

EMI

GridSite 1.x

DOCUMENTATION

Document identifier: **emi-gridsite-doc-1.0.0-1**

Document version: **1.0.0-1**

Date: **April 18, 2011**

Document status: **FINAL**

Document link: **<http://egee.cesnet.cz/cvsweb/SEC/GridSite.pdf>**

Abstract: In its simplest application, GridSite uses X.509 certificates loaded into unmodified versions of web browsers like Internet Explorer, Netscape or Mozilla to authenticate users, and then grants read and write authorization on this basis. HTML and text files can be edited within a browser window, or pages and binary files can be uploaded from local disk. Certificate based authentication of users is now far more practical with the start of large scale issuing of X.509 certificates within Grid projects.

GridSite architecture

The current 1.x series is a substantial rewrite of GridSite, consisting of `mod_gridsite`, `gridsite-admin.cgi`, a toolkit `libgridsite` and the `htcp` command-line HTTP(S) client. Sources of the development versions are available from the CVS Gridsite area, and are published under the Modified BSD License. Sources are available in the GridSite download area.

`mod_gridsite` is a loadable module for the Apache web server which provides access control and page formatting for GridSite HTTP(S) Fileservers, Websites and Web Services hosts. `mod_gridsite` also intercepts some processing in the standard `mod_ssl` module to support GSI Proxies and VOMS attribute certificates, as well as the normal X.509 client certificates. The verification of these credentials is handled by functions within `mod_gridsite` and the main GridSite shared library, without the need to patch or rebuild `mod_ssl`.

CONTENTS

ACCESS CONTROL MODEL	6
AUTHENTICATION	6
VIRTUAL ORGANISATIONS	6
ACCESS POLICIES	6
CVS POLICY	6
CREATING A STABLE CVS BRANCH	7
CONVERT P12	7
HOST CERTIFICATES	7
USER CERTIFICATES	8
GSOAP	8
GRIDSITE	8
GSEXEC	9
HTCP COMMAND	9
MODULE ARCHITECTURE	9
SERVICE DETECTION	10
SLASHGRID HTTP PROFILE	10
DIRECTORY LISTINGS	10
POSIX FILE OPERATIONS	11
USING PHP ON GRIDSITES	11
WORLDS WORST FILE SYSTEM - WWFS	13
WORLDS WORST FILE SYSTEM - WWFS	13
THE FEATURES OF THE WWFS THAT MAKE IT A CANDIDATE FOR THE WORLDS WORST FILE SYSTEM ARE:	13
FEATURES THAT MAKE IT USEFUL:	13
TODO:	14
INTERESTING ISSUES	15
XML NAMESPACES	15
DELEGATION PORTTYPE	15
GACL	15

ADMINISTRATION GUIDE	15
GROUPS AND DN LISTS	16
ACCESS CONTROL LISTS	16
COMPACT CREDENTIALS	16
DELEGATION PROTOCOL	17
INTRODUCTION	17
DEFINITIONS	17
BRIEF DESCRIPTION OF THE GRIDSITE DELEGATION SERVICE	17
USER SCENARIOS	18
GETTING STARTED	18
QUICK USER GUIDES	19
COMPILING THE DELEGATION SERVICE	19
INSTALLATION AND CONFIGURATION OF THE DELEGATION SERVICE	19
RUNNING THE DELEGATION SERVICE	19
DELEGATION SERVICE CLIENTS	19
GRIDSITE DELEGATION SERVICE PROTOCOL SPECIFICATION	20
PROTOCOL OVERVIEW	20
PROTOCOL OPERATIONS	20
GRIDSITE DELEGATION SERVICE MESSAGE STRUCTURES	20
GRIDSITE DELEGATION SERVICE WSDL	21
DELEGATION SERVICE REMOTE METHODS SPECIFICATION	23
IMPLEMENTATION OF THE REMOTE METHODS	24
GRIDSITE X.509 CERTIFICATE FUNCTIONS	24
REFERENCES	25
GACL	25
CREDENTIALS	25
PERMISSIONS	25
ENTRIES	26
ACCESS CONTROL LISTS	26
GRIDFTP AND CHROOT	26
GRIDSITEWIKI	27
HTTP	27
IP PORTS	27

PERL CLIENTS	28
PERL TO ACCESS GRIDSITE OVER HTTPS USING STANDARD X.509 CLIENT CERTIFICATES	29
PERL TO ACCESS GRIDSITE OVER HTTPS USING GSI PROXIES	30
SOAP::LITE TO ACCESS GRIDSITE OVER HTTPS USING GSI PROXIES	31
PERL PEARLS	31
SITECAST	32
USING HTCP FOR FILE LOCATION	32
GROUPS, DOMAINS AND URL-SPACES	32
OPERATIONAL ADVANTAGES FOR STORAGE FARMS	33
SUPPORT IN HTFIND AND HTCP COMMANDS	33
SUPPORT IN SLASHGRID	33
MOD_GRIDSITE DIRECTIVES	33
SUBVERSION	33
VOMS2GACL	34
README	35
RELEASES	36
CRON JOB	36
X.509	36
SEE ALSO	37
BUILD AND INSTALL GUIDE	37
INSTALLING WITH RPM	37
REQUIREMENTS FOR BUILDING GRIDSITE FROM SOURCE	37
BUILDING GRIDSITE WITH MAKE	38
BUILDING GRIDSITE WITH RPM	38
CONFIG GUIDE	38
INSTALLATION	38
CERTIFICATES	39
HTTPD.CONF	39
HTTPD-FILESERVER.CONF	40
HTTPD-WEBSERVER.CONF	40
GRIDSITE DIRECTIVES	40
EU GRID PMA	41

GRACE PARADIGM	41
JOBS AND POOL ACCOUNTS	41
SCRIPTS AND SUEXEC	41
COMBINING POOL ACCOUNTS AND SUEXEC	42
TWO NEW MODES OF OPERATION	42
GRACE PARADIGM	42
GRIDHTTP	43
PROTOCOL	43
ADVANTAGES	43
SERVER SIDE CONFIGURATION	44
CLIENT SIDE EXAMPLES	44
GRIDSITE TOOLBAR	45
INSTALLATION	46
USING THE DELEGATION SERVICE	46
GRIDHTTP USAGE	46
HTTP DOWNGRADE	46
PERL CLIENTS2	47
PERL PEARLS	47
SLASHGRID	47
USER GUIDE	48
READING FROM HTTP AND HTTPS SERVERS	48
AUTHENTICATING	48
AUTHORIZATION	49
MANAGING DIRECTORIES AND 9ILES	49
HTML 9ORMATTING IN GRIDSITE	50
COMMAND LINE USE	51
WEBSERVICES ON GRIDSITE	52
EXAMPLE: WEBSERVICE DELEGATION PROTOCOL	52
X.509 FILE LOCATIONS	53
X.509 USER CERTIFICATES	53
X.509 PROXY CERTIFICATES	53
CERTIFICATION AUTHORITIES	53
HOST CERTIFICATES	53
SEE ALSO	53

ACCESS CONTROL MODEL

GridSite uses an **Access Control model** based on X.509 authentication, various virtual organization / authorization systems, and GACL or XACML policy files.

AUTHENTICATION

Users are identified by X.509 certificates loaded into unmodified web browsers. With commonly used browsers such as Firefox or Internet Explorer, several ways of storing the private key may be used, such as in an encrypted keys file or in an external hardware token.

VIRTUAL ORGANISATIONS

Virtual Organisation and group memberships are defined by lists of members' certificate names ("DN Lists") or by the Fully Qualified Attribute Names of VOMS attribute certificates.

GridSite stores DN Lists in plain text files, and refers to them by LDAP or HTTPS URLs. DN Lists can be retrieved asynchronously from remote authorization servers, or managed locally using the tools described below. The certificate DNs of authenticated users are simply matched with the file containing the relevant DN List to determine their VO and group memberships.

For handling VOMS attribute certificates, a simple parser of X.509 attribute certificates in ASN.1 format is included. This relies on the invariant layout of the ASN.1 tree of objects in attribute certificates. Rather than define the full set of call back functions for all the ASN.1 objects which these certificates can contain, a simpler approach was used: the ASN.1 tree of data nodes is unrolled, each node assigned a co-ordinate and the invariant co-ordinates of each node is used to find its value when required.

ACCESS POLICIES

An XML Grid Access Control Language (GACL) was developed for use with GridSite and other components of the European DataGrid project. For use with websites, the GACL file governing a particular directory or hierarchy is simply stored in that directory as a file .gacl. This policy file allows read, write, list and admin access (giving the ability to modify the policy itself) to be granted or denied on the basis of X.509 identity, GSI proxies, DN List membership, or the possession of a VOMS attribute certificate.

As an alternative, GridSite now also supports simple XACML policies, which are restricted to have the same content as GACL policies. Nevertheless, they are syntactically correct and can also be evaluated by Sun's reference XACML implementation in Java, for example.

When GridSite reads a policy, stored in the .gacl file or otherwise, it determines whether GACL or XACML is present and transparently uses the correct parser. When writing policies, the choice of GACL or XACML can be set by the webserver administrator on a per-directory basis.

CVS POLICY

The **CVS Policy** for GridSite follows the usual Major.Minor.Patch versioning scheme, and roughly follows the numbering scheme policy used by the Linux kernel.

Changes to the Major Version involve a significant change to the APIs or design.

Changes to the Minor Version may involve additions to the API, or a reinterpretation of semantics of the previous API, and you should read the administrator or developer release notes before upgrading.

For even number stable releases (1.0.x, 1.2.x, ...) you should be able to upgrade to a new version of the same Minor Version (eg 1.2.1 -> 1.2.2) without modifying your configuration or code. These changes will principally be bugfixes (but may occasionally involve backwards compatible extensions to the APIs.) Our intention is to branch the code at 1.2.0, 1.4.0 etc and then only make changes to the even numbered branches which are bugfixes.

For odd number development releases (1.1.x, 1.3.x, ...) new APIs may be introduced, modified or removed within the same Minor Version. These releases are intended as preparation for the next stable Minor Version. These odd numbered releases are made by tagging the CVS HEAD branch.

The [ViewCVS graphical tree](http://jra1mw.cvs.cern.ch/cgi-bin/jra1mw.cgi/org.gridsite.core/VERSION?graph=1) (<http://jra1mw.cvs.cern.ch/cgi-bin/jra1mw.cgi/org.gridsite.core/VERSION?graph=1>) of VERSION in org.gridsite.core makes it easier to see the revisions and branches.

CREATING A STABLE CVS BRANCH

For example, to make a stable 1.2 branch, starting from 1.1.19 on the HEAD.

Edit VERSION, CHANGES and project/version.properties to version 1.2.0 then

```
$ cd $HOME/cvspro/org.gridsite.core [or wherever]
$ cvs commit -m 'Start 1.2' .
$ cvstag gridsite-core_R_1_2_0
$ cvs tag -b gridsite-core_branch_1_2
```

Edit VERSION, CHANGES, doc/index.html and project/version.properties to version 1.3.0 then

```
$ cvs commit -m 'Start 1.3' .
```

HEAD is now 1.3.0 and the 1.2 branch can be accessed for bugfixes with

```
$ mkdir $HOME/cvs_1_2
$ cd $HOME/cvs_1_2
$ cvs co -r gridsite-core_branch_1_2 org.gridsite.core
$ cd org.gridsite.core
```

You can then do your bugfixes and commit on the 1.2 branch with

```
$ cvs commit -m 'Important bug fixed' .
```

The [Wikified Cederqvist](http://ximbiot.com/cvs/wiki/index.php?title=Cederqvist) (<http://ximbiot.com/cvs/wiki/index.php?title=Cederqvist>) has more about revisions and branches.

CONVERT P12

If you use a CA like the [UK e-Science CA](http://ca.grid-support.ac.uk/) (<http://ca.grid-support.ac.uk/>) that issues X.509 certificates via web browsers, you may find yourself exporting key/certificate pairs from your browser as .p12 files.

HOST CERTIFICATES

To convert a **host** key/certificate pair, use the following commands:

```
openssl pkcs12 -in host.domain.p12 -clcerts -nokeys -out host.domain.cert.pem
```

```
openssl pkcs12 -in host.domain.p12 -nocerts -nodes -out host.domain.key.pem
```

These files should then be placed in `/etc/grid-security` and `httpd.conf` modified accordingly. `host.domain.cert.pem` can safely be world readable but **host.domain.key.pem must only be readable by root!**:

```
chown root.root host.domain.key.pem
```

```
chmod 0400 host.domain.key.pem
```

USER CERTIFICATES

It is conventional to store user certificates and keys which are used by command line programs like `htcp` in the directory `$HOME/.globus`

```
openssl pkcs12 -in export.p12 -clcerts -nokeys -out $HOME/.globus/usercert.pem
```

```
openssl pkcs12 -in export.p12 -nocerts -out $HOME/.globus/userkey.pem
```

The user certificate can safely be world readable, but **userkey.pem must only be readable by you!**

```
chmod 0400 $HOME/.globus/userkey.pem
```

FUSE - **Filesystem in Userspace** (<http://fuse.sourceforge.net/>) - is a Linux kernel module which forms the basis of GridSite's **SlashGrid** filesystem HTTP(S) client.

FUSE became a part of the official Linux kernel with version 2.6.14, but had existed as a third-party module for several years (originally as part of the AVFS project.)

RedHat's community supported distribution, Fedora Core, includes the kernel module for the FC4 and FC5 releases (and FC6 in development), with the libraries and commands available as part of Fedora Extras.

Since RedHat Enterprise Linux (and therefore Scientific Linux) has not yet reached kernel 2.6.14, it is necessary to produce FUSE binaries ourselves. A suitable `fuse.spec` is in the core GridSite doc directory for version 1.3.0 and above, and some binary RPMs are published in [the download area](http://www.gridsite.org/download) (<http://www.gridsite.org/download>

GSOAP

gSOAP is an web services development toolkit for C/C++, which is used by GridSite's Web Services module to handle the (de-)serialisation of C structures and XML.

gSOAP is only required for building GridSite WS, and a suitable `gsoap-devel.spec` is in the GridSite-WS doc directory for version 1.1.2 and above, and some binary RPMs are published in the GridSite [download area](http://www.gridsite.org/download/gsoap/) (<http://www.gridsite.org/download/gsoap/>).

