system that can be generic enough to work most of the times, or flexible enough to be able to adapt to any given situation.

Armed forces, government agencies, non-government organisations and companies are all possible partners in a collaboration, all with their own way of communicating within themselves. Most of them however use computer-based systems, either for direct command and control (although the term command and control is hardly used by them all) or as a support system.

Using the computer as the common denominator, it can be assumed that they all have and use a web browser and an Internet connection. By providing web-based applications for, for instance, command and control it would be possible to form a loosely-coupled system for collaboration – a coalition.

The web-based applications would be available without enforcing any upgrades or special equipment to be bought or installed. It is thus fast to deploy.

Security-wise, such a system must be able to separate between friend and outsider. But it would also be preferable to be able to discriminate between users how the system is being used. All coalition members do not necessarily have or need the same amount of information or ability to use applications within the

system. Members may come and go as the problem is being solved and some members may even be considered enemies any other day but have to cooperate to solve the present situation.

1.2 Approach

This report presents techniques to handle access control in a web-based system. The focus is on existing and openly available technologies which can be configured and put into service within a relative short time for preparations.

A binding requirement for the technologies presented is that they should not demand anything from the client-side of the system apart of being open to an Internet connection and being able to use a regular web browser.

1.3 Organization of the report

The report is organized as follows. Chapter 2 provides a background on web technologies and presents the coalition model used throughout the report. Chapter 3 focuses on access control in web applications. In chapter 4, an application for access control on a web server is presented and analysed. Chapter 5 presents a discussion and conclusions.

2 Background

A coalition, of forces or organisations, may be formed temporarily to solve a problem or more permanently as a long-term agreement. The temporary, or more short-termed, coalition is characterised by being quick to establish and only existing as long as the problem exists. Partners may join or leave the coalition as the problem changes or partially is solved. It is not possible beforehand to decide who should be part of the coalition, as the partners depends on what the problem is and where it is. The long-term coalition is more generic in the sense that it does not aim to solve a current problem but rather, in the long run, to achieve some kind of higher goal. The time available to form the coalition naturally demands different levels of openness and flexibility in each partner's command and control system. If the time is short, it will be necessary to be able to communicate immediately without having to adapt the equipment.

This report focuses on the more temporary agreement and therefore aims to describe the properties of a temporary coalition.

2.1 Open technologies

An international crisis that calls for a multi-nation reaction is difficult to predict or plan for. A crisis may be natural such as flooding, drought, wind, famine or caused by man such as large-scale accidents. Another type of international crisis is collapsing nations and civil war. Whatever the cause, it is difficult to prepare for a supporting action. In order to provide help to the areas that is suffering, nations and private organisations must be able to cooperate to solve the issues at hand.

The collaboration may very well be between organisations that normally have very little to do with each other and, thus, have few or no prepared ways of communicating. Countries, which may even regard each other as enemies, may be forced to work side by side to solve a particular incident.

For such temporary collaborations to function, means of communication have to be established. As time for preparations is short or none, the equipment used must be common enough to be widely spread. Open communication lines and protocols, widely used technologies, and open or at least sharable formats form the baseline for communication command and control.

Modern command and control system are based on computer systems. The complexity of the system may vary from the most simple, such as an email client, to a fully integrated command and control system. Wherever on the complexity

scale the system at hand is, it still uses a computer. Information is transferred, sought and analysed, both from proprietary systems as well as openly through ordinary mail clients and search engines.

Even though computers are widely spread, the applications, and thus the format of the information, may vary. Manual support can, to a certain extent, handle different data formats. Manual methods are too slow and error prone in the long run, though. A technology that can traverse different computer system, regardless of operating system and applications, is needed. This problem exists not only within military organisations but just as well in government organisations, non-government organisations, commercial organisations and companies; how to spread information to a diverse and unknown population.

The choice falls easily on Internet technology. The protocols used in the network communication are then naturally compatible (more or less) regardless of the end system – the receiving computer. Most computers regardless of operating system are equipped with a web browser.

Using Internet technology for command and control systems balances the level of technology between organisations. A small organisation with a small budget can communicate with a large and well-financed organisation if the web browser is the interface. Also, and possibly more important, it is a technology that is ready to use immediately without having to purchase or install any equipment or applications, nor prioritise extra training.

2.2 Web applications

With the use of Rich Internet Applications, server-based applications and a web browser as the interface the functionality and likeness of a local application may be offered. This is collectively named a web application. A web application is commonly designed as a 3-tier system with a presentation layer (web browser), a logic layer (web and application servers) and a data layer (databases and file servers).

The result is an application presented through the web browser. Scripts provide the necessary functionality in the browser for the user to be able to do what would be possible to do in a locally installed application. The logic layer delivers the functionality needed and handles the interaction with the user by receiving a request and providing an answer using the data and files in the data layer.

The logic layer also provides an opportunity to control access to data, not just acting as an intermediary.

2.2.1 Web sessions

A user sends requests and receives replies from a web server during a session. The session is the time the user and the server communicate with each other. It is not necessarily an active period only. A session lasts, for instance, from the time a user logs in on the server to the time she logs out regardless of the intensity of the session.

The notion of a session, however, does not come naturally to HTTP as it is a stateless protocol. This means that a server following the protocol has no way of knowing if a user has visited the server before or if the current call is the first one. The server does not know the current state of the user, e.g. if the user has logged-in, personal preferences, etc.

Cookies can be used to induce state for the user and thus creating a session. A cookie is a small text file with a required field, `NAME=VALUE`, and some optional such as `path`, `domain`, and `expires`. Other fields are possible as long as the overall size does not exceed four kilobytes.

The cookie is created by a server and programmed with information, for instance, a log-in value. The cookie is sent to the user along with the requested web page. The server can demand that the browser includes the cookie when the browser requests further pages. With the cookie, the user can view the site as a logged-in user for as long as the cookie is valid and no log-out cookie is sent. By having a cookie containing log-in information sent to and from the web server a session is created as the user does not have to log-on again with each request.

If the optional cookie field `expires` is set to NULL (default) the cookie will be used as a session cookie and will be stored in temporary memory at the client side. When a log-out cookie is sent, the browser should remove the cookie from memory. If the cookie has a `expires=YY-MM-DD 00;00;00GMT` value encoded the cookie will remain stored on the hard drive for as long as the session is valid. These are called persistent cookies.

Just about every browser can handle cookies, although some may be configured not to.

2.3 Security issues in web applications

In a web application the middle layer provides logic and interoperability between the user and the data in the back-end system. The logic makes it possible to provide the data as a result of requests, calculations and personal preferences. It is also a possibility to protect the data. As the middle layer is responsible for

control it can validate authentication, grant and deny access, log events, and more, based on the security policy that is implemented at the moment.

The security policy is implemented as rules for how users and information are to be handled and how they are allowed to interact. Every request has to comply with the policy before it is allowed access to the information or application.

Web applications transform the application to a service. The security issues are similar to an ordinary application with the addition that the user accesses an external service instead of a local application/service. The web application will theoretically be available for anyone to reach, although it is not necessarily searchable. The application should not, however, grant everyone access to the application and to the data. Security wise this puts high demands on communication security and access control.

