Institutionen för systemteknik
Department of Electrical Engineering

Examensarbete
Role-based access control and single sign-on for Web services

Master thesis performed in information coding
by

Jerry Falkcrona

LITH-ISY-EX--08/4107--SE
Linköping 2008-03-20

TEKNISKA HÖGSKOLAN
LINKÖPINGS UNIVERSITET

Department of Electrical Engineering
Linköping University
S-581 83 Linköping, Sweden

Linköpings tekniska högskola
Institutionen för systemteknik
581 83 Linköping
Role-Based access control and single sign-on for Web services

Master thesis in information coding
at Linköping Institute of Technology
by

Jerry Falkcrona
LITH-ISY-EX--08/4107--SE

Supervisor: Lars Westerdahl
Examiner: Viiveke Fåk
Linköping 2008-03-20
Abstract

Nowadays, the need for sharing information between different systems in a secure manner is common, not only in the corporate world but also in the military world. This information often resides at different locations, creating a distributed system. In order to share information in a secure manner in a distributed system, credentials are often used to attain authorization.

This thesis examines how such a distributed system for sharing information can be realized, using the technology readily available today. Accounting to the results of this examination a basic system is implemented, after which the overall security of the system is evaluated. The thesis then presents possible extensions and improvements that can be done in future implementations.

The result shows that dynamic roles do not easily integrate with a single sign-on system. Combining the two technologies leads to several synchronization issues, where some are non-trivial to solve.

Number of pages: 43

Keywords
DACS, Single sign-on, RBAC, Web services, authentication
Abstract

Nowadays, the need for sharing information between different systems in a secure manner is common, not only in the corporate world but also in the military world. This information often resides at different locations, creating a distributed system. In order to share information in a secure manner in a distributed system, credentials are often used to attain authorization.

This thesis examines how such a distributed system for sharing information can be realized, using the technology readily available today. Accounting to the results of this examination a basic system is implemented, after which the overall security of the system is evaluated. The thesis then presents possible extensions and improvements that can be done in future implementations.

The result shows that dynamic roles do not easily integrate with a single sign-on system. Combining the two technologies leads to several synchronization issues, where some are non-trivial to solve.
Contents

1. INTRODUCTION ................................................................................................................................. 1
   1.1 BACKGROUND ................................................................................................................................. 1
   1.2 PURPOSE ........................................................................................................................................ 2
   1.3 METHODOLOGY ............................................................................................................................. 2
   1.4 SCOPE AND LIMITATIONS ........................................................................................................... 2
   1.5 THESIS OUTLINE .......................................................................................................................... 3

2. THEORY .................................................................................................................................................. 5
   2.1 ACCESS CONTROL .......................................................................................................................... 5
      2.1.1 Role-Based Access Control ...................................................................................................... 6
   2.2 HTTP COOKIES ............................................................................................................................. 6
   2.3 SINGLE SIGN-ON ........................................................................................................................... 8

3. DACS ..................................................................................................................................................... 9
   3.1 INTRODUCTION ............................................................................................................................... 9
   3.2 ARCHITECTURE ............................................................................................................................. 9
   3.3 AUTHENTICATION .......................................................................................................................... 11
   3.4 ACCESS CONTROL ......................................................................................................................... 13
   3.5 SINGLE SIGN-ON ........................................................................................................................... 14

4. SYSTEM IMPLEMENTATION ................................................................................................................. 17
   4.1 OVERVIEW ...................................................................................................................................... 17
      4.1.1 System structure ....................................................................................................................... 17
      4.1.2 Software .................................................................................................................................. 18
      4.1.3 Roles ....................................................................................................................................... 18
      4.1.4 Single sign-on ......................................................................................................................... 19
   4.2 SYSTEM SETUP .............................................................................................................................. 19
      4.2.1 DACS configuration ................................................................................................................. 19
      4.2.2 Apache configuration .............................................................................................................. 20
   4.3 AUTHENTICATING JURISDICTION ................................................................................................. 21
      4.3.1 Authentication ......................................................................................................................... 21
      4.3.2 Roles ....................................................................................................................................... 22
      4.3.3 Web service ............................................................................................................................ 25
   4.4 APPLICATION JURISDICTION ......................................................................................................... 25
      4.4.1 Access control for Web services .............................................................................................. 25
      4.4.2 Alfresco and Apache Tomcat .................................................................................................. 26
      4.4.3 Connecting Apache and Apache Tomcat ................................................................................. 27
      4.4.4 Single sign-on ......................................................................................................................... 28
   4.5 EXTENDING THE SYSTEM ........................................................................................................... 29
      4.5.1 Adding Web services .............................................................................................................. 29
      4.5.2 Adding jurisdictions .............................................................................................................. 30

5. DISCUSSION ......................................................................................................................................... 31
   5.1 IMPLEMENTATION COMPONENTS ................................................................................................. 31
   5.2 SYSTEM LIMITATIONS AND POSSIBLE SOLUTIONS .................................................................. 31
   5.3 SECURITY EVALUATION .................................................................................................................. 32
      5.3.1 Credential revocation .............................................................................................................. 32
      5.3.2 Federation key revocation ...................................................................................................... 33
      5.3.3 Theft and tampering of credentials ....................................................................................... 33
      5.3.4 Replay attacks ....................................................................................................................... 34
      5.3.5 Alternative paths to resources ............................................................................................. 34
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>6. RESULTS</td>
<td>37</td>
</tr>
<tr>
<td>6.1 SECURITY CONSIDERATIONS</td>
<td>37</td>
</tr>
<tr>
<td>6.2 CONCLUSIONS</td>
<td>37</td>
</tr>
<tr>
<td>6.3 FUTURE WORK</td>
<td>37</td>
</tr>
<tr>
<td>BIBLIOGRAPHY</td>
<td>39</td>
</tr>
<tr>
<td>PUBLICATIONS AND BOOKS</td>
<td>39</td>
</tr>
<tr>
<td>MANUAL PAGES</td>
<td>39</td>
</tr>
<tr>
<td>RFC</td>
<td>40</td>
</tr>
<tr>
<td>INTERNET RESOURCES</td>
<td>40</td>
</tr>
<tr>
<td>APPENDIX A</td>
<td>41</td>
</tr>
</tbody>
</table>
Figures

FIGURE 1: HTTP SET COOKIE HEADER .............................................................................................................. 7
FIGURE 2: SINGLE SIGN-ON USING HTTP COOKIES .......................................................................................... 8
FIGURE 3: DACS HIERARCHY ................................................................................................................................ 10
FIGURE 4: AUTHENTICATION METHODS ......................................................................................................... 11
FIGURE 5: STANDARD AUTHENTICATION MODULES FOR DACS ................................................................ 12
FIGURE 6: DACS CREDENTIAL CONTENTS .......................................................................................................... 12
FIGURE 7: A SIMPLE ACCESS CONTROL RULE ............................................................................................. 13
FIGURE 8: ACCESS CONTROL RULE IMPLEMENTING A DACS EXPRESSION .................................................... 14
FIGURE 9: SINGLE SIGN-ON USING DACS .......................................................................................................... 15
FIGURE 10: SYSTEM STRUCTURE OVERVIEW .................................................................................................. 17
FIGURE 11: JURISDICTION CONFIGURATION .................................................................................................. 20
FIGURE 12: VIRTUAL HOST CONFIGURATION FOR APACHE ........................................................................... 21
FIGURE 13: AUTHENTICATION CONFIGURATION EXAMPLE .......................................................................... 22
FIGURE 14: AUTHENTICATION CONFIGURATION USING A CUSTOM ROLE MODULE .................................... 23
FIGURE 15: AUTHENTICATION PROCESS FOR A USER SIGNING ON TO A ROLE ........................................ 24
FIGURE 16: RECORD OF CURRENTLY AUTHENTICATED USERS ....................................................................... 24
FIGURE 17: AUTHENTICATION WEB SERVICE .................................................................................................. 25
FIGURE 18: SAMPLE ACCESS CONTROL RULE FOR A WEB SERVICE .......................................................... 26
FIGURE 19: CONNECTION BETWEEN APACHE AND APACHE TOMCAT .................................................................. 27
FIGURE 20: MOD_JK CONFIGURATION FILE EXCERPT ..................................................................................... 27
FIGURE 21: APACHE CONFIGURATION PROVIDING ACCESS TO A WEB SERVICE ....................................... 28
FIGURE 22: EXPORTING DACS ENVIRONMENT VARIABLES TO MOD_JK .......................................................... 28
FIGURE 23: CONFIGURATION ADDING A NEW WEB SERVICE ......................................................................... 29
AES – Advanced Encryption Standard
AJAX – Asynchronous JavaScript and XML
AJP – Apache JServe Protocol
CAS – Central Authentication Service
CGI – Common Gateway Interface
CMS – Content Management System
DAC – Discretionary Access Control
DACS – Distributed Access Control System
DNS – Domain Name System
HMAC – Keyed-Hash Message Authentication Code
HTML – Hypertext Markup Language
HTTP – Hypertext Transfer Protocol
JSON – JavaScript Object Notation
LDAP – Lightweight Directory Access Protocol
MAC – Mandatory Access Control
MITM – Man-in-the-middle
NTP – Network Time Protocol
PAM – Pluggable Authentication Modules
RBAC – Role-Based Access Control
RFC – Request for Comments
SSL – Secure Sockets Layer
TLS – Transport Layer Security
URI – Uniform Resource Identifier
WWW – World Wide Web
XML – Extensible Markup Language
1.1 Background

The Swedish Armed Forces are participating more and more in international efforts, leading to coalitions of combined units with foreign nations. Collaboration between nations illustrates the need of an effective way of electronically sharing information. Since different nations use different systems and software, a system for sharing information accessible from as many computer environments as possible is desired. Another use of such a system is to share information in the event of a natural disaster.

