

Adapting Globus and Kerberos for a Secure ASCI Grid

Patrick C. Moore
pcmoore@sandia.gov

Wilbur R. Johnson
wrjohns@sandia.gov

Richard J. Detry
rdetry@sandia.gov

Author's address:

Sandia National Laboratories
P.O. Box 5800 MS 1137
Albuquerque, NM 87185-1137

ABSTRACT

Porting a complex secure application from one security infrastructure to another is often difficult or impractical. Grid security associated with the Globus toolkit is supported by a Grid Security Infrastructure (GSI) based on a Public Key Infrastructure where users authenticate to the grid using X509 certificates. Kerberos security is based on a trusted third party, secret key infrastructure where users authenticate using encrypted tickets. However, both GSI and Kerberos provide a Generic Security Services Application Program Interface (GSSAPI) for source code portability. We describe the porting of our Globus system from GSI security to Kerberos V5 security, and the Kerberos modifications necessary to achieve that portability. Our case study provides details and insights that will be of value to developers and designers interested in GSSAPI portability. We conclude, based on our results, that designers of network security software should strive to accommodate the GSSAPI.

Keywords

ASCI, Globus, Kerberos, GSSAPI, grid, security.

1. INTRODUCTION

The Accelerated Strategic Computing Initiative (ASCI) is a U.S. Department of Energy (DOE) sponsored, multi-laboratory program dedicated to pushing the envelope in providing computing power at Sandia, Los Alamos, and Lawrence Livermore National Laboratories. The ASCI network architecture provides a production heterogeneous distributed computing environment capable of secure operation over the internet to the three labs, manufacturing plants, and collaborating sites at

Universities. Until recently, this environment lacked services to support complex integrated job workflows with user friendly interfaces. This was addressed by a subproject team of ASCI, the Distributed Resource Management (DRM) team, which was tasked to provide secure technology that would promote the ASCI network to the status of highly functional computational grid. [1] [9]

The DRM team found that much of the available grid technologies provided insufficient security, and those that did provide for strong network authentication and data protection did so based on Public Key Infrastructures (PKI). The ASCI environment had available a cross-certified PKI for digitally signed and encrypted data, but this infrastructure was not deployed or accredited as a user authentication infrastructure. For network authentication, the ASCI solution was (and still is) Kerberos Version 5.[9][11][17]

Fortunately, a principal grid technology favored by the DRM team was the Globus Toolkit [8], which is built upon a security abstraction layer called the Generic Security Service Application Program Interface (GSSAPI). [13] The GSSAPI provides some degree of source code portability across security protocols and mechanisms. Kerberos V5 implementations in use at ASCI support GSSAPI. Furthermore, the Globus developers anticipated from the beginning that portability to Kerberos would be important and were careful to design their system to accommodate this portability.

This paper describes work by the ASCI DRM team assisted by Globus developers to provide a Kerberos secured Globus for the ASCI Grid.

2. OVERVIEWS

2.1 The ASCI Security Environment

The ASCI participating laboratories spent over three years selecting, testing, and accrediting their inter-site security infrastructure. This infrastructure uses Kerberos for user authentication. Although ASCI does have an inter-site cross-certified Public Key Infrastructure (PKI) for supporting digitally signed objects and persistent encryption, PKI is not deployed or accredited as a network authentication infrastructure for ASCI.

© 2001 Association for Computing Machinery. ACM acknowledges that this contribution was authored or co-authored by a contractor or affiliate of the U.S. Government. As such, the Government retains a nonexclusive, royalty-free right to publish or reproduce this article, or to allow others to do so, for Government purposes only.

DCE cells have been implemented at each of the ASCI sites and are accredited by DOE for cross-cell authentication. Thus a user may login to their home cell, acquire Kerberos/DCE credentials, and use those credentials to login to resources and access data at the other sites without an additional login. Centrally controlled user account mappings are in place to assure that users at different sites always execute in the ASCI environment with a locally unique UNIX user ID.

All sites participating in ASCI adhere to formal inter-site security assurance agreements which spell out how the site securely manages its cell and the accounts, keys, and passwords of users and entities in that cell. The DCE/Kerberos systems support such things as machine-generated passwords, password expiration, and, for service accounts, periodic key randomization.

A number of Kerberos and DCE enabled applications are available in the ASCI environment:

- Secure Shell (SSH) with Kerberos 5 authentication for:
 - Secure interactive login (including the tunneling of X-windows back to the client),
 - Secure remote execution (essentially UNIX rsh over an SSH tunnel.)
 - Secure file transfer (essentially UNIX cp over an SSH tunnel.)
- Kerberized GSSAPI FTP (GSSFTP) and parallel GSSFTP for secure file transfers,
- DFS, a secure global distributed file system,
- HPSS, for secure persistent storage of very large files,
- Kerberos/DCE enabled Web Servers for secure web services. (Currently these use SSL to authenticate the server and

- encrypt the session, and Basic HTTP Authentication with a Kerberos/DCE password. The password is converted to a Kerberos/DCE credential at the web server.)
- An object oriented framework, the Generalized Security Framework (GSF), for adding Kerberos/DCE security to applications constructed with technologies such as CORBA IOP, Java RMI, and sockets.[6]
- A Distributed Resource Management (DRM) system for supporting a Kerberos/GSF secured ASCI Grid, [1][2]
- Kerberos/GSSAPI/SASL secured LDAP information servers and clients,
- Evolving applications based directly or indirectly on Kerberos or DCE security libraries.

