Practical access control using NDG-security

Bryan N Lawrence¹, Philip Kershaw¹, Jon Blower²

¹NCAS/British Atmospheric Data Centre, Rutherford Appleton Laboratory, STFC
²Reading e-Science Centre, Environmental Systems Science Centre, University of Reading

Abstract

Access control in the NERC DataGrid (NDG) is accomplished using a combination of WS-Security to ensure message level integrity, X509 proxy certificates to assert identity, and bespoke XML tokens to handle authorization. Access control decisions are handled by Gatekeepers and mediated by Attribute Authorities. The design of the NDG-security reflects the reality of building a deployable access control system which respects pre-existing user databases of thousands of individuals who could not be asked to reregister using a new system, and pre-existing services that need to be modified to take advantage of the new security tooling. NDG-security has been built in such a way that it should be able to evolve towards the use of community standards (such as SAML and Shibboleth) as they become more prevalent and best practice becomes clearer. This paper describes NDG-security in some detail, and provides details of experiences deploying NDG-security both in the e-Science funded NDG and the DTI funded Delivering Environmental Web Services (DEWS) projects. Issues to do with securing large data transfers are discussed. Plans for the future of NDG-security are outlined; both in terms of application modification and the evolution of NDG-security itself.

1. Introduction

The NERC DataGrid (NDG), which originally consisted primarily of a partnership between the British Atmospheric and Oceanic data centres, along with the CCLRC e-Science department, was established as a NERC funded e-Science project with the aim of improving integration of data holdings in the atmospheric and oceanographic sciences to the point where users could mix and match data from multiple sources in a “single-sign-on” context. The NDG is now nearing the end of a second tranche of funding, with partners extended to include the Plymouth Marine Laboratory and the National Oceanography Centre, Southampton. The mandate has also been extended to look at integrating data holdings on a wider scale, although the NDG is still only committed to delivering integration of some classes of atmospheric and oceanographic data.

There is a common assumption in the environmental sciences community that access control on data is something that is not necessary and positively gets in the way of “doing science”. Many academics have asserted (and continue to assert) that the first thing the NDG should have done was to throw away the single-sign-on requirement in favour of uncontrolled access. None of those academics has the responsibility of providing services that are used by a significant number of people, or that deliver a significant amount of data. Experience tells us that it doesn’t take many users before I/O subsystems are completely overloaded (never mind CPUs or networks). In such an environment, it is crucial to have methods to constrain access to priority users. When a data provider is also holding data which is either owned by, or for which the IPR resides with, third parties, there is also a requirement to protect the legal rights of those third parties (whether or not some individuals want it done at the time, their employing organizations may well take a different position at a later date). Sometimes too, data providers may be holding data that is in an interim form that is not suitable for general use; in this case too access needs to be limited. There are thus two driving requirements for access control: to protect access to finite resources, and to ensure that the use of those resources is appropriate (legally and/or scientifically).

The remainder of this paper describes the access control mechanism developed for the NDG, and its application both in the NDG and the Delivering Environmental Web Services (DEWS) project. DEWS is a technology demonstrator project, that is using the same underlying technology to (1) take marine forecasts from the Met Office and deliver them into Search and Rescue applications at British Maritime Technology, and (2) take primary data from the health sector (e.g. GP admissions etc) and feed it into a prediction system which returns predictions of hospital admissions broken down into geographical areas. (The prediction systems themselves are not part of
the DEWS project; it is the web services and data standards that share the same technology heritage.)

1.1 Security Requirements

Early in the design of the NDG activities it became clear that a number of guiding principles were going to affect what could be accomplished:

1. Data holders could not, and would not, consider changing the way they stored data and fundamental metadata (including user information).
2. User information could not be shared between partners without explicit authorization by the users on a case-by-case basis.
3. Real existing licensing constraints would affect what common “authorization roles” could be established, and that with more than two partners, common roles across all partners would be so few as to be pointless.
4. Real existing users could not be asked to work directly with X509 certificates.
5. Any new access paradigm had to support what was already possible and did not necessarily have to be any more secure than current practice (although all recognized better security would be good).
6. The technology used had to be portable and easily deployed in groups that have little resource for “tinkering”. (Often such groups have little or no Java experience.)

At the time (2002/03), there was no existing access control paradigm that addressed all these principles satisfactorily, and so the NDG team reluctantly took a decision to engineer their own! Given the resource available a further principle emerged:

7. NDG security had better be simple to devise and build, as well easy to deploy and use.

2. NDG-Security

NDG security provides access control by ensuring that users are granted permission to access a resource by a “gatekeeper”, which has access to (1) an authentication token produced by means of an X509 certificate issued by a “Login-Server” at one of the partners, and (2) an authorization token issued by the “Attribute Authority” associated by the gatekeeper with the resource in question. At that level the NDG-security mechanism is not fundamentally different from other solutions in this space.

