

Authorizing Remote Job Execution based on Job Properties

Sang-Min Park, Glenn Wasson, and Marty Humphrey

Department of Computer Science, University of Virginia, Charlottesville, VA USA
{sp2kn, gsw2c, humphrey}@cs.virginia.edu

Abstract: *E-Science often requires access to remote Grid computing platforms. Current authorization systems on these remote systems have largely based decisions solely on the identity of the submitter -- the job is permitted to execute on the local resource if the job originates from an authenticated and authorized end-user. The problem with this approach is that there is no consideration to what the job will/should do when executed, so an errorful or malicious job -- even from what purports to be a trusted user -- can create significant damage before an operator notices and can kill or suspend the job. This paper presents a novel end-to-end job execution framework in which the properties (behavior) of the job are taken into account for the authorization decision. Experimental results show the duration to perform the authorization and to establish a subsequent restrictive execution context is sufficiently low -- our observed overhead of 253.1 ms on commodity hardware is an acceptable cost for most Grid applications to pay to achieve this more secure execution environment.*

1. Introduction

Remote job execution is one of the most important use cases for e-science and the Grid and therefore authorization is one of the most critical issues facing Grid system designers and implementers. Current authorization and account provisioning systems [2][4][6][11][12][18][24] have largely based decisions on the *identity* of the job submitter -- if the job submitter is authenticated and a member of the set of authorized users, then the job is permitted to execute under a pre-established account. Every job by a particular user is given the *same* set of permissions irrespective of the job's anticipated behavior or requirements.

This approach, while relatively easily implemented, is limited in three interrelated ways. First, policy-based provisioning and control (e.g., using a sandbox to enforce that "no program that reads/writes from the */home/biodata* file system is allowed to open a socket") is difficult to achieve based on the submitter's identity

alone. Second, the job submitter's identity is often a secondary concern -- resources are often provisioned and pre-allocated for specific purposes (such as "35% of the cluster should be used to run Matlab"), not for specific users (particularly for large VOs). Third, giving each job the same privileges fails to achieve the critical security principle of "least-privilege", which is to give someone or something only those rights that are truly necessary in order to complete a task or job. An errorful or malicious job -- even from what purports to be a trusted user -- can create significant damage before an operator notices and can kill or suspend the job!

More informed authorization and provisioning decisions can be made based on a combination of the job submitter's identity and "job properties". Examples of such job properties include: "The application to be executed is Matlab", "The application will write files only under \$HOME", "The binary may open sockets to 'cs.virginia.edu', but no where else", and "The binary requires 100 MB of memory". Job properties can be asserted by the end-user, asserted by the VO to which the user belongs, or determined through static or dynamic analysis by the end system that is performing the authorization and provisioning decision. A representative, end-to-end scenario is as follows:

1. A client, Fred J. Smith, wishes to submit a job (Matlab) to a particular Grid resource.
2. The Virtual Organization (VO) knows the properties and behaviors of Matlab when used for the purposes of the VO. For example, it will run at least 10 hours on a single machine, require 20 MBs of memory, write files only in job's working directory, and etc.
3. Fred J. Smith retrieves an assertion from the VO regarding this use of Matlab.
4. Fred J. Smith submits the job request along with this VO job property assertion to the particular Grid resource.
5. The resource uses the <DN, Job property> evidence pair for authorization.
6. Additional policies, including the site policy, may be used to determine whether to grant the

necessary permissions requested (as denoted by the job's properties).

7. The job properties are translated into the appropriate fine-grained permissions on the resource and the job is started with that permission set, thereby running with fewer rights than any existing general-purpose approach.

The scenario above has four important components. The first is the language for describing relevant job properties. The second is the mechanism by which the system executing the job retrieves these property descriptions (possibly from job submitters, possibly from the executor's own storage system). The third is the mechanism by which the actual permissions for the job on the local resource are resolved from the job property description and the potentially multiple policies that govern resource utilization (for example, the local site policy and the virtual organization policy). The fourth is the mechanism that enforces those permissions on the executing job.