Modifying or installing new software in military systems is generally very restricted and therefore should be avoided, if possible. It is also important that the participating nations feel that they can trust the new system in order to be willing to use it for sharing information. Thus, a new system for sharing information between different nations should to be built on current available technology, since it is well known and can easily be obtained. One obvious choice of currently available technology that meets these requirements and is platform independent is the well known WWW (World Wide Web). By using Web technology, clients only need a Web browser in order to be able to access the system.

Since the purpose of such a system is to share information between nations, possibly sensitive, the system has to handle authentication and authorization for information residing at different locations. It is also desired to have a fast and efficient setup and configuration. Due to the distributed nature of the system, dynamic handling of hosts sharing information is also needed.

In order for the system to be applicable for military purposes the use of role based access control is desirable. Using roles for access control suits the hierarchical military system, but it also has another use; users can be replaced by simply assigning their roles to other users.

Throughout this thesis, the term Web service will often be encountered. Web service in this thesis corresponds to REST (Representational State Transfer) Web services. REST is a network architecture style that aims to minimize network communication and latency. [Fielding, Taylor, 2002]

The REST architecture is much more lightweight than the ones specified by OASIS, which demands XML-formatted messages exchanged by the SOAP protocol.

1 For further details on OASIS, see http://www.oasis-open.org
1.2 Purpose

This thesis aims to examine how an information sharing system such as the one described in section 1.1 can be realized, using the general access control utility DACS (Distributed Access Control System).

An additional purpose of this thesis is to implement a concept system fulfilling the properties and functionality described in section 1.1. The implemented conceptual system is to be evaluated from a security point of view and possible future improvements are to be presented.

1.3 Methodology

Initially, documentation describing the background information regarding why a Web-based authentication system is needed was studied. Documentation describing the general architecture and requirements of such a system was also studied. After that, the theory behind the technologies to be used was studied, leading to a conclusion about the overall architecture of the system to be implemented.

On the basis of this knowledge, a concept system was implemented. The implementation was done in several stages, where each stage improved the functionality from the previous stage. When the implementation was complete the system was evaluated. The overall security of the system was evaluated, trying to uncover potential weaknesses. Based on the evaluation, possible future improvements and extensions were presented. An installation guide, describing how to set up the implemented system was also created.

1.4 Scope and limitations

- The implemented system is not intended to be a finished product, but an evaluation system showing the basic desired functionality.

- This thesis only covers the use of DACS for access control and authentication, since it was a requirement from the company the conceptual system was designed for.

- Role specifications and role assignment rule definitions are not handled; it is outside the scope of this thesis.

- Methods for distributing cryptographic keys are not covered in this thesis.

- It is assumed that the reader has basic knowledge in computer security.

---


3 A. Bengtsson, L. Westerdahl (2006), ”Publicering i webbapplikationer”, FOI-R—2142—SE
1.5 Thesis outline

Chapter 1 presents the background and purpose of this thesis. The scope and limitations of the thesis are also described. Chapter 2 describes the basic theory of the technologies that are used in the implemented system. The later chapters of this thesis assume that the reader is familiar with this theory. Chapter 3 presents DACS; the system used for authentication and access control. Chapter 4 presents the system implementation. Overall system architecture and technical details of the implementation are described. Chapter 5 discusses the overall security and limitations of the implemented system. Chapter 6 presents the results of the thesis; the possible future improvements and security issues.
2.1 Access Control

Access control is a form of authorization; it determines how and when resources are allowed to be accessed. In order for an access control system to be able to determine whether a user should be granted access or not to a resource, the user must be authenticated.

Access control methods usually are described formally, using the notations of users, subjects, objects, operations and permissions as well as the relations between them. The term users corresponds to human users, while the term subjects corresponds to computer programs acting on behalf of users. [Ferraiolo et al, 2003]

There is a well known principle when configuring access control for a system; the principle of least privilege. By following this principle, users are not assigned higher privileges than they need to perform their tasks. Should this principle be followed strictly, users would possess different permissions at different times depending on their current task. [Ferraiolo et al, 2003]

There are two main types of access control, defined by the U.S. Department of Defense in 1983; Discretionary Access Control (DAC) and Mandatory Access Control (MAC). Discretionary Access Control lets the users own objects in the sense that they can control who should be allowed to access their objects. As users in military and corporate systems normally do not own the information they are working with, DAC is not often a suitable access control method.

“Discretionary Access Control – A means of restricting access to objects based on the identity of subjects and/or groups to which they belong. The controls are discretionary in the sense that a subject with a certain access permission is capable of passing that permission (perhaps indirectly) on to any other subject (unless restrained by mandatory access control).” [U.S. DoD, 1985]

Mandatory Access Control does not allow users to change permissions of objects, preventing them from giving away access to objects to other users. In a system using MAC, objects are owned by the system itself rather than by the users. As information can not be passed to lower security levels, MAC is often more suitable for military systems than DAC.

“Mandatory Access Control – A means of restricting access to objects based on the sensitivity (as represented by a label) of information contained in the objects and the formal
authorization (i.e., clearance) of subjects to access information of such sensitivity.” [U.S. DoD, 1985]

In many organisations it is not very efficient to set access control rules using static access control lists. Should users need to be added, removed or be given new access permissions, the access control lists have to be edited. The least privilege principle is also hard to maintain when privilege changes involves editing access control lists. Section 2.1.1 describes a solution to those problems presented in 1992 by D. Ferraiolo and R. Kuhn; Role-Based Access Control. [Ferraiolo, Kuhn, 1992]

2.1.1 Role-Based Access Control

Role-Based Access Control (RBAC) takes advantage of the fact that individuals in an organisation usually have roles assigned. Access control decisions are based on which roles users are acting in, rather than individual user permissions.

When an organisation is implementing RBAC, an RBAC framework is normally created. A RBAC framework describes which roles are needed, as well as which objects specific roles need access to. The RBAC framework should also describe which operations given roles need to perform on objects. Because the access control configuration specifies which roles are entitled to do what, RBAC is said to be a weak form of mandatory access control. Thus, users can not give away access to objects themselves.

Administrating the privileges of users now simply consists of deciding which roles a specific user should possess, and then assign the user the corresponding roles in the system. Should users get new responsibilities within the organisation, the only thing an administrator has to do to keep the privileges up to date, is to edit which roles the users are assigned to in the system. [NIST CSL, 1995]

A role can consist of other roles, making it possible to create hierarchies of roles. Role hierarchies can be used to make the overall role framework more foreseeable and easier to manage. It also suits the military hierarchy where an individual of a higher rank usually indirectly possesses the privileges of a lower rank.

In order to enforce the least privilege principle more efficiently, users are only assigned the roles they need to be able to perform their tasks. Since users normally have different tasks to perform at different times, they sometimes possess roles with greater privileges than needed to complete the current task. To solve this, users can be assigned roles dynamically; they are only assigned the roles needed to complete the current task in a session. [Ferraiolo et al, 2003]

2.2 HTTP Cookies

The hypertext transfer protocol (HTTP) is a stateless protocol by design, meaning that the HTTP server keeps no relation between consecutive HTTP requests. The stateless design makes HTTP a fast protocol for communicating hypermedia resources. The HTTP protocol is a simple request/response protocol, where the communication is mostly initiated by the client. [RFC 2068]
As the World-Wide Web grew larger and became more popular, it was realized that support for keeping a connection state in HTTP sessions was needed, to be able to create more dynamic content. With support for HTTP states it would be possible to create Internet shopping carts remembering the items placed in them. It would also be possible to keep track of user customizations for websites. In order to solve this, a proposed specification for handling HTTP sessions was presented by Netscape Communications in 1994. [Netscape, 1994]

The proposed solution is using small text files, initially sent to the clients from the server. The text file, also called a cookie, is then sent back to the originating server whenever the client sends another HTTP request to the server. By adjusting the information in the cookie and sending it back to the client, the server is able to save the state. The state can be tracked because the information set by the server is sent back along with each HTTP request from the client. The method became more and more popular and eventually evolved into an Internet standard in 1997. The standard states that a client only should send back a cookie to a server residing on the same domain as the domain the cookie was received from. [RFC 2109]

Since the HTTP protocol is a clear text protocol, the contents of cookies could be intercepted by a malicious user. In order to minimize the risk of getting cookie contents stolen, the connection to the server should be via SSL/TLS. The cookie standard defines an option called secure, which tells the browser not to send the cookie using a connection less secure than the connection the cookie was received on. Using this option, clients that have received a cookie through an SSL/TLS connection will not send it through a normal unencrypted HTTP connection. Figure 1 presents a typical Set Cookie header sent from a server to a client using the secure option.

```
Set-Cookie: OPTION=value; path=/; expires=Monday, 23-Jan-08
21:12:43 GMT; secure
```

Figure 1: HTTP Set Cookie header.

A common use for cookies nowadays is to let users send credentials, proving that they have been authenticated. After authenticating and accessing a protected Web service, the user obtains a cookie containing information stating that the user has been authenticated, and possibly authentication details. The information in the cookie is often encrypted or signed so that the server can detect changed or faked cookies. The client attaches this cookie along with the HTTP requests, letting the server know that the client has already been authenticated and does not need to be authenticated again.

