Globus Restricted Proxies and Classified Advertisements as a Policy Language*

Babu Sundaram**
Department of Computer Science
University of Houston, Houston
Babusun@bayou.uh.edu

Christopher Nebergall**
Department of Computer Science
Western Illinois University, Macomb
ct-nebergall@wiu.edu

*Work performed at Argonne National Laboratory, a contract Laboratory of the United States Department of Energy.

**Participants in the Summer 2000 Student Research Participation Program, Argonne National Laboratory, Argonne, IL.

Abstract:

Computer security is an important concern to scientists working in the area of distributed computing. With many different institutions sharing resources over an open network the possibility of a user's credentials being compromised becomes a real threat. Security policies along with a language to describe these policies needs to be developed in order to reduce the risks involved in participating on a distributed network, and to reduce damage that can be done through the use of stolen credentials.

This paper describes how security policies can be implemented into the Globus Project using the extensions in the digital certificates. The framework described allows both the owner of a credential and a Globus site administrator to develop policies on how these improved credentials can be used.

Restrictions researched include policies that limit where, when, and how credentials can be used. The new restrictions can be placed both in the credentials themselves and in the servers that use these credentials for authentication.

A matching system called Classified Advertisements or classads developed by the University of Wisconsin was evaluated for its abilities to implement these security policies. In our work, the matching abilities of classads are used to match resource requests against security policies. Classads were found to be very capable of describing these policies and with a few new additions to the language they will support all the recommendations.

Introduction

Globus is a software suite designed to allow remote access to resources on Globus enabled machines. A person merely logs onto a machine running the Globus software, and they have access to computational resources, advanced storage devices, or immersive environments such as the CAVE located anywhere in the world.

Globus uses a credential passing mechanism to gain authorization to remote sites. The user first obtains a certificate signed by a Certification Authority. In order to access these remote sites the user creates a temporary credential called a proxy credential and digitally signs it with his or her private key. This credential also identifies the user, but it will expire with in a specified number of hours. The use of temporary credentials adds a layer of security to Globus, but in some situations it is not sufficient. Within that time span a compromised credential can be used to impersonate its owner on any site for which he or she has an account.

Restricted proxies are temporary credentials that contain information about how or where the credential can be used. Very little research has been done on how to implement restricted proxies and what constraints they should contain. We, Christopher Nebergall and Babu Sundaram, were assigned to develop and test possible constraints for restricted proxy usage in Globus. We explored the constraints and the possibilities of implementing restricted proxies for a system like Globus and the feasibility of using classads as the policy language to express the restrictions in such proxies. This document illustrates our work.

Proxy Chaining

In Globus a user can submit a job to any machine on which he or she has an account. A proxy is delegated to this machine so that it can then act on behalf of the user. This process of delegating proxies creates a proxy chain. The entire proxy chain is needed for authentication. A site must check the signing trail to ensure the identity of the owner.

With the use of restricted proxies every site that delegates a proxy can add more restrictions, but previous restrictions cannot be removed. In order to ensure that restrictions have not been altered or removed during delegation, the restrictions are added as extensions in the certificate which are always digitally signed. To ensure the action requested by the user does not violate any of the restrictions listed in the proxies, the requested action must be compared against every restriction present in the chain. If at any point a restriction is violated the operation is disallowed. A non-ambiguous method for presenting and evaluating restrictions must be utilized for this purpose.

Classified Advertisements (Classads)

Classified Advertisements or classads have long since been used in Condor, a system used for distributed computing at the University of Wisconsin. Classads are a matchmaking scheme used to match a set of requirements against a second set of possible matches. They have been used intensively by Condor for resource matching.

Classads are implemented with a set of attributes and logical expressions that evaluate a match. This is done by specifying one's own requirements with references to its own attributes using the notation self.<attribute name> (self is assumed if not present) and referring to the attributes of the other classad with the notation other.<attribute name>. Each specifies its matching requirements by having a requirements expression that evaluates to either true or false. If the requirements of both of the classads being matched evaluates to true, the match is successful.

Example:

```
Resource Classad
[
StorageSpace = 800M;
RAMAvailable = 40M;
Architecture="x86";
OperatingSystem = "Linux";
Requirements = true;
]

Request Classad
[
RAMRequired = 30M;
OperatingSystem = "Linux";
Requirements = other.RAMAvaliable >= RAMRequired && self.OS == other.OS;
Rank = RAMAvailable - RAMRequired;
]
```

These sample classads demonstrate a resource match similar to the matches performed in Condor. The strength of classads is that a single request can be matched against a large set of possible matches. This is an efficient method for searching through a large collection of possibilities in order to find a single match. In the event that there are multiple matches the Rank attribute is used. The higher the value of the Rank attribute the better the match.