GRIDSITE

GridSite was originally a web application developed for managing and formatting the content of the GridPP website. Over the past three years it has grown into a set of [extensions to the Apache web server](#) and a toolkit for Grid credentials, GACL access control lists, HTTP(S) protocol operations and Grid/Web service protocols, such as the [delegation protocol](#).

GSEXEC

gsexec is a modified version of Apache's **suexec** (<http://httpd.apache.org/docs/2.0/suexec.html>), and is used as part of the GRACE model of CGI service hosting.

HTCP COMMAND

htcp is part of a family of Unix commands for performing file operations on remote HTTP and HTTPS file servers. The **htcp man page** (<http://www.gridsite.org/1.1.x/htcp.1.html>) explains the command syntax in detail, along with htls, htll, htrm, htmkdir and htmv.

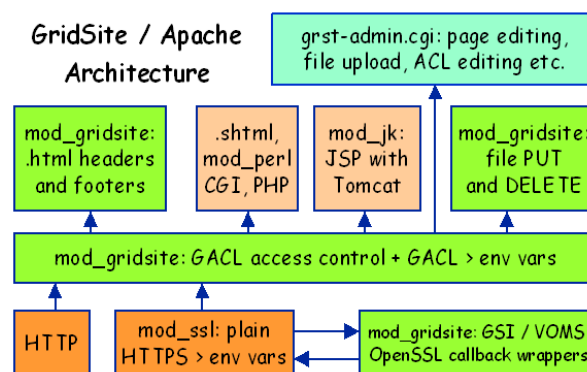
These commands search the standard **X.509 File Locations** for a client certificate to use.

In particular, these commands provide client-side support for the GridSite extensions to the Apache web server, include GSI proxies, authorization by X.509 certificate, PUT / DELETE / MOVE support, and **GridHTTP** bulk data transfers.

MODULE ARCHITECTURE

The **Module Architecture** implemented by GridSite extends Apache by adding extra components at the access control and HTTP method execution layers, and supports CGI services supplied with GridSite or written by third parties.

mod_gridsite is a loadable module for the Apache web server which provides access control and page formatting for GridSite HTTP(S) File servers, Websites and Web Services hosts. **mod_gridsite** also intercepts some processing in the standard **mod_ssl** module to support GSI Proxies and VOMS attribute certificates, as well as the normal X.509 client certificates. The verification of these credentials is handled by functions within **mod_gridsite** and the main GridSite shared library, without the need to patch or rebuild **mod_ssl**.



SERVICE DETECTION

Traditional web services require that the client is either configured for one particular instance of the service, with a hard-coded location (as is the case with the command line client for the GridSite [Delegation protocol](#) , once compiled), or that the user of the client provides the URL of the service they wish to use. This can cause problems if locations of services change or users wish to locate alternative services providing the same functionality.

A system was developed to allow a browser (and in turn the [GridSite Toolbar](#)) to be notified of an available delegation service by a web site. The URL of the service is provided either in HTTP headers or in a META element in the HTML source of the page being viewed. The META element method is similar to the method employed by sites and browsers to enable automatic location of RSS feeds, which uses a LINK element.

The header used for the notification is "Proxy-Delegation-Service", the value of which should be the URL of the delegation service. The insertion of this header can be easily achieved with GridSite Apache module and setting the GridSiteDelegationURI directive in the web server's configuration file. To deliver the same information without using HTTP headers, a META element containing the http-equiv attribute can be inserted into the HEAD element of a web page. The META element would have the following format:

```
<META http-equiv = "Proxy-Delegation-Service" content = "https://eg.com/delegate.cgi">
```

The combination of the two methods above provides great flexibility for different groups of people to alert users to a delegation service. The HTTP header method allows a web server administrator to inform all visitors of a related delegation service and META element method allows an individual page (a personal home page, for example) to do the same. The latter may be useful in an environment where enabling server-wide options are not possible (e.g. pages located on unrelated servers) or on a server not using the GridSite software.

SLASHGRID HTTP PROFILE

The **SlashGrid HTTP Profile** defines a way of using components of [HTTP](#), [WebDAV](#) and [HTML](#) to achieve filesystem semantics similar to POSIX, via HTTP(S) requests. This profile is implemented by [GridSite](#)/Apache servers, by GridSite's [http](#) command-line clients, and the [SlashGrid](#) virtual filesystem client. As a profile rather than a new standard, interoperation degrades gracefully when using third-party clients and servers which are unaware of the profile.

DIRECTORY LISTINGS

Directory listings are obtained by making an HTTP GET request for the directory URL, including the trailing slash. The server returns an HTML file, and the HREF attributes of A elements are parsed to construct a list of canonicalised URLs. Those URLs which represent files and immediate subdirectories of the directory are interpreted as its members.

'Long listings' and file stat() operations - including file modification times, length and type - can be constructed by issuing HTTP HEAD requests for the member URLs to obtain the Last-Modified (ctime and mtime) and Content-Length (size) header values. URLs ending in slash are interpreted as directories (and an HTTP 301 redirect response and Location header may be received if a trailing slash is missing from the request URL for a directory.)

Servers may improve the efficiency of this process by: (1) always including a trailing slash when linking to subdirectories; (2) include a content-length attribute in the A element, to give the size in bytes; (3) include a last-modified attribute in the A element, to give the modification time of the file, as seconds since 1 Jan 1970.

POSIX FILE OPERATIONS

To be completed

open - null operation (or file descriptor / file name book keeping)

close - null operation

read - GET /... with Range: x-y

write - PUT /... either with Content-Range: x-y; or without range for O_TRUNC; if creating, then create temporary file, seek to end (ie to final size), write one dummy byte, write file from the start, rename to final name on success or destroy. This provides space reservation (based on counting st_size values of all files to find space used/reserved.)

unlink - DELETE /...

rmdir - DELETE /.../

mkdir - PUT /.../

rename - MOVE /... Destination: /...

trunc - PUT with Content-Range: *-*/nnn

stat - HEAD /... (Last-Modified: Content-Length: and trailing slash if directory)

scandir - GET /.../ (HREF="/..." links used to construct list of directory members.)

USING PHP ON GRIDSITES

PHP (<http://www.php.net/>) is a popular server side scripting language, which is typically used to generate web pages dynamically by mixing sections of PHP code (similar to C or Javascript) inside HTML. PHP works well on Apache / GridSite servers, and access to PHP pages can be controlled in the same way as CGI programs.

The environment variable GRST_PERM, which is exported to CGI, PHP and other dynamic content by `mod_gridsite`, holds a bitmask with the permissions the client has in the context of the current page. The permissions defined in `gridsite.h` are currently (GridSite version 1.1.11):

```
GRST_PERM_NONE      0
GRST_PERM_READ      1
GRST_PERM_EXEC       2
GRST_PERM_LIST       4
GRST_PERM_WRITE      8
GRST_PERM_ADMIN     16
```

Page access is denied by `mod_gridsite` if GridSite access control is enabled and the client does not have the `GRST_PERM_READ` permission. PHP scripts can test `GRST_PERM` for additional permissions. `GRST_PERM_EXEC` is especially useful, since it is not used by `mod_gridsite` and can be assigned a

meaning by PHP or CGI scripts (eg the ability to modify a database through a PHP/HTML form, without giving users GRST_PERM_WRITE and the ability to modify the PHP script itself.)

This example shows a skeleton PHP page which uses the GRST_PERM_LIST permission to decide whether to show the Manage link in the page footer. The rest of the PHP near the footer simulates the standard GridSite footer inserted into HTML pages when mod_gridsite's page formatting is enabled:

```
<title>Dynamic page title!</title>
<?php include "/var/www/htdocs/gridsitehead.txt"; ?>

<h1>Welcome to this page!</h1>

<p>Interesting, dynamic page content, written in PHP.

<?php

// Do a GridSite-like admin footer, but in PHP

if ($_SERVER["SSL_CLIENT_S_DN"] != "")
    print "<small>You are " . $_SERVER["SSL_CLIENT_S_DN"] . "</small><br>";

print "<small>";

if ($_SERVER["GRST_PERM"] & 4)
    print "<a href=\"/gridsite-admin.cgi?cmd=managedir\">Manage directory</a> . ";
if ($_SERVER["HTTPS"] == "")
    print "<a href=\"https://www.dom.ain/\">Switch to HTTPS</a> . ";
else
    print "<a href=\"https://www.dom.ain/\">Switch to HTTP</a> . ";

print "<a href=\"/website/gridpp-user.html\">Website help</a> . ";
print "Built with <a href=\"http://www.gridsite.org/\">GridSite</a></small>";

include "/var/www/htdocs/gridsitefoot.txt";

?>
```

But remember, when you give someone the ability to upload PHP on your site, you are effectively giving them command-line access as the user which runs the PHP - probably apache! It may be better to use [gsexec](#) and PHP as a CGI script in some situations.

WORLDS WORST FILE SYSTEM - WWFS

WORLDS WORST FILE SYSTEM - WWFS

The idea is to build a simple file staging area on top of GridSite for the OMII [Application Hosting Environment Project\(AHE\)](#) (<http://kato.mvc.mcc.ac.uk/rss-wiki/ApplicationHostingEnvironment>) project. The key extension to GridSite is that support extensive meta data about the files has been added. The code has been written in Perl and is run as a CGI script in GridSite.

This is only a prototype but someone might find it useful.

WWFS was built according to the constraints of [REST](#) (http://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style) defined in Roy Fieldings doctoral thesis. From the thesis "Another conflict with the resource interface of REST occurs when software attempts to treat the Web as a distributed file system. Since file systems expose the implementation of their information, tools exist to "mirror" that information across to multiple sites as a means of load balancing and redistributing the content closer to users. However, they can do so only because files have a fixed set of semantics (a named sequence of bytes) that can be duplicated easily. In contrast, attempts to mirror the content of a Web server as files will fail because the resource interface does not always match the semantics of a file system, and because both data and metadata are included within, and significant to, the semantics of a representation."

[Web Dav](#) (<http://www.ietf.org/rfc/rfc2518.txt>) was not used because it was decided that it was not RESTful: it uses stateful interactions for locking, URI's are not opaque (Web Dav uses a hierarchical namespaces and URIs) and the introduction of the extra Web Dav verbs is in conflict with the concept of a uniform interface. It is possible to argue that Web Dav is RESTful but that is another story...

THE FEATURES OF THE WWFS THAT MAKE IT A CANDIDATE FOR THE WORLDS WORST FILE SYSTEM ARE:

- No directories - allows URIs to be opaque
- No file locking
- No Posix interface
- All access through HTTP (GET, POST, DELETE, HEAD, PUT)

FEATURES THAT MAKE IT USEFUL:

Extensive Meta Data support

MD5 checking support, any files written are MD5 checked after writing for errors

Tags - allows you to create "tags", unique URI's, to tag a file, useful for searching for files. A file can have many Tags associated, for example you could mark all the files for a particular job with a unique Tag. The Tags are resources so you can get/set data about them.

CommonName - you can give your files common names, many files can have the same name (no concept of directory so no name clashes).

Cache-Control - you can set the cache control for the file, allows conditional gets and makes HEAD useful

Support conditional GETs, can use HEAD for checking file changes.

All meta data stored in MySQL database including file location - files can be stored in places other than the Web server and files can be moved around behind the scene without the clients being aware

Database handles concurrency issues.

Use "stable writes" for PUTs to handle write failures - when a PUT is invoked a new file is created and only if the new file is written correctly is the database updated. Database transactions used in case of failure during database update.

Everything has an ACL (Access Control List) expressed as GACL ([Access Control model](#)) even ACLs. If an ACL is not set for a resource then the owner of the resource is the only person with access.

Use links to ACLs to set access control ie. you can create your favourite ACL and use the URI for that ACL to set the access control for lots of things (this adds a layer of indirection, which adds complexity, but seems to be worth it).

Extensive use of Xlink - this allows you to search/navigate to files, you can select files by common name, last modified date, creation date, by "tag", by file type etc.

Supports user defined meta data, ie you can create and add your own meta data, eg for an output file you could set a JPEG as the meta data.

Logging information provided for all file access.

You can chain files together - there is a place in the meta data for identifying a "previous version" of the file.

The ACLs are serialised into the database - making searching fast and simple.

TODO:

Output file logs as ATOM/RSS

Loggin available only for files at the minute, access to all resources IS logged but just not available.

Add support for CSS/XSLT/XHTML so browsers display the data nicely

Add support for uploading files through browser

Add support for XACML - use Accept header in request to decide which format to return

Add support for conditional gets on ACLs

Add RDF and Semantic Web technology

Add support for remote ACLs, use caching, cache-control, HEAD and conditional get for performance

Add users to upload meta data about them selves.

Add MD5 header support for everything not just files

Collections - a user can create a file with a set of Xlinks in it to create a collection, should formal support be added?

Create client test suite.

Add support for authentication support for other things than X509

There is far too much code repeated - it should be refactored, especially the authorisation step.

If you have a UK e-Science certificate you can upload a file to the service at <https://garfield.mvc.mcc.ac.uk/cgi-bin/zcgmka/File>, simply use HTTP to POST a file to the address (you will need to use mutual authentication) - you should get XML with Xlinks that will allow you to navigate around the WWFS. When you POST a file for the first time a resource is created that holds information on all your files, from this point you can access all the files you have created. When you do a GET on a file there is an extra HTTP Header added, Meta-Location, this is set to a URI that identifies the resource that holds the

meta data for a file - so if someone sends a pointer to a file in the WWFS it is possible to access the meta data for the file.

INTERESTING ISSUES

GACL did not have a "delete" permission tag, this is an example of the mis-match that Fielding pointed out when comparing a file system to the Web. There is no delete permission in a UNIX file system, if you have write access to a directory you can delete files in it.

When you POST a file should you get back as the Location HTTP header the location of the file or the location of the meta-data? Or do you use the Content-Location header? The WWFS returns the location of the meta-data, part of the meta data is the location of the file.

Code is written as a Perl CGI script - didn't use mod_perl, the mod_perl books reckon it is not worth while using mod_perl for file upload/download services as the bottleneck is uploading/downloading files not parsing the script.

