Certificate Management in AG 2.0

Robert Olson

March 5, 2003

Introduction

The Access Grid 2.0 software suite utilizes the Globus Toolkit mechanisms for authentication and user identification. These mechanisms are based on a public key infrastructure, and hence the use of X.509 identity certificates. While users of supercomputers and grid-based computing systems may be willing to pay the price of inconvenience in requesting, installing, renewing, and otherwise managing their certificates, casual users of the Access Grid are more likely to find these requirements baffling and annoying.

This document attempts to define the requirements induced on the toolkit by the use of Globus and its public key infrastructure and to chart a path for solutions that lead to greater convenience for users of the Access Grid.

Managing Certificates: The User's Perspective

Ideally, a user should have no idea that he has an identity certificate. He should be able to start the software, and it works properly. In reality, of course, the process is more complex. To understand the problem more completely let us consider the classes of user that we wish to enable to use the Access Grid software, and what the implications with respect to authentication, identification, and certificate management are.

The first user type we consider is the *hit and run* user. This is a person who is trying out the software for the first time, or who doesn't know or care about the details of certificate-based authentication.

A hit and run must be able to use the AG software to connect to a public venue server and see its full functionality in action. However, because he would not have a verifiable identity presented to the venue server, he would not be able to participate in any closed sessions or access any protected data or other resources.

Next, we consider a *basic* user. This is a user who is working at or in collaboration with an institution that uses the Access Grid on a regular basis, and which expects that its collaborators using the Access Grid have identities that are verifiable through the AG security mechanisms.

This user should only be required to do the minimum of work necessary to acquire the credentials required to participate at this level. This implies that the AG software should be the primary interface he uses to view any credentials he may have, to determine from what provider he should request any credentials he does not have, and to perform the actual credential request and installation of credentials when they become available.

Finally, we consider an *advanced* user. This user is familiar with public key certificates (perhaps he is a supercomputing Grid user already, or has credentials that have been created for him by his home institution for use in other applications) and has experience in other applications in their use and manipulation.

We now discuss how the certificate management framework needs to support each of these user types.

Hit and run users

A hit and run user will have no knowledge of certificates or indeed of any authentication mechanisms. If we assume that an AG venue server requires the user to have a valid certificate, one must be provided to this user.

We envision two possible mechanisms for this. The first, and simplest, is that an identity certificate is shipped with the AG software and the user certificate repository is initialized with this certificate to be used as the default certificate.

This certificate would have a Common Name that makes it clear that it is a “known anonymous” certificate.

A second, more involved solution, would involve the dynamic creation of the user’s proxy with an online CA that does not require prior registration. We will discuss this option later.

Basic users

A basic user will be required to request an identity certificate for his use with the Access Grid. [Words about the system needing to know what CA to request the cert from, where the information needs to be sent, etc].

Advanced users

[Words about importing existing certs into the user’s cert store]

[Words about trusted CA certs, and users’ importing them into their trusted CA store]

Basic background

The Globus Toolkit model for certificate management includes the following components. We use $HOME to denote the user’s home directory.

- A user’s identity certificate, an X.509 certificate (see 0 for an example of a Globus user certificate).
- The private key for the identity certificate. The private key is protected by a passphrase which must be entered each time the key is used.
- The certificates for the Certificate Authorities (CA) that are trusted by this installation of Globus (see Appendix A for an example of a CA certificate).
- The user’s proxy certificate. This is a certificate whose private key is not protected by a passphrase, and hence can be used without user intervention. It is created from the user’s identity certificate. (See Appendix B for an example of a user proxy).