ClassAd Implementation in Globus

Classad's matching abilities are capable of more than matching for resource requirements. Classads present a capable language for describing security restrictions. In this project classads where implemented into Globus to test security policies that are not possible in the current system. New security restrictions are possible such as the ability to specify when, where, and how a proxy can be used. There are several key areas where classads could be implemented into Globus, first into the proxies themselves as extensions, second into the Gatekeeper, and third into the Job Manager. Alterations to the Gatekeeper and Job Manager allow these programs to perform the matching services needed to adhere to the restrictions presented in the proxies, and secondly, the alterations allow the Globus administrator to implement restrictions on a global or per-user basis.

User Proxies enhanced with ClassAds

Several new restrictions are made possible through the use of classads in the user proxy. These restrictions are in place to protect the owner of the proxy in the event that the credential is compromised.

In this model the server that the proxy is presented to does all the security and matching services. There would be no security benefit from performing the match on the client side. The act of submitting the certificate to the server is evidence that the user wishes to gain access. A second reason for this implementation is that the classad in the proxy cannot be changed. The server's classad needs to be more dynamic, so that the Globus administrator can easily alter the server's policies on a global or per-user basis.

User Proxy:

```
[
    requirements = true;
    limited = false;
    operatefrom =
      {
        "moleman.mcs.anl.gov"
        "pitcairn.mcs.anl.gov"
      };
    targetsites =
      {
        "/O=Grid/O=Globus/CN=moleman.mcs.anl.gov",
         "/O=Grid/O=Globus/CN=machine.isi.edu/"
      };
        LoginRequirements = Member(targetsites,other.serverstring) &&  !limited &&
        Member(operatefrom, dynamic.hostname);

     dynamic*
        [
                hostname = "moleman.mcs.anl.gov";
        ]
  ]
```

*Attributes added to the dynamic classad are added by the gatekeeper before the match. Any dynamic attributes that the user may have added to this section are automatically cleared before new attributes were added by the gatekeeper.

Gatekeeper Proxy

```
[
        serverstring = "/O=Grid/O=Globus/CN=moleman.mcs.anl.gov"
        requirements = other.AuthenticationRequirements;
]
```

Among the new restrictions possible are limits on the locations or "target sites" of where a particular proxy can be presented. In the example used above the target site list contains a list of subject names. These are the subject names in the certificates of machines with Globus installations. When a user presents his certificate to be matched by the Gatekeeper the match will fail if the Gatekeeper's certificate subject is not in the proxy's target site list.

Another restriction added to the proxy is the "operate from" list. This lists the sites for which a resource could expect to receive this proxy. After the Gatekeeper reads the classad into memory it appends the host name of the location the request is coming from into the user's proxy. This information is for this match only and the changes are not written back to disk. To prevent a user from placing possibly incorrect data into the list, all the dynamic attributes are cleared before new attributes are added. If the request comes from a site not present on this list, it will not be accepted. If a person who steals a compromised proxy wishes to use it they must first have access to a site that the proxy can legitimately be presented from.

Another benefit of using an "operate from" list is that it effectively makes the proxy limited for the list of sites on the "target sites" list that are not on the "operate from" list. These sites can use the proxy for authentication purposes, but they cannot delegate it to a new site. This allows a person who initially sends out a job to specify which sites he trusts to have the ability to delegate his proxy. If a limited proxy is compromised it not as able to cause as much damage as a full proxy, because it cannot be used directly to run processes on remote machines.

Setting the limited attribute in a flag to true is equivalent to cause the proxy to be limited from that point on. It is functionally the same as removing all sites from the "operate from" list and the "target sites" list, but allows the "operate from" and "target sites" lists to not be left off.

A third attribute is possible called Authentication Requirements. This attribute could be used as a way of specifying for which sites or for what reasons the proxy could be used for authentication. This is a slightly different problem than the one addressed by log in Requirements. A situation is possible where a proxy was created for the sole purpose of checking the status of a particular job. Currently limited proxies can be used for authentication (but not delegation) at any site. This new method would allow for users to specify exactly where and when a limited proxy could be used for authentication. The requirements used for the match would either be the same as for login requirements by setting them equal, or a new list of allowed sites could be used for communication.