The code is available at <http://garfield.mvc.mcc.ac.uk/cgi-bin/zzcgumk/a/File/118> and the database structure is available at <http://garfield.mvc.mcc.ac.uk/cgi-bin/zzcgumk/a/File/117> - when you do a GET on the files you can check the HTTP Headers for the Meta-Location ;-)--Mark mc keown 16:55, 4 Oct 2005 (BST)

XML NAMESPACES

GridSite currently uses the following XML Namespaces:

DELEGATION PORTTYPE

This Web Service uses the namespace <http://www.gridsite.org/namespaces/delegation-1>

GACL

The Grid Access Control Language uses the namespace <http://www.gridsite.org/namespaces/gacl-0.0.1>

ADMINISTRATION GUIDE

This Guide is intended for people administrating areas of GridSite websites or file servers, or managing GridSite's DN List groups - that is, how to use GridSite to manage other people's access to parts of the site - for example, people's write access to areas devoted to specific subprojects.

There is a separate [User Guide](#) which explains how to authenticate to the server with X.509 certificates, and how to manage files via a standard web browser or with command-line HTTPS clients. You should be familiar with the User Guide to fully understand this Admin Guide.

You may also find the [Config Guide](#) useful to understand how the Apache webserver is configured with GridSite extensions. If you are also the Apache webmaster for your site, you will definitely need to read the Config Guide to create the httpd.conf file. However, if you only need to manage webpages and files, then this Admin Guide and the User Guide should be sufficient.

GROUPS AND DN LISTS

GridSite defines groups of people using plain text DN Lists - that is, lists of people's certificate DNs. Each DN List has a URL which uniquely identifies the list (and may also allow other sites to obtain the list and use it themselves.) For example, the list of all GridPP members is <https://www.gridpp.ac.uk/dn-lists/gridpp> (note that it's https:// not http:// - this means that other sites that download the list can check the certificate of www.gridpp.ac.uk and know they're talking to the authoritative source of the lists.)

The system can also have a number of other DN Lists which are associated with specific groups of people and perhaps with specific areas of responsibility of the website. If the DN List directory URI is /dn-lists/ then there is a full list of the DN Lists exported by the server at that URI (for example, <https://www.gridpp.ac.uk/dn-lists/>)

If you have permission to modify a DN List, you can start changing it by going to /dn-lists/ (via HTTPS), using the "Manage directory" button and finding the URL of your DN List in the listings. You may need to go down into a subdirectory to find your list. For example, <https://www.gridpp.ac.uk/dn-lists/atlas> is in the atlas subdirectory of /dn-lists/ (You may wish to bookmark the listing of such a directory if you frequently work with one.)

DN List directories are managed by the ACLs described in the next section, and if you have write permission, you can edit the lists already there, and add new lists with the same prefix (this means you can readily create your own subgroups.)

ACCESS CONTROL LISTS

DN Lists appear in the Grid Access Control Lists (GACL) used by GridSite. These are stored as .gacl files in directories: if the .gacl file is present, it governs access to the directory; if it is absent, then the parent directories are searched upwards until a .gacl is found.

The GridSite [GACL Reference](#) explains the XML format of these files, but they can be edited using the ACL editor built into the GridSite system by people who have the Admin permission within the ACL.

If you have this permission in a given directory, when you view directory listings or files in that directory you will see the option "Manage Directory" in the page footer. This allows you to get a listing of the directory and the .gacl file will appear at the top if it's present. If not, then there will be a button to create a new .gacl file with the same permissions as have been inherited by that directory from its parent.

GACL allows quite complex conditions to be imposed on access, but normally you can think of an ACL as being composed of a number of entries, each of which contains one condition (the required credential) and a set of allowed and denied permissions.

Credentials can be individual user's certificate names or whole groups of certificate names if a DN List is given. (You can also specify hostname patterns using Unix shell wildcards (eg *.ac.uk) or EDG VOMS attribute certificates - see the GACL Reference for details.)

Permissions can be Admin (edit the ACL), Write (create, modify or delete files), List (browse the directory) or Read (read files.) Permissions can be allowed or denied. If denied by any entry, the permission is not available to that user or DN List (depending on what credential type was associated with the Deny.)

COMPACT CREDENTIALS

Compact Credentials are a way of representing GridSite's internal GRSTcred structures as strings. They are used in the GRST_CRED_n environment variables exported by mod_gridsite to CGI, SHMTL, PHP etc.

It always starts with the type of credential (X.509, GSI PROXY, VOMS Attribute), the next two numbers are the times that the credential are valid between (as seconds since 00:00:00 1970-01-01 UTC), the 4th value is the level of delegation (for that particular credential) and the final value is the DN or the VOMS FQAN.

DELEGATION PROTOCOL

INTRODUCTION

Delegation is a common requirement for a wide range of Grid applications. In line with the Web Services based Open Grid Services Architecture, and considering the fact that current in-place security mechanisms will continue to be used, defining a standard delegation mechanism is an essential effort.

By describing delegation as a standalone Web Services portType, and by providing ready-to-use library implementations of this portType, service implementers do not need to deal with the details of the delegation mechanisms, and can factor this functionality out of the internal application logic. Furthermore, delegation can be instantiated as a standalone service, front-ending a (trusted) credential repository, from which the target service can then extract the delegated credential in a controlled and secure manner.

The Delegation portType implementation included in GridSite illustrates how web services can be built using C and gSOAP (<http://www.cs.fsu.edu/~engelen/soap.html>) toolkit. The protocol was agreed within the EGEE project, and both C (GridSite) and Java implementation are available as part of EGEE's gLite framework. This document describes the GridSite implementation of the framework.

DEFINITIONS

- *Delegation* is the act of transferring rights and privileges to another party (the Delegatee).
- *The Delegatee* is the requestor of delegation. It is the entity that the Delegation Credential is delegated to.
- *The Delegator* is the entity that delegates the abilities and/or rights to the Delegatee.
- *A proxy certificate* is an X.509 certificates issued by the end-user (the Delegator) to a process acting on end-user's behalf (the Delegatee). The proxy certificate carries the identity of the Delegator. That is, it can be used to establish SSL connections and is interpreted using the GSI as authority to perform work on the Delegator's behalf.
- *gSOAP* is a set of tools for generating the WSDL description of a Web Service from the C header files of the functions that implement it or vice versa.

BRIEF DESCRIPTION OF THE GRIDSITE DELEGATION SERVICE

The GridSite Delegation Service (GridDeS) operates as a service with a single portType. It can be operated as a CGI program with a WSDL description and signing functions or standalone service. As a CGI program, the SOAP request received by Apache will be fed into the standard input of the CGI program and the SOAP response will be taken from the standard output. All the required authentication information by the CGI Web Service program is made available as environment variables by the GridSite.

The GridDeS is linked to the GridSite library to obtain access to its private and certificate signing functions because of the two levels of credential processing taking place i.e. authentication and authorization of the client attempting the delegation.

USER SCENARIOS

GSI delegation is basically exchange of messages between two partners. To perform delegation, the client sends a Get Proxy Request to the server, which causes the server to generate a public and private key and return an X.509 certificate signing request containing the public key.

The client then signs this request using its own private key and certificate, and sends a Put Proxy message back to the server, containing the signed certificate. Together the private key which the server generated and the new certificate form the proxy certificate (RFC3820 / GSI proxy).

Some of the use cases for delegation that are not well covered by X.509 public key certificate alone are:

- *Dynamic delegation:* It is often the case that a Grid user needs to delegate some subset of their privileges to another entity on relatively short notice and only for a brief amount of time.
- *Dynamic entities:* In addition to delegation to persistent services and entities, the requirement exists to support delegation of privileges to services that are created dynamically, often by the user them self, that do not hold any form of identity credential. A common scenario is that a user submits a job to a computational resource and wants to delegate privileges to the job to allow it to access other resources on the user's behalf, for example, to access data belonging to the user on other resources or start sub-jobs on other resources.
- *Repeated Authentication:* It is common practice to protect the private keys associated with X.509 public key certificates either by encrypting them with a pass phrase (if stored on disk) or by requiring a PIN for access (if on a smart card). This technique poses a burden on users who need to authenticate repeatedly in a short period of time, which occurs frequently in Grid scenarios when a user is coordinating a number of resources.

GETTING STARTED

The followings are required for building the GridDeS:

- the GridSite source code.
- a C or C++ compiler.
- the gSOAP 'stdsoap2' stub and skeleton compiler.
- the gSOAP 'wsdl2h' WSDL parser, which convertst WSDL into a gSOAP specification header file for the 'stdsoap2' stub and skeleton compiler.
- the 'stdsoap2.c' and 'stdsoap2.h' files containing the runtime library to be linked with the GridDeS application.

The gSOAP can be downloaded from [4].

QUICK USER GUIDES

The user guide provides a quick way on how to get started with the GridDeS using gSOAP toolkit. You need a basic understanding of the SOAP protocol and some familiarity with C and/or C++. The GridDeS and its client are developed in C using the gSOAP tools. You don't need a detailed understanding of the SOAP protocol before using GridDeS. In this section, we describe the GridDeS and its client.

COMPILING THE DELEGATION SERVICE

The delegation service implementation is included with the GridSite download. After downloading the gSOAP, set up the directory location containing gSOAP soapcpp2 and stdsoap2.h in the Makefile located in the source sub-directory (src) of the GridSite. To compile the service 'type make gridsite-delegation.cgi' in the 'src' directory. This will compile the delegation service as a CGI program.

The compilation of the CGI program starts with the invocation of the gSOAP stub and skeleton compiler on the delegation.h header file from within the Makefile. The compiler generates the skeleton routines for the getProxyReq and putProxy remote methods specified in the delegation.h header file, and also produces a WSDL that advertises the delegation service. The skeleton routines are respectively, soap_serve_ns__getProxyReq and soap_serve_ns__putProxy and saved in the file soapServer.c. The generated file soapC.c contains serializers and deserializers for the skeleton. The compiler also generates a service dispatcher-the soap_serve function. See [4] for further details of how to develop and compile a web service with gSOAP.

INSTALLATION AND CONFIGURATION OF THE DELEGATION SERVICE

The service is designed to run as run as a CGI application at GridSite location (<https://grid7.hep.man.ac.uk/gridsite-delegation.cgi>) for example. To install the service, copy the compiled gridsite-delegation.cgi to the 'cgi-bin' directory of your Web Server or set the installation directory in the http.conf of the Web Server using ScriptAlias.

You will also need to create a new directory called 'proxycache' in the www directory of your Web Server where the proxy certificates are going to be stored. See documentation for GridSite [6] on how to get and configure the host certificate and host key for your Web Server machine.

RUNNING THE DELEGATION SERVICE

After the installation and configuration, the GridDeS is ready to serve request from the clients. The soap_serve function as mentioned above acts as a service dispatcher. It listens to client requests on standard input stream and invokes the remote method implementation function via a skeleton routine to serve a SOAP client request. The response is encoded in SOAP and send to standard output after the request is served. Note that the function prototype of the remote method implementation function is specified in the header file that serves as input to the gSOAP compiler.

DELEGATION SERVICE CLIENTS

The GridDeS is designed to work with any type of clients that use SOAP protocol. There is an example client that comes with the GridDeS, which is command line based client. To compile this client, type 'make htproxypu' in the delegation.h header file directory location. To run or use the client, you must copy your valid certificate (usercert and userkey) into the '.globus' directory of your machine. The certificate is used to sign the proxy certificate. The [GridSite Toolbar](#) is a browser based client for the GridDeS in the form of a [Mozilla Firefox](http://www.mozilla.com) (<http://www.mozilla.com>) extension.

More types of clients possible?

- Browser based with Applet/Java Script/PHP?
- Small Java application agent that can be downloaded and installed through the Internet?
- other methods?

GRIDSITE DELEGATION SERVICE PROTOCOL SPECIFICATION

This section defines the message exchange required for the GridDeS implementation.

PROTOCOL OVERVIEW

GridDeS protocol is based on SOAP Request/Response message pair from a Delegator to a Delegatee or vice versa. The followings show the GridDeS Request/Response steps:

- The Delegator issues a `getProxy` request and send that to Delegatee, who verifies the validity of the request, generates a pair of key (public and private keys) and sends the response containing the public key back to the Delegator.
- The Delegator receives the response message, signs the public key with its certificate and issues a `putProxy` request to the Delegatee. The public key, which is signed using the Delegator's private key associated with its long time public key certificate constitutes the proxy certificate which is kept together with private key in a file with the Delegatee.

PROTOCOL OPERATIONS

The GridDeS presently supports two operations that can be requested by the Delegator to be performed by the Delegatee:

getProxyReq creates a new delegation request. The end result is a pair of key for the proxy certificate.

putProxy sends a new signed public key back to the Delegatee. The end result is a proxy certificate.

GRIDSITE DELEGATION SERVICE MESSAGE STRUCTURES

The GridDeS message model as described consists of two SOAP messages: Request and Response.