These certificates and keys are stored in the following locations by default, on the stock Unix-based Globus distribution:

User’s identity certificate	`$HOME/.globus/usercert.pem`
User’s private key	`$HOME/.globus/userkey.pem`

CA certificates	`$HOME/.globus/certificates/<keyname>` where `<keyname>` is derived from the OpenSSL hash of the certificate
	`/etc/grid-security/certificates/<keyname>`
User’s proxy certificate	`/tmp/x509up_<uid>` where `<uid>` is the Unix user id of the owner of the proxy

Alternative locations can be defined by manipulating the process’ environment variables: [1]

User’s identity certificate	`X509_USER_CERT`
User’s private key	`X509_USER_KEY`
CA certificates	`X509_CERT_FILE` Stores one or more trusted CA certificates
	`X509_CERT_DIR` Directory containing trusted CA certificates.
User’s proxy certificate	`X509_USER_PROXY` File containing the user’s proxy certificate. If set, this variable will override X509_USER_CERT.
Proxy override	`X509_RUN_AS_SERVER` If this variable is set, and a proxy certificate is not explicitly set, the system will not look for a user proxy in the default location. This allows a user to run a service under an explicitly-defined service certificate while still having a user certificate present.

The Windows version of the Globus Toolkit honors the environment variables discussed above. In addition, it looks in the Windows registry for overrides of the default settings. It will also use the value of the HOME environment variable to search for default settings. If HOME is not set, it will use the directory C:\WINDOWS instead.

The following registry keys will be searched for in the HKEY_CURRENT_USER hive, in the directory software\Globus\GSI.

User’s identity certificate	`x509_user_cert`
User’s private key	`x509_user_key`
CA certificates	`x509_cert_file`
	`x509_cert_dir`
User’s proxy certificate	`x509_user_proxy`

It is clear that while the Globus-defined mechanisms are quite flexible, they can be quite confusing to the user in general.

Certificate Management Requirements

Let us discuss the requirements that the AG software has for the management of certificates.

1. To participate in an AG session, a user must have an identity certificate of some sort. We will discuss the options for this later.
2. If the certificate has a passphrase, the user must only be required to enter this passphrase once at the beginning of a session.
3. A user must be able to use multiple certificates if he chooses; however, the system may require that only one certificate may be used in any particular invocation of the client software.
4. The installation of the AG software will ship with a default set of trusted CA certificates.
5. This set of certificates must be modifiable by the user. A user can have a customized set of CA certificates that his sessions will trust.
6. Administrators of a site can install new trusted CA certificates, which will be inhered by users if they desire.
7. Services execute with their own identity certificate, separate from that of any user.
8. A service installation also has its own set of trusted CA certificates.
9. These certificates are modifiable by the “Administrator” for the service installation.

Due especially to requirements (3) and (5), we propose that the AG software define its own repository of certificates for both users’ identity certificates and for trusted CA certificates.

Resolution of requirements (4) and (6) requires that the AG client software be aware of a site-wide repository of trusted CA certificates, and handle the import of those certificates into a user's environment properly. The exact definition of proper import may not be obvious, and will hence need to be specified clearly in the definition of this process.

Requirements (7), (8), and (9) imply that a service, while not being directly attached to a GUI and driven by a user, has similar requirements for the management of identity and trusted CA certificates. However, services typically do not use proxy certificates; rather, their identity certificates are created with private keys that are not protected with passphrases.

Recommendations

Certificate Repository

We define a *certificate repository* as a collection of X509 certificates that are related in some manner. These may be a user's set of identity certificates, a set of trusted CA certificates, etc.

A repository is a directory in the local computer's filesystem. [1] Each certificate is stored in a file whose name is based on a hash of the certificate. If a certificate has an associated private key, that key can either be stored in the file with the certificate or in a separate file, whose name is based on the certificate's filename.

User Certificates

Each user has certificate repository for his personal identity certificates. The repository directory is located in the user's Access Grid per-user configuration directory ($HOME/.AccessGrid on Unix systems, \Documents and Settings\username\AccessGrid on Windows systems).

Service Certificates

Service certificates are stored in a per-system, per-service certificate repository.