The contribution of this paper is a general-purpose architecture and implementation of a Grid authorization system based on a combination of identity and job properties. To address the first two requirements above, we modify the Community Authorization Service, CAS [11], to create and encode SAML [21] assertions regarding job properties on behalf of the VO. The final two aspects, establishing permission sets based on multiple policies and subsequently enforcing those permissions are implemented via a novel use of .NET Common Language Runtime (CLR) sandbox capability in the context of remote job execution using WSRF.NET [26][28]. The CLR sandboxing capabilities are similar in principle to Java's sandboxing capabilities and our system could, in the future, be utilized for JVM-based remote execution as well. Experimental results show the duration to perform the authorization and to establish a subsequent restrictive execution context is sufficiently low – our observed overhead of 253.1 ms on commodity hardware is an acceptable cost for most Grid applications to pay to achieve this more secure execution environment.

While authorization based on identity alone will remain sufficient in many situations, there are situations in which specific properties of the job matter as much or more than properties of the job submitter. Our approach is applicable for a wide class of emerging Grid applications and facilitates finer control of system resources in a general framework --- thus better preventing accidental/malicious use of Grid resources --- than is possible using existing approaches.

The rest of paper is organized as follows. Section 2 describes related work. In Section 3, we describe our

remote execution framework which implements authorization based on job properties. Section 4 provides a discussion and evaluation of the developed system. Section 5 concludes the paper.

2. Related Work

Work in both grid authorization systems and virtual machines and sandboxing are relevant to this project.

2.1. Grid Authorization Systems

In the majority of Grid deployments today, authorization is based solely on the identity of the requestor. In this so called *grid-mapfile* approach, the X.509 DN is mapped to a specific account on the target machine. Unfortunately, its static nature causes significant administration overhead because the administrator of a Grid site must be involved in the creating and maintenance of the mapping between DNs and local user accounts. This has become a significant factor in limiting VO scalability.

More advanced approaches have been taken in recent research on VO authorization infrastructures such as CAS [11], VOMS [2], and PRIMA [18]. These infrastructures use one or more policy decision points (PDP) wherein policy information regarding a community member (often identified by DN) is determined. For example, the resources in a VOMS-based Grid can access group/role information about a user (in addition to their DN) when determining the account under which a job will run, and thus the permissions the job will have. However, the policy enforcement point (PEP), typically the resource itself, makes the decision on how to actually enforce the decisions made by the PDP. This means that the enforcement capabilities of the PEP effectively determine the granularity with which permissions can be granted by the PDP. Even if the PDP is willing to grant fine-grained permissions (e.g., access control lists to individual files), it is not applicable if the actual resource enforcing the policy cannot recognize and enforce the same permission granularity! Thus many policy systems including CAS, Akenti [24], and PERMIS [6] are only used with specific applications that are designed and developed as the enforcement point of policy system (e.g., GridFTP with CAS). However the remote job execution domain requires an enforcement mechanism that is applicable to general binaries submitted by the user.

The approaches that address the enforcement of dynamic and fine-grained privilege for remote execution can be classified into three categories:

1. **Dynamic Account** [14][16][17][18]: In this approach, the account is dynamically created or leased from the account pool, rather than

determined by the static one-to-one mapping based on the user's DN. After making a policy decision, an account is either newly created or leased from the account pool in a way that matches the policy as much as possible (for example, if the PDP determines that the user is in high priority group, a highly privileged account would be leased from the pool). This approach allows the use of dynamic policy, but does not allow fine-grained policy (without creating an account pool containing accounts with all possible privilege sets).

2. **OS-level Virtual Machine** [10]: In this approach, a job runs on a newly instantiated "guest Operating System (OS)", and from the security perspective, the guest OS (such as an instantiation of Xen) becomes the sandbox that isolates the job from the host OS. While the authorization policy can play a part in the customization of the guest OS, the granularity of the customization remains an issue. Also, the performance overhead (especially the instantiation delay of the guest OS) is an issue.
3. **Application-level Sandbox** [4][5][23]: This technique relies on an agent that examines the system calls invoked by the submitted job. A system call can be simply rejected or redirected to routines that enforce fine-grained policy. However, intercepting system calls incurs significant performance overhead. Also, since a particular system call may be allowable with certain parameters and disallowed with other parameters, and because a particular activity to be controlled is often achievable via multiple individual system calls, this approach can be difficult to constrain and/or configure.