Cookies induce state by creating a session. The information in the cookie is sent in clear text to any server within the same domain as the issuing server. As the cookie is created by the server and the client receiving it only stores it for future use without any modification, the cookie can be stored on any computer and still keep the session alive. In other words, a cookie has no binding to the user it was created for in the first place, and thus the server has no possibility to identify if a cookie is sent from a valid user or if it is stolen and sent from an unauthorized user.

It is possible for the server to encrypt the cookie, which will protect the content of the cookie. It will not, however, stop the use of a stolen cookie. An unauthorized user does not need to understand or being able to read the content to be able to use it.

The remaining sections of this report are focused on access control.

2.4 The coalition model

For simplicity reason, a model of a coalition has been developed and is depicted in Figure 1.

The model is comprised of loosely coupled systems within the coalition and a gateway as an entrance point to the coalition. The members (users) are all physically outside the coalition system and access the coalition services through the gateway. At the gateway, a member is authenticated. After authentication, a policy decides what the member is allowed to access or do. Before a member is allowed to access resources or services the policy has to be consulted. A Policy Enforcing Point (PEP) can be located at the gateway or at each server/application. The job of the PEP is to enforce the compliance with the

request and the active policy. The PEP however, is only a guard. The decision is made by the Policy Decision Point (PDP). The PDP decides whether or not to authorize the members request based on the credentials sent by the PEP, see Figure 1.

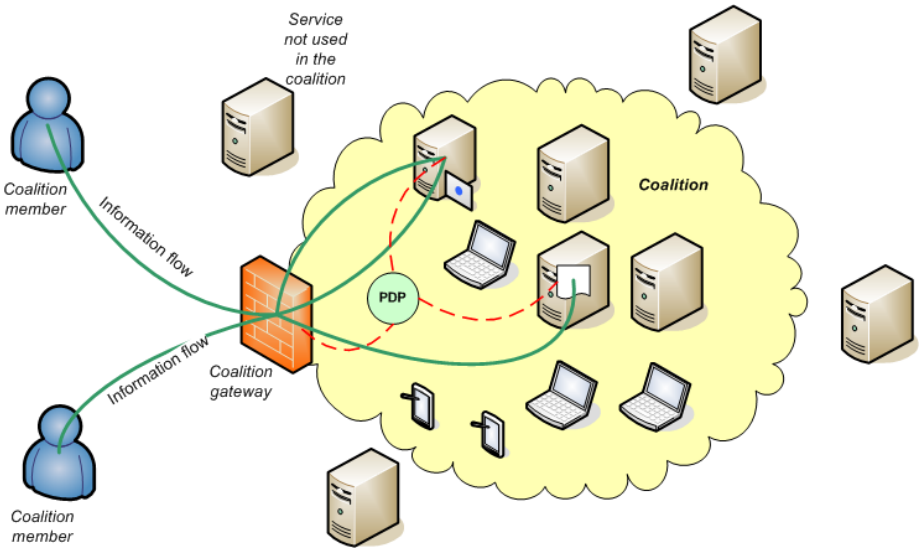


Figure 1: Schematic view of a coalition.

The level of granularity in the authorization is not defined in the model. At the most coarse grained level, the PDP may only decide if the member is allowed to access a service or a server. On a more fine grained level, the PDP would handle authorization on single objects such as, for instance, database entries.

Initially, this project will focus on access to other applications and servers.

3 Access control in web-based applications

The owner of an information system wants to share the information that resides within the system. If there is no secrecy involved, the information can be spread without any restrictions through an open channel with public access. However, if some of the information is to be delivered only to a restricted group of users some control function has to be established.

The first step, when it comes to regulating user access, is to provide an authentication mechanism. How the authentication is performed depends mostly on what is convenient and necessary. However, the amount of trust that can be placed on the user identity is not only a question of method, it is just as much a question of how the user was brought into the system. There is a huge difference in a user remotely creating an account on her own compared with a user who has to be physically present and identify herself to an administrator.

Passwords and certifications are common authentications methods. Passwords are often regarded as unsafe due to the ease with which they can be copied or forged. Certificates usually provide more information about the user compared to a password. However, a self signed certificate does not provide more assurance than a password.

3.1 Access control

Authentication confirms the identity of a user. It does not provide any information to the system of what the user is allowed to do. Different users have different needs and purposes when accessing a system. They may want to read a file, write a new file, change an existing one, delete a file or execute a program. Other possibilities are of course possible but these are just to name a few.

There is nothing however that says that a user should have total access to all resources in the system. The owner of the system may have a security policy regulating access to information and applications. The security policy forms the basis for an access control system. The access control system is an implementation of the security policy.

An access control system is comprised of subjects, objects, rights and permissions. The subject does not have to be a human user; it can just as well be an agent or another program. The object can be a data file, a database entry, a program, or whatever is accessible in the system. A subject has rights to perform

different operations on an object. An object has permissions to carry out different operations requested by the subject.

How the rights are set varies with the security model chosen. The system owner may decide how and by whom information is allowed to be accessed in the system. This is called Mandatory Access Control (MAC). In a MAC-system, the subjects do not have any special rights to an object even if the subject has created the object herself. An alternative model is the Discretionary Access Control (DAC) model. Here, it is the subject that has created the object which sets the permissions to access it.

3.2 Role-based access control

In a large system a subject usually has several duties and needs. Over time, as a subject may change her position within the organization, the duties and needs may vary. It will, quite easily, become a difficult task for a system administrator to set the appropriate rights for each individual user. Another way of viewing the subject can come in hand. A subject performs her duties based on, for instance, her position in the organization. Another alternative is to view what tasks the subject is performing. By viewing the subjects by what role they have in the system it can be easier to tie the access rights needed to that role.

A security policy is usually written by using roles as it would be infeasible to write it based on individuals. Thus it is quite easy to think of an access control system based on the same notation. It is not, however, obvious what a role is. The definition varies depending on the purpose and needs. It is easy to think of roles as group of users. In its simplest form this may work but it is not really correct. A role is a set of permissions, and subjects are assigned roles. A role focuses on the permissions whereas a group focuses on its members. A simple example of this is separation of duties. Assume that a subject is allowed to sign off on a purchase order by a subordinate but not for purchases made by herself. In the group version the subject belongs to a group that is allowed to sign the order but at the same time she is not a part of that group. Viewing the problem as a role-based system, the subject has a role as leader which permits her to sign orders. However, she cannot act in that role when ordering herself. A group is something a subject belongs to regardless of activity, whereas a role is something a subject acts in when performing specific duties.

The capabilities of a role-based system are defined in the RBAC reference model [Sandhu, 1996] (Figure 2). The base model (RBAC0) comprises of users, roles, permissions and sessions, which forms the basis for a role-based system. Users

create a session during which they may activate roles to which they belong. Each role has a set of permissions associated with it.