The DCE/Kerberos ASCI security infrastructure has proved to be a robust, scaleable, adaptable, and manageable system for securely providing (and denying) inter-site access.

Figure 1 summarizes the software layers provided by this infrastructure and shows how these layers are integrated into the Globus and DRM software that support grid security.

In the illustration, each layer uses security features provided by layers below. At the higher layers, security is provided in a more transparent fashion than at the lower layers. For example, an application that uses Internet Inter-Orb Protocol (IOP), Java Remote Method Invocation (RMI), or socket class extensions provided by GSF can get authenticated and protected communications with very simple modifications to the C++ or Java code.

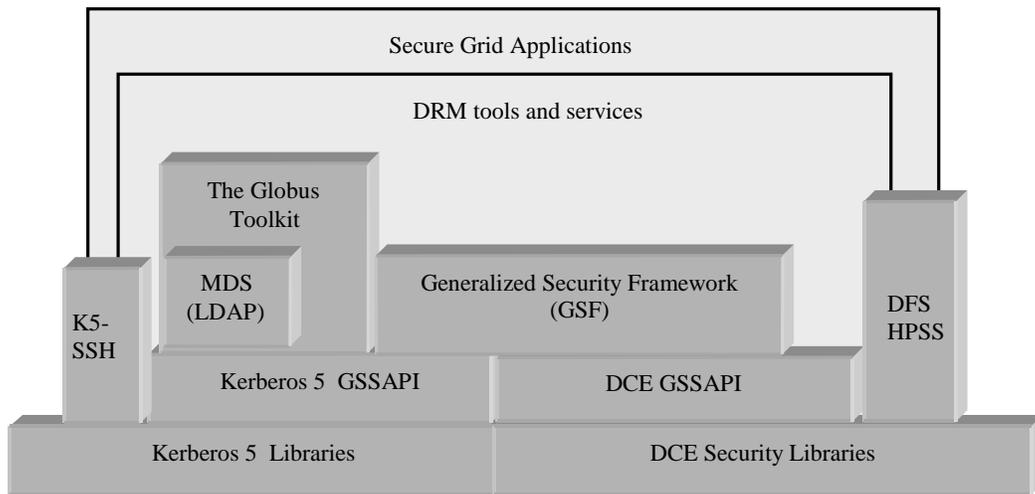


Figure 1: ASCI/DRM Security Software Layers.

2.2 The Generic Security Service Application Programming Interface (GSSAPI)

The common denominator between the security system underlying Globus and the Kerberos infrastructure used to secure ASCI is the GSSAPI layer. GSSAPI is an IETF standard programming and message protocol abstraction, which specifies how two parties can acquire security credentials and use those credentials to establish a shared security context over untrusted communication channels. In addition to the high level specification [13], there is also an IETF RFC [22] that standardizes the C language bindings of the GSSAPI – essentially providing stable function prototypes that should be used by a standard GSSAPI application. Some examples of secure systems that provide GSSAPI interfaces are:

- Kerberos V5 [12]
- Open Group Distributed Computing Environment (DCE) [18]
- The Globus Toolkit / Grid Security Infrastructure [8][9][5]
- Entrust™ [7]

Microsoft® Windows® 2000, which uses Kerberos V5 authentication, provides a “GSSAPI-like” portability layer in their Security Software Provider Interface (SSPI). This API is compatible with GSSAPI over Kerberos V5. [15]

GSSAPI is not a network protocol, it is a generic abstraction that can layer on top of a variety of network security protocols. A GSSAPI client application is not network interoperable with a GSSAPI service application unless both the client and the server are using compatible protocols and mechanisms. These are identified at run time by mechanism Object Identifiers (OIDs). [14] Both the client and the server must be using compatible OIDs in order to establish a connection. For example, until recently, GSSAPI client applications built upon MIT Kerberos V5 were unable to interoperate with GSSAPI server applications built upon Open Group DCE, because the latter was using an older version of Kerberos, and an older mechanism OID. (The latest versions of DCE are compatible with the latest Kerberos libraries.)