Since cookies consist of text files stored at a client, caution must be taken when using them as credentials proving authentication. Even if the cookie contents have been encrypted using a cryptography key only the server has access to, nothing can prevent the client from sharing the cookie with other clients. Several solutions exist that can make sharing of cookies among unauthenticated clients more complicated. One solution would be to embed the IP address of the client into the cookie at authentication time. The server can now verify that the credential is sent from the same IP address as the one authenticating. A problem with this method is that a client with a dynamic IP address might be considered unauthenticated if a new IP address has been obtained since the authentication. The lifetime for the credentials should, however, not be too long, minimizing the risk of having valid credentials stolen.
2.3 Single sign-on

One of the most popular ways for a user to authenticate nowadays is to provide a username and a password. The username and password combination are often unique to each service; leading to a lot of passwords to remember, as users normally need to access a variety of different services. Single sign-on is a technique created to reduce the amount of authentications needed in a system to a minimum. A single sign-on system handles the authentication for services without user interaction.

The use of single sign-on can benefit both the users and the overall system security. The benefit for users is that they need to remember fewer passwords, increasing the possibility to choose strong passwords. The authentication method used by the single sign-on system can also, and should if possible, be improved as it is the main authentication point in the system. [Novell, 2006]

There are several different methods that can be used by a single sign-on system to achieve single sign-on properties. The ticket based method, used in Kerberos from MIT, is probably one of the most well known. In Kerberos users authenticate against a Kerberos server, receiving an encrypted credential, also known as a ticket. When accessing services that require authentication, the obtained ticket is automatically sent to the service, transparent to the users. Another method, commonly used on the World-Wide Web is the cookie method, where users obtain HTTP cookies as credentials after authenticating. The obtained cookies are then sent along with HTTP requests to prove that authentication already has occurred. Figure 2 illustrates the use of the HTTP cookie method. There are several other methods for implementing single sign-on; they are however, not of particular interest to this thesis since DACS provides single sign-on functionality.

![Single sign-on](image)

*Figure 2: Single sign-on using HTTP cookies.*

Since credentials are used as a means of authentication in single sign-on systems it is important to keep the integrity of the credentials protected. In order to prevent theft of credentials it is usually a good idea to use a secure channel for transmission. It is also common practice to implement some kind of encryption or checksum of the credentials in order to detect and prevent forged credentials.
3.1 Introduction

DACS is a general purpose authentication processing and access control system. DACS can be used to implement single sign-on properties along with access control and authentication, for both Web services and applications. In order to implement authentication and access control functionality for applications, the DACS suite provides a C/C++ toolkit and command line utilities. Together with a DACS module for the popular open source Apache web server, access control and authentication for Web services can be implemented. [WEB1]

DACS is open source software, developed by the Canadian company Distributed Systems Software, Inc. The DACS suite supports UNIX-type platforms, where the officially supported operating systems include FreeBSD, GNU/Linux and Solaris. At the time of writing the latest stable version is DACS 1.4.20.

The documentation for DACS consists of mostly UNIX manual pages together with a quick start guide and installation instructions. For further assistance, commercial support contracts can be bought from Distributed Systems Software, Inc. There is also an option to purchase DACS under a different license than the default open source license, if desired. A copy of the DACS open source license can be found in appendix A.

3.2 Architecture

A DACS system consists of a federation, containing one or several jurisdiction members, see Figure 3. Federations and jurisdictions can be seen as abstract entities that describe the structure and configuration of a DACS enabled system. Thus, a single host can serve an arbitrary amount of jurisdictions and federations. There is no participating part controlling the federation itself; the federation is effectively the name of the system.

The federation is the highest level of the DACS hierarchy; it defines the member jurisdictions and the federation crypto key. The federation cryptographic key is used to encrypt and decrypt credentials between the participating jurisdictions within a federation. To realize this, the federation crypto key must be securely distributed to all federation members. Should a malicious party gain access to the federation cryptographic key, seemingly valid credentials can be crafted.

Since the loss of the federation cryptographic key leads to the security of the whole system being breached, it is extremely important to keep the key secure in a deployed system.
The lower level of the DACS hierarchy consists of the jurisdictions; entities that authenticate jurisdiction users, perform access control, and possibly, provide Web services. Each jurisdiction within a federation is assigned a jurisdiction name that is unique. The jurisdiction names are short and capitalized by convention, and can together with a username identify an authenticated user within the federation. [WEB1]

Because of the modular design of DACS, a jurisdiction can be configured in many different ways. For example, a jurisdiction can consist of a single DACS protected Web service or even a whole company server protected by DACS.

The usernames are unique within their jurisdictions, meaning that users in the same federation, but at different jurisdictions can have the same username. The jurisdictions manage the access control for resources protected by DACS. In a federation, authentication and access control for a resource can be handled by different jurisdictions, provided that the jurisdictions have been configured accordingly. Users can be assigned groups at run time or roles at authentication time, making this information part of the users’ credentials. The role and group membership of users can be used in access control decisions for easier management of access control rules. Groups can include other groups, even groups at foreign jurisdictions if desired. The assignment of roles can be implemented in several different ways, the simplest being a file containing mapping of roles and usernames. DACS also supports the use of LDAP (Lightweight Directory Access Protocol) for obtaining role memberships and authenticating users. Since DACS is a highly modular system, custom external modules for assigning roles can be easily integrated.

In order to complicate eavesdropping and credential theft, the communication between nodes running DACS is secured using the transport layer security protocol SSL/TLS. The use of SSL/TLS can be disabled if explicitly stated in the DACS configuration, but it is not recommended. The reason DACS uses SSL/TLS in favour of other transport layer security methods is that every modern Web browser of today supports it out of the box. The following quote from the DACS FAQ describes the ambition of DACS only requiring standard software:

“DACS was originally devised as a system that would: ...

...Be usable with standard web browsers (primarily newer releases of Internet Explorer and Netscape) that support secure communication via SSL, and requiring few or no browser
configuration changes by the user; it must not depend on user agents (the interface by which a user invokes web services) having anything other than the simplest computational abilities; DACS is required to work with a "lowest common denominator" user agent, namely an SSL-capable web browser; DACS must not require any custom code at the user's end. Note that this does not mean that a user can't be sitting in front of some other kind of interface, interact through a servlet, or use a browser proxy, only that basic DACS functionality must be accessible from a standard browser.” [WEB1]

### 3.3 Authentication

Authentication at a DACS jurisdiction can be performed using several different authentication methods. The DACS authentication modules provide an interface between DACS and the existing authentication methods of a system, allowing jurisdiction users to authenticate themselves using the same methods they normally employ. Different users at a jurisdiction do not have to authenticate themselves using the same authentication method. Figure 4 presents the authentication methods that are currently available for DACS.

- "Simple authentication"
- Digest authentication
- CAS
- HTTP
- Certificate-based authentication
- Native authentication
- Prompted authentication
- Expression-based authentication
- Delegated authentication"

**Figure 4: Authentication methods. [WEB1]**

Simple authentication can be used to sign in via an HTML form or similar. A username and password is required to sign on using simple authentication. Digest authentication is an implementation of Digest Authentication\(^4\) for DACS. Usually this authentication method is realized by letting the browser show a pop-up where the users type in their username and password. CAS\(^5\) (Central Authentication Service) is an open protocol for authentication and single sign-on. The CAS authentication method relays the authentication process to a CAS server. HTTP authentication contacts a Web service and decides whether to grant or deny authentication judging from the answer of the invoked Web service. Certificate-based authentication uses X.509 certificates for authentication via SSL. Native authentication makes it possible to let native Apache authentication modules take care of the authentication. DACS imports and uses the identity provided by the Apache authentication module.

---

\(^4\) A detailed description of this standard can be found in RFC 2617 - HTTP Authentication: Basic and Digest Access Authentication.

\(^5\) See, http://www.ja-sig.org/products/cas/index.html for more information on CAS.
Prompted authentication is similar to digest authentication, with the difference being that the prompt description is set at authentication time. PAM (Pluggable Authentication Modules) authentication is an implementation of prompted authentication. Expression-based authentication uses a DACS expression for authentication. The expression can consist of a computation or even running an external application. Delegated authentication does not authenticate a user, but contacts an outside entity that vouches for the user in question. Delegated authentication can be used to transfer credentials between different DACS federations. [WEB1]

The modular design of the DACS authentication phase allows new authentication methods to be added by simply creating a new authentication module. At build time DACS can be configured to enable several different authentication modules, Figure 5 describes the authentication modules that are available.

- "local_apache_authenticate: Password-protected accounts maintained by Apache utilities"
- local_cas_authenticate: Central Authentication Service (CAS)
- local_cert_authenticate: SSL-based X.509 client certificates
- local_grid_authenticate: Grid-based one-time passwords
- local_http_authenticate: Generic authentication via HTTP
- local_ldap_authenticate: Lightweight Directory Access Protocol (LDAP) / Microsoft Active Directory
- local_native_authenticate: Importing an identity established by Apache
- local_ntlm_authenticate: Microsoft Windows NT LAN Manager usernames and passwords
- local_pam_authenticate: Pluggable Authentication Modules (PAM)
- local_passwd_authenticate: Password-protected DACS accounts
- local_simple_authenticate: Account name without a password
- local_token_authenticate: Hardware token-based one-time passwords
- local_unix_authenticate: Unix usernames and passwords"

Figure 5: Standard authentication modules for DACS. [dacs_authenticate(8) manual]

When users successfully authenticate, the invoked authentication module reports success to the DACS authentication service, which issues a credential. DACS credentials can be used within a DACS federation to prove that users have been authenticated, representing DACS identities of authorized users. When implementing DACS as a web service the credential is often sent to the user client using HTTP cookies. However, cookies are not mandatory. DACS credentials contain the roles assigned to the user at authentication time, authorized username and information such as the name of the authorizing jurisdiction. Figure 6 presents the contents of a typical DACS credential. Credential elements such as expiration date and hash values are validated at decryption time and are not located inside the DACS part of the credential.

```
<dacs_current_credentials xmlns="http://fedroot.com/dacs/v1.4"
    federation_name="COALITION" federation_domain="foi.se">
    <credentials federation="COALITION" jurisdiction="PROXY"
        name="jerry" roles="Author" auth_style="passwd,password"
        cookie_name="DACS:COALITION::PROXY:jerry"/>
</dacs_current_credentials>
```