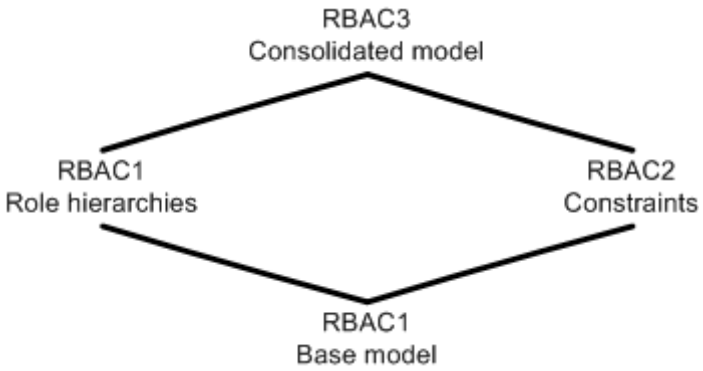


Figure 2: RBAC reference model [Sandhu, 1996]

Sometimes permissions follow the lines of authority within an organization. For this purpose the concept of role hierarchies where established (RBAC1). With a hierarchy permissions can be inherited.

Constraints (RBAC2) are extra conditions that have to be met. Here, for instance, comparisons with roles can be made, like in the example above with separation of duties.

The consolidated model (RBAC3) fulfils all the properties of the others.

3.2.1 RBAC in operation

Viewed from a high level, a RBAC-system comprises of three parts; the client, role-server and web/application server. The client acts in a certain role when requesting access to an object on the web/application server. The role server is the directory which administrates the user-role connection. The role server is accessed either by user-pull or server-pull architecture [Park et al., 2001].

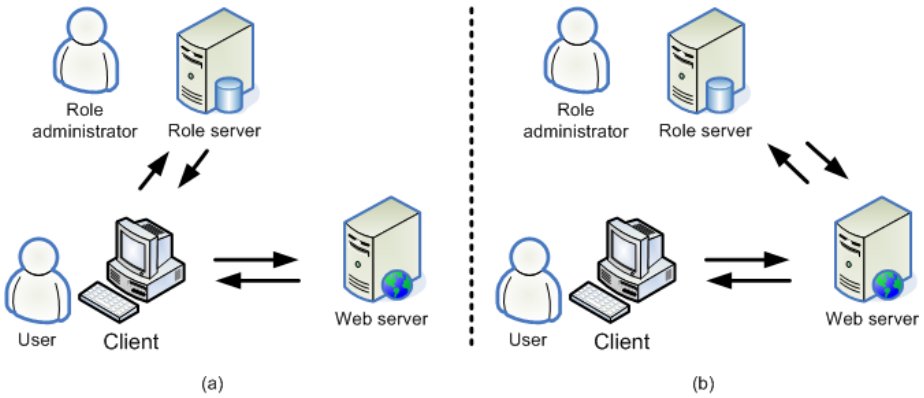


Figure 3: (a) User-pull architecture, (b) Server-pull architecture.

In the user-pull architecture, the user downloads the roles needed and admitted from the role server and stores them locally. The roles are presented to the web/application server along with a request for access. In the server-pull architecture, the web/application server pulls the roles from the role server when needed.

With user-pull, the user has to take active part in obtaining her roles. After they are stored on her local computer, she can use them in different sessions and on different web/application servers until they expire. This is a good solution when flexibility is required.

For systems where the freshness of the roles is a priority, the server-pull is preferred. It demands a reliable connection with the role server but the accuracy of the information will be higher.

3.2.2 Benefits of using RBAC

Role-bases access control can describe an organizational-wide security policy in the terms of how the organization is viewed. The main idea behind RBAC is that users do not own the objects they use [Ferraiolo et al., 1995]. Thus, a user cannot grant access to objects they use or even to those they themselves have created as in discretionary access control. However, a system that grants access to objects based on a security clearance like mandatory access control does not provide flexibility that RBAC does. RBAC-systems are process-oriented, that is, access is allowed to the objects needed to perform a duty. In a process the RBAC-system ask “who can perform what actions on what information”? In doing so, the integrity of the information is protected [Softpanorama].

The ability to describe processes in terms of roles is particularly beneficial for large organizations. It greatly reduces the labour for an administrator if a user can be assigned a set of roles instead of having to define each and everyone of the access rights the user may need. It also reduces the risk of a user being granted more privileges than was intended. RBAC typically enforces separation of duties. The system can control that a user is not given any roles which are mutually exclusive with roles the user is already a member of (static separation of duties). Also, the system can prevent a user of being granted a role which is mutually exclusive with roles the user is currently active in (dynamic separation of duties).

With the use of delegated administration, user profiles and privileges can easily be maintained and rapidly updated if needed [Murrell, 2001].

3.3 Role-based access control for the web

Web-based systems such as web sites, web services and web applications all use the HTTP-protocol for communication. As mentioned in section 2.2.1, the HTTP-protocol is stateless. Techniques for inducing state have been developed, such as cookies and URL-based sessionID. RBAC-systems are heavily dependant on the ability of preserving the state of a user and the roles in which the user acts. It is therefore natural for a web-based RBAC-system to user these techniques.

A session within a RBAC-system regulates the workspace a user can access using the available roles. It is by defining the session possible to enforce dynamic separation of duties. Using a cookie to create a session is a solution in a web environment that is common and in many cases more dynamic than using a URL-based sessionID-string. The cookie does not, however, come without controversies.

A cookie is created by a web server and used within the domain of the web server. This is not a problem for a web-based system which has all its applications, pages and files within a single domain. However, if the request is directed to another domain, the cookie will not be forwarded.

Cookies have also been criticized for containing too much information. The purpose with cookies is, among others, to allow a user to use resources on a web server without having to re-authenticate with each request. Thus, the cookie becomes the authentication credential. If the cookie is stolen, another user can use it and the web server has no way of knowing that there is another user now using the cookie. Partly another problem, although it does not prevent nor aid the authentication problem, is the expiration date of the cookie. A web server can set the expiration date in the `set-cookie` field. If no expiration date is set the

cookie will (should) be deleted when the web browser is closed down. As far as the web server is concerned, a non-expired cookie is valid and, as it is not locked to a specific host, the web server will not pay any attention to where the cookie comes from.

Work has been done by for instance Ye et al. [2007] on revocation schemes for cookies. When a user signs out of a session (and thus returning her roles) a new cookie is sent to the user. For the same user to get access again, log-in and role assignment procedure is necessary. However, for the web server there is no notion of signed-in or signed-out. If another user presents a stolen cookie, the cookie will be regarded as valid even if a log-out cookie has been sent. Ye et al. [2007] suggest a server-side cookie state database that will keep track of sign-out cookies. The cookie ID in a request will be compared to signed-out cookie IDs in the database. If the cookie ID in the request is older than the sign-out cookie ID the cookie will be rejected and the user is sent to a log-in page.

The main attraction with using web applications is for them to be accessible anywhere. Thus, users may log-on from both controlled and uncontrolled locations. The coalition should be aware of location when granting a user access. The system should take notice of how a user is authenticated (several methods may be available) and from where. Wolf & Schneider [2003] suggest that the available roles in a web-based collaboration environment should depend on how a user authenticates herself. Authentication method, user and location should all be evaluated, in order to decide the amount of trust the system is willing to grant at the moment.