2.1 NDG Authorisation

One key area where NDG differs from other solutions is in the detail of how Attribute Authorities issue authorization tokens (and in the format of the token itself, but that is a minor issue, and one that is likely to disappear: while the current format is a bespoke XML document, it is highly likely that the next version of NDG will use a SAML format token).

Principles 3 and 7 listed above led the NDG to decide on a bilateral trust mechanism based firstly on a simple triple defining the access control for a resource: For a Resource A, access control depends on Role B known by Attribute Authority C. Any gatekeeper that is presented with an authorization token (known as an “attribute certificate”) obtained from Attribute Authority C that asserts that the user has role B can then grant access to A.

The bilateral step comes from including within each Attribute Authority the mechanism to establish (zero, one or more) mappings between local roles and those known by remote Attribute Authorities. Thus, if Attribute Authority C trusts Attribute Authority D, and C has an equivalence mapping between role B (known by C) and role E (known by D), then Attribute Authority C can issue a “mapped attribute certificate” to a user (or their software agent) in response to an attribute certificate from D with role E. This “mapped attribute certificate” is an assertion from C that the user has role B, and can thus be used by the gatekeeper process to authorize access.

It is currently a design decision that only one level of mapping is allowed, since the NDG participants could then only envisage a situation where trust could only reliably be extended after an actual human bilateral relationship between the data providers (in practice many of the roles are intended to reflect real legally binding contracts by the users as to what they can do with the data and so the initial role mapping has to be done by a human). However, it is clear now that there are use-cases, particularly for international collaborations, where a second level of mapping from “federation attribute authorities” would be helpful. A future version of NDG security would allow one further mapping from specially designated mappings that identify federation servers.

Currently the implementations in both DEWS and NDG have the gatekeeper and Attribute Authority functionality independent, however it would not be a major step to merge that functionality into a “Policy Enforcement

---

1 In some cases the gatekeeper is a standalone process, in others it is simply a module within the application.
Point” similar to that in PERMIS\(^2\). In such a case, the (standard, non-federated) mapped attribute certificate would no longer have to actually exist as a document in its own right, although the conceptual function would remain. Nonetheless there are use cases where the independence is useful: not all data providers will want to run Attribute Authorities, but they might want to exploit a third party Attribute Authority. This provides a useful level of functionality for real virtual organizations: For example, the constituent universities which make up the National Centre for Atmospheric Science (NCAS) may want to run Gatekeepers locally which bind roles understood by an NCAS Attribute Authority deployed at the BADC to resources which they deploy locally.

### 2.2 NDG Authentication

Authenticating users across multiple domains is now a common problem, and there are multiple solutions: ranging from Shibboleth\(^3\) assertions to OpenID\(^4\). The NDG mechanism is not fundamentally different from either of those.

NDG users start by logging in, usually in response to a request to access to restricted data in the situation where the portal doesn’t yet have access to the required user credentials to present to the gatekeeper. 

The key components in the process are shown in Figure 1. In this case, for browser access to a portal, the gatekeeper process will launch a login page that has much the same functionality as a Shibboleth WAYF (Where Are You From). In the NDG case, the options presented will be for “Login Servers” associated with the attribute authorities that are listed in the mappings available to the Attribute Authority protecting the resource in question. So for example, continuing with the previous nomenclature: Should a user be trying to access resource A, the login page would allow the user to choose between two login servers: those associated with Attribute Authorities C and D.

Each Login Service utilizes a MyProxy\(^5\) server via its own Session Manager service. In NDG that MyProxy server is populated by lightweight\(^6\) certificates issued by a SimpleCA.

---

\(^2\) www.permis.org  
\(^3\) shibboleth.internet2.edu  
\(^4\) openid.net  
\(^5\) grid.ncsa.uiuc.edu/myproxy  
\(^6\) Lightweight: no real effort has been made to secure the SimpleCA.
itself populated and updated directly from existing user databases, but more reliable certificates could be used.

On response to a login request, the Login Server requests a “Session Manager” to create a “User Session” and populate a “Credential Wallet” within it with a proxy certificate. The Session Manager returns the Login Server a handle to the User Session it has created. In the case of a browser process, the handle is then returned via a redirect to an HTTPS GET to the original application (the address of which is passed in the redirect to the Login Server). The application can then populate a cookie with the Session Manager address and User Session details for subsequent use.