Figure 6: DACS credential contents.
Before the DACS authentication service issues a credential, the content of the credential is encrypted and provided with a hash value to detect modifications of the credential content. DACS credentials are encrypted using AES (Advanced Encryption Standard) with a default key length of 128 bits and the hash value is calculated using the HMAC (Keyed-Hash Message Authentication Code) algorithm. At build time DACS can be configured to use larger AES key lengths and other hash algorithms. However, all members of a federation must use the same key length and hash algorithm. [dacs_authenticate(8) manual]

3.4 Access Control

Access control for Web services in DACS is achieved using two main parts; the Web server module (mod_auth_dacs) and the access control service (ACS). The Web server module, currently only available for the Apache web server, communicates the requested URI (Unified Resource Identifier) to the ACS. The ACS parses the access control rules and together with the requested URI and the user provided credential, decides whether the request should be granted or not. Using the user supplied credentials, access control rules can use information such as username, roles, authenticating jurisdiction and authentication style when deciding whether the user should be granted access or not. If the ACS decides that the request should be granted, the URI is passed on to Apache and handled in the usual way. Should the ACS decide that the request should not be granted, Apache is given the URI of a document stating that authorization failed instead of the document the client initially requested. The DACS access control is transparent to the Web services, requiring no special implementation or configuration of such resources in order to be protected by DACS. In this sense DACS can be seen as a filter for the Web content, only granting access to authorized users. [dacs_acs(8) manual]

Access control in DACS is applied on a requested URI, access control rules are therefore created with regard to specific URI patterns. Access control rules in DACS are written as XML (Extensible Markup Language) expressions as seen in Figure 7. The rules can also take into consideration user passed CGI (Common Gateway Interface) parameters, supporting both the HTTP POST and HTTP GET methods.

```
<acl_rule>
  <services>
    <service url_pattern="/secret/garbage_file"/>
  </services>
  <rule order="allow,deny">
    <allow>
      user("%PROXY:Tech")
    </allow>
  </rule>
</acl_rule>
```

*Figure 7: A simple access control rule stating that only users of the group/role Tech, authenticated at the jurisdiction PROXY are granted access to the service.*

Access control rules can also include expressions written in the DACS expression language; a simple language with expressions resembling those of C/C++. The DACS utility *dacsexpr* can validate expressions, proving it to be a great aid when writing expressions for access control
rules. Figure 8 presents a simple example on how expressions can be used in access control rules. The DACS expression language provides several built-in functions to perform DACS related tasks. User defined functions are however, at this time, not supported in the DACS expression language. [dacs.exprs(5) manual]

```xml
<acl_rule>
  <services>
    <service url_pattern="/secret/garbage_file"/>
  </services>
  <rule order="allow,deny">
    <allow>
      user("auth") and ${Args::CODE} eq "HackThePlanet"
    </allow>
  </rule>
</acl_rule>
```

*Figure 8: Access control rule implementing a DACS expression. Grant access to the service for all authenticated users if the argument CODE is equal to HackThePlanet.*

Access control rules are saved in a directory configured by the administrator and individual rules are named as acl-rulename.integer. The rule name is used to aid the administrator to tell rules apart when making changes to rules, while the integer is used by the ACS to determine in which order the rules should be processed. The rules are always processed in ascending order with regard to the integer. DACS comes with a utility named dacsacl, which can parse access control rules in order to find syntactical errors. [dacs.acls(5) manual]

As with virtually every system where access control is applied, DACS provides the ability to revoke access for users. Given that DACS is a distributed system providing users with credentials, it is easily realized that enforcing removal of credentials already handed out is not an option when revoking access. The revocation implementation in DACS consists of a revocation file that the DACS utilities employ to decide whether the access privileges for the user in question has been revoked or not. The file specifying the revocations is written using DACS expressions, which enables four different types of revocation. Revocations are written in the form type expression, where type can be either deny, revoke, disable or block. The type deny simply denies access for a request if the expression should evaluate to true. In the case of the expression evaluating to true for the type revoke, the user supplied credentials are ignored and treated as if they did not exist. The type disable prevents credentials from being issued, effectively disabling authentication, if the expression should be validated to true. The last revocation type, block, is essentially a joint of the two types deny and disable. [dacs.acls(5) manual]

### 3.5 Single sign-on

Single sign-on in DACS can be implemented for all members of a federation, since all the jurisdictions share the same federation cryptographic key. A DACS jurisdiction uses the federation cryptographic key when encrypting and issuing a credential for an authenticated user. Thus, all other jurisdictions within the federation can decrypt the credential and verify that it has not been tampered with. The access control rules for resources at a jurisdiction can be set to grant access for users authenticated at a foreign jurisdiction within the same federation. Figure 9 presents a simple example on how single sign-on can be realized in a
DACS enabled system; a client authenticates at a jurisdiction, then requests access to a Web service located at another jurisdiction within the same federation.

Figure 9: Single sign-on using DACS.
4.1 Overview

4.1.1 System structure

The structure of the implemented system consists of a federation, containing a jurisdiction handling authentication and several different jurisdictions providing Web services. Figure 10 shows an overview of the system structure.

![System structure overview](image)

Users authenticate at the authenticating jurisdiction to obtain credentials, containing their username and assigned roles. When accessing Web services, the credentials are automatically sent by the authenticated users so that the DACS service protecting the Web services can decide whether to grant or deny access. The different jurisdictions providing Web services could for example represent different countries participating in a coalition.

It should be noted that only one authenticating jurisdiction is used in the implemented system, where in reality several different authenticating jurisdictions can be used. Using several different authenticating jurisdictions however, can lead to synchronization problems. The synchronization problems arise as the jurisdictions need to share a common state in order to be able to implement conditions and rules for assigning roles. Another issue with using several authenticating jurisdictions is that they all can create valid credentials, and thus more
participating members have to be trusted. The main advantage of letting several jurisdictions handle authentication is that the administration of users can be distributed among them instead of being handled by a single jurisdiction.

4.1.2 Software

The software used in the implemented system is open source. The advantage of the software being open source is that the participating parts are able to read the source code before running it. Because one of the intended uses for a future system is to share information between nations, being able to trust the software in use is very important.

Since one of the main goals was that the resources should be accessible through a standard Web browser, the implemented system needs a Web server. The popular open source Web server Apache is used, as it is supported by DACS and can run on many different platforms. In order to employ access control for the Web services, DACS together with the DACS Apache module are used.

The various CGI applications that have been implemented are using the Perl programming language, thus a Perl interpreter is needed. Perl is also used to implement the role module used by DACS. It is however, possible to implement both the DACS role module and the CGI applications using any other language of choice. The reason for using Perl is that it is a simple language, good at handling strings. Perl code was originally designed to be fairly compact and easily readable; something that is useful for a concept implementation such as this one.

In order to implement a working system, a Web service to provide access control for is needed. The Web service used in the implemented system is Alfresco\(^6\), an open source content management system. Alfresco provides a community version free for download, which is the version used as the Web service of the system. There is also a commercial enterprise version of Alfresco available that includes support and extended functionality. Alfresco consists of a Java servlet, requiring Apache Tomcat and a Java runtime environment in order to run. Alfresco needs a database for storing information; the implemented system uses the MySQL database system for this purpose. Section 4.4 describes the implementation as well as the issues that had to be solved when applying Alfresco to the system.

4.1.3 Roles

Since the roles a user possesses are part of the credentials, roles can be taken into account when implementing access control at jurisdictions serving Web applications. The implemented system uses the assigned user roles rather than the usernames to decide whether access to resources should be granted or not. By using roles in access control rules, existing rules does not need to be changed when new users are added or when existing users are assigned new roles. Since the DACS protected resources can consist of Web applications it is also desired to have the possibility of letting Web applications inherit the roles assigned by DACS. However, the implementation and use of roles in a Web application is not necessarily equivalent to their implementation and use in DACS.

\(^6\) For more information on Alfresco, see http://www.alfresco.com
As it is roles, rather than the usernames that are taken into account by the access control service, the system needs to handle roles in a dynamic way. The possibility for users to sign on and off roles on the fly, effectively affecting their privileges is a necessity. The dynamic handling of roles also provides the possibility of using constraints when assigning roles to users. An example of such a constraint could be that in order to sign on to role A, no user can be signed in as role B. Unfortunately, the use of constraints when assigning roles creates a need of keeping track of the already assigned roles; leading to the need of keeping a common state within a system. Section 4.3.2 further describes this issue and presents the implemented solution for it.

4.1.4 Single sign-on

The shared information is often served by a CMS (Content Management System) or similar Web application. When accessing Web services, the DACS credentials can be used to automatically sign on; effectively transferring the DACS privileges to the Web service. As described earlier, it is not certain that the Web service handles roles in the same way as the overall DACS system. Thus the implementation for integrating DACS with Web services to provide single sign-on properties generally differ from service to service.

The amount of details that are possible to transfer between DACS and the Web services depends heavily on the Web service in question. Should the Web service source code be available, the integration is possible; it is more a question of how much work it involves. For Web services where the source code is not available, a suiting integration might not even be possible to implement.

4.2 System setup

4.2.1 DACS configuration

Authentication modules have to be explicitly enabled at compile time in order to be available when running DACS. The supported authentication modules are described in section 3.3. Configuration of the activated authentication modules together with the implementation details of the authenticating jurisdiction is described in section 4.3.