Trusted CA Certificates

We envision two categories of trusted CA certificate repository. A system will have a centralized store of trusted certificates available to each user of the system, and to the services running on the system. Each user can also maintain a customized store of trusted certificates, so that he may manipulate the set of trusted CAs that are appropriate to his use.

Due to the way that Globus authentication is implemented, a process may only specify a single directory for lookups of trusted CA certificates. Thus, an

[1] I considered defining another layer of abstraction here, in the event that a repository could be kept in a database or HTTP server, etc. However, as neither possibility is likely in the near term having the additional abstraction is not useful at the moment.

implementation that allows a per-user trusted CA certificate repository must provide a means for updating the per-user trusted CA repository from the central repository.

API definition

We define the application interface to the certificate management mechanism using an object-oriented interface. We define the following classes:

- A *CertificateRepositor*y represents a certificate repository as described above. Operations include adding and removing certificates from the repository, querying the repository for current contents, and for choosing certificates appropriate for various purposes.
- A *Certificate* represents a single X509 certificate. It provides methods for loading the certificate data from a file, writing to a file, and querying for the information kept in a certificate.

CertificateRepository Methods

CertificateRepository(*directory*)

Initialize a CertificateRepository instance. *directory* is the directory holding the repository itself.

Certificate Methods

Certificate()

Initialize a Certificate instance.

LoadFromFile(file)

Load the certificate instance given the PEM-formatted certificate in *file*.

LoadFromFileHandle(fp)

Load the certificate instance given an open file handle in *fp*.

Alternatives to Locally Stored Certificates

[Discuss proxying of standard certificates using MyProxy. [2, 3]]

[Discuss self-contained online CA with username/password based registration]

Globus identity certificate

```
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number: 13230 (0x33ae)
        Signature Algorithm: md5WithRSAEncryption
        Issuer: C=US, O=Globus, CN=Globus Certification Authority
        Validity
            Not Before: Jan 10 21:45:06 2003 GMT
            Not After : Jan 10 21:45:06 2004 GMT
        Subject: O=Grid, O=Globus, OU=mcs.anl.gov, CN=Bob Olson
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
            RSA Public Key: (1024 bit)
                Modulus (1024 bit):
                    00:e8:17:f4:b9:c7:4b:5c:01:09:95:9f:29:80:c1:
2a:b0:d9:01:61:3d:0e:ce:0e:02:0d:eb:09:f8:0d:
ba:f3:ef:f0:60:24:42:62:9d:88:c0:1c:17:26:98:
95:c1:23:c4:9e:1f:0e:36:ea:88:db:3d:2f:d9:fc:
8b:5b:5c:1c:cc:4f:7b:50:26:79:20:dc:80:ac:c1:
c0:cf:9f:8a:5a:19:53:60:fa:51:c8:fc:1d:cd:7f:
d6:76:c0:3a:3a:b9:f2:f3:29:d4:26:e0:d1:b2:54:
d5:9c:71:ed:e9:9e:9e:33:c9:b0:9b:79:b6:99:8a:
17:28:2e:cd:9d:ab:f4:fe:ff
                Exponent: 65537 (0x10001)
        X509v3 extensions:
            Netscape Cert Type:
                SSL Client, SSL Server
    Signature Algorithm: md5WithRSAEncryption
        4f:48:12:25:f7:77:fa:a9:fb:0f:9e:28:8e:6a:96:1d:6e:10:
24:40:15:47:01:88:3e:1e:f5:72:67:3b:b3:2e:10:4d:39:26:
4e:7a:4e:f8:2f:cf:18:f5:14:3a:d5:e5:5b:b1:da:b8:c7:6b:
b6:ff:20:04:49:32:16:4c:7b:2d:12:30:e8:f6:fd:b1:06:c3:
b2:28:4d:fb:a1:10:f0:7d:f6:11:e4:b7:02:d6:77:7d:68:70:
63:40:d2:a1:60:f1:d0:2c:8f:82:28:f3:ee:a7:82:9f:d6:a7:
0a:56:97:57:a6:0c:bc:5c:3f:f0:e9:3f:3b:20:3d:58:49:e2:        ac:cf
```