2.3 Obtaining Credentials

Now requests from the portal to access the resource are accompanied by the (Session Manager, User Session handle) tuple. The gatekeeper can then request the Session Manager for a specific attribute certificate associated with the appropriate Attribute Authority.

If the user session does not hold an attribute certificate from the Attribute Authority associated by the gatekeeper with the resource, then the Session Manager can use the proxy certificate to request an attribute certificate from the Attribute Authority. If the user is not known by that Attribute Authority, the session manager can obtain a list of trusted attribute authorities for that resource, approach them with the user proxy, take that remote attribute certificate and use it to obtain a mapped attribute certificate from the original Attribute Authority.

A walk through of the entire sequence from login through to a Gatekeeper decision based on a mapped an attribute certificate from Attribute Authority C (based on Attribute Authority D) is show in Figure 2.

2.4 Known Exploitation Strategies

One of the strengths of the NDG-security mechanisms is that apart from applications which have obtained individual credentials, the Session Manager is the only entity with access to credentials which needs to be accessible outside a firewall: the MyProxy Servers do not need to be visible outside. This means that all exploits are limited to obtaining proxy and/or attribute certificates at best.

There is, however, an obvious opportunity for a form of phishing in NDG-security: anyone could produce a “rogue” service which when a user attempts to access it passes a request to a “proper” Login Server. That rogue service, exploiting the open access NDG source code, could then get access to the use of a proxy certificate via the redirect holding the Session Manager handle. The rogue could then purport to require access to a “rogue Attribute Authority”; at which point the Session Manager would give up the proxy certificate to that rogue in an attempt to obtain an attribute certificate. While the NDG currently sees this as a low risk, the most obvious way to avoid the problem is to give all login servers the capability to register associated portals (and/or applications), and require redirections to be signed. This has not yet been implemented.

2.4 Software Implementation

The core NDG security software consists of two components: the server suite, which is needed to deploy an Attribute Authority etc, and the client suite, which is used to build applications.

Both components have been been engineered in Python and have been recently rebuilt as Python eggs to minimize problems with dependencies. The server component currently uses Twisted\(^7\) as the execution environment. The client egg provides a Session Manager client interface that can be plumbed into existing applications, and NDG-security middleware exploiting that client in a WSGI\(^8\) package is nearing completion.

3. NDG-security in DEWS

As outlined in the introduction, the NDG software has been deployed in DEWS, which introduced a new set of requirements over and above those in the original NDG project.

Within DEWS, the major requirements were to (1) streamline the security for a case where the data transfers between one organisation and the servers on the boundary of another were to be controlled by NDG-security, but “the final mile” was internal to the consumer organisation, and (2) deploy NDG-security to protect a specific implementation of the OGC-Web Coverage Service (WCS\(^9\)). While this latter case is definitely an NDG use-case, the NDG experience had thus far been on securing Python applications that could be heavily modified by including the NDG-client code directly.

In both DEWS cases the consumers of NDG-security are Java applications, but using

---

\(^7\) twistedmatrix.com
\(^8\) www.wsgi.org
\(^9\) www.opengeospatial.org/standards/wcs
differing execution environments. While the project set out to use IBM WebSphere exclusively, various practical issues with application portability led to a different strategy. Now IBM WebSphere is used as the container for the gatekeepers and for the portal, and Axis WSS4J is used as an additional client for the Marine stream. In addition, the DEWS-WCS itself is run in Tomcat.

Using the native filtering capability of the containers to do the WS-Security, and having them interact with the NDG-security Python server implementations of WS-security has been an interesting exercise in the real state of
interoperability in this area. In practice, the actual configuration of what, and how, blocks in the messages are signed was a major stumbling block with progress held up by weeks on rather trivial steps. One further unexpected difficulty was also discovered with establishing the chain of trust for proxy certificates: because the standard web service containers expect to have static key stores it is difficult to set up signature verification with dynamically generated certificates.

In DEWS, one of the “users” is in fact a downstream server platform (BMT SeaInfo), which performs search and rescue predictions, thereby adding value to the data served by DEWS-WCS. As was expected, modelling SeaInfo as a dedicated “user” with a static certificate was easy to setup with NDG security, however, more complicated problems were encountered with the WCS access, primarily because of the data volumes.

4. Large Data Issues

One of the major issues faced is how to secure access to both applications and the data they deliver. While the NDG-security paradigm can deal with the logical issues associated with making a policy decision, the practical issue still remains how to deal with the transactions themselves, and in particular key transactions that result in large data transfers (potentially of gigabytes to terabytes).