When configuring DACS, a federation is created together with one or more jurisdictions. The member jurisdictions of a federation are specified in a DACS configuration file named jurisdictions.grp. The jurisdictions.grp configuration file must be shared among all the participating jurisdictions of the federation. Figure 11 shows the jurisdictions.grp configuration of the implemented system, specifying two jurisdictions; APPSERV and PROXY.

<groups xmlns="http://dss.ca/dacs/v1.4">
  <group_definition jurisdiction="PROXY" name="jurisdictions"
    mod_date="Tue, 11-Dec-2007 15:10:00 GMT" type="public">
    <group_member jurisdiction="PROXY" name="Login proxy"
      type="meta"
      alt_name="Login proxy for foi.se"
      dacs_url="https://lamp.foi.se/cgi-bin/dacs"
The jurisdiction configuration file specifies whether a jurisdiction is allowed to authenticate users or not. As seen in Figure 11 the jurisdiction PROXY is allowed to authenticate users while the jurisdiction APPPSERV is not. The directive dacs_url is needed for DACS to be able to invoke DACS services at a jurisdiction properly.

When configuring a system implementing DACS it is important that the domain names of the participating jurisdictions are subdomains of the same domain. In Figure 11, the domain names of the jurisdictions is lamp.foi.se and appserv.foi.se, both being subdomains of foi.se. The need for using subdomains is due to the HTTP cookie standard; clients should only send cookies to hosts residing at the same domain they were received from. [RFC 2109]

Encryption keys for the federation have to be created and distributed among all participating jurisdictions. The federation encryption key is used to encrypt and validate DACS credentials. There is no built in support for distributing keys in DACS, the distribution of keys must be done manually. All jurisdictions also need a private encryption key, which is not shared between jurisdictions. The jurisdiction encryption key is used to encrypt local data for a jurisdiction. The DACS utility dacskey is used to create both federation and jurisdiction keys. [dacskey(1) manual]

### 4.2.2 Apache configuration

The Apache configuration has to be carefully set up to not allow alternative URI’s to services protected by DACS. The implemented system uses a virtual host for services, defining the path to documents and DACS services. Thus, the services can only be accessed via the virtual host; lowering the risk of alternative non DACS protected access. The issue with alternative paths is further discussed in section 5.3.5. Figure 12 presents the fragment of the Apache configuration defining a virtual host implementing DACS.

```plaintext
NameVirtualHost lamp.foi.se:443
Listen lamp.foi.se:443
<VirtualHost lamp.foi.se:443>
    ServerName lamp.foi.se
    DocumentRoot "/opt/dacs/www"
    ErrorLog "/opt/apache-dacs/logs/error_log"
    TransferLog "/opt/apache-dacs/logs/access_log"
    ScriptAlias /cgi-bin/ "/opt/apache-dacs/cgi-bin/"
    ScriptAlias /script/ "/opt/dacs/www/script/"
    Alias /css "/opt/dacs/www/css/"
    Alias /handlers "/opt/dacs/www/handlers/"
```
AddDACSAuth dacs-acs /opt/dacs/bin/dacs_acs "-t -v"
SetDACSAuthMethod dacs-acs external
SetDACSAuthConf dacs-acs "/opt/dacs/federations/dacs.conf"

<Location />
  AllowOverride AuthConfig
  Require valid-user
  AuthType DACS
  AuthDACS dacs-acs
</Location>
</VirtualHost>

Figure 12: Virtual host configuration for Apache.

The configuration in Figure 12 defines a virtual host, lamp.foi.se:443 with the document root /opt/dacs/www. The documents in /opt/dacs/www can only be accessed by clients connecting to the host lamp.foi.se:443. As seen in the figure, Apache has to be configured to know the location of the DACS configuration file as well as the location of the dacs_acs service.

The directive ServerName defines the domain name of the server Apache is run on. For the same reasons as described in section 4.2.1, the Apache configuration directive ServerName for jurisdictions of a federation must be subdomains of the same domain.

In order to lower the risk of cookie theft and eavesdropping, Apache is set up to listen for SSL connections only. When creating SSL certificates for the Apache servers it is important to enter the correct domain name of the server Apache is run on. Should the domain name in the certificate mismatch the domain name of the server, DACS will return an error, as an attempt to prevent Man-in-the-middle (MITM) attacks.

4.3 Authenticating jurisdiction

4.3.1 Authentication

The authenticating jurisdiction let users authenticate themselves using one or more of the available authentication modules described in section 3.3. After successfully authenticating, users obtain DACS credentials which can be used within the federation to prove that they have been authenticated. DACS can be configured to require successful outcome from several different authentication modules in order for users to be authenticated.

Figure 13 presents a simple authentication configuration implementing two modules; authentication using a local user database for DACS and normal UNIX authentication on the host DACS is running on. The directive CONTROL is set to sufficient for both of the authentication modules, meaning that users can authenticate using any of those two modules. If the CONTROL directives should be set to require instead, both authentication modules would have to return success in order for users to be authenticated. The configuration also specifies the federation name and domain together with the name of the jurisdiction that authentication is set up for. Since the name of the jurisdiction is specified in the configuration, several different authenticating jurisdictions can be set up in the same DACS installation.
The actual authentication process is started when the DACS Web service \texttt{dacs\_authenticate} is invoked. Input information from users such as a username and password are sent as arguments to \texttt{dacs\_authenticate}. Typically user input is handled by HTML forms, but this is not a necessity. Users do not need to specify which of the authentication modules to use when authenticating. For example, take the configuration in Figure 13; \texttt{dacs\_authenticate} will try both authentication modules using the user provided input and authentication will succeed as long as one of the modules returns success.

### 4.3.2 Roles

In order to realize the dynamic role handling described in section 4.1.3, a custom module for assigning roles has been implemented. The custom role module is contacted at authentication time by the Web service \texttt{dacs\_authenticate}. Since roles are assigned at authentication time in DACS, roles are configured together with the other authentication settings. Figure 14 shows a DACS authentication configuration implementing a custom role module.

\begin{verbatim}
<Configuration xmlns="https://fedroot.com/dacs/v1.4">
  <Default>
    FEDERATION_DOMAIN "foi.se"
    FEDERATION_NAME "COALITION"
  </Default>

  <Jurisdiction uri="lamp.foi.se">
    JURISDICTION_NAME "PROXY"

    <Auth id="passwd">
      URL "https://lamp.foi.se/cgi-bin/dacs/local_passwd_authenticate"
      STYLE "pass"
      CONTROL "sufficient"
    </Auth>

    <Auth id="passwd2">
      URL "https://lamp.foi.se/cgi-bin/dacs/local_unix_authenticate"
      STYLE "pass"
      CONTROL "sufficient"
    </Auth>

  </Jurisdiction>
</Configuration>
\end{verbatim}

\textit{Figure 13: Authentication configuration example.}
The role configuration in Figure 14 uses a DACS expression to invoke a Web service; the custom role module, which assigns roles. The Web service is sent the username together with the desired role as arguments and replies with a list of roles the user will be assigned. Should the Web service return an empty list, the authentication fails. Since the role module is in fact a Web service, it is possible to let an external service query it to obtain information on assigned roles. Figure 15 presents the authentication process for user jerry trying to sign on as the role Boss.
Since roles are handled dynamically, the role module has to keep track of the authenticated users and their assigned roles. The list of currently authenticated users together with their roles are stored in a plain text file in the implemented system, see Figure 16. As seen in the figure, expire date for the credentials are also saved in UNIX epoch time. In a production system, authentication information records should probably be handled by a database system in order to ensure consistency and reliability.

Since the role module stores a record of the assigned roles, rules for assigning roles can be implemented. Expire date for credentials must be stored in order to prevent the possibility of locking roles due to expired credentials. The first action the custom role module performs when invoked is to clean up the record from expired credentials. The role assignment module in the implemented system contains some hard coded assignment rules for evaluating the possibility and details of implementing such rules. In a future system, it is probably preferred to implement a parser with support for external custom role assignment rules.

As explained earlier, roles in DACS are assigned at authentication time and are part of users’ credentials. Therefore, in order for users to sign out of roles, new credentials must be issued. It is not desirable to require users to reauthenticate by providing a password in order to sign out of roles. The implemented system solves this by taking the current user credentials,
removing the role in question and generating new credentials. The newly created credentials are then sent to the client as HTTP cookies. Thus, in the implemented system, users can sign out of their current roles by selecting which role to sign out from and pressing a button to confirm.

4.3.3 Web service

The authenticating jurisdiction provides a Web service allowing users to authenticate themselves using an HTML form. The authentication Web service is accessed using a standard Web browser over an SSL connection. The form requires users to provide a username and a password together with a selected role. To get a better picture of the Web service, a screenshot can be found in Figure 17 below.

![Figure 17: Authentication Web service.](image)

After providing the required information and successfully signing on, the user retrieves a DACS credential via a HTTP cookie. As seen in Figure 17, the actual authentication modules that are invoked are fully transparent to the users, just as described in section 4.3.1.

4.4 Application jurisdiction

4.4.1 Access control for Web services

The application jurisdiction performs access control for Web services; it is the application jurisdiction that decides which information users have access to. The access control is realized by using DACS together with Apache. Access control is performed on the actual access of Web services, not on actions within the Web services themselves.
The access control rules in the implemented system have been written to base their access control decisions on the roles within the received credentials. Figure 18 presents a sample access control rule for a Web service, only granting access for users possessing the role Engineer or Captain.

```
<acl_rule>
  <services>
    <service url_pattern="/alfresco/"/>
  </services>
  <rule order="allow,deny">
    <allow>
      user("%Captain") or user("%Engineer")
    </allow>
  </rule>
</acl_rule>
```

*Figure 18: Sample access control rule for a Web service.*

As only the roles and not the usernames are used for access control rules, the application jurisdiction does not need to worry about added or removed users at the authenticating jurisdiction. The application jurisdiction does not authenticate any users; it is expected that users have authenticated themselves using the authentication jurisdiction before accessing Web services.