Another restriction is possible that is based on Globus version. The owner of a proxy may not want his proxy used with a particular Globus version. This may be because he or she knows that a particular version of Globus is not suitable for his needs or because a particular version is known to contain security holes. Every Globus server should present their version numbers in their classads in the following manner.

```
Globus Server
[
        ....
        Version =
        [
                Release = 1;
                Major = 1;
                Minor = 3;
                Beta = 5;
        ]
]
```

A client can use this information in their requirements in any manner that he or she wishes.

For example:

```
User Proxy
[       ...
        LoginRequirements =
                (other.Version.Release == 1) &&
                (other.Version.Major == 1);
]
```

Another new restriction that can be applied is detailed time restrictions of when a proxy can be used. Proxy credentials have always contained their issue and expiration dates, but currently there are no additional time capabilities.

```
User Proxy
[
        ...
        StartProxyTime = 'Jan 20, 2000, 05:00';

        UserTimeList = {[valid = false, start = '08:00:00';end = '18:00:00'],
                        [valid = true,  start = '08:00:00';end = '18:00:00'],
                        [valid = true,  start = '10:00:00';end = '20:00:00'],
                        [valid = false, start = '08:00:00';end = '18:00:00'],
                        [valid = true, start = '08:00:00';end = '18:00:00'],
                        [valid = true, start = '08:00:00';end = '18:00:00'],
                        [valid = false, start = '08:00:00';end = '18:00:00']};

        ctime = CurrentTime();

        DayOfWeek = GetDayOfWeek(ctime);
        Requirements = UsertTimeList[DayOfWeek].valid &&
                       ctime >= UserTimeList[DayOfWeek].start &&
                       ctime <= UserTimeList[DayOfWeek].end &&
                       ctime >= StartProxyTime;
]
```

The UserTime list is a list that represents time restrictions for every day of the week. The index of the array is equivalent to the day of the week (0= Sunday, 1 = Monday, etc.). The valid attribute states whether the proxy is valid for a particular day. The StartProxyTime attribute is to aid the user in issuing a certificate that is not valid until a particular time occurs. This is useful for applications such as MyProxy which allow users to store their proxies for future use. The end time of the proxy is taken care of by the expiration date on the certificate.

A third recommendation is that when a proxy is delegated the name of the site that the proxy was delegated to be automatically be placed in the classad. It would produce a trail of where that proxy has been used. This information is useful if a gatekeeper only trusts particular sites and does not trust a proxy if it has been through any other sites.

Changes recommended for the Gatekeeper

Several recommended changes can be made so that the gatekeeper can use classads to give the Globus administrator the same control over security as the owner of the proxy is given. The Gatekeeper will have a Global Security Policy that all certificates must match against in order for them to be allowed access to the system.

Gatekeeper Global Classad

```
[
        NotOperateFrom = {
                                "pitcairn.mcs.anl.gov",
                                "clarinet.mcs.anl.gov,
                           }
        NotIntermediateSites =
                         {
                                "/O=Grid/O=Globus/CN=moleman.mcs.anl.gov"
                         }
        requirements = !Member(NotOperateFrom,other.dynamic.hostname)
                       && !(NotIntermediateSites, other.dynamic.hostsite)
]
```

All other restrictions are possible such as Global Time Restrictions, but Global policies cannot be overridden by user specific policies. So whatever is implemented in this policy is applicable for every user who logs into Globus.

The "non-intermediate sites" list is designed to examine sites for which a user's certificate has been delegated through, which will be listed in the proxy chain. For example the following diagram shows User proxy (U) and each of the following proxies delegated in the chain (U', U", etc.)

```
U
[
        ...
        hostsite = "/O=Grid/O=Globus/CN=moleman.mcs.anl.gov";
]

U'
[
        ...
        hostsite = "/O=Grid/O=Globus/CN=clarinet.mcs.anl.gov";
]

U"
[
        ...
        hostsite = "/O=Grid/O=Globus/CN=pitcairn.mcs.anl.gov";
]
```

As the gatekeeper goes through the whole proxy chain it will check each of the intermediate sites against its own list.

The Delegation List will not contain the name of the site where the original proxy was created. There is no secure way of addressing the problem. The grid-proxy-init program could be changed to append the host name into the user proxy when it is first created, but the user ultimately controls what value is entered into the user proxy.

Classads should be implemented to replace the gridmap file and append user specific security policies. This allows flexible policies to be created.