2.2. Virtual Machines and Sandboxing

In order to understand the VM technology used in our authorization system, we describe the base sandboxing capabilities (in Table 1) of the .NET VM (CLR) [1] and the JVM [15]. Note that these capabilities are different than OS VMs such as Xen and VmWare, which construct complete operating system stacks as opposed to a more light-weight virtual machines.

Table 1. Resources Sandboxed in Virtual Machines

VM	Resources
.NET CLR	Execution(whether allowed to execute), File, Socket, DNS, Web, Database, MessageQueue, Environment variable, Registry, Thread, Calling Unmanaged Code, Control of AppDomain
JVM	File, Socket, Network, GUI, Runtime (configuration of runtime), Database, SSL

The operation of the sandbox mechanisms in the .NET CLR and the JVM are similar. As the .NET and Java sandboxes are motivated by the execution of mobile code, such as the applet [19], the default set of permissions granted is based on the "identity" of code (e.g., the author of the code, the location from which the code originated, etc). The runtime security manager reads policy information which maps the identity of the code to a set of permissions and grants the mapped permissions to the container in which the code runs.

3. Job Property Authorization System

In this section, we present the job execution framework and the authorization subsystem. Authorization and provisioning is based on *who* is doing the requesting and *what* is requested, rather than simply who is doing the requesting. The system is built on the .NET Framework and WSRF.NET, and leverages Globus CAS, GSI, GRAM, and the SAML protocol.

Figure 1 presents the architecture and the sequence of interactions between the four major components during the lifetime of the job: the VO Authorization Service, the Client, the Remote Execution Service (GRAM.NET), and the Managed Execution Service (MES.NET). In the subsequent sections, we present the design of each component in detail and explain how we address the four identified problems for the job property authorization (i.e., the language for expressing the job properties, the secure delivery of job properties, resolution of multiple governing policies, and the fine-grained enforcement).

3.1. VO Authorization Service and Client

As shown in steps 1-3 of Figure 1, the VO Authorization Service is responsible for asserting requirements or behavioral constraints on applications and/or users in the VO, manifested as job properties. Although the procedure by which the VO determines the property of job is important, we do not explicitly address this issue in this paper. The VO administrator defines the properties of various applications and adds them to the authorization server. The client, a member of the VO, fetches the job property to which he/she is entitled to get from the server.

The first two problems, a language for expressing job properties and the secure delivery of job properties, are addressed via the novel use of CAS. CAS, the community authorization server, is a service by which fine-grained permission sets are maintained by the VO and retrieved by community members. To date, CAS's main use has been the GridFTP server, but here we extend it to support job-property authorization. More specifically, we use a SAML assertion to express an application's properties, and the client's GSI proxy