- *Request Message*: This message is issued by the Delegator either to ask the Delegatee for a public key to base the proxy certificate on or to push the proxy certificate to the Delegatee. A sample of the SOAP Request message from the Delegator to the Delegatee for `getProxyReq` is depicted below:

```
. . . . .  
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"  
xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"  
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
```

```
xmlns:ns="urn:delegation"  
SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">  
<SOAP-ENV:Body>  
  <ns:getProxyReqRequest>  
    <delegationID xsi:type="xsd:string">20050001</delegationID>  
  </ns:getProxyReqRequest>  
</SOAP-ENV:Body>  
</SOAP-ENV:Envelope>
```

- **Response Message:** This message contains a generated public key and it is sent from the Delegatee to the Delegator in response to a getProxyReq request. Sample of the above getProxyReq Response message is shown below:

```
. . . .  
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"  
xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"  
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
xmlns:xsd="http://www.w3.org/2001/XMLSchema"  
xmlns:ns="urn:delegation"  
SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">  
<SOAP-ENV:Body>  
  <ns:getProxyReqResponse>  
    <request xsi:type="xsd:string">MI IuuBB33Jjjjj0222MmEppoooddWwwwlll . . . </request>  
  </ns:getProxyReqResponse>  
</SOAP-ENV:Body>  
</SOAP-ENV:Envelope>
```

GRIDSITE DELEGATION SERVICE WSDL

The WSDL description of the GridDeS reproduced below is automatically generated from the header file using gSOAP.

```
<?xml version="1.0" encoding="UTF-8"?>  
<definitions name="delegation"  
  targetNamespace="http://www.gridsite.org/ns/delegation.wsdl"  
  xmlns:tns="http://www.gridsite.org/ns/delegation.wsdl"  
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"  
  xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"  
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"  
  xmlns:ns="urn:delegation"  
  xmlns:SOAP="http://schemas.xmlsoap.org/wsdl/soap/"
```

```
xmlns:MIME="http://schemas.xmlsoap.org/wsdl/mime/"
xmlns:DIME="http://schemas.xmlsoap.org/ws/2002/04/dime/wsdl/"
xmlns:WSDL="http://schemas.xmlsoap.org/wsdl/"
xmlns="http://schemas.xmlsoap.org/wsdl/">
<types>
  <schema targetNamespace="urn:delegation"
    xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:ns="urn:delegation"
    xmlns="http://www.w3.org/2001/XMLSchema"
    elementFormDefault="unqualified"
    attributeFormDefault="unqualified">
    <import namespace="http://schemas.xmlsoap.org/soap/encoding/" />
  </schema>
</types>
<message name="getProxyReqRequest">
  <part name="delegationID" type="xsd:string"/>
</message>
<message name="getProxyReqResponse">
  <part name="request" type="xsd:string"/>
</message>
</message>
<message name="putProxy">
  <part name="delegationID" type="xsd:string"/>
  <part name="proxy" type="xsd:string"/>
</message>
<message name="putProxyResponse">
</message>
<portType name="delegationPortType">
  <operation name="getProxyReq">
    <documentation>Service definition of function ns__getProxyReq</documentation>
    <input message="tns:getProxyReqRequest"/>
    <output message="tns:getProxyReqResponse"/>
  </operation>
  <operation name="putProxy">
    <documentation>Service definition of function ns__putProxy</documentation>
    <input message="tns:putProxy"/>
```

```
<output message="tns:putProxyResponse"/>
</operation>
</portType>
<binding name="delegation" type="tns:delegationPortType">
  <SOAP:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
  <operation name="getProxyReq">
    <SOAP:operation style="rpc" soapAction=""/>
    <input>
      <SOAP:body use="encoded" namespace="urn:delegation" encodingStyle="http://schemas.xmlsoap.org/soap/encoding"/>
    </input>
    <output>
      <SOAP:body use="encoded" namespace="urn:delegation" encodingStyle="http://schemas.xmlsoap.org/soap/encoding"/>
    </output>
  </operation>
  <operation name="putProxy">
    <SOAP:operation style="rpc" soapAction=""/>
    <input>
      <SOAP:body use="encoded" namespace="urn:delegation" encodingStyle="http://schemas.xmlsoap.org/soap/encoding"/>
    </input>
    <output>
      <SOAP:body use="encoded" namespace="urn:delegation" encodingStyle="http://schemas.xmlsoap.org/soap/encoding"/>
    </output>
  </operation>
</binding>
<service name="delegation">
  <documentation>gSOAP 2.7.4 generated service definition</documentation>
  <port name="delegation" binding="tns:delegation">
    <SOAP:address location="http://localhost/delegserver.cgi"/>
  </port>
</service>
</definitions>
```

DELEGATION SERVICE REMOTE METHODS SPECIFICATION

The GridDeS is developed by developing a header file that contains the C service remote methods and data types. The remote methods are based on the above GridDeS protocol specification. These methods must be implemented in the server application. The function prototype in the header file must be a valid

prototype of the method implemented as a C/C++ function. See section on compiling the delegation service on how to compile the delegation service and its WSDL from the header file. The header file for the GridDeS is shown below:

```
//gsoap ns service name:      delegation
//gsoap ns service style:     rpc
//gsoap ns service encoding:  encoded
//gsoap ns service namespace: http://www.gridsite.org/ns/delegation.wsdl
//gsoap ns service location:  http://localhost/delegserver.cgi
struct ns__putProxyResponse { } ;
//gsoap ns schema namespace:  urn:delegation
int ns__getProxyReq(char *delegationID, char **request);
int ns__putProxy(char *delegationID, char *proxy, struct ns__putProxyResponse *unused);
```

The remote method 'ns__putProxy' has a response struct that is empty because it has no output parameters. The WSDL description of the delegation service generated from the header file is also located in the same directory with the Makefile (i.e. src directory).

IMPLEMENTATION OF THE REMOTE METHODS

As earlier stated, the application of gSOAP compiler on the header file produces C/C++ source files that are used to build the delegation service and client applications in C/C++. The gSOAP compiler acts as a preprocessor. The details of the files generated by the compiler are:

<i>File Name</i>	<i>Description</i>
soapStub.h	A modified and annotated header file produced from input header file
soapH.h	Main header file to be included by all client and service sources
soapC.c	Serializers and deserializers for the specified data structures
soapClient.c	Client stub routines for remote operations
soapServer.c	Service skeleton routines
soapClientLib.c	Client stubs combined with local static (de)serializers
soapServerLib.c	Service skeletons combined with local static (de)serializers

The implementation of the remote methods return a SOAP status code. The code SOAP_OK denotes success, while soap_sender_fault returns an exception.

GRIDSITE X.509 CERTIFICATE FUNCTIONS

The GridDeS remote methods implementation uses some function from the GridSite X.509 certificate functions:

The ns__getProxyReq(..) method uses GRSTx590GRSTx509MakeProxyRequest(...) function to make and store a X.509 request for GSI proxy. Its returns GRST_RET_OK on success, non-zero otherwise. The key is stored in proxydir as temporary file.

The ns__putProxy method uses the GRSTx509CacheProxy(...) function to store a GSI proxy chain in the proxy cache along with the private key. It returns GRST_RET_OK on success, non-zero otherwise. The existing private key with the same delegation ID and user DN is appended to make a valid proxy file.

REFERENCES

[1] McNab, A. and Kaushal, S. The GridSite Security Framework, Department of Physics and Astronomy, University of Manchester, Manchester, UK (Limited circulation).

[2] Welch, V., Foster, I., Kesselman, C., Mulmo, O., Pearlman, L., Teucke, S., Gawor, J., Meder, S. and Siebenlist, F. (2004). X.509 proxy certificate for dynamic delegation, Proceeding of the 3rd Annual PKI R&D Workshop.

[3] Ahsant, M., Basney, J. and Mulmo, O. (2004). Grid Delegation Protocol, UK Workshop on Grid Security Experiences, Oxford.

[4] gSOAP: Generator Tools for Coding SOAP/XML Web Services in C and C++, <http://www.cs.fsu.edu/~engelen/soap.ht>

[5] Snelling, D. F., Van den Berghe, S. and Li, V. Q. (2004). Explicit Trust Delegation: Security for the Dynamic Grids, FUJITSU Sci. Tech. J.

[6] GridSite: Grid Security for the Web platforms for Grids, <http://www.gridsite.org/>

GACL

GACL is the main authorization policy language used by **GridSite**. GACL allows policies to be written in terms of common Grid credentials: X.509 identities, GSI proxies, VOMS attribute certificates and lists of X.509 identities.

GridSite both uses GACL policies and provides a GACL manipulation API for C/C++ in the GridSite library.

CREDENTIALS

In GridSite 1.3.x, four credential types and one level modifier are supported:

```
<person> <dn>/O=Grid/CN=Name</dn> </person>
<voms> <fqan>/vo.dom.ain/group</fqan> </voms>
<dn-list> <url>https://www.vo.dom.ain/dn-lists/group</url> </dn-list>
<dns> <hostname>host*.dom.ain</hostname> </dns>
<level> <nist-loa>0-4</nist-loa> </level>
```

In addition, `<any-user/>` and `<auth-user/>` may be used to refer to any user or to any authenticated user.

PERMISSIONS

Five permissions are supported: Admin, Write, List, Exec and Read. Admin is permission to modify the authorization policy itself, but applications can map the other permissions to local methods as appropriate to their environment. For filesystems and file servers, Write, List and Read have their usual meanings: creating or modifying files or directories; browsing directories; reading files. Exec is not used by GridSite itself, and applications are free to give it a meaning within their own contexts.

In 1.0.x, only per-directory GACL files are supported, and the file is stored in the directory in question, or in one of its parent directories. (GridSite searches upwards until it finds one.)

In GACL files, the permissions are represented by single tags: `<admin/>`, `<write/>`, `<list/>`, `<exec/>`, `<read/>`. Permission tags are contained within Allow or Deny blocks. For example: `<allow><read/><list/></allow>` or `<deny><admin/></deny>`.

ENTRIES

Entries associate credentials with permission statements. Entries consist of one or more credential blocks, and either an Allow or a Deny block, or both. If multiple credentials are present in one entry, they must all be held by a user to receive the association permissions. (So Entries provide logical AND of credentials.)

ACCESS CONTROL LISTS

ACLs consist of a list of one or more Entry blocks. When a user's credentials are compared to the ACL, the permissions given to the user by Allow blocks are recorded, along with those forbidden by Deny blocks. When all entries have been evaluated, any forbidden permissions are removed from those granted. (So Deny always wins over Allow, even between different Entries, but otherwise ACLs provide logical OR of credentials.)

GRIDFTP AND CHROOT

Using GridFTP from VDT1.2.2rh9_LCG-1. This is probably the simplest way to set up chroot - using ftp guest accounts. The GridFTP code is still running with root=/, and the chroot is only done once the user has been selected. This isn't as secure as a full chroot would be, but is a lot easier to set up (no need for Globus shared libs in the chroot etc.)

Basic script to start gridftp:

```
#!/bin/sh
export LD_LIBRARY_PATH=/opt/globus/lib
export GRIDMAPDIR=/etc/grid-security/gridmapdir
/opt/globus/sbin/in.ftpd -v -s -p 2811 -a
```

The -a option ensures that /etc/ftpaccess is consulted (it's still looked for there, in /etc, even though the rest of Globus is installed in /opt/globus)

/etc/ftpaccess needs to set all users as guest, and you can do this by copying /opt/globus/etc/ftpaccess to /etc/ftpaccess and adding these lines at the end:

```
guestgroup *
guest-root /grid/local
```

This means all users will be chroot'ed to /grid/local once they have authenticated.

GridFTP will work fine on the /grid/local filesystem provided by [SlashGrid](#), using the pool accounts directory to map UIDs back to DNs.

If this is done, only directories and symbolic links need to be created. Assuming that the SlashGrid option --local-root has value /var/www/html, then:

```
/var/www/html/etc/localtime -> /etc/localtime
/var/www/html/etc/passwd -> /etc/passwd
/var/www/html/etc/grid-security/certificates -> /etc/grid-security/certificates/
/var/www/html/etc/.gac1
```

Where the final GACL file allows any user to read the files (including root which has no DN):

```
<?xml version="1.0"?>
<gac1 version="0.0.1">
```

```
<entry>
<any-user/>
<allow><list/><read/></allow>
</entry>
</gac1>
```

SlashGrid's local filesystem hides that the /etc files are really symbolic links (which GridFTP doesn't like.)

With this done, URLs like `gsiftp://host.name:2811/file.txt` map to `/grid/local/file.txt`, which in turns maps to `/var/www/html/file.txt` - consistent with `http://host.name:777/file.txt` etc if `/var/www/html` is the Apache DocumentRoot. The rest of the / filesystem on the host isn't accessible via GridFTP.

GRIDSITEDIKI

GridSiteWiki is a version of MediaWiki - the same software used by WikiPedia, but with user authentication based on X.509 certificates rather than usernames and passwords. (It is the basis of [this Wiki](#), that you are currently reading.)

See the [GridSiteWiki](http://www.gridsite.org/gridsitewiki/) (<http://www.gridsite.org/gridsitewiki/>) page for an explanation of the software, full sources and a brief installation guide.

HTTP

HTTP has an article, [The HTTP Protocol](http://en.wikipedia.org/wiki/HTTP) (<http://en.wikipedia.org/wiki/HTTP>), in the [Wikipedia](http://en.wikipedia.org/) (<http://en.wikipedia.org/>). There are also some details of relevant extensions to HTTP (such as COPY) in the [WebDAV](http://en.wikipedia.org/wiki/WebDAV) ([http://en.wikipedia.org/wiki/](http://en.wikipedia.org/wiki/WebDAV)) article.

For the IETF specifications, see [RFC2616](http://www.w3.org/Protocols/rfc2616/rfc2616.html) (<http://www.w3.org/Protocols/rfc2616/rfc2616.html>) (HTTP) and [RFC2518](http://www.ietf.org/rfc/rfc2518.txt) (<http://www.ietf.org/rfc/rfc2518.txt>) (WebDAV HTTP extensions.)

IP Ports - recommendations for HTTP and HTTPS ports to use in environments where 80/443 are blocked to prevent denial of service attacks against mainstream websites.

SlashGrid HTTP Profile - using components of HTTP, WebDAV and HTML to achieve filesystem semantics similar to POSIX, via HTTP(S) requests.

IP PORTS

Well known and Registered IP Ports are defined by [IANA](http://www.iana.org/) (<http://www.iana.org/>) in their [Port Numbers](http://www.iana.org/assignments/port-numbers) (<http://www.iana.org/assignments/port-numbers>) document. **GridSite** uses the HTTP family of protocols, and this involves ports for [HTTP](#) and [HTTPS](#).

The vast majority of deployed webservers operate on the relevant well known ports. However, there is a reluctance at many computing centres to allow outbound access to TCP port 80 (HTTP) and TCP

port 443 (HTTPS) from farm nodes, in case these machines are compromised and then used for denial-of-service attacks on prominent websites. Farm firewalls are often configured to block outbound ports 80/443, or only allow them through a web proxy with a corresponding loss of performance.

To deal with this, we propose that GridSite servers listen on 80/443 as standard, but also on **port 777 ("multiling-http") for plain HTTP**; and on **port 488 ("gss-http") for HTTPS**. It is straightforward to configure the underlying Apache server to listen on multiple ports, using the Listen and VirtualHost directives in httpd.conf:

```
# Plain HTTP server
Listen 80
Listen 777
<VirtualHost *:80 *:777>
ServerName host.name
...
</VirtualHost>

# Secured HTTPS server
Listen 443
Listen 488
<VirtualHost *:443 *:488>
ServerName host.name
SSLEngine on
...
</VirtualHost>
```

It is then necessary to ensure that clients which are likely to be behind restrictive firewalls use the 777/488 versions of the site.