This issue manifests itself directly when dealing with the OGC WCS specification in the context of both the marine data server in DEWS and the wider NDG. This specification imposes no limit on the size of data that can be requested. That leads to a number of possible responses to a data request, not all of which are supported by the vanilla WCS specification which was not produced with large met-ocean data transfers in mind.

The WCS specification describes a Web Service that receives requests for data either as a URL (via HTTP GET) or an XML document (via HTTP POST with or without SOAP). The DEWS WCS currently only accepts requests via a URL. There are three possible delivery mechanisms for the DEWS WCS from which the client can choose:

1. If the client requests synchronous delivery of data, it will receive the data as a file as a direct response to the request. This is only suitable for small data requests that require only a short time to process; an error message will be returned otherwise. (This is the vanilla WCS response)

2. If the client requests semi-synchronous delivery, it will receive an XML document that contains the location of the data as a URL: this document is returned when the data extraction is finished. The client then downloads data from that URL. (This is a DEWS extension to the WCS spec.)

3. If the client requests asynchronous delivery, it will immediately receive a “ticket” that contains information about the progress of the data extraction: this ticket can be polled to monitor progress. When the client receives an update that says that progress is complete, it can download the data from the web location given in the ticket. (This is also a DEWS extension.)

In order to use WS-Security, all data requests must be formatted as SOAP messages that contain the required security credentials. The current Gatekeeper is a typical SOAP Web Service in that it receives SOAP messages and responds with SOAP messages (“SOAP in SOAP out”). This is fine for the Health stream of DEWS (in which the data sizes are small enough to be encoded in SOAP) and for most messaging in the Marine stream of DEWS. However, because the WCS allows arbitrarily large file responses, and because in the Marine stream large files are actually needed, a problem arises: such files are too big to be encoded in XML SOAP messages, yet the Gatekeeper can only respond to clients with SOAP messages (SOAP “out”).

The existing DEWS prototype deals with this by handling the access control requests using the Gatekeeper, resulting with the final data request returning an obscured time-limited URL, which can be polled using any standard HTTP GET tool. (The data files are placed in a file-system cache and are removed when the download has succeeded).

10 The polling mechanism chosen is based on the OGC Web Processing Specification.
While fundamental premise 5 in section 1.1, allows a modicum of freedom, this solution is in fact a lower level of protection than is currently offered, and so something needs to be done. When one does not wish to modify the application code stack, there are three possible solutions, (1) Stream binary data through the gatekeeper; (2) Use SOAP extensions for binary data transfer, and; (3) Construct a new data delivery web service to wrap the original service with additional new access control syntax.

The first of these solutions would involve changing the Gatekeeper so that it is able to stream binary data (i.e. NetCDF files) as a direct response to a SOAP message. That is to say, change the Gatekeeper from purely “SOAP in”, “SOAP out” behaviour to allow “SOAP in”, “data out” behaviour for the downloading of data. However this would break all the available software tooling, result in a lot of new code, and result in a particularly unportable solution.

The second option would exploit one of three binary transfer extensions to the SOAP standards. The impracticality of encoding large binary datasets as XML in SOAP is widely acknowledged, and has resulted in (at least) three options: (1) Base-64 encoding, (2) SOAP with Attachments (SwA), and (3) Message Transmission Optimization Mechanism (MTOM). Of these three options, vanilla base-64 encoding is not really a solution for large datasets, although it is satisfactory for binary data. The second two options are not conceptually significantly different, but the team is not aware of reliable Python tooling currently available to exploit them. Accordingly, although software libraries exist in Java for all three options, neither NDG nor DEWS have experimented with them.

The third option, like the first, would result in bespoke solution, but the methodology should be relatively easily extensible for modification into a variety of other applications. In essence any data delivery service, including a “special” one developed for DEWS, would be secured by accepting a “security-token” along with the final HTTP GET. The security token would be constructed using pure SOAP by the normal NDG security mechanisms and the standalone Gatekeeper. The client would then need to digitally sign the token (and all the URL arguments) on each use. This signature would ensure that the token was of no use to an attacker without an appropriate private key.

This last solution has one important limitation: with time-dependent target-dependent tokens, the client will not be able to perform synchronous data requests. This is because the initial data request is always made to the Gatekeeper, which will be unable to return data, and replays would not be allowed. The Gatekeeper must return the token and the client must make a separate request to the custom data delivery Web Service. Semi-synchronous and fully-asynchronous requests are still permitted, and these types of requests are more important for DEWS than synchronous requests.