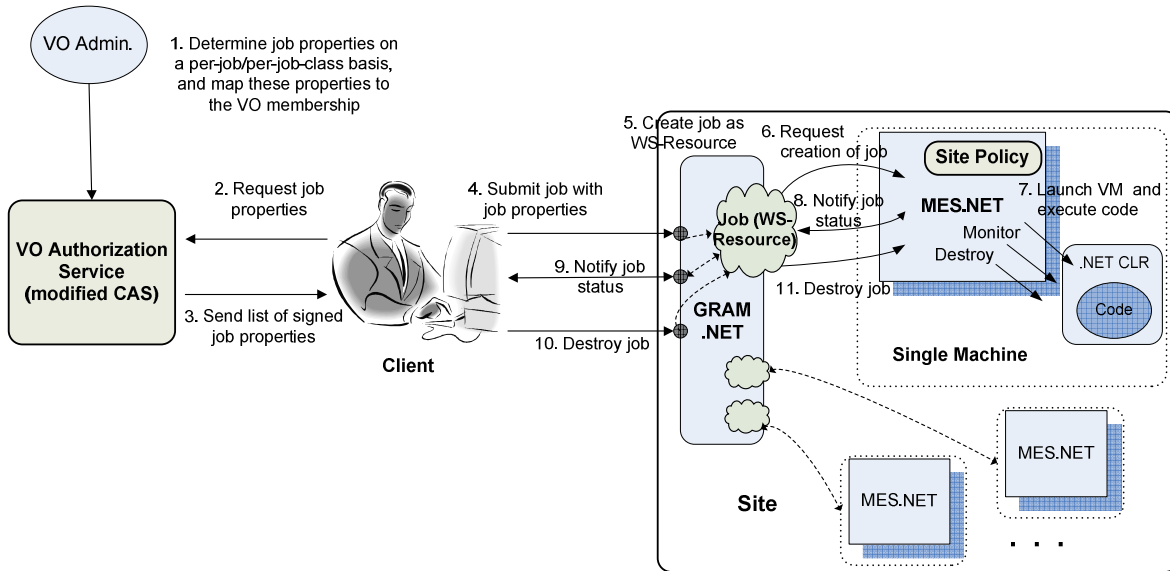


Figure 1. Remote Job Execution Framework

```

1. <Assertion xmlns="urn:oasis:names:tc:SAML:1.0:assertion" AssertionID="788c9180-3c1c-11da-8ad4-
a32bf1308618" IssueInstant="2005-10-13T19:06:34Z" Issuer="C=US,ST=Virginia,L=Charlottesville,O=University
of Virginia,,CN=PKI Master" MajorVersion="1" MinorVersion="0">
2. <Conditions NotBefore="2006-4-8T19:06:34Z" NotOnOrAfter="2006-4-10T19:06:34Z"></Conditions>
3. <AuthorizationDecisionStatement Decision="Permit" Resource="http://sangmin.cs.virginia.edu/gram">
4. <Subject>
5. <NameIdentifier Format="#X509SubjectName"/>C=US/O=University of Virginia/OU=UVA Standard PKI
User/E=sp2kn@virginia.edu/CN=Sang-Min Park</NameIdentifier>
6. <SubjectConfirmation><ConfirmationMethod>urn:oasis:names:tc:SAML:1.0:am:X509-
PKI</ConfirmationMethod></SubjectConfirmation>
7. </Subject>
8. <Action Namespace="http://cs.virginia.edu/gcg/authorization">write/ WORKING_DIR</Action>
9. <Action Namespace="http://cs.virginia.edu/gcg/authorization">read/WORKING_DIR </Action>
10. <Action Namespace="http://cs.virginia.edu/gcg/authorization">execution/WORKING_DIR/bin</Action>
11. <Action Namespace="http://cs.virginia.edu/gcg/authorization">socket/cs.virginia.edu</Action>
12. </AuthorizationDecisionStatement>
13. </Assertion>

```

Figure 2. Example SAML Assertion for Job Property Authorization

certificate is used to securely hold and deliver the SAML assertion to the resources on which the client wishes to run that application.

In CAS, the Security Assertion Markup Language (SAML) is the OASIS standard that defines the format of the security assertion in which statements about authentication and authorization decisions are expressed. The major design goal of SAML is Single-Sign-On, meaning a subject authenticated by one domain, may use resources in other domain without re-authenticating. Figure 2 shows an example of how we express job properties using SAML. Lines 1 and 2 define the assertion issuer and conditions under which

the assertion applies. Line 3 shows that the target resource for which this authorization decision is made is the remote execution service at <http://sangmin.cs.virginia.edu/gram>. Lines 5-6 describe the subject to whom the job property assertion is issued, and how the subject is confirmed. Lines 8-11 illustrate the job properties that the VO has issued to the client (for the execution of a particular job). The first two properties describe the permission to read and write files under a particular directory (expressed as 'WORKING_DIR' which is to be interpreted by the resources). The third property specifies that the client is able to execute only binaries under

'WORKING_DIR\bin'. The last property indicates that the job may setup socket connection to hosts in a particular domain.

3.2. Remote Execution Service (GRAM.NET)

The purpose of the GRAM.NET [27] component is to provide a standards-based remote execution service that leverages the .NET Framework (via WSRF.NET). We developed GRAM.NET to interoperate with GT4 GRAM, which is built on Web services and the concept of "stateful resources" rendered by WSRF/WS-Notification specifications [7]. GRAM.NET services are interoperable with GT4 GRAM -- that is, clients can submit, monitor, and terminate the job using the GT4 GRAM tools. Clients are authenticated by GRAM.NET using GSI proxy certificates.