4 DACS

4.1 Background

Our coalition cloud, Figure 1, essentially means that we want a point, PDP, which controls access to information, and to applications that we want to easily add, remove or modify. The rules for access control should not be overly complicated, to make configuration possible at coalition set up. This means that the access control must be general and coarse grained. Special, fine grained, access control must be carried out inside applications. The access requests should emanate from users at a standard web browser.

These requirements are not unique, so it is natural to search for what has been done elsewhere. One alternative is DACS, Distributed Access Control System [DACS (a)]. It is a Canadian project and its main implementation is for controlled sharing of information within NFIS (National Forest Information System of Canada).

DACS has many advantages. It is a free, open source, simple and flexible system. It has integrated a rich set of existing and tested components.

It has some potential disadvantages. It is only available for Unixes (Windows at ToDoList). It does not conform to OASIS standards (it rather follows the competing REST-standard). It is not military grade security scrutinized. It has no GUI-support.

4.2 Overview

DACS, Distributed Access Control System, consists of two parts. Part 1 is the integration of a configurable set of functions for single-sign-on (see section 4.3). A user, who wants to access some kind of resource that is protected by DACS, must use one of these functions to authenticate the user's identity and, optionally, to set a role to be active. The output from such a function is a credential, which is cryptographically protected. The credential can be stored between requests, for instance as a cookie in the user's browser.

Part 2 of DACS is the evaluation of access control rules (see 4.4). Each protected resource – data, executable, script … – is associated with XML-files, where access control rules are described. Inputs to DACS are these rules, the authenticated credentials, environment variables (like time of day, IP-address, etc) and revocations. Output from the evaluation is a decision; *allow* or *deny*. The

decision can be qualified by a constraint, like *allow if constraint* or *deny if_not other_constraint*. The data flow to and from DACS could be depicted as in Figure 4.

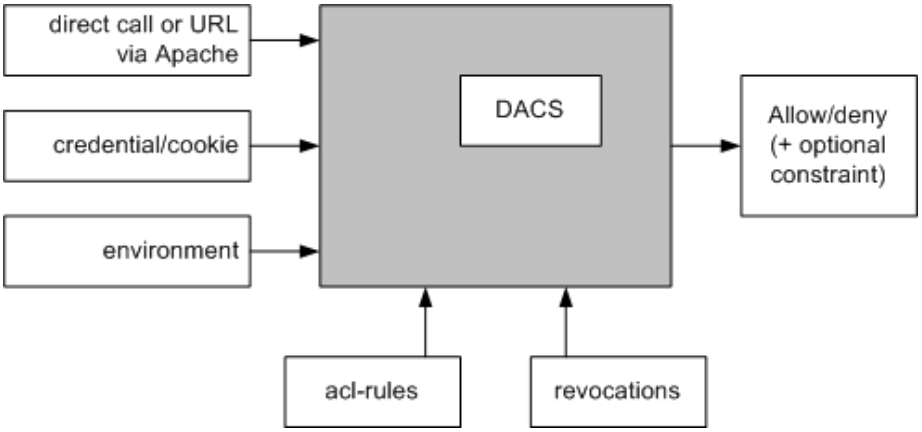


Figure 4: Data flow in DACS.

A feature, which is of particular interest in coalitions, is the federation structure of DACS (see 4.5). DACS uses a hierarchy federation-jurisdiction-user, which means that administration of users and rules can be distributed to different jurisdictions.

4.3 Single Sign On

One of the most valuable components in a distributed security system is a module for single-sign-on. In a system of systems with many applications spread over many computers, like Figure 1, it is most disadvantageous if each application uses its own method for sign-on. The signing-on should be handled by one service that could be called from all applications. The service should generate a *credential* that could be interpreted and trusted by all applications, or by a PDP that acts as a proxy for the application. A credential is a data structure which includes attributes that are essential for the access control. Authenticated user identity is an almost mandatory attribute. Other useful attributes are for instance active role and how long the credential is valid. Since the credential is crucial for the access control, it must not be easily forged. Therefore it ought to be cryptographically protected.

The single-sign-on in DACS is built upon a utility *dacsauth*, authentication check. It is a utility that can be called in the standard Unix manner – with options, stdin etc. Its exit status is binary; true or false. One of the options is *module_spec* that specifies which module should perform the actual authentication check. The set of available modules is configured when DACS is built. There is a fairly long list of Unix authentication modules that are ready to be built in. There are also hooks where a private module could be connected. It is also possible to let *module_spec* point at an external module, even across the network. In this case it is crucial to protect the communication and to set privileges etc.

The utility dacsauth can be called anywhere a Unix process can be called. But it only returns true/false, it does not generate any credential. For this, there is a DACS authentication service, *dacs_authenticate*, which can be called as a Web Service. It uses the same set of authentication modules as dacsauth does. Since input and output to dacs_authenticate are sensitive, and since it is called as a Web Service across a potentially insecure network, the call must be protected (for instance by SSL/TLS). The output, from a positive authentication by dacs_authenticate, is packaged as an encrypted credential. The same encryption key is used by all in a federation, so a credential can be decrypted anywhere inside a federation, but not outside. If dacs_authenticate was called from a web browser, the credential is returned as a cookie, which is used as input when DACS is subsequently called. The credential, when decrypted, is an XML-document. Mandatory element is authenticated user-ID. Optional elements are role, expiration time, user agent (which browser/program that was calling), IP-address. The values of these elements can be used in the evaluation of access control rules, Figure 4.

The actual authentication is done by ready modules, built in or external, which are designed, standardized and tested in other frameworks. Different groups of users can use different modules. The list of modules in [DACS (b)] includes:

- Unix authentication
- Windows/NTLM authentication
- Local password
- SSL-based X.509 certificates
- HTTP authentication
- LDAP/Active Directory
- OpenID
- Apache authentication
- about six more

To summarize the features, most important in our scenario, of DACS' single-sign-on:

- choose authentication modules from an extensive list of tested modules
- successful authentication results in encrypted credential that is stored in user's browser during work session.

4.4 Gateway/Apache and access control

As was mentioned in 4.2, part 2 of DACS is the evaluation of access control rules, Figure 4. Like the single-sign-on in section 4.3, the evaluation of rules invokes many utilities which can be called individually in standard UNIX manner. In [DACS (a)] details of the utilities are extensively documented. In this chapter our scenario, Figure 1, is in focus - how to make a gateway for web requests and how to put up a policy that controls access.

The gateway is achieved by setting up a modified web server. In theory, any web server might be modified. But since the modification is a relatively great task, the tested modification of Apache [Apache] is the practical choice. The modified DACS-wrapped-Apache can be configured to catch specified URLs and ask the DACS access control service if the request should be granted. Standard URLs point at Apache-controlled files or cgi-scripts. But what if the request points at something more complicated, like a Java web application hosted in an application server? Then the DACS-wrapped-Apache can act as a proxy front-end, which catches the request, controls access, and forwards the request to the application server. It is not obvious how fine-grained the access control can be. We plan to test this with the document management system Alfresco [Alfresco], hosted in the application server Tomcat [Tomcat].