PERL CLIENTS

This page contains notes on how to build a perl client for authenticated access to a GridSite. It splits into three sections:

- How to *get* a web page using an X.509 client certificate as the authentication method
- How to *get* a web page using a GSI Proxy as the authentication method
- How to access a web service using SOAP::Lite, authenticating with your GSI Proxy (over TLS).
- Links to examples and applications of this.

PERL TO ACCESS GRIDSITE OVER HTTPS USING STANDARD X.509 CLIENT CERTIFICATES

Notes on mutual authentication in perl using X.509 certs. How to interpret **Client-SSL-Warning: Peer certificate not verified** headers if you see them and even if you don't!

- First get and install Crypt::SSLeay if you haven't already got it.

```
perl -MCPAN -e 'install Crypt::SSLeay'
```

This provides the link between the https methods in LWP and the openssl C libraries.

During installation you will be prompted for an openssl to link against. (At first I linked against the stock openssl 0.9.7a and later discovered problems)... I ended up installing and linking against version 0.9.6m.

- Get an up to date version of the LWP (libwww) perl module.

The stock version that installs on a Fedora Core 1 is old and doesn't do everything needed for mutual authentication. These early versions of the LWP module produced the header "Client-SSL-Warning: Peer certificate not verified", regardless of the it was or not. So update the LWP module:

```
perl -MCPAN -e 'install "LWP"'
```

- Write a program to authenticate using your X.509 Certificate (one like this will do):

```
#!/usr/bin/perl
```

```
use LWP::UserAgent;
```

```
$ENV{HTTPS_CA_DIR} = (defined $ENV{X509_CERT_DIR})?$ENV{X509_CERT_DIR}:"/etc/grid-security/cert
```

```
$ENV{HTTPS_CERT_FILE} = $ENV{HOME}/.globus/usercert.pem;
```

```
$ENV{HTTPS_KEY_FILE} = $ENV{HOME}/.globus/userkey.pem;
```

```
# Print SSL Debug stuff (omit this line if not debugging)
```

```
$ENV{HTTPS_DEBUG} = 1;
```

```
# Instantiate an LWP User Agent to communicate through
```

```
my $agent = LWP::UserAgent->new;
```

```
# Get a response from https://www.gridsite.org/
```

```
my $response = $agent->get( "https://www.gridsite.org/" );
```

```
# Do something with your response
```

```
if ( $response->is_success ) {
```

```
    print $response->as_string;
```

```
} else {
```

```
    print "Something went wrong\n";
```

```
}
```

- To check that it's all working connect to a secure site that has a certificate issued by a CA not in \$HTTPS_CA_DIR

PERL TO ACCESS GRIDSITE OVER HTTPS USING GSI PROXIES

- Write a new program e.g. this time, to authenticate with your GSI proxy.

```
#!/usr/bin/perl

use LWP::UserAgent;

$ENV{HTTPS_CA_DIR} = (defined $ENV{X509_CERT_DIR})?$ENV{X509_CERT_DIR}:"/etc/grid-security/certificates";

# ---- GSI Magic to make it work ----
my $GSIPROXY = (defined $ENV{X509_USER_PROXY})?$ENV{X509_USER_PROXY}:"/tmp/x509up_u${UID}";
$ENV{HTTPS_CA_FILE} = $GSIPROXY;
$ENV{HTTPS_CERT_FILE} = $GSIPROXY;
$ENV{HTTPS_KEY_FILE} = $GSIPROXY;
# ---- End of GSI Magic ----

# Print SSL Debug stuff (omit this line if not debugging)
$ENV{HTTPS_DEBUG} = 1;

# Instantiate an LWP User Agent to communicate through
my $agent = LWP::UserAgent->new;

# Get a response from https://www.gridsite.org/
my $response = $agent->get( "https://www.gridsite.org/" );

# Do something with your response
if ( $response->is_success ) {
    print $response->as_string;
} else {
    print "Something went wrong\n";
}
```

- Anything that you use/build, which is derived from your instance of LWP, will use your proxy if you set the environment variables as specified above.

HTTPS_KEY_FILE

points to the file containing the key

HTTPS_CERT_FILE

points to the file containing the certificate that matches the key

HTTPS_CA_FILE

points to a file containing a list of trusted certificates (for GSI we specify this so that Crypt::SSLLeay knows how to construct your GSI proxy's certificate chain)

HTTPS_CA_DIR

points to the directory containing all your Trusted CA Root Certificates

For GSI proxies the first three should be the same file (normally after a grid-proxy-init command located here: /tmp/x509up_u`id -u`).

SOAP::LITE TO ACCESS GRIDSITE OVER HTTPS USING GSI PROXIES

- Get SOAP::Lite if you haven't already got it.

```
perl -MCPAN -e 'install SOAP::Lite'
```

A Simple SOAP::Lite script might look a bit like this:

```
#!/usr/bin/perl

# Uncomment next line for SOAP debug info
# use SOAP::Lite +trace => debug => sub {};

use SOAP::Lite;

# ---- GSI Magic to make it work ----
my $GSIPROXY = (defined $ENV{X509_USER_PROXY})?$ENV{X509_USER_PROXY}:"/tmp/x509up_u$<";
$ENV{HTTPS_CA_DIR} = (defined $ENV{X509_CERT_DIR})?$ENV{X509_CERT_DIR}:"/etc/grid-security/certificates";
$ENV{HTTPS_CA_FILE} = $GSIPROXY;
$ENV{HTTPS_CERT_FILE} = $GSIPROXY;
$ENV{HTTPS_KEY_FILE} = $GSIPROXY;
# ---- End of GSI Magic ----

# Uncomment next line for SSL debug info
# $ENV{HTTPS_DEBUG} = 1;

# force SSLv3, If you want
$ENV{HTTPS_VERSION} = '3';

# Instantiate a SOAP User Agent to communicate through (NB this need not be HTTPS,
the WSDL will specify HTTP or HTTPS)
$service = SOAP::Lite -> service( "http://www.gridsite.org/AnotherDescription.wsdl"
);

# Call a WS Operation via the SOAP Agent
$response = $service->SomeOperation();

print $response;
```

PERL PEARLS

Assuming you've got to grips with the above stuff, you may like to visit the [Perl Pearls](http://www.kato.mvc.mcc.ac.uk/grid/perl-pearls/) (<http://www.kato.mvc.mcc.ac.uk/grid/perl-pearls/>) page on the Manchester Computing RSS GridSite.

There in you'll find such gems as:

- How to create a legacy GSI proxy without any Globus code.

- How to create a basic yet secure and authenticated Application Hosting Environment.
- How to create a web service and client that will delegate a proxy to that hosting environment.

...

SITECAST

SiteCast is a file-location technology developed for the [GridSite](#) project (and added in version 1.1.12). SiteCast clients use [HTCP](#) messages transmitted over UDP [multicast](#) to query GridSite filesystems for the presence of specific files within a shared SiteCast URL space. The UDP responder is implemented as part of the [GridSite Apache module](#), and if the file is present, the responder replies with an HTTP redirection to the transfer URL of the file on that server. If multiple servers reply, then the client must choose from which to fetch the file.

USING HTCP FOR FILE LOCATION

The [HTCP Protocol](#) was developed as a lightweight way for a group of web caches to query each other for the presence of a given URL. The main operation, TST, presents an HTTP request to the cache. The cache replies by saying whether it has the corresponding file/page, and may return the cached HTTP response headers. The HTCP messages are stateless and sufficiently short that they are naturally transmitted via UDP rather than TCP.

Within SiteCast, HTCP queries are interpreted as the question "how would you respond to this HTTP request?" This interpretation allows HTCP to be used to query file servers rather than web caches.

Since HTCP is naturally transmitted over UDP, then it is straightforward to take it a step further and multicast the queries, rather than direct them to individual servers.

GROUPS, DOMAINS AND URL-SPACES

IP multicast allows hosts to subscribe to **multicast groups**, which take the form of reserved IP numbers in the range 224.0.0.0 - 239.255.255.255. Since groups are mapped to ethernet addresses, most unwanted multicast groups can be ignored in the ethernet hardware or device driver layer, without reaching the level of the IP stack or the userspace code.

SiteCast clients can query multiple multicast groups, perhaps in order with a time delay between each one. This feature allows filesystems to be grouped according to network proximity, CPU/memory or even who owns the machine, and then higher priority given to particular groups. For example, a group could be assigned to each rack of hosts which share the same switch; clients can then query their local group to try to find a "close" source for the file, and if that fails, then query the group containing all hosts in the storage farm.

A transfer URL is needed to really fetch the file, but this will include the specific hostname which has the file. When querying all hosts, the file is expressed as a SiteCast URL, containing a site-wide domain name, scheme, port and the local path on the filesystems:

<https://sitecast.hep.man.ac.uk:488/dir/file.txt>

All of the hosts which will respond to SiteCast queries containing the generic domain name "sitecast.hep.man.ac.uk", are said to be part of the same SiteCast Domain.

To test for the presence of the file, the host maps the path part of the URL into a path in the local filesystem. If the file is found, the host replies to the client, specifying an HTTP **Location:** header which gives the absolute URL of its copy of the file:

<https://host001.hep.man.ac.uk:488/dir/file.txt>

Consequently, the SiteCast domain forms a shared, site-wide URL-space, in which all hosts in the storage farm participate.

OPERATIONAL ADVANTAGES FOR STORAGE FARMS

As well as the grouping of hosts by proximity, this architecture provides a measure of load balancing (unloaded hosts are likely to respond first), failover when hosts fail (if a host goes down, then it stops responding to multicast queries), and natural support for replication of files (multiple hosts can reply if the file is replicated.)

SUPPORT IN HTFIND AND HTCP COMMANDS

This command will copy a file located via SiteCast to the local disk:

```
htcp --domain sitecast.hep.man.ac.uk --groups 224.0.1.111 \  
  https://sitecast.hep.man.ac.uk:488/dir/file.txt /tmp/file.txt
```

See also the [htcp man page](http://www.gridsite.org/doc/HEAD/htcp.1.html) (<http://www.gridsite.org/doc/HEAD/htcp.1.html>)

SUPPORT IN SLASHGRID

[SlashGrid](#) includes support for reading files located via SiteCast, with virtual paths in the local filesystem like `/grid/https/sitecast.hep.man.ac.uk:488/dir/file.txt`

These files can be read just like any other other remote file accessible to SlashGrid via HTTP(S). The SiteCast domain(s) and groups to use must be specified on the command line of the SlashGrid daemon, and this easily can be done via the `OPTIONS` variable in the `/etc/sysconfig/slashgrid` file.

MOD_GRIDSITE DIRECTIVES

[mod_gridsite man page](#)

SUBVERSION

Subversion is a version management system similar to CVS. Unlike CVS, Subversion can use the Web-DAV protocol and an Apache module, which the `svn` command-line tools can communicate with directly. This means one less service to run if you're already running a web server, and due to the modular design of Apache, it can be used with [GridSite](#)-based authorization.

This guide describes setting up Subversion, GridSite and Apache, using the `GridSiteACLPath` directive introduced in GridSite 1.3.4.

You should follow the basic independent installation and configuration steps for the three components. For RedHat derived systems, the Subversion website provides binary RPMs for recent releases. You must include the WebDAV and Subversion modules to the Apache configuration file:

```
LoadModule dav_module      modules/mod_dav.so
```

```
LoadModule dav_svn_module  modules/mod_dav_svn.so
```

and you should check that Subversion is working with anonymous or password protected requests using the svn client.

We will assume you're only going to use Subversion over **HTTPS** rather than **HTTP**. In the HTTPS virtual server, include the `SSLUserName` directive to fill the HTTP user with the client's certificate DN. (This is used by Subversion for recording and displaying who made each commit to the repository.)

```
SSLUserName                 SSL_CLIENT_S_DN
```

For production use, you should create a directory which is writeable by your webserver Unix account (for example, apache) and which can contain multiple repositories. Each repository will be a separate authorization domain, with its own set of users able to read or write within it. It should be outside of the `DocumentRoot` (eg `/var/www/html`) exported by Apache. For example, use `/var/www/svn`

Now add a `Location` section to define the virtual URL space owned by the Subversion Apache module:

```
<Location /svn>
# make sure GridSite doesnt try to fulfill any of the DAV methods
  GridSiteMethods
# work out GACL path from REQUEST_URI
  GridSiteACLPath /var/www/html/svngacls/%2.gacl
# turn on SVN mod_dav
  DAV svn
# point to the repositories
  SVNParentPath /var/www/svn
</Location>
```

These four directives make sure that GridSite doesn't try to fulfill the PUT, MOVE etc methods used by WebDAV for these virtual URLs; tell GridSite what GACL files to use for the virtual URL space; turn on `mod_dav` in Subversion mode; tell `mod_dav_svn` where to find the repositories.

Both `SVNParentPath` and `GridSiteACLPath` work out which GACL or repository to use based on the URL given. The `%2` in `GridSiteACLPath` refers to the 2nd part of the request's URI on the server. For example, **main-project** in <https://www.gridsite.org/svn/main-project/Makefile> In this example, `%0` would refer to the 0th component, or `www.gridsite.org`

This virtualisation allows additional repositories to be added without having to modify the Apache configuration and restart.

VOMS2GACL

The python script **voms2gacl** can be used for creating `.gacl` files by a by contacting a VOMS server and creating a `.gacl` with the members of VOMS group or Role. It is especially useful for restricting access from non-voms enabled clients to `.gacl` enabled services.

The obvious use case is for adding `.gacl` files to gridsite enabled webserver that you wish to restrict page views with a web browser to members of VO.

README

voms2gacl Steve Traylen <steve.traylen@cern.ch>

This utility can be used for creating .gacl files containing a list of DNs that are contained within a voms group or role.