There are two services in GRAM.NET: the ManagedJobFactoryService (MJFS) and the ManagedExecutableJobService (MEJS). The client submits a job by calling the MJFS::createManagedJob method. The job is represented by a *managedJobResource*, which consists of the job's status, owner, path, etc., and which is stored in an XML database managed by WSRF.NET. It is the MEJS's role to tightly link the information stored in the *managedJobResource* with the actual running process. Thus, if the job's state changes, these changes are sent to the MEJS by MES.NET (described below) so that the MEJS can update the fields of the *managedJobResource* accordingly. The final step in job creation is for the MEJS to extract the job property document from the client's proxy certification and send it to MES.NET which will appropriately configure a VM and execute the job (Step 7 of Figure 1).

3.3. Managed Execution Service on .NET (MES.NET)

MES.NET is the container that spawns and manages job processes in our framework. It spawns jobs on the virtual machine (CLR), thus facilitating fine-grained control over the privileges of remotely-executed applications. In the .NET security model, the Application Domain (AppDomain) provides isolation between managed codes and is the basic protection boundaries between tasks (it is analogous to the memory segment of a process without the expense of process creation) [8]. In MES.NET, before running the submitted code, we setup the AppDomain with permissions determined by the process of policy resolution explained below. The code (.EXE and required .DLLs) is loaded into the AppDomain and gets executed after the setup. The AppDomain is dismantled upon completion of the job.

3.4. Policy Resolution and Enforcement

One of the key challenges in our authorization architecture is the process by which the actual rights granted to the sandbox are determined from the job properties and multiple policies that govern the resource usage in the site. As shown in Figure 1, we recognize and support that the site may have a policy separate from the VO's policy. An example of a site policy is shown in Figure 3.

```

<VO Name="VO1">
  <JOB Name="Matlab">
    <MaxJobRunningTime> 100 </MaxJobRunningTime>
    <File Action="Read">$VO_BASE_PATH </File>
    <Socket> Yes </Socket>
    <Thread> Yes </Thread>
  </JOB>
  <JOB Name="Blast"> ...
</JOB>
</VO>

```

Figure 3. Representative Site Policy

The site policy is defined on a per VO basis, since a site can be a member of multiple VOs. In each VO, there are policies per job. Each policy, defined by the <VO, JOB> pair, defines the permissions that the site grants to the job. The site takes the priority of both VO and job into account when it makes the policy decision.

The next step is to resolve permissions from multiple governing policies. The authorization system is given the job property document and the site policy. Figure 4 illustrates the permission resolution process. When the job property document is given, the authorization component of MES.NET maps the entries of the document to the equivalent permissions used in the component by looking in the job property mapping table. This table converts various statements in the job property documents into equivalent VM permissions used at the site.

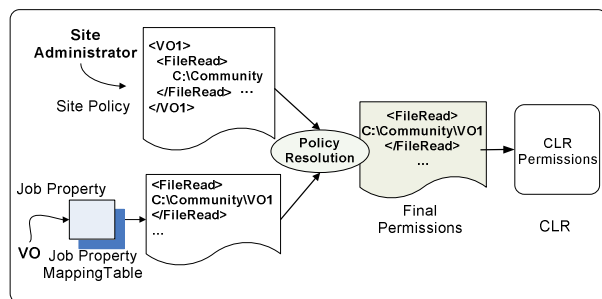


Figure 4. Policy Resolution/Enforcement in the MES.NET Authorization Component

<code><Site-Policy vo="VO1", job="Matlab"></code>	<code><Job-Property name="Matlab"></code>	<code><Final-Permission></code>
<code><Read>C:\</Read></code>	<code><Read>C:VO1</Read></code>	<code><Read>C:VO1</Read></code>
<code><Write>C:\VO1</Write></code>	<code><Write>C:VO1\Matlab</Write></code>	<code><Write>C:VO1\Matlab</Write></code>
<code><Socket> Grant </Socket></code>	<code><Write>C:\Temp</Write></code>	<code><Socket> Grant </Socket></code>
<code><Thread> Deny </Thread></code>	<code><Socket> Grant </Socket></code>	<code><Thread> Deny </Thread></code>
<code>...</code>	<code><Thread> Grant </Thread></code>	<code>...</code>
<code></Site-Policy></code>	<code></Job-Property></code>	<code></Final-Permission></code>