One important purpose of GSSAPI is to provide a simplified abstraction for security, isolating the high-level handshake from the low-level mechanisms and protocols. This simplified abstraction makes application development relatively easy. For example, the client side of a GSSAPI handshake is this simple:

1. The application assumes that the user or process is ‘logged in’ to the security system. (This is provided by the underlying security system via a login utility such as ‘*kinit*’ or ‘*grid-proxy-init*’ in which the user logs in with a password.)
2. The application calls `gss_acquire_cred()`, to get a credential handle. This provides the application with a handle to an object that only the logged-in user can access. The application need not know where the credential is, how it’s protected, and what it contains.
3. The application calls `gss_init_sec_context()`, passing in a token (which is initially null), some properties and flags, the

name of the desired target, and handles to a security context and output token.

4. The application passes the output token prepared by `gss_init_sec_context()` to the server or peer at the other end of the conversation, then reads a reply token from the other end.
5. If the last result from `gss_init_sec_context()` was a `CONTINUE_NEEDED` code, the application repeats steps 2 and 3, passing in the reply token from step 4 as the next input token.
6. When the call to `gss_init_sec_context()` returns a `COMPLETED` code, the application has a ‘security context’ which can be used to sign or encrypt subsequent messages.

The GSSAPI application programmer does not need to be concerned with the arcane details of credential certificates and keys, or with complex protocols involving SSL key exchanges or Kerberos ticket acquisitions. It’s all magically handled by the underlying GSSAPI software.

The simplified abstraction is important, but the primary purpose of GSSAPI is to provide a portability layer for developing or wrapping secure network applications that might be ported to new or different security infrastructures as the need arises.

The GSSAPI specification is designed to accommodate a wide variety of existing and yet-to-be-invented security protocols and mechanisms. To accomplish this, it must be broad and flexible – supporting but not requiring many features in optionally implemented flags and parameters. This aspect of the GSSAPI is the cause of its unfortunate but unavoidable shortcomings as a portability layer. For example, GSSAPI supports a delegated credential in a manner that is compatible with Kerberos and GSI. But many security protocols have no support for delegated credentials. Porting an application to such a system would require eliminating the application’s need for that feature, which may or may not be feasible.

Ideally, a port can be accomplished by making modifications to the application code (the caller of GSSAPI functionality.) But sometimes, as was the case with our Globus/Kerberos port, modifications are necessary in the low-level Kerberos/GSSAPI code.

2.3 Globus and Grid Security Infrastructure (GSI)

The Globus Toolkit [8] is a set of network applications and tools that provide an integrated infrastructure for finding and using resources on a network. The Globus toolkit supports sessions in which a desktop client submits tasks to resource machines. Globus tasks are submitted to a network daemon called the Globus Gatekeeper, which authenticates the user and forks job manager processes on behalf of the user. Job managers publish job metadata to a Metacomputing Directory Service (MDS). Resource machine status reports are also periodically published to the MDS by Globus daemons.

The client application requires two or three network connections with the job manager to receive job I/O and status. In addition, connections are required in order to publish and retrieve MDS information. These connections are authenticated and protected

via a GSSAPI-based security infrastructure. The standard Globus Toolkit uses a GSSAPI implementation called the Grid Security Infrastructure (GSI). [5] The GSI is based on X509 public key certificates issued by certificate authorities. Globus resource machines are configured with a trust relationship to one or more certificate authorities, and use map files to map distinguished names in these certificates to UNIX names and UIDs on the resource machine.

The security services underlying the GSI are based on adaptations of public domain Transport Layer Security (TLS) or Secure Sockets Layer (SSL) cryptographic libraries. (TLS is the newer name for SSL.) All calls used by the GSI to acquire credentials, establish secure sessions between network processes, encrypt and authenticate the sessions, and transfer delegated credentials are made via standard GSSAPI Version 2 C bindings. Strict adherence to GSSAPI interface bindings made it theoretically feasible to port a GSI-secured application to any security infrastructure that provides a GSSAPI Version 2 compliant C binding interface.

As mentioned earlier, the Globus/GSI security developers included Kerberos experts who designed a system that was conceptually similar to Kerberos. For example, a GSI credential is very analogous to a Kerberos user credential. In both cases the user's credential:

- Is maintained in a persistent data object called a credential cache.
- Is typically a local file, readable only by the user.
- Is created when a user supplies a password or passphrase to an initialization program. (GSI uses *grid_proxy_init*, Kerberos uses *kinit*.)
- Contains only time-limited session keys or proxy certificate keys, never the user's long term identity key.

GSI includes optional utilities that provide support for a Kerberos/GSI hybrid authentication environment. One important utility allows a GSI credential to be promoted to a Kerberos credential. This functionality requires a modification of the Kerberos security server. (Our grid did not consider this option because ASCII required the use of RFC compliant and vendor-supported Kerberos security servers.)

2.4 Simple Kerberos-Globus Authentication Scenario

Figure 2 illustrates the network security requirements of a simple DRM session using Globus and Kerberos security.

In this illustration, the user has already logged in (typically using the Kerberos *'kinit -f'* command) and the user's credentials cache is available to the *globusrun* process.

The *globusrun* process initiates a GSSAPI-secured connection to the Globus gatekeeper running on a resource machine. This connection is very standard — the gatekeeper is running as root, and has access to a Kerberos service key in a keytab file. This file contains the keys needed to decrypt the authentication information in the service ticket.