Appendix A. Globus CA public key certificate

```
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number: 0 (0x0)
        Signature Algorithm: md5WithRSAEncryption
        Issuer: C=US, O=Globus, CN=Globus Certification Authority
        Validity
            Not Before: Jan 23 19:20:24 1998 GMT
            Not After : Jan 23 19:20:24 2004 GMT
        Subject: C=US, O=Globus, CN=Globus Certification Authority
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
            RSA Public Key: (1024 bit)
                Modulus (1024 bit):
00:f6:9b:7a:73:64:c6:07:6e:35:c1:10:82:92:f6:
db:aa:a8:92:c5:c0:87:0f:c7:95:eb:37:67:1d:af:
bd:aa:4f:fe:1b:32:b0:4e:52:17:02:ae:5e:68:0c:
47:1c:d5:37:36:67:ef:24:f2:45:c9:b5:e1:eb:b7:
d1:8a:a3:06:c8:36:6a:34:1f:04:15:5c:30:71:28:
31:fa:b9:57:3f:3e:84:06:10:76:d4:b9:93:2f:dc:
82:17:5c:e6:c1:13:5a:6b:69:ca:93:07:47:43:e5:
81:1a:e9:5a:b1:8a:6c:71:45:c0:e2:c8:ed:0c:0d:
94:6c:62:0f:71:53:3a:ef:65
                Exponent: 65537 (0x10001)
        X509v3 extensions:
            X509v3 Basic Constraints:
                CA:TRUE
            Netscape Cert Type:
                SSL CA, S/MIME CA, Object Signing CA
    Signature Algorithm: md5WithRSAEncryption
        5c:b4:79:3e:dd:52:01:39:81:b4:21:a4:ac:18:d3:5d:5e:0e:
54:b5:6d:d4:fd:78:00:d1:1b:89:23:3b:90:7d:67:5d:9d:50:
d5:73:06:df:c7:f2:4f:2e:4b:73:1c:4a:f0:a2:a4:4c:ae:f3:
92:d1:c4:47:a8:b6:46:0b:01:f2:56:33:6b:55:a3:73:f7:ce:
fd:a5:46:4d:97:cb:59:66:ab:8b:54:5e:d8:b6:3d:23:37:b1:
52:31:51:8f:42:7f:96:dd:58:f8:78:b5:8e:74:bb:18:47:ee:
58:ce:81:96:36:2e:8e:f1:f1:7d:58:89:c3:47:d5:da:ff:24:
09:2c
```

Appendix B. A proxy certificate

```
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number: 13230 (0x33ae)
        Signature Algorithm: md5WithRSAEncryption
        Issuer: O=Grid, O=Globus, OU=mcs.anl.gov, CN=Bob Olson
        Validity
            Not Before: Mar  4 16:47:38 2003 GMT
            Not After : Mar  5 04:52:38 2003 GMT
        Subject: O=Grid, O=Globus, OU=mcs.anl.gov, CN=Bob Olson,
CN=proxy
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
            RSA Public Key: (512 bit)
                Modulus (512 bit):
[deleted]
                Exponent: 65537 (0x10001)
    Signature Algorithm: md5WithRSAEncryption
[deleted]
```