Sample Gatekeeper Grid Map File

```
[
        Subject = {"/O=Grid/O=Globus/OU=mcs.anl.gov/CN=Christopher Nebergall"};
        login = "CNebergall";
        UserTimeList = {[valid = false, start = '08:00:00';end = '18:00:00'],
                        [valid = true,  start = '08:00:00';end = '18:00:00'],
                        [valid = true,  start = '10:00:00';end = '20:00:00'],
                        [valid = false, start = '08:00:00';end = '18:00:00'],
                        [valid = true, start = '08:00:00';end = '18:00:00'],
                        [valid = true, start = '08:00:00';end = '18:00:00'],
                        [valid = false, start = '08:00:00';end = '18:00:00']};
        AccountValid = true;
        ctime = CurrentTime();
        DayOfWeek = GetDayOfWeek(ctime);
        TimeRequirements = UsertTimeList[DayOfWeek].valid &&
                        ctime >= UserTimeList[DayOfWeek].start &&
                        ctime <= UserTimeList[DayOfWeek].end;

        Requirements = AccountValid && TimeRequirements && Other.dynamic.Subject ==
                self.Subject;
]

[
        Subject = {"/O=Grid/O=Globus/OU=mcs.anl.gov/CN=Babu Sundaram"};
        login = "sundaram";
        AccountValid = true;
        Requirements = AccountValid;
]
[
        Subject = {"/O=Grid/O=Globus/OU=mcs.anl.gov/CN=Babu Sundaram",
                   "/O=Grid/O=Globus/OU=mcs.anl.gov/CN=John Doe"};
        login = "grpaccount";
        AccountValid = true;
        Requirements = AccountValid && Other.dynamic.Subject == self.Subject ;
]

User Proxy
[
        Rank = (other.hostname == "/O=Grid/O=Globus/CN=moleman.mcs.anl.gov" &&
                                                        login == grpaccount)?1:0;
        Requirement = && Other.dynamic.Subject == self.Subject;

        dynamic
        [
                subject = "/O=Grid/O=Globus/OU=mcs.anl.gov/CN=Babu Sundaram";
        ]
]
```

Each of these is a separate classad that represents different restrictions per subject name and account name. The person must present their credentials for matching to find the appropriate log in name. The Gatekeeper automatically appends the user's subject name from the certificate into the user's classad. The classad's in the grid map file are read into an array and a function matches the array of classads against the top-level classad into the user's certificate. The function looks for the best match in the list of classads. If there are several matches the match with the highest rank will be returned. The matching of the rank value allows users to specify which account they wish to use.

Policy Templates

It is possible to nest one classad inside a separate classad. In order to specify the classad use the notation expression.<attribute name>. The expression evaluates to a classad to find the scope and then the attribute name is evaluated at that scope. If that attribute name is not found the parent scope is searched. This trend will continue till the root scope is reached. If the root classad does not contain the attribute then the expression is undefined.

```
[
        a = 3;
        b = [
                d = 5;
                e = [g = "sample"]
            ]
]
```

Expression	*Value*
a	3

b.d	5
b	[d=5; e = [g = "sample]]
b.a	3
d	undefined
b.e.g	"sample"
b.e.d	5

This allows for the addition of default security restrictions as a type of policy template. These restrictions can be overwritten by specific user policies. In order to accomplish this, all attributes needed for the default policy matching must be placed at the top level. Every operation that compares these attributes must reference the more specific classad. If the user wishes to alter these attributes they can place values into the specific classad. If the attributes are not present the value will convert to the default values.

```
Default policy
[
          login = Specific.login;
          subject = Specific.subject;
          OperateFromList =
                    {"moleman.mcs.anl.gov","clarinet.mcs.anl.gov"};

          UserTimeList =    {[valid = false, start = '09:00:00';end = '17:00:00'],
                             [valid = true,  start = '09:00:00';end = '17:00:00'],
                             [valid = true,  start = '09:00:00';end = '17:00:00'],
                             [valid = true,  start = '09:00:00';end = '17:00:00'],
                             [valid = true,  start = '09:00:00';end = '17:00:00'],
                             [valid = true,  start = '09:00:00';end = '17:00:00'],
                             [valid = false, start = '09:00:00';end = '17:00:00']};

          ctime = CurrentTime();
          DayOfWeek = GetDayOfWeek(ctime);
          TimeRequirements = Specific.UsertTimeList[DayOfWeek].valid &&
                             ctime >= Specific.UserTimeList[DayOfWeek].start &&
                             ctime <= Specific.UserTimeList[DayOfWeek].end;

          IntermediateSites = {"/O=Grid/O=Globus/CN=pitcairn.mcs.anl.gov",
                               "/O=Grid/O=Globus/CN=moleman.mcs.anl.gov",
                               "/O=Grid/O=Globus/CN=clarinet.mcs.anl.gov"};

          LocationRequirements = Member(Specific.OperateFromList,other.dynamic.localhost);

          Req = Specific.TimeRequirements && Specific.LocationRequirements &&
          Other.dynamic.subject = Specific.Subject;
          Specific =
          [
                    Subject = "/O=Grid/O=Globus/OU=mcs.anl.gov/CN=John Doe";
                    login = "jdoe";
                    OperateFrom = {"pitcairn.mcs.anl.gov","homer.mcs.anl.gov"};
                    Req = TimeRestrictions && Subject==other.dynamic.subject;
          ]
          Requirements = Specific.Req;
]
```