While all these mechanisms require an NDG-security aware client, none of them require modification of the original application. If access to the application code stack is also available, then the access control can be done by SOAP messages independently of the data access and the problem does not exist. However, the third mechanism provides much of the same code and syntax that would be used, in particular, the token exchange could be identical, but with an internal gatekeeper process communicating with attribute authorities etc.

Currently, neither NDG nor DEWS intend to encrypt data during transmission. Clearly therefore, a third party could obtain a copy of the data during transmission, but this is not seen as a significant problem at the moment, whereas the drop in performance associated with encrypting large binary datasets would be an issue. Hence, NDG has no plans to implement encryption in the data transfer component of the security architecture. However, what is important about NDG-security is that an unauthorized third party cannot make a new data request.

With large data transfers, reliability of data delivery becomes an issue. Particularly for long transfers, connections can drop and other problems can occur that mean that a client might wish to resume a broken transfer. Hence it is important that the actual software moving the data is capable of restarting transfers from an arbitrary starting point (i.e. the nth byte). The client also needs to know if the data file has been corrupted in transfer.

There are of course many tools that can handle data transfer in such away, ranging from standard FTP tools and the ubiquitous WGET (with check summing) to fully featured tools.
implementations like GridFTP\textsuperscript{12}. The issue for NDG and DEWS is how best to use such tooling, and it is yet to be resolved.

One final issue is that often it is expected that the client has requested an operation that has created a new dataset that will be cached. It is important then to know when the client has succeeded in obtaining the data, to allow effective cache maintenance. The most reliable way to do this would be for the client to send a message to the server saying “download complete”. This too is inconsistent with a completely synchronous standalone HTTP GET, but is completely consistent with the other alternatives.

5. Plans for the Future

The NDG team has no ambition to sustain a complete security package into the indefinite future. At the same time, it is necessary to deploy NDG-security now, with as many applications as possible. This section addresses these two conflicting requirements.

All security paradigms consist of effectively three stages: secure credential acquisition followed by secure production of the credentials in response to, or in anticipation of, a challenge, concluding with the handling of those credentials in the challenge. Hence plans for incremental change are based around handling these independently as well as addressing what it means to be “secure” while doing so.

There are two key protocols for credential establishment garnering widespread acceptance: OpenID and Shibboleth. It should not be too difficult for NDG-security to produce modified Login Servers that can support either protocol by verifying identity and then generating one-time certificates (perhaps based on Version 3 or later of MyProxy which now supports the ability to generate certificates on demand). However, before that is undertaken, there will need to be: (1) acceptance by attribute authorities in terms of supporting roles for individuals not actually registered at known data providers, and (2) gatekeepers which provide resources for such roles.

The most likely use cases for this class of acceptance is in those cases where security is being implemented to limiting access because of resource constraints, or simply to log access. It is less likely in the short term that either will be acceptable where the security constraints are based on imposing licensing or scientific constraints, where detailed data dependent roles will need to have been matched to individual users at some stage by a human.

There are a number of key applications that will need to be modified to utilise NDG-security before there is general acceptance of the benefits. Modifying such applications will depend on three steps: (1) establishing a security context either before applications are invoked, or early within them, and (2) ensuring that the security context is available subsequently within those applications, and (3) passing that context with network calls.

As well as enabling NDG and DEWS versions of the OGC web services to be NDG-security compliant, two other key applications that will be addressed in the near future are Thredds\textsuperscript{13} and OpenDAP\textsuperscript{14}. Both are in common use in the Met-Ocean community, primarily because of their ease of use with NetCDF archives, and providing access control to those tools would open up a large user community.

Finally, once one entertains handling digitally signed tokens as a security mechanism for regular HTTP GET transactions, the opportunity to completely remove the dependency on SOAP becomes possible. Methods of doing this include using the HTTPsec\textsuperscript{15} emerging standard, or given that signed tokens are all that is really needed if encryption is not important, one could use simple (agreed) modifications of the HTTP headers (as is done by Google\textsuperscript{16}). Given the paucity of existing HTTPsec implementations deploying the latter would be a pragmatic first step. That, along with Shibboleth, would result in a much simpler easy to maintain security system with only the attribute authorities and session managers as unique components.

\textsuperscript{12} dev.globus.org/wiki/GridFTP

\textsuperscript{13} www.unidata.ucar.edu/projects/THREDDS

\textsuperscript{14} www.opendap.org

\textsuperscript{15} http://www.httpsec.org/

\textsuperscript{16} http://code.google.com/apis/accounts/AuthForWebApps.html#signingrequests