The gatekeeper then forks a Job Manager process that runs with the UNIX ID of the authenticated user. Since this process will be initiating and accepting network connections, it must have access to the user's Kerberos credentials. The gatekeeper therefore needs to export those credentials in a manner that makes them securely available to forked processes. Delegated credential export was added by GSI developers to their GSSAPI implementation but is not yet part of the standard GSSAPI. This was our first portability challenge.

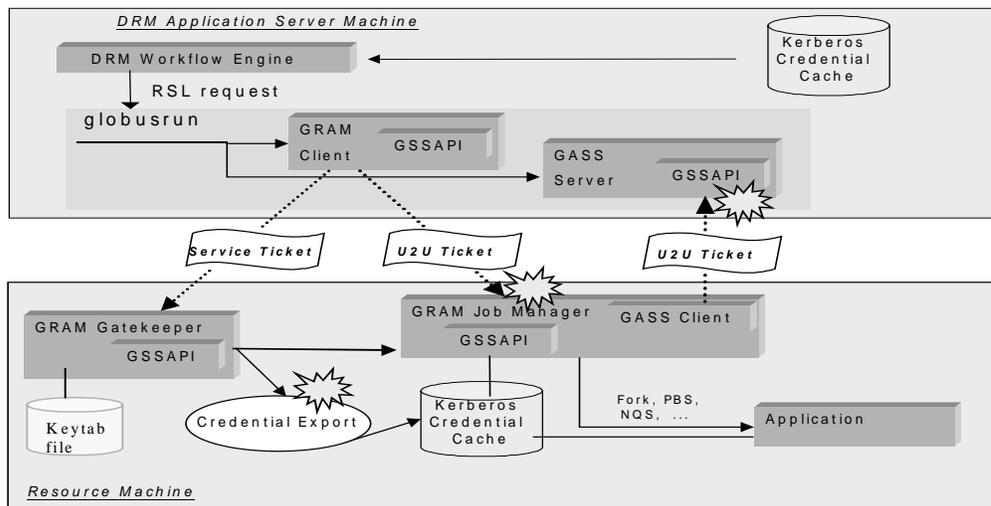


Figure 2: Kerberos-secured Globus session in action. Portability challenges are marked with a star.

Subsequent connections involve user processes that need to accept authenticated secure connections. Standard MIT Kerberos GSSAPI is very traditional client-server, and assumes that the acceptor of a connection is a privileged process with access to a keytab file containing a service's secret key — not a user process with a user's credential cache. Adapting Kerberos' GSSAPI layer to support these user-to-user connections was our major portability challenge.

3. PORTING ISSUES

3.1 Kerberos vs. DCE

The Globus GSI uses 'C' library bindings based on the GSSAPI version 2. This API is supported in the version 1.0 and later implementations of MIT Kerberos version 5, as well as in the latest releases of vendor-supported DCE libraries from IBM/Transarc. Because Globus requires GSSAPI features not found in the standard libraries, and because we have source for Kerberos libraries, not for DCE, our choice was simple. Our Globus will be linked against libraries built from MIT Kerberos V5, release 1.x.

Using Kerberos libraries will not preclude us from using DCE/DFS in Globus processes. Kerberos credentials, including the delegated credential that impersonates the user in the Globus forked processes, can be promoted to DCE credentials, which will permit those processes to use DFS or other DCE applications. (DCE credentials are based on standard Kerberos tickets, but also include a Privilege Attribute Certificate identifying the DCE authorization groups to which the principal belongs.)

3.2 Avoiding Compile-time Kerberos Dependencies in Globus

The current system uses statically linked GSSAPI libraries, so the Kerberos vs. GSI decision is essentially made when you run the *make* command to build the binaries. Nevertheless, a long range hope is that the binding to either GSI or Kerberos might eventually be made at run-time using dynamically loaded libraries.

Toward that end, we are avoiding patching Kerberos-specific or GSI-specific code into Globus. Where necessary to support mechanism-specific behavior (like the handling of delegated credentials) we have opted to carefully modify the Kerberos GSSAPI library source code.

3.3 Authenticated LDAP Access

Much of Globus relies on a Metadata Directory Service (MDS) where information is posted by resource machines and retrieved by web servers and user processes. The original Globus Toolkit did not use GSSAPI authentication in the LDAP connections to the MDS. Our system requires strong Kerberos authentication in all network connections. With help from the Globus developers, and some code licensed from PADL Software Ltd., we adapted our system to use Netscape Directory Servers with a Simple Authentication Services Layer (SASL) plug-in that supports GSSAPI over Kerberos. The plug-in is a conventional GSSAPI service with no delegation, and requires no special adaptations of the underlying Kerberos/GSSAPI code.