An application jurisdiction can provide access control for several different Web services. In an international effort perspective, application jurisdictions can be seen as different nations each sharing their own information. The nations decide the access control rules for their own information; deciding which actions specific roles is allowed to perform.

In the implemented system, access control rules have been created manually for Web services. In a future system it is probably wise to create some kind of user interface in which access control rules can be created and modified. DACS also supports delegation of rules, meaning that access control for specific URI’s can be delegated to individual users. By implementing delegation, users can be responsible for the access control rules protecting their own data. [dacs.acls(5) manual]

### 4.4.2 Alfresco and Apache Tomcat

The application jurisdiction hosts a Web service consisting of the open source Alfresco CMS. As described in section 4.1.2, Alfresco is a Java servlet and requires Apache Tomcat to run. The DACS Apache module does not run on Apache Tomcat, therefore the application jurisdiction needs to run both an Apache server and an Apache Tomcat server. However, the Apache server and the Apache Tomcat server do not necessarily need to be run on the same physical computer. In order to implement DACS protection for the Web services running on the Apache Tomcat server, the two servers must be connected. Details on the connection between Apache and Apache Tomcat are explained in section 4.4.3.

Alfresco uses a database for storing information, there are several different databases supported; PostgreSQL, Oracle, MySQL and DB2 to name a few. The implemented system uses the MySQL database, which is available under an open source license. [WEB2]
Users sign in to Alfresco using a username and password, retrieving a session cookie. The user database and the session cookies in Alfresco are separate from the users and cookies in DACS. Should a user be authenticated within Alfresco and log out from DACS, the Alfresco Web service will not be accessible until a new valid DACS credential is obtained.

4.4.3 Connecting Apache and Apache Tomcat

Apache Tomcat can be connected to several different Web servers using the Apache JServe Protocol (AJP). The latest version of the AJP protocol currently available is version 1.3, which is also the version used in the implemented system. AJP version 1.3 was created to improve performance and security from the earlier version 1.2. AJP version 1.3 improves security by implementing better support for the SSL protocol. [WEB3]

Since the Apache Web server does not have built in support for AJP, an external module is required. The authors Apache, Apache Software Foundation, provide a module named mod_jk which implements the connection between Apache and Apache Tomcat. The connection between Apache and Apache Tomcat is transparent to the clients accessing the Apache server. Figure 19 illustrates the connection between Apache and Apache Tomcat.

![Figure 19: Connection between Apache and Apache Tomcat.](Image)

The Apache module mod_jk is configured using its own configuration file. The mod_jk configuration file specifies connection details for Apache Tomcat servers. An Apache server using mod_jk can be connected to several different Apache Tomcat servers. Figure 20 shows the relevant part of the mod_jk configuration file used in the implemented system. The configuration specifies that the Web service at host localhost, using port 8009 and the protocol AJP version 1.3 should be exported as a resource named alfresco.

```
worker.alfresco.port=8009
worker.alfresco.host=localhost
worker.alfresco.type=ajp13
```

*Figure 20: Mod_jk configuration file excerpt.*

---

7 For further information on mod_jk, see http://tomcat.apache.org/connectors-doc/
In order to provide access to the Alfresco Web service from within the Apache Web server, the configuration directive `JkMount` has to be used. The `JkMount` directive specifies that a Tomcat Web application should be accessible from within this location. Figure 21 presents the section from the Apache configuration that defines a relative location providing the Alfresco Web service. The configuration also enables DACS access control for the location.

```xml
<Location /alfresco/*>
  AllowOverride AuthConfig
  Require valid-user
  AuthType DACS
  AuthDACS dacs-acs
  JkMount alfresco
</Location>
```

*Figure 21: Apache configuration providing access to a Web service.*

### 4.4.4 Single sign-on

Single sign-on is achieved with DACS in the sense that users sign on at the authenticating jurisdiction and are able to access services at an application jurisdiction using the credentials obtained when authenticating. Extending the single sign-on capabilities to incorporate the Web services requires further efforts; the information within the DACS credentials must be made accessible for Web services. The Apache module `mod_jk` enables the Apache server to export internal Apache environment variables to an Apache Tomcat server. The exported environment variables are sent as attributes in the request to the Apache Tomcat server; enabling Web services to access them. DACS can export environment variables containing decrypted authentication details from credentials sent along a request. Figure 22 shows the component of an Apache configuration file that exports the available DACS environment variables to `mod_jk`. In the figure, `NONE` is the default value of the variable.

```xml
<IfModule mod_jk.c>
  JkEnvVar DACS_FEDERATION NONE
  JkEnvVar DACS_JURISDICTION NONE
  JkEnvVar DACS_ACS_JURISDICTION NONE
  JkEnvVar DACS_ROLES NONE
  JkEnvVar DACS_USERNAME NONE
  JkEnvVar DACS_CONSTRAINT NONE
  JkEnvVar DACS_DEFAULT_CONSTRAINT NONE
  JkEnvVar DACS_COOKIE NONE
  JkEnvVar DACS_MOD_AUTH_DACS NONE
</IfModule mod_jk.c>
```

*Figure 22: Exporting DACS environment variables to mod_jk.*

Once Apache has been configured to export the DACS environment variables to the Apache Tomcat server, the Web services can access information such as the username and roles of the currently authenticated user. The implementation details for realizing single sign-on for Web services using the exported DACS variables are application specific.
The open source CMS Alfresco, used in the implemented system has built in support for filters. A filter in Alfresco is run for every received request and provides functionality such as redirecting users to a sign on page if not authenticated. In order to implement single sign-on for Alfresco, a custom filter module was created. The custom filter uses the exported DACS variables to automatically set the authenticated user, circumventing the Alfresco authentication process. Should the DACS username not exist in Alfresco, users are redirected to the Alfresco sign on page for manual authentication.

The community version of Alfresco, used in the implemented system does not support the use of roles; it is only supported in the enterprise version. Therefore, the custom filter in the implemented system does not translate DACS roles into Alfresco roles. In a future system using the enterprise version of Alfresco, support for translating roles should be straightforward to implement. It should be noted that roles might not be suited for direct translation, but that is outside the scope of this thesis.

### 4.5 Extending the system

#### 4.5.1 Adding Web services

In order to add a DACS-protected Web service at a jurisdiction, Apache must be configured to enable DACS authentication for the new service. An Apache configuration snippet defining a new Web service requiring DACS authentication can be seen in Figure 23 below.

```xml
<Location /new_service/*>
    AllowOverride AuthConfig
    Require valid-user
    AuthType DACS
    AuthDACS dacs-acs
</Location>
```

*Figure 23: Configuration adding a new Web service requiring DACS authentication.*

After enabling DACS authentication for the Web service, access control rules must be written for the service. An example of access control rules for Web services can be found in section 4.4.1, Figure 18. Adding Web services at a jurisdiction does not affect other jurisdictions in the federation; each jurisdiction is responsible only for its own services.

The next step is to connect the recently added Web service and DACS to provide single sign-on for the Web service. In order to realize single sign-on for Web services via DACS, modification of the authentication phase for the Web services is most likely needed. For Web services running as servlets or java server pages on an Apache Tomcat server, DACS authentication information can be accessed via the Apache module *mod_jk* as described in section 4.4.4. Should the Web service consist of a CGI script or similar, the DACS command line utilities can be used to retrieve authentication information from a DACS credential.

If the source code for the Web service is not available, single sign-on could be difficult or even impossible to implement for the service. Should the closed source Web service accept authentication information as parameters, it might be possible to write a wrapper script that invokes the service with parameters obtained from a DACS credential.
4.5.2 Adding jurisdictions

Adding a new jurisdiction to a federation involves obtaining information from a participating jurisdiction as well as federation-wide update of information. Since information must be updated on all participating jurisdictions of a federation, adding jurisdictions in large federations can lead to synchronization issues. To solve those issues it is probably preferred to develop a solution for synchronizing the configuration files of jurisdictions automatically. The updating of existing jurisdictions can be done at runtime; there is no need to temporarily shut down the DACS services. Files can be replaced and DACS will notice the changes automatically at the next invocation.

The first thing that needs to be done when adding a new jurisdiction is to configure DACS and Apache at the new jurisdiction, as described in sections 4.2.1 and 4.2.2. A private cryptographic key must be generated for the new jurisdiction using the utility `dacskey`. The jurisdiction key is used to encrypt locally stored data at a jurisdiction.

The federation-wide shared file `jurisdictions.grp` must be obtained from one of the jurisdictions in the federation and be updated to include the new jurisdiction. In order for all the jurisdictions of the federation to be aware of the new jurisdiction, the updated `jurisdictions.grp` must be distributed among all participating jurisdictions in the federation.

The new jurisdiction must also acquire the federation cryptographic key from one of the jurisdictions in the federation. The federation key is needed in order to be able to encrypt and verify credentials issued by jurisdictions within the federation.
5.1 Implementation components

Many parts of the implemented system are built using existing software components. For example, authentication and credentials is handled by DACS. Due to the limited role capabilities of DACS, the implementation of role handling had to be designed and implemented from scratch. The role handling implementation allows users to sign-on and sign-out of roles dynamically and provides the functionality of specifying custom role assignment rules.

In order to realize single sign-on for a DACS authenticated user in the Alfresco CMS, a custom interface between the two components had to be implemented. The interface, a custom Alfresco filter, converts authentication information from within a DACS credential to Alfresco authentication components.