When the user specifies his requirements he must use the req variable to avoid the circular reference problem created by also naming the specific attribute requirements. The specific.req variable if present can be used with the attributes in the parent scope or new requirements can be specified or both could be satisfied if Specific.Req is set in the following manner:

Req = .Req && NewRestrictions

The only attribute that must be present is Subject==other.dynamic.subject because it is required for the grid map file to accomplish its basic operation.

The above diagram is the appearance of the Gatekeeper's classad as it would appear before a match. The specific classad is not incorporated into the default classad until a match needs performed.

Changes recommended for Job Manager

In the job manager changes can be made that restrict which operations a particular user can do. This includes which executables they can run and what operations they are allowed to perform such as a job run or a job cancel. This must be implemented in several steps.

- The type of request being sent to the job manager must be inserted into the client classad. This can contain operations such as "Jobrun" or "jobcancel".
- If the request is a job run after the RSL (Resource Specification Language) is parsed the name of the executable and its command line arguments should be inserted into the client's classad.

Both the classad and the server can limit which operations can be used with the certificate. A user can insert resource policy statements into the classad in his certificate, which places limitations on how the certificate can be used. Also, the job manager can specify per user policies. These per user policies are shown below.

```
User Classad
[
        ...
        executables = {"/bin/ls","/bin/echo"};
        Operations = {"jobrun","jobcancel","statuscheck"};
        requirements = Member(executables, dynamic.executable) &&
         Member(Operations, dynamic.operation);
        dynamic =
        [
                Subject = "/O=Grid/O=Globus/OU=mcs.anl.gov/CN=Christopher Nebergall";
                operation = "jobrun";
                executable = "/bin/ls";
                arguments = "-al";
        ]
]

Job Manager Classads
[
        Subject = "/O=Grid/O=Globus/OU=mcs.anl.gov/CN=Christopher Nebergall";
        executables = {"/bin/ls", "/bin/echo", "/bin/cat"};
        Operations  = {"jobrun","jobcancel","statuscheck"};
        Requirements = other.dynamic.subject == Subject &&
                        Member(executables, other.dynamic.executable) &&
                        Member(Operations,  other.dynamic.operation);
]
[
        Subject = "/O=Grid/O=Globus/OU=mcs.anl.gov/CN=Babu Sundaram";
        executables = {"/bin/ls", "/bin/echo"};
        Operations  = {"jobrun","statuscheck"};
        Requirements = other.dynamic.subject == Subject &&
                        Member(executables, other.dynamic.executable) &&
                Member(Operations,other.dynamic.operation);
]
```

For security reasons the full path of the executable must be specified in the executable list for the Job Manager. If the full path is not specified there is no guarantee that the executable being run is the intended executable. The user could have placed executables by the same name into their account. Because UNIX supports file and directory linking, the Globus administrator will often have to specify several paths to the same executable. Allowing a user to run executables that have the ability to launch other executables is not recommended because if is not properly implemented such abilities will give the user the chance to circumvent the restrictions.

Restricting executables is more difficult in the certificate of a client. The reason is that absolute paths are specific to particular machines, and depending on the how the proxy certificate is used, it may be given to several machines. So either executable paths for several machines must be specified or relative paths must be considered.

It is not possible to specify a list of disallowed executables in current implementations of Globus. A user could merely rename or alias an executable to circumvent the restriction. This implies that if limits are placed on which executables can be run every possible executable the user needs must be present in the list.

Suggested extensions to the Classad Language

It is not possible to place restrictions on command line arguments currently because of limitations in the classad language. Classads cannot adequately handle attribute pairs. There is no method for finding a particular classad in a nested classad that contains these attribute pairs. In addition, there is no method for obtaining the index number of an element in a list. The latter would allow for parallel arrays where related attributes values would be accessible under the same array index value.