The actual access control (compare PDP in Figure 1] is performed by the service *dacs_acs*. This means DACS access control service, which is called by a module *mod_auth_dacs* [DACS (c)] in the modified DACS-wrapped-Apache. Figure 4 is an outline of the inputs to *dacs_acs*. In [DACS (d)] an extensive set of inputs/outputs is described. Important inputs from DACS-wrapped-Apache are for instance URL (path and cgi-arguments) and credential/cookie.

Normally, the first step in the access control is to validate the credential. If it is invalid, or missing, the DACS-wrapped-Apache normally redirects the browser to a page that handles single-sign-on, section 4.3.

Then dacs_acs checks if there is any applicable deny-clauses from environment or revocations. If not, dacs_acs evaluates the access rules for the particular requested resource. In [DACS (e)] the rules are described. The rules are

expressed in a script language, described in [DACS (f)], reminiscent of TCL, Tool Command Language. The language has “the usual” set of basic data types and operators, and 59 embedded functions. A rule is a text in XML-format. Basically, a rule expresses a triple (what, who, how). The following example gives a feeling of the language.

> Every user will be able to invoke the service /cgi-bin/metalogic/group if CGI parameter OP is LIST_GROUPS or SHOW_GROUP. If OP is ADD_GROUP, DELETE_GROUP, or MODIFY_GROUP, only a member of the group SWE:admin can invoke the program. String comparisons are performed without regard to case. If OP has any other value, access will be denied.

```
<acl_rule>
     <services>
          <service url_pattern="/cgi-
          bin/metalogic/group"/>
     </services>
     <rule order="allow,deny">
          <allow>
               ${Args::OP} eq:i "LIST_GROUPS"
                    or ${Args::OP} eq:i "SHOW_GROUP"
          </allow>
          <allow>
          (${Args::OP} eq:i "ADD_GROUP"
               or ${Args::OP} eq:i "DELETE_GROUP"
               or ${Args::OP} eq:i "MODIFY_GROUP")
               and user("%SWE:admin")
          </allow>
     </rule>
</acl_rule>
```

In this example `${Args::OP}` means the value of a variable `OP` passed as an argument, `eq:i` means case insensitive string comparison and the function `user("%SWE:admin")` is true if the credential contains the group/role name `%SWE:admin`.

The rules are expressed as XML-elements, like those of XACML (chapter 5). They are easier to understand than XACML, and the script language makes it easier to formulate able access control rules. However, they must be manually edited and thoroughly tested.

The rules for access control, the acl-rules, are stored as acl-files, following a naming convention. The example used would be stored in a file named acl-group.N, where N is an integer that controls the sequence of evaluation. The files are stored in different directories which mirror the federation structure (section 4.5). Together with the sequencing, this affects the outcome of the final rule.

Altogether, DACS has potential for a competent access control. But the consequence of flexibility and manual editing is that it is error prone and requires thorough testing.

4.5 Federation Structure

Our projected application is focused on information sharing in a coalition, with members from different nations and organizations. It is a great advantage if the information system reflects the coalition structure. Each member wants control of, or at least insight in, its own information and use of the system. This is facilitated by DACS' federation structure. The structure is set up via sets of XML-formatted configuration files [DACS (g)].

The naming of identities, used for instance in access control rules, can be described as hierarchical, *ID = federation-name:jurisdiction-name:username*. [1]

The highest level is the *federation*. A federation must be the first unit to construct at configuration time. It consists of several configuration files, including pointers to underlying jurisdictions. An important object is the common key. This is used for all encryption/decryption of credentials within the whole federation. In the DACS documentation linking of federations is mentioned. But this is not elaborated, and it is not clear how flexible it would be. So every part of a coalition must probably belong to the same federation. The authority that configures the federation can be seen as a "supreme commander".

The second level is the *jurisdiction*. This is where users, groups and rules are managed. To create a jurisdiction, one must have the cryptographic key (securely!) copied from overlying federation.

[1] This hierarchy is, in principle, independent of the domain structure of Internet, e.g. resource.foi.se. But since the Internet structure affects the way standard browsers handle cookies, there must be a mapping between them. This is done in configuration files.

The lowest level is *username*. A username is unique within its jurisdiction. It is not always a name of a person. It can also be an identity of a role or of a group of users.

The most natural way to map a coalition structure to DACS' federation structure is probably to have one federation for the coalition, and one jurisdiction per nation or organization. Another possibility could be to let a military mission be the federation level. Possible ways to link federations or to have sub jurisdictions might give other openings. To conclude this, further testing is needed.

The encryption/decryption and handling of keys within a federation is important for the security. AES-128 is used for encryption, but the handling of keys needs to be further scrutinized.

4.6 Adequate improvements

The handling of roles in DACS is rather primitive. It is part of the RBAC0-level, discussed in section 3.2. It essentially results in a flexible way to group users, and to use roles in acl-rules. Describing roles, and attaching users to roles, are manual processes. It would probably be a straightforward improvement to make a web based tool for this. Likewise, it should be possible to make a tool for manual enabling and controlling of roles, for instance controls for separation of duty. It is harder to make this an automated service, similar to authentication. In this respect, it is particularly hard to withdraw an active role. It is the same problem as the classical revocation of issued and still valid certificates.

The problem with withdrawal of roles can partly be mitigated by time stamping. The DACS credentials are time stamped. It should be observed, that this time stamping is independent of the cookie lifetime mentioned in section 3.3. The credential is handled by the server/PDP, while cookie lifetime is local to the web browser.

5 Other frameworks

5.1 Standardization

DACS, as described in chapter 4, is meant to be an efficient and expressive system for distributed access control. But it is not a standard, scrutinized by some standardization body. So a relevant question is: “Why DACS and not an established standard?”.

Formulating policies and policy rules is very common and very crucial in many situations. This applies to access control, obviously, but also to many other situations, where some sort of flow is to be controlled. Examples range from network management to control of business processes. It is therefore natural that many languages for expressing policies have arisen, also for access control specifically. It is also obvious that standardization would be very valuable.

Standardization is run by a lot of organizations, at different levels and with different perspectives. This leads to the often quoted, teasing statement: "The nice thing about standards is that there are so many to choose from". A bantering overview of standards for policies can be found in a keynote speech at IEEE Policy 2006 Workshop [Anderson, 2006].

The OASIS-standard XACML [OASIS] is the apparent standard for our purposes – to formulate policy for a PDP. Section 5.2 describes XACML very roughly. Certainly, it is evident that DACS and XACML have many common points. Section 5.3 describes an alternative project, GRID/Shibboleth. Section 5.4 summarizes why we, after all, chose DACS. Two significant reasons are

- DACS is more manageable, particularly for services via web browser interface
- the similarities with XACML means that lessons learned from DACS can be transferred to XACML

5.2 XACML

XACML stands for eXtensible Access Control Markup Language, and is the standard for how to express and formulate policies and rules for access control. It is a language, thus it is not a compiled implementation like DACS. So aspects like single-sign-on, credentials etc are not covered by XACML. Such aspects are covered by other standards. But the policy rules from DACS, e.g. section 4.4, can be expressed in XACML.