To use the command:

```
voms2gacl --help
```

```
voms2gacl: Converts a .voms2gacl file to a .gacl file
```

```
voms2gacl -h      Print this help
```

```
voms2gacl [-v] [ -d directory ] [ -k hostkey.pem ] [ -c hostcert.pem ]
```

```
    -v                Enable verbose mode
```

```
    -d directory      Specify a directory to process.
```

```
                        Default directory is .
```

```
    -k hostkey.pem    Location of hostkey.  Default /etc/grid-security/hostkey.pem.
```

```
    -c hostcert.pem   Location of hostkey.  Default /etc/grid-security/hostcert.pem.
```

```
    -t threshold      Threshold, if a VOMS listing contains less than threshold
```

```
users
```

```
                        the program exits.  Designed to cope with broken VOMS servers.
```

```
Default 0
```

or create a .voms2gacl file e.g /var/www/html/.voms2gacl

```
<?xml version="1.0"?>
```

```
<voms2gacl>
```

```
<entry>
```

```
    <vomsserver>vomss://voms.cern.ch:8443/voms/atlas?/atlas/Role=production</vomsserver>
```

```
    <allow><read/><write/></allow>
```

```
</entry>
```

```
<entry>
```

```
    <vomsserver>vomss://voms.cern.ch:8443/voms/dteam?/dteam</vomsserver>
```

```
    <allow><write/></allow>
```

```
</entry>
```

```
</voms2gacl>
```

This can then be processed with

```
voms2gacl -d /var/www/html
```

to create /var/www/html/.gacl file containing:

```
<?xml version="1.0"?>
```

```
<gacl version="0.0.1">
```

```
  <entry>
```

```
    <person><dn>/DC=org/DC=doegrids/OU=People/CN=Dan Schragger 614930</dn></person>
```

```
    <allow><read/><write/></allow>
```

```
  </entry>
```

```
<entry>
  <person><dn>/DC=org/DC=doegrids/OU=People/CN=LEUNG FOOK CHEONG Annabelle 571152</dn></pe
  <allow><read/><write/></allow>
</entry>
```

....

```
<entry>
  <person><dn>/O=GermanGrid/OU=DESY/CN=Christoph Wissing</dn></person>
  <allow><write/></allow>
</entry>
<entry>
  <person><dn>/C=GR/O=HellasGrid/OU=uoa.gr/CN=Aristotelis Glentis</dn></person>
  <allow><write/></allow>
</entry>
<entry>
  <person><dn>/C=CH/O=CERN/OU=GRID/CN=Pal Anderssen 4660</dn></person>
  <allow><write/></allow>
</entry>
</gac1>
```

RELEASES

voms2gacl can be downloaded from <http://hepunix.rl.ac.uk/~traylens/rpms/voms2gacl>

- Version 1.0 Initial Release
- Version 1.1 Bug fix for -t option.

CRON JOB

If you want a cron job to process your web documents converting .voms2gacl to .gacl files then a trivial

```
find /var/www/html -name .voms2gacl -exec dirname {} \; | xargs -i voms2gacl -t 10
-d {})
```

X.509

X.509 was originally an ITU standard for Public Key Infrastructure, and is the cryptographic basis of HTTPS used by GridSite. The most recent version in use, X.509 v3, is defined by the IETF in RFC3280 (<http://www.ietf.org/rfc/rfc3280.txt>).

SEE ALSO

Wikipedia X.509 article (<http://en.wikipedia.org/wiki/X.509>)

[Converting .p12 certificate files to PEM](#)

BUILD AND INSTALL GUIDE

This Guide explains how to build GridSite from source, and how to install the server components alongside an Apache 2.0 webserver. There is a separate [Config Guide](#) which explains how to modify the httpd.conf file, and how to set up other files and directories used by the system. You should look through all of this Building and Installation Guide to decide which is the easiest route for your system.

Table of contents

[showTocToggle\("show","hide"\)\[showhide\]](#)

[1 Installing with RPM](#)

[2 Requirements for building GridSite from source](#)

[3 Building GridSite with Make](#)

[4 Building GridSite with RPM](#)

INSTALLING WITH RPM

If you are installing on Linux with a binary RPM release, you can skip most of this Guide, install the binary rpm(s) and go straight to the [Config Guide](#).

RedHat 9, Fedora, RHEL, Scientific Linux: This is the simpler case, since the standard release includes a suitable version of Apache 2.0: just install the gridsite-...-1.i386.rpm to get the various GridSite components.

Earlier, eg RedHat 7.3: This is more complicated because you must also install a back-ported Apache 2.0 RPM or build it from source.

GridSite also depends on shared libraries from libcurl and libxml2, and the RPMs distributed as part of the standard RedHat, from 7.3 onwards, are sufficient.

With the RPMs installed, you can proceed to the [Config Guide](#).

REQUIREMENTS FOR BUILDING GRIDSITE FROM SOURCE

GridSite is currently only supported on Linux, but should be straightforwardly portable to other Unix platforms where the GNU build tools are available.

GridSite consists of a core library (libgridsite[.sol.a]), an Apache module (mod_gridsite.so), a CGI utility (gridsite-admin.cgi) and some command line tools (htcp, urlencode.)

All of the components use the GridSite library, and this in turn depends on libcurl and libxml2. You will need the development versions of these packages installed before you can proceed.

BUILDING GRIDSITE WITH MAKE

Our download area at <https://www.gridsite.org/download/> includes a tar-ball distribution of the sources, which can be unpacked and used to build GridSite from source. (Bleeding-edge developers can get the current snapshot of the same files from our CVS area.)

GridSite needs a copy of the Apache 2.0 include files to build, and the location of this is set by the MYCFLAGS variable in the top-level Makefile. For manual builds, the default MYCFLAGS=-I/usr/local/include/httpd is used. If you wish to use the GridSite module with Apache 2.0 installed elsewhere, you should change the MYCFLAGS variable to point to the includes directory installed by the development part of that Apache 2.0 distribution.

```
make
```

```
make install
```

will build all components and install them all under the default locations of /usr/local/[lib|bin|include|sbin] The default prefix for manual builds is /usr/local, as set by the prefix variable in the top level Makefile (/usr is the default for RPMs.)

BUILDING GRIDSITE WITH RPM

For RedHat Linux and derivatives, building with RPM is recommended. The command `make rpm` in the top level of the source tree will build the GridSite and htcp binary RPMs in the directory `../RPMTMP/RPMS/i386` relative to the working directory. An SRPM is put into `../RPMTMP/SRPMS` This build assumes the Apache 2.0 includes are in `/usr/include/httpd`.

For other configurations, you can modify the assumed location of the Apache 2.0 includes by changing the MYCFLAGS variable in the rpm target near the foot of the top level Makefile. Building Apache 2.0 If it is not possible to use binary RPMs of Apache 2.0, then it can be built from source using the `build-apache2.sh` script found in the GridSite docs directory. The script includes instructions on how to build from the tarballs distributed by the Apache Foundation. (it removes the `-C` option from "configure -C" in the `.spec` file and builds the RPMs under the current directory.)

If these targets do not work on your build platform, the Makefile and the scriptlets in the included SPEC files are a good starting point for building Apache by hand yourself. The complexities of this are outside of the scope of this Guide, but you are welcome to ask for assistance on the GridSite Discussion List, although <http://www.apache.org/> is a better starting point for purely Apache problems.

CONFIG GUIDE

This guide is intended for webmasters setting up GridSite with an Apache 2.0 webserver. We assume you have root access to the server machine to do this. There is a separate [Administration Guide](#) for people administrating areas of GridSite websites or file servers, or managing GridSite's DN List groups. That is, for people managing files on the server rather than the server itself.

INSTALLATION

We assume you have installed Apache 2.0 and GridSite, using the [Building and Installation Guide](#) where necessary. This Config Guide assumes installation has been done under /usr. For an alternative tree like /usr/local, the relative paths should be the same.

Installation should have given you an Apache 2.0 httpd binary at `/usr/sbin/httpd` and a set of standard Apache 2.0 modules in `/usr/lib/httpd/modules/` including the standard `mod_ssl` and our `mod_gridsite.so` module.

GridSite 1.1.x also includes some commands and man pages in `/usr/bin` and `/usr/share/man/man1` (`urlencode`, `htcp` and other `ht*` commands.)

CERTIFICATES

You must also install the CA root certificates of the CA's used by the users you wish to talk to. These should be installed in `/etc/grid-security/certificates` as files like `01621954.0`, and RPMs and tar files for many common European and North American CAs are available via the [EU Grid PMA](#).

This location also has VOMS server certificate RPMs which install into the `/etc/grid-security/vomsdir` directory. You may also manually install VOMS server certificates into that directory with any filename. (GridSite currently parses the certificate itself when looking for a match, rather than checking the filename.)

The server itself needs a certificate to supply to clients that use HTTPS connections. You should apply for this from your Certification Authority (for example, the UK e-Science CA) and your request must use the advertised hostname of your server (the one that appears in URLs and not, for instance, the canonical name of the host itself.) This advertised hostname should appear in the Distinguished Name of your request. (For example `/C=UK/O=eScience/OU=Manchester/L=HEP/CN=www.gridpp.ac.uk`) For compatibility with standard browsers, the `/CN=` component should not include any Globus-style service name (so not `/CN=host/www.gridpp.ac.uk`) If possible, you should also include the advertised hostname as a DNS Subject Alternative Name. Consult your CA first if you're in any doubt about how to compose your certificate request.

Once you've got your certificate, Apache uses the certificate and private key in PEM format. If you obtained your certificate and key in PKCS#12 or `.p12` format (eg by exporting from a web browser), you can convert the `.p12` file to `.pem` with the following commands:

```
openssl pkcs12 -in ck.p12 -clcerts -nokeys -out hostcert.pem
openssl pkcs12 -in ck.p12 -nodes -nocerts -out hostkey.pem
```

Copy the PEM files to `/etc/grid-security/` as `hostcert.pem` (which should be world readable) and `hostkey.pem` (which should only be readable by root):

```
chown root.root hostkey.pem hostcert.pem
chmod 400 hostkey.pem
chmod 444 hostcert.pem
```

HTTPD.CONF

`/etc/httpd/conf/httpd.conf` is the key to configuring the Apache 2.0 webserver. The directives in this file determine which files the server will publish, how they are handled, which areas are writeable and who can access them. Through `mod_gridsite.so`, the GridSite system itself is configured by directives in this file.

The easiest way to get started is to examine the example `httpd.conf` files we provide with GridSite, in the `doc` directory.

Please note: this version of GridSite is not compatible with the SHM SSL session cache - use the DBM or per-process caches instead.

HTTPD-FILESERVER.CONF

httpd-fileserver.conf is an example configuration file to use Apache/GridSite as a read/write HTTP(S) fileserver, including comments on how to get the server up and running.

HTTPD-WEBSERVER.CONF

httpd-webserver.conf is an example configuration file to use Apache/GridSite as a Web Server (that is, primarily for interactive use with a browser) including comments on how to get the server up and running.

GRIDSITE DIRECTIVES

The mod_gridsite reference lists all the GridSite httpd.conf directives.

To start serving files, make a directory /var/www/htdocs owned by nobody.nobody, including the .gac1 access control file described below, and add the following directive to the HTTPS <Directory> section:

```
GridSiteMethods GET PUT DELETE
```

If you wish to accept Globus GSI Proxies as well as full X.509 user certificates, set GridSiteGSIProxyLimit to the depth of proxy you wish to accept. (As a *_rough_* guide: 0=No Proxies; 1=Proxy on user's machine; 2=Proxy owned by running Globus job; 3=Proxy delegated by a Globus job.) GACL access control

The GACL reference explains the XML access control files used by GridSite. These allow flexible policies to be written, in terms of X.509 user certificates, GSI proxies, VOMS attribute certificates, DN List groups and DNS hostnames.

For example, to give all clients read and list permission:

```
<gac1>
<entry>
  <any-user/>
  <allow><read/><list/></allow>
</entry>
</gac1>
```

To enable writing, add DN List, Person or VOMS entries to the file. For example:

```
<gac1>
<entry>
  <any-user/>
  <allow><read/><list/></allow>
</entry>
<entry>
  <person>
  <dn>/C=UK/O=eScience/OU=Manchester/L=HEP/CN=Andrew McNab</dn>
  </person>
  <allow><write/></allow>
</entry>
</gac1>
```

The GACL file that governs a directory is stored as .gacl in that directory. If no .gacl is present, then GridSite will search the parent directories in ascending order until one is found.

EU GRID PMA

The **EU Grid PMA** (<http://www.eugridpma.org/>) is a body to establish requirements and best practices for grid identity providers to enable a common trust domain applicable to authentication of end-entities in inter-organisational access to distributed resources. As its main activity the EUGridPMA coordinates an **X.509** Public Key Infrastructure (PKI) for use with Grid authentication middleware. The EUGridPMA itself does not provide identity assertions, but instead asserts that - within the scope of its charter - the certificates issued by the Accredited Authorities meet or exceed the relevant guidelines.

GRACE PARADIGM

GRACE (GridSite - Apache - CGI - Executables) is a method of using Unix pool accounts to provide partial "sandboxing" of services, which allows remote users to deploy services in the form of scripts and native executables, and so enables third-party **service hosting**, built on top of Apache/GridSite.

JOBS AND POOL ACCOUNTS

In large production grids such as the LHC Computing Grid, there has been a focus on providing support for jobs written as scripts and native binary executables. This has partially reflected the heritage of the applications of these grids, such as High Energy Physics, with its large investment in Fortran/C/C++ analysis and simulation codes.

For this reason, effort has been put into providing native execution environments at remote sites on the grid. One of the issues this approach must deal with is the danger that careless or malicious jobs from one user will interfere with other users' files or programs, and the **pool account** system developed at Manchester HEP for EDG and adopted by LCG and EGEE has provided one solution.

SCRIPTS AND SUEXEC

A not dissimilar problem has been faced in the mainstream web world, where web server administrators have needed to host CGI executables provided by multiple users (perhaps in commercial, third-party service hosting, where no trust relation exists beyond monthly credit card payments of hosting charges.)

The Apache software provides a solution for this by allowing CGI scripts or executables to be run as different Unix users at the level of each virtual host (each apparent website.) This mechanism, named **suexec** after the wrapper program which it relies on, leads to a very powerful separation of roles using Unix account privileges:

root

server started as root, binds to privileged ports (80 and 443), access to X.509 certificate private key, creates log files

apache

processes requests, reads HTML and other files with a URL

CGI users

accessed via suexec when dynamically creating a CGI response, files protected from other CGI users via Unix filesystem permissions

This mechanism is widely used but in standard Apache is tied to fixed configuration decisions made when the Apache web server is started.