If a function that found the index of an attribute in a list was created, they could be used in the following manner:

Job Manager Classad

```
[
        Subject = "/O=Grid/O=Globus/OU=mcs.anl.gov/CN=Christopher Nebergall ";
        executables = {"/bin/ls","/bin/echo","/bin/cat"};
        arguments = {"-[al]","[a-z,A-z \t \n]*]","[a-zA-z~\.][a-zA-Z~\. ]"};
        Index = IndexOf(executables,other.dyamic.executable)];
        Requirements = Other.dynamic.subject = Subject && Member(executables,
              Other.dynamic.executable) &&
                      Regexp*(other.dynamic.arguments,arguments[Index]);
].
```

*Regular expressions are being used for representing the allowed arguments. This operation is supported in the current implementation of classads.

If a function was created that was able to find a classad nested inside a second classad the following more elegant layout would be possible.

Job Manager Classad

```
[
          Subject = "/O=Grid/O=Globus/OU=mcs.anl.gov/CN=Christopher Nebergall";
          execList =
          {
                    [exec = "/bin/ls";
                     args = "[-al]*"],
                    [exec = "/bin/echo";
                     args = "[a-z,A-z \t \n]*]"]
          }
]
```

As listed above in the document another suggest addition to the classad language is a subset function. This operation would allow another way for a site to determine whether intermediate sites are acceptable to the server.

Gatekeeper Classad

```
[
          IntermediateSites =
          {
                    "/O=Grid/O=Globus/CN=pitcairn.mcs.anl.gov",
                    "/O=Grid/O=Globus/CN=moleman.mcs.anl.gov"
                    "/O=Grid/O=Globus/CN=yukon.mcs.anl.gov"
          }         requirements = Subset(IntermediateSites,other.dynamic.IntermediateSites)
]
```

Client Classad

```
[
          ....
          dynamic =
          [
                    intermediateSites =
                    {
                              "/O=Grid/O=Globus/CN=pitcairn.mcs.anl.gov",
                              "/O=Grid/O=Globus/CN=moleman.mcs.anl.gov",
                    }
          ]
]
```

Another useful addition to classads would be if a new version of the Member function that took a string and matched it against a list of regular expressions. It would prevent users from having to list each specific machine name in a given domain.

Client Classad

```
[
          OperateFrom = {"[a-zA-z0-9.]*.mcs.anl.gov", "[a-zA-z0-9]*.isi.edu"};
          requirements = MemberRegexp(dynamic.hostname,OperateFrom);
]
```

<u>**Work that can be carried upon the present work:**</u>

- The gatekeeper should be extended to read a list of classads from a file for the purpose of replacing the gridmap file.

- More attributes representing further fine-grained security policies can be identified and added (on either the client side or server side or both).
- The job-manager capabilities can be extended so as to restrict even the operations on files requested, authorize requests against job-status request/cancel and so on.
- An authorization server could be set up for a set of services so that this authorization could be further simplified without each individual server needing to know about the restrictions.

Conclusion

If an adequate policy language is used it is possible to place very tight restrictions on credentials and their use. These restrictions make it possible to limit when, where, and how proxy credentials can be used. Because of these new restrictions, it is possible make distributed computing even more secure.

Classified advertisements would be an excellent mechanism for implementing security policies into Globus. They allow for greater control of security restrictions than is currently possible. Several new functions need added to classads to make them more suitable for implementing security concerns, but overall they are well suited for the task.

References

Neuman, B. Clifford. *Proxy-Based Authorization and Accounting for Distributed Systems.* In Proceedings of the 13th International Conference on Distributed Computing Systems, pages 283-291, May 1993

Raman, Rajesh, *ClassAds Programming Tutorial (C++)*, http://www.cs.wisc.edu/condor/classad/c++tut.html, June 2000

Raman, Rajesh, Miron Livny, and Marvin Solomon, *Matchmaking: Distributed Resource Management for High Throughput Computing* Proceedings of the Seventh IEEE International Symposium on High Performance Distributed Computing, July 28-31, 1998, Chicago, IL.

Acknowledgements

We thank the following people in the Distributed Super Computing Lab, MCS Division Argonne National Laboratory for their priceless support and guidance during the course of our work:

Douglas E Engert, Steve Tuecke, Ian Foster, Gregor Von Lazsweski, Darcy Quesnel, Stuart Martin, Jarek Gower.