4. PORTING IMPLEMENTATION

4.1 Delegated Credentials

Kerberos and GSSAPI have some shortcomings in their support for making delegated credentials available to forked processes. Because security systems differ so widely in whether and how delegation and impersonation are supported, the GSSAPI does not encapsulate the delegated credential inside the generic abstraction of security context. The GSSAPI provides an abstraction (delegated credential handle) but assumes that any operations dealing with that handle will be in implementation-specific code.

The GSSAPI has a single routine, `gss_export_context()`, which can be called to export a security context that can be imported by another process or child process. The importing process can continue the secure conversation with the client. Unfortunately, after an export/import the process has no access to the client's delegated credential. There is currently no GSSAPI routine that allows a process to export a delegated credential to another process. In the Kerberos V5 release 1.x environment, a delegated credential is only available in a memory cache of a GSSAPI context-accepting service.

The ideal solution to this dilemma is to extend the GSSAPI specification, and members of the Globus GSI team are pursuing that with a new IETF draft (not yet submitted to IETF, but in review by the Global Grid Forum [21]). We are participating in the development of this draft, but for now, Globus will rely on an interim modification to an existing GSSAPI function.

The modified function is `gss_inquire_cred()`, which is normally used to get information about a credential. The prototype for that function call is as follows:

```
OM_uint32 gss_inquire_cred (
OM_uint32          *minor_status, /* out */
const gss_cred_id_t cred_handle, /* in */
gss_name_t         *name,        /* out */
OM_uint32         *lifetime,    /* out */
gss_cred_usage_t  *cred_usage,  /* out */
gss_OID_set       *mechanisms  /* out */ );
```

The `minor_status` variable is output only and is not expected to be initialized to any particular value when the routine is called. In the Globus GSS, however, if a value of 0xDEE0 is assigned to `*minor_status` before making the call, then the function will have a side-effect: The credential will be cached to storage and the name of the credential cache returned in the `*name` parameter. In the Kerberos implementation, the storage is on-disk (so that forked processes can read the credential cache). In the Kerberos version the value in `*name` has the format: "KRB5CCNAME=/tmp/krb5ccname.xxxxxx.x. If this call detects the 0xDEE flag in `*minor_status`, it assigns the code 0xDEE1 to `*minor_status`, thus providing the caller an assurance that this version of GSSAPI supports this code modification.

4.2 Acquiring an Initiator/Acceptor Credential

The *globusrun* example session (described in section 2.4) requires that a user's job process acquire a credential for accepting a context. This is not supported by the current MIT Kerberos 1.x implementation, which assumes that initiators of a context are always user processes (with a credential cache from a previous

'kinit) and that acceptors are services with access to a keytab key. The Globus code in many places calls `gss_acquire_cred()` with a flag set to `GSS_C_BOTH`, which is used to acquire a credential that can be used to either initiate or to accept a secure connection. For user-to-user to work, the Kerberos GSSAPI code had to be modified so that these acceptor credentials could, in some cases, be in a user's credential cache (i.e., the on-disk cache that is referenced by the environment variable `$KRB5CCNAME`).

We modified the Kerberos library so that when a `GSS_C_BOTH` or `GSS_C_ACCEPT` credential is requested, new logic in `gss_acquire_cred()` will always retrieve a keytab credential if one is available, but will otherwise retrieve a cached credential (typically a delegated user credential.) This logic allows the keytab identity (which is a longer-lived key) to take preference over the credential cache identity when both exist. This modification complies with the current RFCs for Kerberos and GSSAPI.

4.3 User-to-User

Callbacks from resource-side processes (which are impersonating the user using delegated credentials) must establish GSSAPI secured connections back to the user's listening sockets on the client desktop or DRM agent. There will probably be a future requirement that job managers accept connections from other job managers, all using delegated user credentials. These communications require an authentication feature called user-to-user authentication.

A ticket protocol that supports user-to-user has been in Kerberos for a long time, primarily to permit X-11 client applications to initiate secure sessions with X display servers (which are user sessions.) Unfortunately, user-to-user connections have never been supported in Kerberos' GSSAPI.

The user-to-user ticket protocol involves a special type of ticket request in which the client includes two ticket-granting-tickets (TGTs) in the service ticket request: the usual TGT from the credential cache, plus the TGT belonging of the intended session peer. For this to work in GSSAPI there must be an additional handshake message that can transfer the acceptor's TGT to the initiator.

There was some guidance on how to approach this in expired IETF drafts and in an Open Group RFC[4]. Our modifications comply with that guidance, and we have co-authored a new IETF informational draft [19] (along with Kerberos developers at Microsoft and at University of Washington) that describes our approach, which should be wire compatible with the Microsoft® Windows® 2000 Kerberos user-to-user handshake.

The strategy involves introducing two new message tokens to the underlying `gss_init_sec_context()` and `gss_accept_sec_context()` handshake. These tokens will be opaque to the caller of these functions – the additional token exchanges will be supported using the `GSS_C_CONTINUE_NEEDED` status flag. (As discussed previously, GSSAPI supports complex multi-stage handshakes with a generic model that hides the implementation of the handshake from the calling functions.)