XACML version 2.0 [XACML (a)] was established march 2005. The standard is split up to cover the core components [XACML (b)] and several extensions. One extension covers RBAC, Role Based Access Control [XACML (c)].

The core standard is 141 pages of definitions and XML-code. To get a feeling of XACML, and to claim similarity with DACS, a simple policy-rule is cut and pasted from the core standard.[2]

> Any user with an e-mail name in the "med.example.com" namespace is allowed to perform any action on any resource.
>
> An XACML policy consists of header information, an optional text description of the policy, a target, one or more rules and an optional set of obligations.

```
[a02] <?xml version="1.0" encoding="UTF-8"?>
[a03] <Policy
[a04] xmlns="urn:oasis:names:tc:xacml:2.0:policy:
      schema:os"
[a05] xmlns:xsi="http://www.w3.org/2001/XMLSchema-
      instance"
[a06] xsi:schemaLocation="urn:oasis:names:tc:xacml:
      2.0:policy:schema:os http://docs.oasis-
      open.org/xacml/access_control-xacml-2.0-
      policy-schema-os.xsd"
[a07] PolicyId="urn:oasis:names:tc:example:
      SimplePolicy1"
[a08] RuleCombiningAlgId="identifier:rule-combining-
      algorithm:deny-overrides">
[a09] <Description>
[a10] Medi Corp access control policy
[a11] </Description>
[a12] <Target/>
[a13] <Rule
[a14] RuleId="urn:oasis:names:tc:xacml:2.0:example:
      SimpleRule1"
[a15] Effect="Permit">
```

[2] Line [a12] is resource target. This is empty, which will match any resource. Lines [a20]-[a35] describes which target subjects will match, those with a particular e-mail name. Response context is not covered in the example. It MUST consist of "Permit", "Deny", "Indeterminate" or "NotApplicable".

```
[a16] <Description>
[a17] Any subject with an e-mail name in the
      med.example.com domain
[a18] can perform any action on any resource.
[a19] </Description>
[a20] <Target>
[a21] <Subjects>
[a22] <Subject>
[a23] <SubjectMatch
[a24] MatchId="urn:oasis:names:tc:xacml:1.0:
      function:rfc822Name-match">
[a25] <AttributeValue
[a26] DataType="http://www.w3.org/2001/
      XMLSchema#string">
[a27] med.example.com
[a28] </AttributeValue>
[a29] <SubjectAttributeDesignator
[a30] AttributeId="urn:oasis:names:tc: xacml:1.0:
      subject:subject-id"
[a31] DataType="urn:oasis:names:tc: xacml:1.0:data-
      type:rfc822Name"/>
[a32] </SubjectMatch>
[a33] </Subject>
[a34] </Subjects>
[a35] </Target>
[a36] </Rule>
[a37] </Policy>
```

The DACS triple *what* is to be accessed by *whom* and *how*, corresponds to *target resource*, *target subject* and *response context* with optional obligations.

An realistic policy, with combined rules and logical conditions, will be much more complicated. As is usual with XML, the rules get very verbose. This is not a problem, rather a prerequisite, for computer evaluation. But the need for capable tools for human interaction is obvious, even more than with DACS.

The data flow in access control is in the standard depicted as Figure 5. Its central structure is analogous to DACS Figure 4.

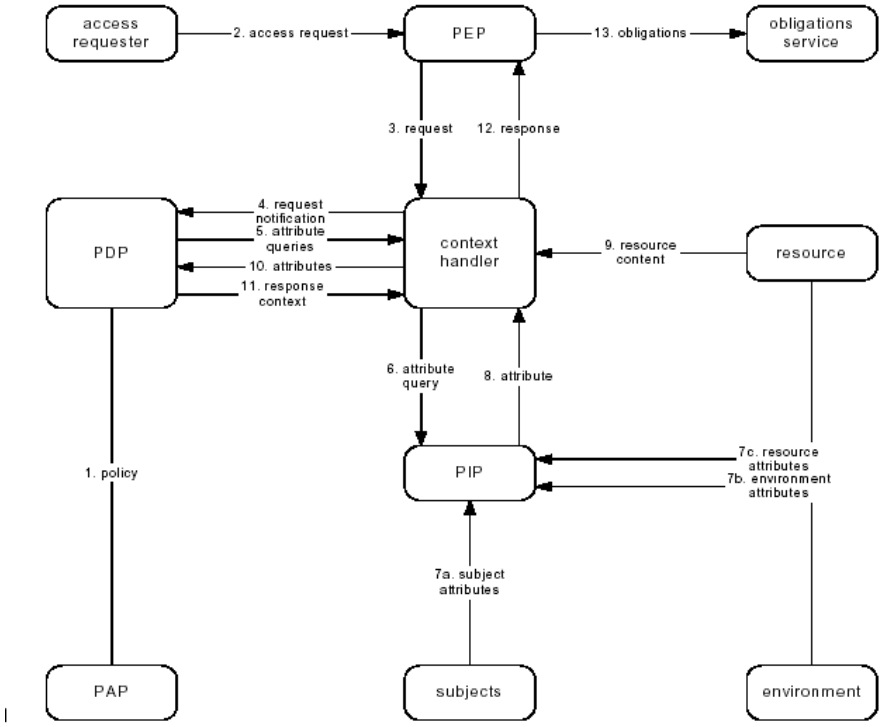


Figure 5: Data flow for policy decision.

The acronyms are:

PDP – Policy Decision Point (comparable to the central box DACS in Figure 4)

PEP – Policy Enforcement Point

PAP – Policy Administration Point, where policies are formulated (compare acl-rules in Figure 4)

PIP – Policy Information Point – where attributes (credentials, environment, …) are collected (compare credential, environment and revocations in Figure 4)

The XACML-standard is primarily about handling in the PDP. A vital part, not dealt with in the standard, is the context handler, connecting PDP and PEP. The DACS parallel would be the modifications to Apache. Outside the standard is the important PIP. The PIP, in turn, must cooperate with vital services like SSO – Single Sign On – and REA – Role Enabling Authority. The functionality in the context handler and the PIP must somehow be implemented in an application, to make it ready to call a PDP. This is likely to be problematic when existing applications must be tailored.

XACML has been implemented, more or less. The most used open source implementation is Sun's XACML Implementation[3] [Sun] in Java. It is an implementation of XACML version 1, with version 2 coming. Parts like tools, context handler etc, are not included.

5.3 GRID and Shibboleth

The goal of DACS, as described in section 4.2, is to provide users with a single-sign-on which creates attributes (credentials) that are associated to further requests for access to resources and objects. The attributes control the decision to allow/deny the access. This is a very common goal in distributed systems. In fact, it must be implemented, more or less, in any distributed system. If the system is self contained, to solve a specific task, it could be most efficiently implemented in a platform specific architecture, for instance Java security Acegi [Acegi] which is used by the document management system Alfresco [Alfresco]. But for our purpose, to quickly be able to integrate with unknown systems, the openness is most important.