Appendix C. Detailed description of proxy determination

This is taken from the Globus source code:

```
Function: proxy_get_filenames()
Description:
    Gets the filenames for the various files used
    to store the cert, key, cert_dir and proxy.


    Environment variables to use:
        X509_CERT_DIR   Directory of trusted certificates
                        File names are hash values, see the SSLeay
                        c_hash script.
        X509_CERT_FILE  File of trusted certifiates
        X509_USER_PROXY File with a proxy certificate, key, and
                        additional certificates to makeup a chain
                        of certificates used to sign the proxy.
        X509_USER_CERT  User long term certificate.
        X509_USER_KEY   private key for the long term certificate.

    All of these are assumed to be in PEM form. If there is a
    X509_USER_PROXY, it will be searched first for the cert and key.
    If not defined, but a file /tmp/x509up_u<uid> is
    present, it will be used, otherwise the X509_USER_CERT
    and X509_USER_KEY will be used to find the certificate
    and key. If X509_USER_KEY is not defined, it will be assumed
    that the key is is the same file as the certificate.

    If windows, look in the registry HKEY_CURRENT_USER for the
    GSI_REGISTRY_DIR, then look for the x509_user_cert, etc.

    Then try $HOME/.globus/usercert.pem
    and $HOME/.globus/userkey.pem
        Unless it is being run as root, then look for
        /etc/grid-security/hostcert.pem and /etc/grid-security/hostkey.pem

    X509_CERT_DIR and X509_CERT_FILE can point to world readable
    shared director and file. One of these must be present.
    if not use $HOME/.globus/certificates
        or /etc/grid-security/certificates
        or $GLOBUS_LOCATION/share/certificates

    The file with the key must be owned by the user,
    and readable only by the user. This could be the X509_USER_PROXY,
    X509_USER_CERT or the X509_USER_KEY

    X509_USER_PROXY_FILE is used to generate the default
    proxy file name.

    In other words:

    proxy_get_filenames() is used by grid-proxy-init, wgpi, grid-proxy-info and
    Indirectly by gss_acquire_creds. For grid-proxy-init and wgpi, the proxy_in
    is 0, for acquire_creds its 1. This is used to signal how the proxy file is
    to be used, 1 for input 0 for output.

    The logic for output is to use the provided input parameter, registry,
    environment, or default name for the proxy. Wgpi calls this multiple times
    as the options window is updated. The file will be created if needed.

    The logic for input is to use the provided input parameter, registry,
    environment variable. But only use the default file if it exists, is owned
    by the user, and has something in it. But not when run as root.

    Then on input if there is a proxy, the user_cert and user_key are set to
    use the proxy.

    Smart card support using PKCS#11 is controled by the USE_PKCS11 flag.

    If the filename for the user key starts with SC: then it is assumed to be
    of the form SC:card:label where card is the name of a smart card, and label
    is the label of the key on the card. The card must be using Cryptoki
    (PKCS#11) This code has been developed using the DataKey implementation
    under Windows 95.

    This will allow the cert to have the same form, with the same label as
    well in the future.
```

Appendix D. Evaluation of Python OpenSSL Bindings

At the time of this writing (where Python 2.2.2 is the current stable release, and 2.3a2 is the current alpha release), there is not a standard OpenSSL binding present in the Python standard library. The work described in this document requires the ability of Python code to peer into an X509 certificate to extract information for display to the user and for analysis of correctness, expiry information, etc.

Thus we turn to the open source community. There are currently three OpenSSL bindings available: m2Crypto [4], pyOpenSSL [5], and POW (Python OpenSSL Wrappers) [6]. Each of these bindings provides the basic functionality required in this document; namely, the reading of X509 certificates and the examination of the information in them. Each also provides additional functionality such as providing encryption, secure socket connectivity, creation and signing of certificates and certificate requests, etc. This additional functionality may become useful in the future as well.

Each package is linked against the OpenSSL libraries, and has a body of C code to link these libraries to Python objects. The m2Crypto library uses a SWIG wrapper for this interface, and hence requires SWIG to build. I have successfully built all three packages using a statically-linked build of OpenSSL 0.9.6i on a Windows XP computer using Visual Studio 6.0. Building with Visual Studio.NET led to problems with undefined symbols that were not present in the application code and were undocumented in the Microsoft documentation.