Since DACS is not a complete system but rather a collection of components and services, the implemented system provides an interface for interacting with those services. The interface consists of Perl scripts and HTML code. The Perl scripts implements parsing of the credential XML structure to present information based on credential contents. The credential parsing also allows the system to update user credentials together with the record of currently assigned roles.

5.2 System limitations and possible solutions

There are some limitations in the implemented concept system. The participating jurisdictions of a federation must use hostnames that are subdomains of the same domain, as described in section 4.2.1. The hostnames of jurisdictions must resolve to their corresponding IP addresses when being accessed by clients in the system. In order to circumvent the need for using subdomains, all jurisdictions could be set up as individual federations instead and be configured to accept credentials from foreign federations. Credentials can be transferred between federations using the DACS utility `dacs_auth_transfer`. [dacs_auth_transfer(8) manual]

Since the federations would not have a common federation key anymore, credentials from foreign federations has to be imported in order to be validated successfully. In order to provide single sign-on in the same manner as described earlier, credentials must be exported to all federations at authentication time. In the same way, credentials must be updated from every federation when users sign out of roles. Revocation also has to be handled by every federation in the system. As realized, the solution with multiple federations generates a lot of potential synchronization problems. It is probably less error prone to use the original solution consisting of one big federation and several member jurisdictions with the restriction that all jurisdictions need to share a common domain name.
The issue with resolving subdomains can be solved by using a federation-wide DNS (Domain Name System) server. All the participating jurisdictions and clients would have to be configured to use the federation DNS server. The DNS server is configured to map the hostnames in the current coalition to the corresponding IP addresses of the participating jurisdictions. By using a federation DNS server, there is no need for a local file mapping hostnames to IP addresses at every node in the system.

Another limitation is the handling of roles. DACS does not support dynamic usage of roles out of the box, but rather uses roles as extended groups. The implemented system supports dynamic usage of roles by using a custom written role assignment module. In order to provide the possibility of defining role assignment rules, the previous assignments must be recorded. Should a client close the Web browser without signing out from the system, roles could get locked until they expire and get cleared out from the record. Therefore, support for revoking users from roles is desired. Revoking users from roles can easily be implemented by modifying the local record of assigned roles; however, the user possessed credentials would still imply that users occupy the role. A way of solving this would be to implement some kind of automatic credential revocation at the jurisdiction performing the revocation of roles. The user provided credential would be considered invalid by the jurisdiction, forcing the user to authenticate again. The revocation would of course need to be synchronized between all jurisdictions in the federation.

The fact that a record of the currently assigned roles needs to be kept can also lead to problems when using more than one authenticating jurisdiction in a system. At authentication time the authenticating jurisdictions would need to communicate with each other in order to be able to determine the outcome of the role assignment request. The communication between the authenticating jurisdictions needs to implement a distributed synchronization algorithm in order to avoid role assignment decisions based on inaccurate data.

The usage of the system time is another limitation in the implemented system. When credentials are created at authentication time, the expiration date is saved inside the credential. The expiration date for credentials is calculated by adding the configured credential lifetime to the current system time. Thus, the expiration date inside a credential is based on the system time of the jurisdiction creating it. Should the system date of other jurisdictions within the federation differ more than the lifetime of the credential from the authenticating jurisdiction, the credential will be considered invalid. To avoid the problem caused by non synchronized system times, NTP (Network Time Protocol) can be used. An NTP server can be configured for the whole federation, allowing all jurisdictions to synchronize their system time regularly.

All the jurisdictions within the federation are stand alone, in the sense that if a jurisdiction should crash, the other jurisdictions are not affected. However, should the authenticating jurisdiction crash, users would of course not be able to authenticate themselves.

5.3 Security evaluation

5.3.1 Credential revocation

The probably most evident security issue in the implemented system is revocation of credentials. Since credentials are stored in users’ Web browsers, the system itself can not
guarantee their removal. A user held credential will be valid until the expiration date within the credential itself has passed. DACS supports revocation of credentials, as described in section 3.4. However, revocation in DACS is very limited; revocation is performed by adding a DACS expression in a local file named \textit{revocations} at a jurisdiction. When performing access control DACS will read the \textit{revocations} file and if the expression matches the current credential, the credential will be ignored and the request will be treated as sent by an unauthenticated user. \cite[dacs.acls(5) manual]

Revocations in DACS are therefore more a revocation based on credential contents rather than a revocation of the credentials themselves. The revocation file is jurisdiction specific; the credential will only be revoked at jurisdictions that have a \textit{revocations} file specifying it. Should a user be revoked at a jurisdiction and reauthenticate successfully at the same jurisdiction, no new credential will be issued since the user is listed as revoked by the jurisdiction.

As described above, there are two main problems with the revocation of credentials; annulling the revocation when reauthenticating and distributing the revocation among all jurisdictions in a federation. Annulling a user revocation at a successful reauthentication will likely have to be implemented by modifying the DACS authentication utility \texttt{dacs\_authenticate} to remove the revocation before trying to issue a credential. Distributing the revocations among all jurisdictions in a federation require the jurisdictions to communicate with each other. However, the communication only needs to involve addition and annulling of revocations, since the jurisdictions themselves will automatically revoke the credentials once the proper expression has been added to the \textit{revocations} file.

### 5.3.2 Federation key revocation

Revocation of the federation cryptographic keys is also an issue in the implemented system. In the event of a jurisdiction leaving a federation, the federation cryptographic key must be replaced and distributed among all remaining jurisdictions. Should the federation cryptographic key be compromised, the federation must also replace the key, of course. Since there is no built-in support in DACS for distributing keys, the distribution must be done manually. Currently authenticated users in a federation would have to reauthenticate themselves in the event of a federation key change, since their credentials can not be validated using the new federation key. Therefore, to avoid problems with invalid credentials, changing the federation cryptographic key in a federation must be done as simultaneously as possible on all the participating jurisdictions. To realize this, a system implementation for distributing a new federation key together with information such as a timestamp when to perform the key change must be designed and implemented.

### 5.3.3 Theft and tampering of credentials

Another potential security threat in the implemented system is theft and tampering of credentials. DACS credentials are embedded in cookies, which could potentially be stolen by a malicious part. DACS do not set the expiration date in cookies; the expiration date is specified inside the credential itself only. Cookies without an expiration date are kept in the memory by the Web browser and are deleted once the browser exits. Since the cookies are not written to disk, stealing cookies from a client is a bit more difficult, but not impossible. All
DACS communication is performed over an SSL protected HTTP connection, making theft of cookies during communication difficult.

The weak part in the system is the Web browser; cookies could potentially be stolen and there is a risk of cross site scripting attacks. Cross site scripting attacks could possibly enable a malicious part to steal cookies. To avoid cross site scripting attacks as much as possible, Web browsers should be up to date and user provided data input for Web services should be sanitized. Sanitizing data means that all HTML special characters are encoded, preventing them from being treated as code.

Since DACS credentials are encrypted and validated using the federation cryptographic key, a malicious part needs access to the federation key in order to tamper with the credential contents. Should a user be able to access the federation key, contents of a credential such as the roles the user possess could be altered. However, if a user attains access to the federation key, the overall security of the system is compromised and the key needs to be changed. In order to reduce the risk of leaking the federation key, an administrator must ensure that the permissions of the federation key file are set correctly. It is also recommended not to allow any users to log on to the server where DACS is running. The weak spot when it comes to the federation key is therefore the distribution of the key.

### 5.3.4 Replay attacks

Another security concern is the possibility of replay attacks. A replay attack is when a malicious part records a message from a session, to resend it at a later time. Replay attacks could possibly be used to record an encrypted authentication message, to later resend the recorded data in order to achieve authentication. [Syverson, 1994]

The implemented system uses the SSL protocol for all communication sessions. SSL/TLS implements a random nonce as a counter-measure against replay attacks. The nonce is an at least 128 bit long connection identification, generated from random connection-based events. The nonce is sent to clients at session initialization, and is used by clients when communicating with the server. Should clients use a different nonce than the one chosen by the server, the connection will be dropped. The possibility for an attacker to have previously recorded sessions using the nonce needed for a specific connection is very low, due to the vast amount of possible nonce values. Since the right nonce is needed when initiating a session, replay attacks are very difficult to perform even if the attacker has recorded a lot of sessions. [Hickman, 1995]

### 5.3.5 Alternative paths to resources

As described in section 3.4, DACS operates on URI’s when performing access control. When configuring the Apache server it is important to ensure that no alternative paths to DACS protected resources exist. Take for example an Apache configuration specifying a virtual host using SSL and DACS to protect resources. Users connecting using SSL and the hostname specified in the virtual host would have to provide a valid DACS credential in order to gain access to the resources. Should the Apache configuration provide a server listening for a connection not using SSL and DACS, users can access this server without providing a valid DACS credential. If the Apache server not using SSL and DACS is configured to serve the
same resources as the secured virtual host, users can gain access to the resources without being DACS authenticated.

Another case for alternative paths to resources could be when an Apache server specifies an alias for a URI that is not protected by DACS. For example; Apache is configured to perform DACS access control for `servername.com/secrets/` and specifies `/files` as an alias for `servername.com/secrets/files/`. Access control will not be performed on `servername.com/files`, since DACS is not configured to protect that URI.