In explaining this strategy, the terms 'client' and 'server' can be confusing because in many cases it is the Job Manager side (typically called the 'server') that will initiate a user-to-user

connection. This discussion uses the terms *initiator* and *acceptor* to avoid confusion.

- Initiator: The process initiating a secure connection. In the case of callbacks, the initiator may be running on the Job Manager's host
- Acceptor: The process accepting a secure connection. In the case of Globus callbacks, the acceptor might be running on the user's desktop.

Our user-to-user strategy requires two error messages that inform an initiator that user-to-user is required for this session. The first error code may be returned by the KDC in the initial `gss_init_sec_context()` call. (DCE and Kerberos 5 security servers already return this error if an account is appropriately flagged as requiring user-to-user authentication.) The other error code was needed so that the acceptor could tell the initiator that user-to-user is required. We modified the GSSAPI code so that if an acceptor receives a conventional service ticket, and has no access to a service keytab (because it is a user process) it will return an error code indicating that user-to-user is required (this is a new KRB5 error message.)

We modified the GSSAPI initiator code to detect either of these errors and handle them by preparing a `TGT_REQUEST` token (one of the two new tokens.) This token is sent to the peer by the initiator application, which treats it just like any other opaque GSSAPI token.

We also modified the underlying code of `gss_accept_sec_context()` so that when the acceptor receives this `TGT_REQUEST`, it replies with another newly defined token. The type of this token is `TGT_REPLY`, and it contains a principal name and a "second ticket". The second ticket is a TGT obtained from the credential cache. This is just a TGT and not a credential (a credential is a TGT and its associated session key.) Passing a TGT in an unprotected session is an accepted part of the Kerberos protocol, and is demonstrated in the MIT code base example, `uuclient.c`.

After receiving a `TGT_REPLY` token, the next pass through `gss_init_sec_context()` extracts and uses the second TGT to acquire a user-to-user service ticket, which is placed in the next outgoing GSSAPI token instead of a conventional service ticket. The rest of the handshake iterations proceed the same as in standard GSSAPI.

5. RESULTS

5.1 Current Status

The system has been fully functional since August 2000. The compile-time modifications (`#ifdef KRB5` code) are minimal, and we may eventually eliminate these to support a system that could be converted at link-time from GSI security libraries to Kerberos 5 libraries.

The complete DRM system was accredited for use in DOE classified networks in January 2001. The security plans and the security test plans make the assertion that all network services use Kerberos authentication and Kerberos/GSSAPI sessions to provide transport layer security. This simple assertion provided for a relatively fast and painless DOE approval process. We deployed the system on top of the existing DCE/Kerberos account

infrastructures, so no user of the system needed to generate a new certificate or learn a new password. Existing cross-cell account systems are used to provide Globus mappings (gridmap files) that convert principal names to UNIX user Ids on the resource machines. ASCI resources needed no additional access control infrastructure because our system relies on existing DCE/DFS/Unix access controls for need-to-know protection. The system is currently in production, having successfully completed tri-lab acceptance testing in April 2001.

5.2 Error Reporting Difficulties

GSSAPI provides an interface, `gss_display_status()`, that may be used to help decipher a GSSAPI failure code, converting it to a message string that is provided by the underlying security mechanism and, ideally, makes sense to a user or administrator. The Globus system logs these messages in the gatekeeper logs, which assists with troubleshooting by the administrator. But generally speaking, the GSSAPI tends to isolate higher-level software layers such that they are unable to interpret failures and provide meaningful feedback to the user. For example, when a Globus GASS service attempts to connect back to the work manager, if the connection handshake fails for any reason the user gets a “failed to open I/O” message, rather than something that indicates the true problem. (Typically, the problem is that the user’s DCE/Kerberos account is missing some required flags.)

This could be improved somewhat by additions to exception handling code in Globus, making sure that the `gss_display_status()` string and the context of the GSSAPI failure makes it all the way back to the user interface whenever possible.

Our best option will be to provide detailed troubleshooting guidance in the DRM User’s Guide and to include tools and utilities to detect and report security problems. Common problems include: incorrect or incomplete user account setup, user’s failure to acquire a forwardable credential, missing globusmap authorization, and a user inadvertently running a client with root privileges, using keytab rather than user credentials.

5.3 Code Maintenance

ASCI Kerberos developers will maintain the patches necessary to support Globus along with other patches of the MIT code base that we develop and maintain for use by ASCI laboratories. We will work to assure that IETF standards and the MIT base code incorporates our changes as much as possible. It is in our interest to encourage the evolution of both Globus and Kerberos in a direction that supports the needs of ASCI.