COMBINING POOL ACCOUNTS AND SUEXEC

One of our goals in the GridSite project has been to provide support for thirty party hosting of [Web Services](#) for grids, even when the service is written as a script or native executable. This requires some form of sandboxing of users, to prevent them interfering with each other's files or programs, in the same way as must be prevented for remote batch jobs.

To do this, we have combined the pool account system with the Apache suexec mechanism (renamed to [gsexec](#).) As well as providing legacy support compatible with Apache's default, this allows two new modes of operation, which can be configured appropriately for each directory on the GridSite server.

TWO NEW MODES OF OPERATION

First, a CGI web service can be executed as a Unix pool user associated with the authenticated identity of the client. That is, based on their [X.509](#) certificate or [GSI proxy](#). The lock files associated with the pool mechanism mean that the same client certificate will be associated with the same pool account on subsequent requests (until the account lease expires, and the file space associated with the account is recycled.) This allows services to maintain internal session information in the form of temporary files owned by pool users, and protected from interference by the Unix file permissions system. (It can also be used for other user-like permission systems, such as MySQL databases.)

In the second mode, a pool account is associated with the CGI web services stored in a particular directory. This means that for every remote client, the same Unix account will be used (and the CGI services are therefore responsible for maintaining separation between the sessions of different authenticated users.)

This mode is intended to support third-party services, where user A is given write access to a directory capable of hosting CGI services. Service scripts or executables can be deployed by simply uploading them using GridSite's manual or programmatic interfaces, and then the service can access requests from other users, B1, B2, Because A's service runs as a dedicated pool account, if another user C also has the ability to deploy services to their own directory, then C still cannot interfere with A's files from their distinct pool account.

Without these mechanisms, either all the services must run as the same "apache[FFFFD?]" or "nobody[FFFFD?]" Unix account, which permits conflicts between users' actions, or each user must be configured individually by the site administrator, which requires that the server is shut down, all sessions are stopped and the server is started with the new configuration.

GRACE PARADIGM

This combination of the ability to manage grid-facing access permissions through GridSite, and local file access permissions via pool accounts allows us to define a new execution model for Web Services in grid environments, which we refer to as GRACE ("GridSite - Apache - CGI - Executables.")

GRACE offers an alternative to the reliance on Java for webservices, and is especially attractive to applications which have a large investment in executable code, or have performance requirements which are not suited to current implementations of Java.

Furthermore, the ability to use standard scripting languages such as Perl, Python and even PHP to provide Web Services offers possibilities of rapid prototyping of simple services, in languages which site administrators and scientists typically use for day to day automation tasks.

GRIDHTTP

GridHTTP is a protocol implemented in **GridSite** version 1.1.11 onwards, which supports bulk data transfers via unencrypted **HTTP**, but makes use of authentication and authorization with the usual grid credentials over **HTTPS**.

PROTOCOL

To initiate a GridHTTP transfer, clients set an Upgrade: GridHTTP/1.0 header when making an HTTPS request for a file. This header notifies the server that the client would prefer to retrieve the file by HTTP rather than HTTPS, if possible. The authentication and authorization are done via HTTPS (X.509, VOMS, GACL etc deciding whether it's ok) and then the server may redirect the client to an HTTP version of the file using a standard HTTP 302 redirect response giving the HTTP URL (which can be on a different server, in the general case.) For small files, the server can choose to return the file over HTTPS as the response body. When contacting a legacy server, the Upgrade header will be silently ignored and the file will be returned via HTTPS as normal.

For redirection to plain HTTP transport, a standard HTTP Set-Cookie header is used to send the client a one-time passcode in the form of a cookie, GRIDHTTP_PASSCODE, which much be presented to obtain the file via HTTP. This one-time passcode only works for the file in question, and only works once: the current implementation stores it in a file and deletes the file when the passcode is used. (This mechanism is no worse than GridFTP for providing an unencrypted data channel: it's vulnerable to man-in-the-middle attacks or snooping to obtain a copy of the requested file, but not vulnerable to replay attacks or to other files being obtained by the attacker.)

As you can see, GridHTTP is really a profile for using the HTTP/1.1 standard, rather than a new protocol or a set of extensions: no new headers or methods are involved.

Ways of extending it to support variable TCP window sizes so it can be used for a mix of long and short distance connections (currently the TCP window size has to be set in the Apache configuration file), and support for third-party transfers using the HTTP COPY method from WebDAV are being added to the GridSite implementation.

ADVANTAGES

One big advantage of redirecting to a pure HTTP GET transfer is not just that the server and client don't have to spend CPU en/decrypting it, but that Apache can use the sendfile() system call to tell the kernel to copy it directly from the filesystem to the network socket (or you can use the Linux kernel module HTTP server, which has much the same effect.) This means the data never has to be copied through userspace (the so-called zero copy mode.)

As far as client side APIs go, any client side library which supports HTTP redirects and cookies and lets you add your own headers is sufficient (even the curl command line tool lets you do this, with the `-H` and `-c` options, without having to make any modifications to its code.)

From GridSite version 1.1.11, `htcp` supports GridHTTP redirection, by using the `--grid-http` option.

SERVER SIDE CONFIGURATION

`mod_gridsite` adds two new Apache configuration file directives to enable GridHTTP support: `GridSiteGridHTTP` and `GridSiteSessionsDir`.

GridSiteGridHTTP should be specified in `Directory` or `Location` sections of the Apache `httpd.conf` to enable GridHTTP transfers for the files governed by those sections. (ie GridHTTP can be selectively enabled at the level of individual directories if required.) For HTTPS virtual servers, setting `GridSiteGridHTTP on` will enable redirects to the file on the corresponding HTTP virtual host, when a request is made to the HTTPS server with the header `Upgrade: GridHTTP/1.0` present. For both HTTPS and HTTP virtual servers, the directive `GridSiteGridHTTP on` will also allow requests to be made with the `GRIDHTTP_PASSCODE` cookie: if the cookie value matches a valid onetime passcode created by making a request via HTTPS, then the request will be acted upon.

GridSiteSessionsDir is used to change the directory name holding the onetime passcodes (the default is `/var/www/sessions`) This directory must be writable by the Unix account which the `httpd` server runs as, and should only be readable by that account for security reasons. `GridSiteSessionsDir` may only appear once, as part of the main/default server configuration (ie not inside a virtual server.) `mod_gridsite` will create this directory at startup, but suitable permissions / ownership can be produced manually on variants of RedHat Linux with:

```
chown apache.apache /var/www/sessions
chmod 0700 /var/www/sessions
```

CLIENT SIDE EXAMPLES

To copy a remote file to local disk using `htcp`, with your X.509 credentials in the standard location (`.globus/usercert.pem` and `.globus/userkey.pem`):

```
bash: htcp --verbose --grid-http \
  https://test.hep.man.ac.uk/1234.txt /tmp/1234.txt
htcp version 1.1.11
https://test.hep.man.ac.uk/1234.txt -> /tmp/1234.txt
Add Upgrade: GridHTTP/1.0
Enter PEM pass phrase:
Received GridHTTP Auth Cookie: GRIDHTTP_PASSCODE=f269452c1c00b235ukH86g
Received Location: http://test.hep.man.ac.uk/1234.txt
... Found (302)
GridHTTP redirect to http://test.hep.man.ac.uk/1234.txt
... OK (200)
```

(Without the `--verbose` option, the command produces no output on success.)

You can achieve the same transfer using the `curl` command (and a lot more options!):

```
bash: curl -s --verbose --capath /etc/grid-security/certificates/ \
```

```
--cert $HOME/.globus/usercert.pem --key $HOME/.globus/userkey.pem \  
--location --header 'Upgrade: GridHTTP/1.0' \  
https://test.hep.man.ac.uk/1234.txt > /tmp/1234.txt  
* About to connect() to test.hep.man.ac.uk port 443  
* Connected to test.hep.man.ac.uk port 443  
Enter PEM pass phrase:  
> GET /1234.txt HTTP/1.1  
User-Agent: curl  
Host: test.hep.man.ac.uk  
Upgrade: GridHTTP/1.0  
< HTTP/1.1 302 Found  
< Date: Sat, 10 Sep 2005 20:11:09 GMT  
< Server: Apache mod_ssl OpenSSL mod_gridsite  
< Set-Cookie: GRIDHTTP_PASSCODE=f48e19e8a48e00719M2ZpV;  
expires=Sat, 10 Sep 2005 20:16:09 GMT;  
domain=test.hep.man.ac.uk; path=/1234.txt  
< Location: http://test.hep.man.ac.uk/1234.txt  
< Content-Length: 0  
< Content-Type: text/plain  
* Issue another request to this URL: 'http://test.hep.man.ac.uk/1234.txt'  
* Connected to test.hep.man.ac.uk port 80  
> GET /1234.txt HTTP/1.1  
User-Agent: curl  
Host: test.hep.man.ac.uk  
Cookie: GRIDHTTP_PASSCODE=f48e19e8a48e00719M2ZpV  
< HTTP/1.1 200 OK  
< Date: Sat, 10 Sep 2005 20:16:10 GMT  
< Server: Apache mod_ssl OpenSSL mod_gridsite  
< Content-Length: 4  
< Content-Type: text/plain
```

The [GridSite Toolbar](#) is a Firefox extension that allows the use of the GridHTTP protocol from within a web browser.

GRIDSITE TOOLBAR

The **GridSite Toolbar** is a [Mozilla Firefox](http://www.mozilla.com) (<http://www.mozilla.com>) extension designed to allow the use of various GridSite protocols from within a web browser. Currently, [GridHTTP](#) and the [Delegation protocol](#) are supported.

INSTALLATION

The extension is available [here](http://www.gridsite.org/toolbar/gridsite.xpi) (<http://www.gridsite.org/toolbar/gridsite.xpi>). After you click on the link the Firefox extension installer should take over and all that is required is to click "Install". A notification at may appear at the top of the screen with a message saying that the installation was blocked - to allow installation, click "Edit Options" and then "Allow" in the window that appears, followed by "Close". This will authorise www.gridsite.org to install the extension in to your browser. After doing this the [installer](http://www.gridsite.org/toolbar/gridsite.xpi) (<http://www.gridsite.org/toolbar/gridsite.xpi>) will run as described above. A resart of the browser is required for the install to be completed.

USING THE DELEGATION SERVICE

After installation, the toolbar will add an icon to the system tray. The Toolbar makes use of the [Service Detection](#) method to find delgation services and the icon becomes "lit" when one is found.

[Warning: Image ignored]

Upon clicking the delegation button in the status bar, a short dialogue box appears informing the user that the browser is attempting to connect to the delegation service.

The user may then be prompted for up to 2 passwords. The first is for the Firefox software security device and will only be asked once per Firefox session. This is the standard Firefox behaviour when using the user's certificate to authenticate to a secure server. The second password requested is the PEM passphrase for the `~/globus/userkey.pem` file. For security purposes, this passphrase is requested every time the GridSite toolbar is used to delegate to a service. Once the proxy request has been signed and sent back to the server, a confirmation dialogue box will appear.

GRIDHTTP USAGE

To grab files with [GridHTTP](#) simply right click the link to the file and select "Get with GridHTTP", as shown below.

[Warning: Image ignored]

HTTP DOWNGRADE

HTTP Downgrade has now been superceded by [GridHTTP](#), which uses the same concepts and much of the same code, but with a slightly different set of HTTP header and cookie names.

HTTP Downgrade is a protocol supported by [GridSite](#) which supports bulk data transfers via unencrypted [HTTP](#), but still retaining the support for authentication and authorization with the usual grid credentials over [HTTPS](#).

The protocol allows clients to set an `HTTP-Downgrade-Size:` header when making an [HTTPS](#) request for a file. This header gives the minimum size of file the client would prefer to be retrieved by [HTTP](#) rather than [HTTPS](#) if possible. The authentication and authorization are done via [HTTPS](#) (X.509, VOMS, GACL etc deciding whether its ok) and then the server may redirect the client to an [HTTP](#) version of

the file using a standard HTTP 302 redirect response giving the HTTP URL (which can be on a different server, in the general case.) For small files, the file can just be returned over HTTPS as the response.

For the redirection to HTTP response, a standard HTTP Set-Cookie header is used to send the client a one-time passcode in the form of a cookie, which much be presented to obtain the file via HTTP. This one-time passcode only works for the file in question, and only works once (the current implementation stores it in a file and deletes the file when the passcode is used.) This is no worse than GridFTP for providing an unencrypted data channel: it's vulnerable to man-in-the-middle attacks or snooping to obtain a copy of the requested file, but not vulnerable to replay attacks or to other files being obtained by the attacker.

Ways of extending it to support variable TCP window sizes so it can be used for a mix of long and short distance connections (currently the TCP window size has to be set in the Apache configuration file), and support for third-party transfers using the HTTP COPY method from WebDAV are being added to the GridSite implementation.

One big advantage of redirecting to a pure HTTP GET transfer is not just that the server and client don't have to spend CPU en/decrypting it, but that Apache can use the `sendfile()` system call to tell the kernel to copy it directly from the filesystem to the network socket (or you can use the Linux kernel module HTTP server, which has much the same effect.) This means the data never has to be copied through userspace (the so-called zero copy mode.)

As far as client side APIs go, any client side library which supports HTTP redirects and cookies and lets you add your own headers is sufficient (even the curl command line tool lets you do this, with the `-H` and `-c` options, without having to make any modifications to its code.)

PERL CLIENTS2

PERL PEARLS

Assuming you've got to grips with the stuff on the [Perl_Clients](#) page. You may like to visit the [Perl Pearls](#) (<http://www.kato.mvc.mcc.ac.uk/gridsite/GridPerlPearls.html>) page on the Manchester Computing RSS GridSite.

There in you'll find such gems as:

- How to create a legacy GSI proxy without any Globus code.
- How to create a basic yet secure and authenticated Application Hosting Environment.
- How to create a web service and client that will delegate a proxy to that hosting environment.

SLASHGRID

SlashGrid is an HTTP(S) client added to [GridSite](#) in version 1.3.0, which allows remote HTTP(S) servers to be presented to users as part of their local filesystem. The SlashGrid daemon communicates via the Linux [FUSE](#) kernel module, and uses functions from [libcurl](#) and [libgridsite](#) to access HTTP(S) web/file servers. SlashGrid will use an [X.509 Proxy Certificate](#) in HTTPS requests if the user has one. If the remote server is based on GridSite, the full access control model based on X.509 and VOMS credentials can be used.