The most open existing project is for collaboration in the academic community. The term GRID was coined in the early 1990's. The term was chosen as an analogy to an electric grid, since collaborating computers are connected in a network. The main focus has been to distribute computations to utilize idle time between computers. But to do so, also data must be distributed and accessible. This means information sharing with access control. Many projects have emerged, for different aspects of grid computing. The de facto standard for developers is the Globus Toolkit [Globus]. Version 4, GT4, includes a prototype XACML engine that can be configured as a PDP for access control. It also includes handling of certificates (credentials) that are fitting the standard SAML [SAML], Security Assertions Markup Language.

[3] Used for instance by SICS in their project to incorporate delegation of rights as an extension to XACML.

The basic access control in grid computing has been identity based, with access control lists where permissions are stated for user identities. This was found being hard to scale. A service is wanted for single-sign-on that assigns an authenticated user attributes, like active role, to be used in access control.

Internet2 [Internet2] is a consortium, established in 1996, of mainly universities but also commercial network companies. Their objective is to form the next generation of internet technology by research and development in high capacity networking, security, network services etc. One of their projects is Shibboleth [Shibboleth (a)]. The term shibboleth has an old Hebrew origin, it was a word that foreigners could not pronounce properly. It has now a transferred sense as “ways to recognize one of us”.

The main purpose of Shibboleth is to support ad-hoc collaboration among users across the Internet. When a user wants access to some away resource, there obviously must be some trustworthy way to give the resource provider information about the requesting user. There are quite a few initiatives around that accomplish this, one is Shibboleth. Its main target is academic users, but it also has other potential, for instance e-business. To give a short overview of Shibboleth, Figure 6 is copied from [Shibboleth (b)].

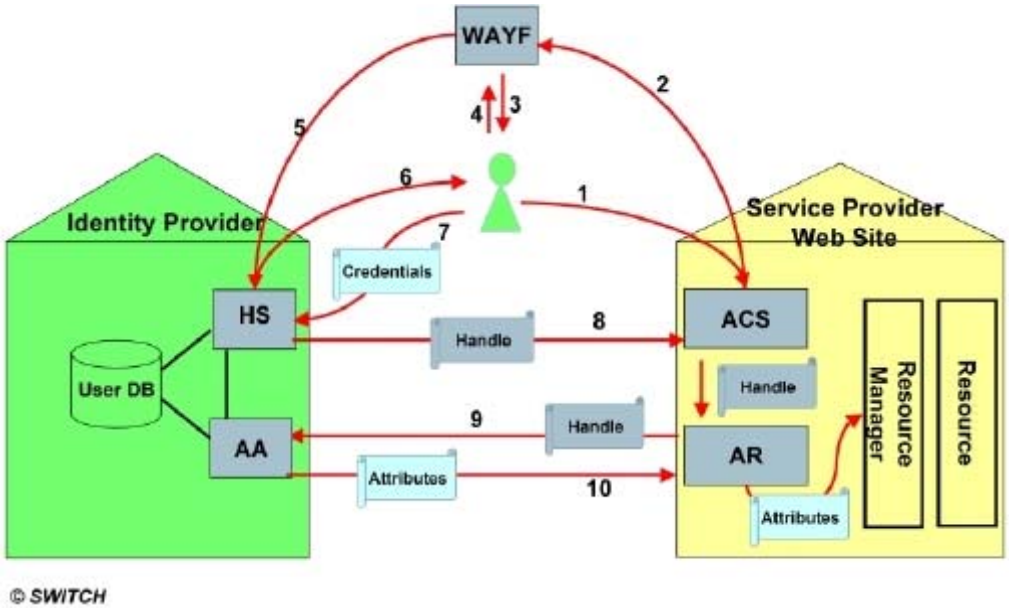


Figure 6: Data flow in Shibboleth.

Users must be registered at one, or many, Identity Provider (IP). This is where the actual authentication, by some trustworthy method, takes place. A very rough description of the flow is as follows:

- The user (in the middle of Figure 6) calls (1) SP, the service providing the requested resource. There the call is handled by ACS, the assertion consumer service. The ACS checks if it has any pointers to attributes for the user, saved from some previous call. If so, the flow jumps to (9), otherwise the user is redirected to the service WAYF, Where Are You From.
- The user tells WAYF which IP should be assigned to ascertain the user's rights to access the resource. The user is redirected to this IP (5).
- The IP demands the user to authenticate, by presenting some valid credential (7). If authentication is OK, the IP asks its Attribute Authority, AA, to create attributes which describe the user's rights, roles etc. However, the attributes are not sent directly to SP, only a pointer to the attributes (8).
- The SP asks the IP if the attributes are still valid. If so, they are sent to the SP (10). They can then be used by SP for access control, or maybe sent to a PDP, Policy Decision Point. The message with attributes follows the standard SAML.

Since grid computing needs a service for single-sign-on, it is very natural to merge Shibboleth with the Globus Toolkit. This is also under way, in the project GridShib [GridShib]. For one thing, it can use attributes from a European grid project, VOMS, Virtual Organization Membership Service [VOMS].

The GridShib initiative has many similarities with our scenario. Among its qualities, the two most attractive for us are:

- The user dialog in Shibboleth, steps 1-7 in Figure 6, can be handled from a standard web browser (which is also true for DACS)
- Established standards, notably XACML and SAML, can be used (not true for DACS).

However, there are also aspects that make GridShib less attractive:

- The aspiration to make it applicable globally, to the whole Internet, results in properties not essential in a controlled coalition
- This also means a lot of suggested extensions, which results in "a moving target"
- The total complexity is orders of magnitudes larger than DACS

5.4 Summary

The two main alternatives for our scenario are DACS, described in chapter 4, and GridShib, described in sections 5.2 and 5.3. Each alternative has its pros and cons. Table 1 is a very rough summary of our assessments.

Table 1: Comparison of DACS and GridShib.

	DACS	GridShib
Browser dialog	+(+)	+
Standards compliant	-	++
Single Sign On	+(+)	+
Complexity	+	--
Build and configure	++	-

The (+) is due to our, not verified, feeling that both browser dialog and single sign on are more directly integrated in DACS.

The most significant advantage of GridShib is that XACML is used as policy language. Standard compliance is obviously a very important aspect in coalitions. But DACS' policy language is also XML-based, and we think that lessons learned from DACS-policies could be transformed to XACML-policies.

The greatest advantages of DACS are lower complexity and that it is much easier to build and configure. This outweighs GridShib's advantages.

6 Discussion and conclusion

As was discussed in chapters 2 and 3, our overarching goal has been to implement access control in a future information system supporting an unknown coalition. A set of requirements for such a system can be expressed, and the requirements lead to some consequences. The most important requirements are:

- It should be easy to connect existing and legacy systems. This implies use of Internet technology.
- It should be easily used and managed, no long training should be necessary. This implies use of pervasive components, like standard web browsers.
- It should be built from COTS and affordable and standardized components. This is accomplished through web applications.
- It should be possible to have role based access control to the information. The access control should be as fine grained as possible, but most important is that it should be easily managed.