Figure 5. Example Policy Resolution

The resolution of the site policy and the job properties are the basis for making the decision whether to execute the job or not. In order to preserve site autonomy, the job must not be given privilege beyond what is granted by the site. If the site policy does not grant the permissions requested in the job properties, it is likely (although not definite) that the job will not be able to complete its task under the restricted configuration. When this occurs, there are two possible options - immediately reject the job or let the job run and see if it actually requests a forbidden action. In our current system, we take the second option, since determining if the job properties are accurate (and not, say, overly pessimistic) is the subject of continuing research. To ensure least-privilege, the combined policy is the intersection of the site policy and the job properties. Figure 5 shows an example of this process. Note that there is no explicit "VO policy" as the VO policy is used by the VO authorization service and reflected in the job property document. In the example, the site grants the <VO1, Matlab> read permission of 'C:\' and write permission of 'C:\VO1'. The Job property describes that this job is likely to read from 'C:\VO1', and write to 'C:\VO1\Matlab' and 'C:\Temp'. The final permissions after the integration are read permission of 'C:\VO1', and write permission of 'C:\VO1\Matlab'.

The last aspect of the job property authorization system is the mechanism by which the actual enforcement of permissions is achieved. For this, the abstract permissions used in the authorization component are translated to the actual .NET permission objects and the CLR sandbox is configured in a way that enforces the permissions. In this case, any attempt to write files under 'C:\Temp' by the executable is rejected by the CLR sandbox.

4. Evaluation and Discussion

To evaluate the run-time overhead of our system, we executed "Hello World", utilizing the site policy shown in Figure 6 and the Job Properties shown in Figure 7.

We chose these configuration parameters in the policy language in Figure 6 in part because of the ease with which they can be mapped to CLR capabilities; we are in the process of evaluating the applicability of these parameters as a sufficient set for the wide class of Grid applications. The job properties, expressed via SAML assertions, are shown in Figure 7. The resulting configuration of the CLR is shown in Figure 8.

<code><MaxJobRunningTime</code>
<code>Type="Number">100</MaxJobRunningTime></code>
<code><File Type="FilePath" Action="Read">\$VOBasePath</File></code>
<code><File Type="FilePath" Action="Write">\$VOBasePath</File></code>
<code><DatabaseConnection</code>
<code>Type="Bool">NO</DatabaseConnection></code>
<code><Diagnostics Type="Bool">No</Diagnostics></code>
<code><DirectoryService Type="Bool">NO</DirectoryService></code>
<code><Printing Type="Bool">No</Printing></code>
<code><SocketConnection Type="Bool">Yes</SocketConnection></code>
<code><Dns Type="Bool">Yes</Dns></code>
<code><Web Type="Bool">No</Web></code>
<code><MessageQueue Type="Bool">No</MessageQueue></code>
<code><EnvironmentVar Type="Bool">Yes</EnvironmentVar></code>
<code><FileDialog Type="Bool">No</FileDialog></code>
<code><IsolatedStorage Type="Bool">No</IsolatedStorage></code>
<code><Reflection Type="Bool">No</Reflection></code>
<code><Registry Type="Bool">No</Registry></code>
<code><ControlAppDomain Type="Bool">No</ControlAppDomain></code>
<code><ControlEvidence Type="Bool">No</ControlEvidence></code>
<code><ControlPolicy Type="Bool">No</ControlPolicy></code>
<code><ControlPrincipal Type="Bool">No</ControlPrincipal></code>
<code><ControlThread Type="Bool">Yes</ControlThread></code>
<code><Execution Type="Bool">Yes</Execution></code>
<code><CallUnmanagedCode</code>
<code>Type="Bool">Yes</CallUnmanagedCode></code>
<code><SkipVerification Type="Bool">No</SkipVerification></code>

Figure 6. Site Policy for "Hello World" Example

```

<Action>read/VO_APPLICATIONS_PATH</Action>
<Action>read/VO_LIBRARY_PATH</Action>
<Action>read/VO_SHARED_PATH</Action>
<Action>read/VO_TMP_PATH</Action>
<Action>read/VO_UTILS_PATH</Action>
<Action>read/VO_BIN_PATH</Action>
<Action>read/DEFAULT_WORKING_DIR</Action>
<Action>write/DEFAULT_WORKING_DIR</Action>
<Action>write/VO_SHARED_PATH</Action>
<Action>write/VO_TMP_PATH</Action>
<Action>socket</Action>
<Action>execution</Action>
<Action>registry</Action>
<Action>db_connection</Action>
<Action>call_unmanaged_code</Action>
<Action>environment_var</Action>