SlashGrid is multi-threaded with proper separation of user credentials (including allowing multiple users of a machine to access remote hosts simultaneously with different credentials); features HTTP connection reuse; SSL session reuse; a full set of GET, PUT, DELETE, MOVE, mkdir, trunc read/write operations - following the [SlashGrid HTTP Profile](#); read/write of subsections of a file, using HTTP range headers; caching of data blocks read from servers; caching of directory information; and location of files within a storage farm via [SiteCast](#).

Remote server URLs appear under /grid in the filesystem (hence 'slash grid'.) The URL <https://www.example.com/dir/file.txt> would appear as /grid/https/www.example.com/dir/file.txt

A previous version of SlashGrid, based on the coda kernel module, was developed during 2002/2003 with the principal aim of providing sandboxes for running jobs. The current version is a total rewrite, and aims to provide transparent, secured access to bulk storage on neighbouring machines in storage farms.

USER GUIDE

This Guide is intended for people using [GridSite](#) websites with conventional web browsers, especially people with write access to areas of the site. There is a separate [Administration Guide](#) with additional information for people managing access control and group membership. This Guide assumes you are familiar with basic Web and HTML concepts. Towards the end we discuss how to access servers with command line tools like curl and httpc.

READING FROM HTTP AND HTTPS SERVERS

GridSite servers are usually accessible both via HTTP and via HTTPS. You can always tell which version you are using by looking at whether the URL in your browser's location window starts with "http://" or "https://" HTTPS means that the connection to the server is encrypted, that you can verify you're talking to the real server and not an imposter, and gives you the option to authenticate to the site and perhaps gain write access.

Simple browsing of the website via HTTP or HTTPS is reasonably self-explanatory. If configured, additional links may appear in the footer of each webpage with links to this help, and to switch between HTTP and HTTPS versions of the page. Pages may also have a link to the page History, showing the dates of changes to that page and names of its authors.

When looking at HTTPS pages, you may find your browser reports it cannot verify the server's certificate since it does not recognise the Certification Authority (CA) it uses. You should attempt to load the CA's root certificate into your browser to stop these warnings. (This means your browser will be able to identify any servers using fake certificates which you shouldn't trust.) How you obtain the CA Root Certificate from a trust-worthy source depends on the CA. For example, the UK e-Science CA lets you download it [from their website \(http://ca.grid-support.ac.uk/\)](http://ca.grid-support.ac.uk/) and others can be obtained from the [EU Grid PMA](#).

AUTHENTICATING

To go beyond reading pages you need to obtain a user certificate and load it into your web browser. How you do this again depends on the Certification Authority you have access to (for most Grid projects, CAs are organised on a national basis.) To use the UK e-Science CA example again, their website has links to the procedure for applying for a certificate from within a web browser.

A user certificate usually has a version of your name and affiliation as its Distinguished Name (DN) - for example, "/C=UK/O=eScience/OU=Manchester/L=HEP/CN=Andrew McNab"

Once you've obtained a user certificate in your name from your CA, you need to make sure it is loaded into the browser you normally use to browse the web. How you do this is different for different browsers and to some extent for different CAs (but if you applied for the CA through your browser, you may already have it there.)

Browsers want the certificate and private key in the PKCS#12 format, which is normally a single file with the extension ".p12". Many programs which are based on OpenSSL, such as Globus and curl, prefer the PEM (".pem") format for certificates, with separate certificate and key files ("usercert.pem" and "userkey.pem", for example.) If you only have the files in .pem format and have access to openssl, you can use its command line tools to convert PEM to PKCS#12:

```
openssl pkcs12 -in usercert.pem -inkey userkey.pem -export -out certkey.p12
```

Be very careful not to accidentally overwrite .pem or .p12 files when doing this kind of thing! In particular, if you lose your private key, you cannot retrieve it from your CA.

Once your user certificate is loaded, you should be able to see your certificate name appear when you look at an HTTPS GridSite page which has the page footers enabled - for example, the "Switch to HTTP" link present. If GridSite understands your user certificate, it displays a "You are ..." line in the footer. (However, the Apache webserver must also be set up with your CAs root certificate for this to work. The GridPP HTTPS home page is set up to recognise a good range of European and North American Grid CAs.)

AUTHORIZATION

Once users can prove their identity to the web server, it then becomes possible to give them appropriate rights depending on that identity. GridSite allows site administrators to specify these rights for individuals and groups using [GACL](#) access control files. (The [Administration Guide](#) explains how to manage these files.) GACL defines who can read files, who can list directories, who can write or create files and who can modify the GACL policy files. To get increased access to an area of a site, you need to contact the administrator for that area and give the DN of your certificate (it's not necessary to send any certificate files.)

MANAGING DIRECTORIES AND FILES

If you have list permission for the directory containing a page, you should see an extra link "Manage Directory" in the page's set of footer links, which allows you to browse the directory even if the normal index.html is present. If page histories are available, this listing view also has links to them.

The real power of GridSite becomes available if you have write access to a directory. In that case, the "Manage Directory" page has additional links to Delete or Rename pages and other files, and to Edit HTML and plain text files. An Edit link also appears in the footer links of HTML pages.

If you use the Edit function, you are presented with an HTML form containing the current filename and the full HTML or plain text of the page for you to edit. This allows you to maintain the content of the site "in place" and to see the result of your changes immediately, in context.

If you modify the filename in the form before saving, GridSite will make a new file with that name, and the old file will still be present, unmodified. (However, you cannot use this feature for creating a file in a different directory.) As you make changes, the history of the changes and your certificate DN are recorded, and available in the history page for that file.

For people with write access, the "Manage Directory" page also has options to upload a file from the computer your browser is running on, and to create files and directories. If it's enabled, you can also view the contents of WinZIP / PKZIP / .zip files, and unpack their contents into the current directory. (This feature is very useful if you have several files to upload at one time.)

HTML FORMATTING IN GRIDSITE

As well as providing access control and file management, GridSite provides some simple formatting of HTML pages by adding standard headers and footers. (If this isn't sufficient, GridSite will happily coexist with HTML preprocessor languages like SSI, PHP and JSP.)

If HTML formatting is enabled for the current directory, GridSite looks for the files gridsitehead.txt and gridsitefoot.txt in that directory, or goes up through the parent directories until they are found.

The <body> and </body> tags from the HTML file are replaced with the contents of the gridsitehead.txt and gridsitefoot.txt files, which should normally be chunks of HTML including a replacement <body> or </body> tag. If either tag is absent from the original page, then the header or footer is just added rather than being inserted in place of the tag. (One consequence of this absence is that HTML header tags like <title> can end up after a <body> tag, and can get ignored by browsers - so always include <body> ... </body> in your pages.)

This simple system is surprisingly flexible, and allows a variety of top and bottom, or sidebar navigation layouts of pages. Since the <body ...> tag is under full control of the author of the gridsitehead.txt file, backgrounds, colour schemes and style sheets can easily be specified.

For example:

Source	HTML
page.html	<title>PAGE TITLE</title>
page.html (replaced)	<body>
gridsitehead.txt	<body text=blue> Heading text <table border=1> <tr> <td>Standard sidebar</td> <td>
page.html	<p> Page content...
page.html (replaced)	</body>
gridsitefoot.txt	</td> </tr> </table> Footer text </body>

produces pages with a layout like:

Heading text	
Standard sidebar	Page content...
Footer text	

COMMAND LINE USE

GridSite adds support for the HTTP PUT and DELETE methods, and this makes it easy to create or delete files from within programs and commands without using a web browser and HTML forms. It is straightforward, although slightly awkward, to use a standard HTTPS-aware client like curl to upload files, but GridSite provides `htcp` as a more convenient client program, which is easier to use with GSI Proxies and X.509 user certificates, and has a syntax closer to the familiar `scp` command.

The following examples assume the GridSite server has GSI support and use a GSI proxy as the client certificate. For non-GSI use, just skip the `grid-proxy-init` stage, and replace the proxy filename with `$HOME/.globus/usercert.pem` and `$HOME/.globus/userkey.pem` (or wherever your PEM format certificate and key are stored.)

First generate a GSI proxy with `grid-proxy-init`. This will create a proxy file in `/tmp/x509up_uXXXXX` where `XXXXX` is your Unix UID (also given by `id -u`.) The GSI proxy contains a temporary private key and certificate signed by your long-term user certificate.

You should make sure you have a copy of the CA root certificates of the CA's used by the servers you wish to talk to. These are usually installed in `/etc/grid-security/certificates` as files like `01621954.0`, and RPMs and tar files for many common European and North American CAs are available from the [EU Grid PMA](#).

To upload a file with curl:

```
curl --cert /tmp/x509up_u`id -n` --key /tmp/x509up_u`id -n` \  
      --capath /etc/grid-security/certificates \  
      --upload-file /tmp/new.file.txt https://server/new.file.txt
```

The equivalent `htcp` command is:

```
htcp /tmp/new.file.txt https://server/new.file.txt
```

since `htcp` looks for the GSI proxy and CA certificates automatically. `htcp` can also be used to copy remote files to the local machine by reversing the arguments. For more details, see the `htcp(1)` man page.

`htcp` also has options for deleting files, and doing short or long listings, and these can also be accessed using the `htrm`, `htls` and `htll` commands (which are normally symbolic links to `htcp`.)

Directory indexes are based on parsing the index returned by the web server and by using the HTTP HEAD method to obtain the file size and modification times.

All of the `ht**` commands can accept multiple source file arguments, and this allows you to copy multiple files to or from the server. Shell wildcard expansion on the local machine is especially useful:

```
htcp /tmp/new.*.txt https://server/
```

WEBSERVICES ON GRIDSITE

Protocols based on **Web Services** provide important benefits for Grids, particularly in avoiding the tendency that proprietary binary protocols frequently become closely tied to particular implementations or languages. The language neutrality of Web Services has received less emphasis so far, since most development of Web Services for Grids has been done in the Java language. However, due to Apache's support for dynamic content created by a wide variety of languages, [GridSite/Apache](#) is able to support secure web services written in C, C++, Perl, Python and other scripting languages.

Apache supports two main classes of content: static web pages and files, and dynamic content obtained by running a [CGI](#) program and returning its output to the client. Each request is processed by a chain of modules which modify the input or output stream of data in some way.

The GridSite extensions have been implemented as an [Apache module](#). Other architectures were considered, such as filters (in which an external program is called to transform the output stream) or providing library functions which CGI programs can call to parse grid security credentials. However, these either suffer from performance penalties or complicate the CGI interface, or both.

The GridSite module makes preliminary access control decisions as with static web pages, but also exports the parsed grid security credentials and the permissions granted according to the governing GACL or XACML policy. This information is exported as environment variables, using the same mechanism as used by the CGI API itself: the request and response are communicated via the stdin and stdout of the CGI process, and out of band information, such as authentication or remote network addresses, are communicated via environment variables.

This means that all the languages suitable for writing CGI executables and scripts are immediately able to access GridSite's evaluation of X.509, GSI, VOMS, GACL and XACML credentials and applicable policies.

EXAMPLE: WEBSERVICE DELEGATION PROTOCOL

The [Delegation protocol](#) implementation included in GridSite illustrates how web services can be built using C and the [gSOAP](#) toolkit.

This protocol was agreed within the EGEE project, and both C (GridSite) and Java implementations are available as part of EGEE's gLite framework.

To perform a delegation, the client sends a Get Proxy Request message to the server, which causes the server to generate a public and private key and return an X.509 certificate signing request containing the public key. The client then signs this request using its own private key and certificate, and sends a Put Proxy message back to the server, containing the signed certificate. Together the private key which the server generated (and which has not crossed the network) and the new certificate form an RFC3820 / GSI proxy.

gSOAP provides tools for generating the [WSDL](#) description of such a Web Service from the C header files of the functions which implement it; or vice versa. With a consistent WSDL description and populated callback functions which implement the actual X.509 key and signing functions, an executable can be built which will operate as a CGI program.

This means that SOAP requests received by Apache will be fed into the stdin of the CGI program and the SOAP response will be taken from the stdout. Since all the required authentication information is made available as environment variables by GridSite, the CGI Web Service program can obtain these directly without needing to be linked to the GridSite library.

However, in the case of the delegation service, there are two levels of credential processing taking place: authentication and authorization of the client attempting the delegation, and then generation of the new

credentials themselves. For this reason, in the special case of the GridSite delegation service, the executable **is** linked to the GridSite library, to obtain access to its private key and certificate handling functions.

The delegation service operates as a service with a single portType. However, since the bulk of the code necessary for delegation is part of the GridSite library, it is straightforward to add a delegation portType to other services which require delegation to function. If the standalone delegation service is used, then a mechanism is needed to share the credentials with other services. For CGI-based services running as native code or scripts, it is natural that the delegated credentials are stored in the local filesystem - where they are identified by a Delegation Session ID specified during the delegation process.

X.509 FILE LOCATIONS

In Grid environments, [X.509](#) certificates are usually stored in PEM format in files, in the following standard locations.

X.509 USER CERTIFICATES

An End Entity Certificate issued by a [Certification Authority](#) is stored in `$HOME/.globus/usercert.pem` with the corresponding private key in `$HOME/.globus/userkey.pem` (which should have Unix permissions 0400.)

This location can be overridden by the environment variables `X509_USER_CERT` and `X509_USER_KEY`

X.509 PROXY CERTIFICATES

`grid-proxy-init` and `voms-proxy-init` create a proxy directly from the user's EEC, in the file `/tmp/x509up_uUID` where UID is the user's Unix userid on that machine. Some applications (including GridSite's [htcp](#) etc) look for proxies in a file of that name.

The environment variable `X509_USER_PROXY` may be used to override this location.

CERTIFICATION AUTHORITIES

Self-signed CA X.509 root certificates are usually stored in `/etc/grid-security/certificates`, and should have filenames corresponding to the hash of their contents. The command `c_rehash` supplied with OpenSSL can be used to generate these, but most CAs distribute their CA root certificates with appropriate filenames already.

HOST CERTIFICATES

We recommend that host certificates and keys are stored in `/etc/grid-security` with filenames `domain.name.cert.pem` and `domain.name.key.pem`, both owned root.root and with permissions 0444 and 0400 respectively.

SEE ALSO

[Converting .p12 certificate files to PEM](#)