5.4 User-to-User Account Flags

Enabling user-to-user will require that both the ‘server yes’ flag and the ‘dupkey yes’ flag be set on the Globus user’s Kerberos/DCE account. This may have security implications. Setting the ‘server yes’ flag in some environments may make user keys more vulnerable to offline cracking, because conventional service tickets contain plaintext encrypted in the service’s private key. (Note: If the ‘preauth-required’ flag is not set on a user account, the risk of offline cracking is already there, and would be made no worse by adding the server flag.) In environments that use the latest DCE 1.2 security servers, setting the ‘server yes’, ‘dupkey yes’ and ‘usertouser yes’ account flags will eliminate the

security risk, because this flag causes the KDC to allow user-to-user tickets but never issue a conventional service ticket for that account principal. We have tested and verified this functionality. In the ASCI environment we will enforce user-to-user authentication on user accounts. This is an efficiency feature as well as a security feature, as it potentially eliminates a round-trip in the GSSAPI token exchange when user-to-user is required.

6. FUTURE WORK

6.1 Expiring Credentials

There may be cases where a Globus user requests a job, and the job may not start for a long while, or may require a long time to complete. If these jobs require Kerberos/DCE network credentials, they may fail. The standard credential is only valid for 10 hours after the user has used the ‘*kinit*’ command. A “*kinit -f -l 24h*” command, which specifies a lifetime of 24 hours, can be used to get longer-lived credentials. A command such as “*kinit -r 30d -f -l 24h*” can be used to acquire a credential that can be used for 30 days, provided a process periodically renews the credential. These limits are subject to site-imposed maximums.

We need a Globus resource utility that can properly refresh renewable credentials regularly. The idea of a generic credential refresh agent is being investigated by the ASCI DCE group and has been solved in some cases within specific application code.

6.2 Public Key Infrastructure (PKI) Integration

There is a task underway in ASCI to investigate and pilot Kerberos-PKI integration via an IETF standard[20]. One motive for this is to enforce two-factor authentication via private keys embedded in smart cards. Some have misinterpreted this as a technology that would enable interoperability between ASCI/DCE and GSI PKI. The important thing to understand is that this Kerberos-PKI integration will not change the way clients authenticate to servers, only the way initial Kerberos credentials are acquired. For example, with PKI integration, the users’ ‘*kinit*’ command would require that they login to Entrust using their smart card and pin. Once logged in, authentication between clients and servers would be based on Kerberos tickets. So even after this integration, our Globus implementation would continue to require Kerberos tickets, not PKI certificates.

ASCI has a DOE accredited inter-site PKI based on Entrust™. This infrastructure currently supports a relatively small number of people who need to encrypt or digitally sign e-mail messages or persistent data files. But eventually the ASCI PKI will be widely deployed such that every user has an identity certificate.

Once such an infrastructure is in place, it might be possible to retool the ASCI Globus system to use PKI. This would probably entail a system that uses signature keys from our Entrust™ PKI to sign GSI proxy certificates. Such an infrastructure might pave the way to interoperability between ASCI and sites on the open grid that have deployed GSI-secured Globus. Interoperability will require more than just basing our Globus on GSI; it will require establishing trust relationships and cross-certifying with outside certificate authorities. This is technologically simple, but extremely difficult in practice. Our security requirements will

demand formal agreements with trusted sites to assure that they have high assurance certificate practices and policies, and reliable, high availability certificate revocation services.

Despite the touted advantages of PKI as an authentication infrastructure, Kerberos has some clear advantages over GSI:

- Kerberos' GSSAPI handshake is more efficient, being based on secret key.
- Kerberos enforces central trust. If a foreign site is compromised, a Kerberos security administrator can instantly disable access by any user from the compromised site.
- Kerberos is a mature and respected standard.
- Kerberos is becoming (with Microsoft Windows 2000) a very widely deployed infrastructure.

Despite these advantages, ASCI teams working on security policy and technology have not ruled out GSI as an option. As the GSI-secured global grid becomes widespread, based on mature, high assurance certificate authorities, perhaps we can pursue cross certification between the ASCI PKI and certificate authorities in the global grid. This would pave the way to making at least some resources on the open ASCI grid available to trusted collaborators that use GSI rather than Kerberos.

7. CONCLUSIONS

We chose Globus for our grid services partly because it was based on GSSAPI, and that made it adaptable (we hoped) to our security requirements. Our success in implementing a Kerberos-secured Globus demonstrates that GSSAPI, for all its shortcomings, is a viable portability layer. In retrospect, the developers of Globus and GSI made a wise choice in building their security around GSSAPI, and we hope other developers of secure systems follow their example.

We can't predict at this point whether the long-term future of the open ASCI grid lies with Kerberos or with GSI. Fortunately, we have a system that can be built upon either.

8. ACKNOWLEDGMENTS

Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the United States Department of Energy under Contract DE-AC04-94AL85000. We are grateful to the designers and developers of Globus Security for building a system that ported so nicely. This team included Doug Engert, Steve Tuecke, and Joe Bester at Argonne National Laboratory, Mei-Hui Su at USC Information Sciences Institute, and Von Welsh at NCSA. We especially want to thank Doug Engert for working closely with us on this project, providing us with Kerberos expertise, guidance, and some critical code examples and patches.