In this report we describe two evolving architectures, the two that arguably best fulfils the requirements. One is GRID/Shibboleth, used for information sharing in the academic society. The other is DACS, Distributed Access Control System. Each of them has its pros and cons. The main shortcoming of DACS is that it is not fully compliant with emerging Web Services standards. We argue that this is outweighed by DACS being much simpler and more versatile. Above all, it is so alike the standards, that lessons learned from DACS should be easily transformed to other systems. We therefore intend to use DACS in future work.

In the report we do not describe any implemented coalition system based on DACS. But we have implemented and tested DACS to an extent that we consider future work to be fruitful. We are fully aware, that an implementation will not comply with military grade requirements for confidentiality of secret data. We rather believe that such requirements cannot be fulfilled by a system that is quickly set up with the requirements above. Secret data must be end-end protected by some method for object security, where a secret information object is encrypted at the data source and not decrypted until it reaches the final destination. Our systems architecture does not hinder such methods.

6.1 Future work

A type of application, most likely to be used in coalitions, is DMS and CMS [Westerdahl & Bengtsson, 2006]. It stands for Document Management Systems and Content Management Systems, respectively. We have installed Alfresco, a

Java-based DMS/CMS, and we know that DACS can be used as a front system to Alfresco. We intend to balance coarse grained access control in DACS, and fine grained in Alfreso, and document lessons learned.

DACS contains role handling, but in a rather minimal way. We would like to enhance the capabilities for role handling.

7 References

7.1 Papers and presentation

Andersson, A. (2006), ”Policies in the Alphabet Soup”, IEEE Policy 2006 Workshop, London, Canada, 5-7 June 2006
URL: http://research.sun.com/projects/xacml/POLICY06_keynote.pdf (2007-11-23)

Ferraiolo, D.F., Cugini, J.A., Kuhn, D.R. (1995), "Role-Based Access Control (RBAC): Features and Motivations", in Proceedings of the 11th Annual Computer Security Applications Conference, New Orleans, Louisianna, 11-15 December 1995
URL: http://hissa.ncsl.nist.gov/rbac/newpaper/rbac.html (2007-12-03)

Murrell, L. (2001), Role-based access control has benefits for security, August 1 2001

URL: http://securitysolutions.com/mag/security_rolebased_access_control (2007-12-03)

Park, J.S., Sandhu, R.S. (2001), “Role-based Access Control on the Web”, ACM Transactions on Information and System Security, vol. 4, no. 1, February 2001, pp. 37-71

Sandhu, R.S., Coyne, E.J., Fernstein, H.L., Youman, C.E. (1996), “Role-Based Access Control Models”, IEEE Computer, vol. 29, no. 2, February 1996, pp. 38-47

Ye, R., Chan, A., Zhu, F. (2007) “Efficient Cookie Revocation for Web Authentication”, IJCSNS International Journal of Computer Science and Network Security, vol. 7, no. 1, January 2007
URL: http://paper.ijcsns.org/07_book/200701/200701B17.pdf (2007-12-03)

Westerdahl, L., Bengtsson, A. (2006) “Publicering i webbapplikationer”, FOI-R--2142--SE, November 2006

Wolf, R., Schneider, M. (2003), “Context-dependent Access Control for Web-based Collaboration Environments with Role-based approach”, in Proceedings of the Second International Workshop on Mathematical Methods, Models, and Architectures for Computer Network Security (MMM-ACNS 2003), St. Petersburg, Russia, September 21-23, 2003 URL: http://www.sicari.de/fileadmin/content/verwandteArbeiten/context_dependent_collaboration_mmm-acns03.pdf (2007-12-03)

7.2 Web pages

Acegi
Acegi Security System for Spring
URL: http://www.acegisecurity.org/ (2007-11-23)

Alfresco
Alfresco the Open Source Alternative for Enterprise Content Management
URL: http://www.alfresco.com/ (2007-11-23)

Apache
Apache HTTP Server Project
URL: http://httpd.apache.org/ (2007-11-23)

DACS (a)
DACS: The Distributed Access Control System
URL: http://dacs.dss.ca/ (2007-11-23)

DACS (b)
dacs_authenticate — DACS authentication service
URL: http://dacs.dss.ca/man/dacs_authenticate.8.html (2007-11-23)

DACS (c)
Apache/DACS authentication and authorization module
URL: http://dacs.dss.ca/man/mod_auth_dacs.html (2007-11-23)

DACS (d)
DACS access control service
URL: http://dacs.dss.ca/man/dacs_acs.8.html (2007-11-23)

DACS (e)
DACS access control rules
URL: http://dacs.dss.ca/man/dacs.acls.5.html (2007-11-23)

DACS (f)
DACS expression language
URL: http://dacs.dss.ca/man/dacs.exprs.5.html (2007-11-23)

DACS (g)
DACS configuration files and directives
URL: http://dacs.dss.ca/man/dacs.conf.5.html (2007-11-23)

Globus
The Globus Toolkit
URL: http://www.globus.org/toolkit/ (2007-11-23)

GridShib
GridShib
URL: http://gridshib.globus.org/about.html (2007-11-23)

Internet2
Internet2
URL: http://www.internet2.edu/ (2007-11-23)

OASIS
OASIS eXtensible Access Control Markup Language (XACML)
URL: http://www.oasis-open.org/committees/tc_home.php?
wg_abbrev=xacml (2007-11-23)

SAML
Assertions and Protocols for the OASIS, Security Assertion Markup Language (SAML) V2.0, OASIS Standard, 15 March 2005
URL: http://docs.oasis-open.org/security/saml/v2.0/saml-core-2.0-os.pdf (2007-11-23)

Shibboleth (a)
Shibboleth
URL: http://shibboleth.internet2.edu/ (2007-11-23)

Shibboleth (b)
Shibboleth Technical Overview
URL: http://shibboleth.internet2.edu/shib-tech-intro.html (2007-11-23)

Softpanorama
RBAC, SOX and Role Engineering in Large Organizations
URL: http://www.softpanorama.org/Access_control/
role_engineering.shtml (2007-11-26)

Sun
Sun's XACML Implementation
URL: http://sunxacml.sourceforge.net/index.html (2007-11-23)

Tomcat
Apache Tomcat
URL: http://tomcat.apache.org/ (2007-11-23)

XACML (a)
XACML 2.0 Access Control Markup Language
URL: http://xml.coverpages.org/XACMLv20-Standard.html (2007-11-23)

XACML (b)
eXtensible Access Control Markup Language, (XACML) Version 2.0, OASIS Standard, 1 February 2005
URL: http://docs.oasis-open.org/xacml/2.0/access_control-xacml-2.0-core-spec-os.pdf (2007-11-23)

XACML (c)
Core and hierarchical role based access control (RBAC) profile of XACML v2.0, OASIS Standard, 1 February 2005
URL: http://docs.oasis-open.org/xacml/2.0/access_control-xacml-2.0-rbac-profile1-spec-os.pdf (2007-11-23)

VOMS
VOMS: Virtual Organization Membership Service
URL: http://www.globus.org/grid_software/security/voms.php (2007-11-23)