```

Figure 7. Job Properties for "Hello World" Example

```

<IPermission
class="System.Security.Permissions.EnvironmentPermission"/>
<IPermission class="System.Net.SocketPermission"/>
<IPermission class="System.Net.DnsPermission/>
<IPermission
class="System.Security.Permissions.SecurityPermission
Flags="Execution"/>
<IPermission
class="System.Security.Permissions.FileIOPermission,
Read="C: \Community\VO1\Applications; C: \Community\VO1;
C: \Community\VO1\Users\Sang-Min Park" C:
\Community\VO1\Utils;
Write="C: \Community\VO1; C: \Community\VO1\Users\Sang-
Min Park" C: \Community\VO1\Users;/>

```

Figure 8. Resulting Configuration of CLR for "Hello World" Example

We measured the execution time of our authorization system as well as the baseline case in which the same job was executed without the policy resolution and sandbox configuration of the CLR. The server has two AMD Opteron 240 CPUs and 2G memory, and runs Windows Server 2003 and .NET Framework 1.1. Table 2 shows the results averaged over 50 runs. The column, GRAM.NET, represents the time that is taken inside the two GRAM.NET Web services (MJFS and MEJS). CLR creation time represents the time that is taken to invoke new CLR process in order to run the user's submitted job. It should be noted that it consumes the majority of the execution time in both cases. Note that policy resolution time and CLR sandbox configuration time are present only in the authorization system case, and they constitute the majority of the overhead caused by our authorization system. As shown in the table, the overhead of the system is 953.1 ms - 700.6 ms = 253.1 ms. We believe that this overhead is acceptable for all but trivial jobs and is justified in the enhanced control

and security this system provides. We assert that even as policies become more complex and realistic, this overhead will remain small (and caching of policy resolutions is certainly possible if needed).

Table 2. Running time of Authorization System for "Hello World" Example

	No Authorization System (ms)	Authorization System (ms)
GRAM.NET	62.2	76.7
CLR Creation Time	619.0	650.8
Policy Resolution Time	-	43.1
Sandbox Configuration Time	-	176.6
Etc	19.4	5.9
Total	700.6	953.1

5. Conclusion

To date, authorization and provisioning have relied too heavily on the identity of the job submitter. In this paper, we have presented an authorization system based on a combination of job submitter identity and job properties. We adopt existing security protocols to express and manage VO-wide job properties, develop the policy resolution process abiding by the least-privilege principle, and exploit the .NET Framework CLR VM sandbox to dynamically enforce the fine-grained permissions resolved from the property and policy. Experimental results show tolerable overhead of 253.1 ms on commodity hardware, which we believe is acceptable for all but trivial Grid and e-science application scenarios.

In the future, we will address numerous issues, particularly the language framework to provide general and abstract syntax and semantics for resource providers and communities to describe the properties of the job independent of actual enforcement mechanisms.

References

- [1]. Adam Freeman and Allen Jones. Programming .NET Security. O'Reilly, June 2003.
- [2]. Alfieri, R., et al. VOMS, an Authorization System for Virtual Organizations. First European Across Grid Conferences. Santiago de Compostela, Spain, Feb. 14 2003.
- [3]. Bull, J., Smith, L., Pottage, L., and Freeman, R. Benchmarking Java against C and Fortran for