The choice between these packages would be largely arbitrary, but for one distinguishing characteristic. The class that m2Crypto and pyOpenSSL use for the management of X509 names (like the issuer and subject names in certificates) do not appear to properly handle distinguished names that include multiple occurrences of the same component. That is, Globus identity certificates look like this:

/O=Grid/O=Globus/OU=mcs.anl.gov/CN=Bob Olson

Note that there are two organizational elements. Only POW provided accessor methods on its X509 Name object that made the multiple elements available.

It appears that none of these modules provide an interface for returning the hash value of an X509 name. However, this is a straightforward modification to be made, and such modifications can be fed back to the maintainers of the library used.

The pyCrypto module [7] includes related cryptographic technology, but does not support the handling of X509 certificates.

Appendix E.   Patch for POW-0.7 to support subject hashing

```
--- POW.c~  2002-09-18 04:54:28.000000000 -0500
+++ POW.c   2003-03-07 11:18:32.000000000 -0600
@@ -1091,6 +1091,27 @@
    return NULL;
 }

+static PyObject *
+X509_object_get_subject_hash(x509_object *self, PyObject *args)
+{
+   PyObject *result_list = NULL;
+   X509_NAME *name = NULL;
+   char hstr[16];
+   unsigned long h;
+
+   if (!PyArg_ParseTuple(args, ""))
+       goto error;
+
+   h = X509_subject_name_hash(self->x509);
+   snprintf(hstr, sizeof(hstr), "%x", h);
+   printf("hash to '%s'\n", hstr);
+   return PyString_FromString(hstr);
+
+error:
+
+   return NULL;
+}
+
 static char X509_object_set_subject__doc__[] =
 "<method>\n"
 "   <header>\n"
@@ -1596,6 +1617,7 @@
    {"getIssuer",     (PyCFunction)X509_object_get_issuer,      METH_VARARGS,
NULL},
    {"setIssuer",     (PyCFunction)X509_object_set_issuer,      METH_VARARGS,
NULL},
    {"getSubject",    (PyCFunction)X509_object_get_subject,     METH_VARARGS,
NULL},
+   {"getSubjectHash",    (PyCFunction)X509_object_get_subject_hash,
METH_VARARGS,  NULL},
    {"setSubject",    (PyCFunction)X509_object_set_subject,     METH_VARARGS,
NULL},
    {"getNotBefore",  (PyCFunction)X509_object_get_not_before,  METH_VARARGS,
NULL},
    {"getNotAfter",   (PyCFunction)X509_object_get_not_after,   METH_VARARGS,
NULL},
```

Appendix F. Patch for pyOpenSSL to support subject hashing

```
--- src\crypto\x509name.c~    2002-07-09 08:54:52.000000000 -0500
+++ src\crypto\x509name.c     2003-03-07 11:05:36.000000000 -0600
@@ -147,6 +147,16 @@
 {
     int nid;

+    if (strcmp(name, "hash") == 0)
+    {
+     char hstr[16];
+     unsigned long h = X509_NAME_hash(self->x509_name);
+     snprintf(hstr, sizeof(hstr), "%x", h);
+     printf("hash to '%s'\n", hstr);
+     return PyString_FromString(hstr);
+
+    }
+
     if ((nid = OBJ_txt2nid(name)) == NID_undef)
     {
         PyErr_SetString(PyExc_AttributeError, "No such attribute");
```

Bibliography

1. *Globus Toolkit Security: Environment Variables*.http://www.globus.org/security/environment.html
2. *MyProxy project website.*
3. Novotny, J., S. Tuecke, and V. Welch. *Tenth International Symposium on High Performance Distributed Computing (HPDC-10)*. 2001: IEEE Press.
4. Siong, N.P., *M2Crypto*.http://www.post1.com/home/ngps/m2/
5. Sjögren, M., *pyOpenSSL*.http://sourceforge.net/projects/pyopenssl
6. Shannon, P., *Python OpenSSL Wrappers*.http://sourceforge.net/projects/pow
7. Kuchling, A.M., *Python Cryptography Toolkit*.http://www.amk.ca/python/code/crypto.html