Our work was accomplished as a part of a team effort to provide ASCI grid services. The ASCI Grid Services team includes: Esther Baldonado, Judy Beiriger, Hugh Bivens, Dwight Coles, Kathy Hiebert-Dodd, Clark Haskins, Steven Humphreys, Ann Hodges, Wilbur Johnson, Lois Lauer, Ellen Lemen, Don McLaughlin, Pat Moore, Ron Rhea, Mary Roehrig, Bryan Spicer, and Ruthe Vandewart. At Los Alamos, the team includes Randal

Reinheimer and William Barber. The Lawrence Livermore team members are Moe Jette, Keith Fitzgerald, Lori Eastment, and Amy Pezzoni.

9. REFERENCES

- [1] Beiriger, J. et. al, Constructing The ASCI Grid, *Proceedings of 9th High Performance and Distributed Computing Conference*, Pittsburgh, PA, August 2000.
- [2] Bivens, H., Beiriger, J., GALE: Grid Access Language for High Performance Computing Environments. Work in progress, hpbiven@sandia.gov, Sandia National Laboratories, 2001.
- [3] Brunett, S., Czajkowski, K., Fitzgerald, S., Foster, I., Johnson, A., Kesselman, C., Leigh, J. and Tuecke, S., Application Experiences with the Globus Toolkit. In Proc. 7th IEEE Symp. on High Performance Distributed Computing, 1998, IEEE Press, 81-89.
- [4] Burati, M. Pato, J., User-to-User Authentication -- Functional Specification. OpenGroup RFC 91.1 1996. <http://www.opengroup.org/tech/rfc/rfc91.0.html>
- [5] Butler, R., Engert, D., Foster, I., Kesselman, C., Tuecke, S., Volmer, J. and Welch, V. Design and Deployment of a National-Scale Authentication Infrastructure. *IEEE Computer*, 33(12):60-66. 2000.
- [6] Detry, R., Kleban, S., Moore, P., and Berg R., The Generalized Security Framework. Presented at CSCORE 2000. <http://www.ccs.bnl.gov>, Brookhaven National Laboratory, NY.
- [7] Entrust Inc., The EntrustSession™ Toolkit FAQ. Online documentation. 2001, <https://www.entrust.com/developer/session/faqs.htm>
- [8] Foster, I., and Kesselman, C., Globus: A Metacomputing Infrastructure Toolkit, *International Journal of Supercomputer Applications*, 1997.
- [9] Foster, I. and Kesselman, C. (eds.). The Grid: Blueprint for a New Computing Infrastructure. Morgan Kaufmann, 1999.
- [10] Kohl, John T., Neuman, B. Clifford, T'so, Theodore Y. The Evolution of the Kerberos Authentication System. In Distributed Open Systems, pages 78-94. IEEE Computer Society Press, 1994.
- [11] J. Kohl, J., C. Neuman, C., The Kerberos Network Authentication Service (V5), IETF RFC 1510. 1993. <http://www.ietf.org/rfc/rfc1510.txt>
- [12] Linn, J., The Kerberos Version 5 GSS-API Mechanism, IETF RFC 1964. 1996, <http://www.ietf.org/rfc/rfc1964.txt>
- [13] Linn, J., Generic Security Service Application Program Interface Version 2, Update 1, IETF, RFC 2743, 2000. <http://www.ietf.org/rfc/rfc2743>.
- [14] Mealling, M., A URN Namespace of Object Identifiers. IETF RFC 3061, 2001. <http://www.ietf.org/rfc/rfc3061>
- [15] The Microsoft Corporation. Answers to Frequently Asked Kerberos Questions. Online documentation Q266080, 2000. <http://support.microsoft.com/support/kb/articles/Q266/0/80.ASP>

- [16] Myers, J, SASL GSSAPI Mechanisms. IETF Draft (Work in progress, 2001), <http://search.ietf.org/ID.html>
- [17] Neuman, B. Clifford and Ts'o , Theodore. Kerberos: An Authentication Service for Computer Networks, IEEE Communications, 32(9):33-38. September 1994.
- [18] Rosenberry, W., Ed., DCE Today – An Indispensable Guide to DCE. The Open Group 1998.
<http://www.opengroup.org/publications>
- [19] Swift, M., Brezak, J., Moore, P., User to User Kerberos Authentication using GSS-API. IETF Informational Draft (work in progress, 2001), <http://search.ietf.org/ID.html>
- [20] Tung, B. et.al, Public Key Cryptography for Initial Authentication in Kerberos. IETF Draft (Work in progress, 2001) <http://search.ietf.org/ID.html>
- [21] Welch, V, Tuecke, S, Engert, D. GSS-API Extensions, GGF Draft (work in progress, 2001)
- [22] Wray, J., Generic Security Service API Version 2,; C-Bindings. IETF, RFC 2744, 2000.
<http://www.ietf.org/rfc/rfc2744>.