An administrator setting up a DACS enabled system must be very careful when configuring Apache, not to accidentally allow alternative paths to resources. It is wise to only use virtual hosts, where each virtual host use different file system directories for resources. It is also recommended to avoid aliases as much as possible, since mistakes resulting in alternative paths to resources can be easily overlooked. Generally, one should consider enabling DACS protection for the path `/` in virtual hosts to make sure that no utterly unprotected paths to resources exist.
6.1 Security considerations

Apache and Apache Tomcat are connected to each other using the AJP protocol as described in section 4.4.3. Since Apache Tomcat acts as a server listening for connections using the AJP protocol, proper precautions must be taken to ensure that no third party can connect and access the Web services. The server hosting the Apache Tomcat server should restrict which hosts are allowed to connect to the Apache Tomcat service. The restrictions could for example be realized using a firewall.

As stated earlier, DACS only perform access control on the actual access to a Web service; the security of the Web services themselves is not enhanced by DACS. Thus, DACS authenticated users allowed access to a Web service are only restricted by the security of the Web service itself. As realized, the security of the Web services have to be kept in mind when implementing a DACS enabled system.

Due to the single sign-on properties, different participators in the implemented system need to trust each other. There is no supreme commander in the system since no participator controls the federation itself. Each participating jurisdiction is in command of the access control rules for its own Web services. However, the roles and usernames which are used in the access control rules are determined at the authenticating jurisdictions. Thus, participants controlling application jurisdictions must trust the participants controlling the authenticating jurisdictions.

6.2 Conclusions

This thesis shows that it is possible to implement a system for sharing information between different nations using DACS. Single sign-on between DACS and Web services is also possible, but the implementation details are application specific. The overall security of the implemented system is good, much due to SSL/TLS being used for communication. The greatest security threat in the system is the risk of credentials being stolen from users’ Web browsers; the weakest spot in the system.

6.3 Future work

The implemented system can be improved in several different areas. The most prominent issues exist in the area of distributed synchronization. The revocation of credentials must be synchronized among all jurisdictions in a federation, as described in section 5.3.1. The synchronization method should be able to handle non-responding jurisdictions and similar errors. In the same manner, a method for securely distributing configuration files and federation keys is also of interest.
The custom role module in the implemented system is only a concept, consisting of a simple role assignment record and some hard-coded role assignment rules. A natural extension of the custom role module is to implement a parser for role assignment rules. The ability to create and modify custom role assignment rules at runtime would make handling of roles much more flexible. The record of currently assigned roles should be improved by using a real database system, making it transaction safe. Before the role module assigns roles, it is desirable to communicate with the other authenticating jurisdictions within the federation, taking their assigned roles into account.

As described earlier, DACS does not offer the functionality to perform access control within Web services. However, it might still be possible to implement access control using DACS within Web services by modifying them to invoke HTTP requests for external DACS services. Web services could then base their access control decisions on the outcome of the HTTP requests. The actual access control and rules can be handled by DACS, since the request from the Web services will be validated by the DACS access control system. Note that the DACS services that are invoked could be located at a remote host, if desired.

It is desirable to let DACS handle access control within Web services since no translation between DACS credentials and access control rules in different Web applications would be needed. Possible problems with this method are that DACS cookies must be relayed by Web services and that the modification of Web services might not be straight forward. Further research is needed in order to decide whether this method should be considered or not.

Another area that might be of interest to explore is the combination of DACS and so called Web 2.0 technology, such as AJAX or JSON. Since the new Web technologies grow more popular each day, it is of interest to see whether they integrate easily with a DACS system. DACS relies heavily on the use of HTTP cookies, something that could possibly lead to problems with modern Web technologies.
Bibliography

Publications and books


D. Ferraiolo, R. Kuhn, R. Chandramouli (2003), “Role-Based Access Control”, Artech House Incorporated


Manual pages


“dacs_acs(8)”, Distributed Systems Software, Inc,

“dacs_authenticate(8)”, Distributed Systems Software, Inc,

“dacs_auth_transfer(8)”, Distributed Systems Software, Inc,
http://dacs.dss.ca/man/dacs_auth_transfer.8.html, Accessed 2008-02-06

“dacskey(1)”, Distributed Systems Software, Inc,

RFC


RFC 2109, “HTTP State Management Mechanism”, D. Kristol, L. Montulli (1997), The Internet Society


Internet resources


Appendix A

DACS license

$Id: LICENSE 1589 2006-12-23 22:28:54Z brachman $

==============================================================================
1. This copy of the DACS software and documentation is provided by
DSS Distributed Systems Software, Inc. (hereafter "DSS") under the following
license.

/*
 * Copyright (c) 2003-2007 DSS Distributed Systems Software, Inc.
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification are permitted provided that the following conditions are met:
 * 1. Redistributions of source code must retain the above copyright
 *    notice, this list of conditions, and the following disclaimer.
 * 2. Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions, and the following disclaimer in the
 *    documentation and/or other materials provided with the distribution.
 * 3. Redistributions in any form must be accompanied by information on
 *    how to obtain complete source code for the DACS software and any
 *    accompanying software that uses the DACS software. The source code
 *    must either be included in the distribution or be available for no
 *    more than the cost of distribution plus a nominal fee, and must be
 *    freely redistributable under reasonable conditions. For an
 *    executable file, complete source code means the source code for all
 *    modules it contains. It does not include source code for modules or
 *    files that typically accompany the major components of the operating
 *    system on which the executable file runs.
 *
 * NO WARRANTY
 *
 * THIS SOFTWARE IS PROVIDED BY DSS `AS IS' AND ANY EXPRESS OR IMPLIED
 * WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
 * MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT,
 * ARE DISCLAIMED. IN NO EVENT SHALL DSS BE LIABLE FOR ANY DIRECT, INDIRECT,
INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
*/

Product licensing information:

a) This open source license permits you to use DACS at no charge under the condition that if you use the software in an application that you redistribute, the complete source code for your application must be available and freely redistributable under reasonable conditions.

If you do not want to release the source code for your application, you may purchase a license from DSS. For pricing information, if you have further questions on licensing, or to purchase support for this software, please contact Distributed Systems Software at dacs@dss.ca or visit http://www.dss.ca

b) The term "redistribution" in the open source license means your application is distributed to one or more third parties. Giving an application to customers, even in alpha or beta releases, is redistribution. Giving contractors, affiliates, parent organizations or subsidiaries, business partners or support vendors a copy of the application is generally redistribution. Whether or not you charge money for your application does not matter, the important factor is whether you redistribute it.

Neither building an application for use internal to your organization, deployed and managed on your company servers, nor off-site backups or other software archival procedures constitute redistribution.

c) Under the open source license, you must release the complete source code for the application that uses DACS. You do not need to release the source code for components that are generally installed on the operating system on which your application runs, such as system header files or libraries.

d) In general, licenses recognized by the Open Source Initiative (www.opensource.org) meet the requirements of "freely redistributable under reasonable conditions" in the DACS open source license. In particular, the common open source licenses, such as the GPL and the BSD and MIT licenses, are acceptable.
e) Releasing an application which includes DACS under an open source license does not change the requirements of the DACS license, and the DACS source code remains subject to the terms of the open source license governing its use and redistribution. The DACS open source license is compatible with the GPL, so GPL'ed software can incorporate DACS without violating the terms of either license.

f) Redistribution of the complete DACS distribution, by itself or as part of a collection of other software, whether free of charge or for a fee, and by any electronic means (such as ftp) or on any physical media (such as on hard disk, CD-ROM, or DVD), is permitted provided the terms of the DACS open source license are met.

g) Redistribution of the complete DACS distribution after changing its packaging, archive format, or compression scheme from the original one, with or without modifications and additions (such as bug fixes or enhancements), is permitted provided the terms of the DACS open source license are met.

==============================================================================

2. Portions of this software distribution may also be subject to other copyrights and licenses. For details, please refer to the NOTICES file included with the DACS distribution.

==============================================================================

3. This software distribution may include public domain software; proper attribution has been retained within the associated files. For details, please refer to the ACKNOWLEDGEMENTS file included with the DACS distribution.

==============================================================================

Copyright (c) 2003-2007
Distributed Systems Software. All rights reserved.
På svenska

Detta dokument hålls tillgängligt på Internet – eller dess framtida ersättare – under en längre tid från publiceringsdatum under förutsättning att inga extra-ordinära omständigheter uppstår.

Tillgång till dokumentet innebär tillstånd för var och en att läsa, ladda ner, skriva ut enstaka kopior för enskilt bruk och att använda det oförändrat för icke-komersiell forskning och för undervisning. Överföring av upphovsrättar vid en senare tidpunkt kan inte upphäva detta tillstånd. All annan användning av dokumentet kräver upphovsmannens medgivande. För att garantera äktheten, säkerheten och tillgängligheten finns det lösningar av teknisk och administrativ art.

Upphovsmannens ideella rätt innefattar rätt att bli nämnd som upphovsman i den omfattning som god sed kräver vid användning av dokumentet på ovan beskrivna sätt samt skydd mot att dokumentet ändras eller presenteras i sådan form eller i sådant sammanhang som är kränkande för upphovsmannens litterära eller konstnärliga anseende eller egenart.

För ytterligare information om Linköping University Electronic Press se förlagets hemsida http://www.ep.liu.se/

In English

The publishers will keep this document online on the Internet - or its possible replacement - for a considerable time from the date of publication barring exceptional circumstances.

The online availability of the document implies a permanent permission for anyone to read, to download, to print out single copies for your own use and to use it unchanged for any non-commercial research and educational purpose. Subsequent transfers of copyright cannot revoke this permission. All other uses of the document are conditional on the consent of the copyright owner. The publisher has taken technical and administrative measures to assure authenticity, security and accessibility.

According to intellectual property law the author has the right to be mentioned when his/her work is accessed as described above and to be protected against infringement.

For additional information about the Linköping University Electronic Press and its procedures for publication and for assurance of document integrity, please refer to its WWW home page: http://www.ep.liu.se/

© Jerry Falkcrona