- scientific applications. *Java Grande 2001*: 97-105
- [4]. Butt, A., Adabala, S., Kapadia, N., Figueiredo, R., Fortes, J. Fine-Grain Access Control for Securing Shared Resources in Computational Grids. 2002 International Parallel and Distributed Processing Symposium (IPDPS).
 - [5]. Calder, B., Chien, A., Wang, J., and Yang, D. The Entropia Virtual Machine for Desktop Grids. In *Proceedings of the First ACM/USENIX Conference on Virtual Execution Environments (VEE'05)*, June 2005.
 - [6]. Chadwick, D., and Otenko, O. The PERMIS X.509 role based privilege management infrastructure. *Future Generation Computer Systems*, 19(2):277-289, February 2003.
 - [7]. Czajkowski, K., Ferguson, D., Foster, I., Frey, J., Graham, S., Sedukhin, I., Snelling, D., Tuecke, S., Vambenepe, W. 2004. The WS-Resource Framework. <http://www-106.ibm.com/developerworks/library/ws-resource/ws-wsrf.pdf>
 - [8]. Demien Watkins. An Overview of Security in the .NET Framework. <http://msdn.microsoft.com/security/securecode/default.aspx?pull=/library/en-us/dnnetsec/html/netframesecover.asp>
 - [9]. ECMA. Standard ECMA-335 Common Language Infrastructure (CLI) 3rd edition. June 2005. <http://www.ecma-international.org/publications/files/ECMA-ST/Ecma-335.pdf>
 - [10]. Figueiredo, R., Dinda, P. A Case for Grid Computing On Virtual Machines. *Proc. International Conference on Distributed Computing Systems (ICDCS)*, May 2003.
 - [11]. Foster, I., Kesselman, C., Pearlman, L., Tuecke, S., and Welch, V. The Community Authorization Service: Status and Future. In *Proceedings of Computing in High Energy Physics 03 (CHEP '03)*, 2003.
 - [12]. Foster, I., Kesselman, C., Tsudik, G., Tuecke, S. A Security Architecture for Computational Grids. *Proc. 5th ACM Conference on Computer and Communications Security Conference*, pp. 83-92, 1998.
 - [13]. Globus Toolkit. <http://www.globus.org>
 - [14]. <http://www.gridsite.org/gridmapdir/>
 - [15]. Java Security Overview. April 2005. http://java.sun.com/developer/technicalArticles/Security/whitepaper/JS_White_Paper.pdf
 - [16]. Kapadia, N., Figueiredo, R., Fortes, J. Enhancing the Scalability and Usability of Computational Grids via Logical User Accounts and Virtual File Systems. 2001 International Parallel and Distributed Processing Symposium (IPDPS).
 - [17]. Kirschner, B., Hacker, T., Adamson, W., and Athey, B. WALDEN: A Scalable Solution for Grid Account Management. 5th IEEE/ACM International Workshop on Grid Computing (Grid 2004), 5 July 2004.
 - [18]. Lorch, M., Kafura, D., Fisk, I., Keahey, K., Carcassi, G., Freeman, T. Authorization and Account Management in the Open Science Grid. *Proceedings of the Sixth International Workshop on Grid Computing (GRID'05)*
 - [19]. Loureiro, S., Molva, R., and Roudier, Y. Mobile Code Security. In *proceedings of ISYPAR 2000 (4ème Ecole d'Informatique des Systèmes Parallèles et Répartis)*, Code Mobile, Toulouse, France, February 2000.
 - [20]. Mono project. <http://www.mono-project.com>
 - [21]. OASIS. Security Assertion Markup Language (SAML) 1.0 Specification. November 2002.
 - [22]. OASIS. eXtensible Access Control Markup Language (XACML). <http://www.oasis-open.org/committees/xacml>
 - [23]. Thain, D. Identity Boxing: A New Technique for Consistent Global Identity. In *Proceedings of Supercomputing*, Nov 2005.
 - [24]. Thompson, M., Johnston, W., Mudumbai, S., Hoo, G., Jackson, K., Essiari, A. Certificate-based Access Control for Widely Distributed Resources. *Proceedings of the Eighth USENIX Security Symposium (Security '99)*, Washington, D.C., August 23-26, 1999, pp 215-227.
 - [25]. Vogels, W. HPC.NET - are CLI-based Virtual Machines Suitable for High Performance Computing? *Proceedings of the 2003 ACM/IEEE conference on Supercomputing*.
 - [26]. Wasson, G., and Humphrey, M. Exploiting WSRF and WSRF.NET for Remote Job Execution in Grid Environments. 2005 International Parallel and Distributed Processing Symposium (IPDPS 2005), Denver CO, April 4-8, 2005.
 - [27]. Watson, J., Park, S-M., and Humphrey, M. Toward GT3 and OGSI.NET Interoperability: GRAM Support on OGSI.NET. 2005 International Conference on Computational Science (ICCS 2005), May 22-25, 2005. Emory University, Atlanta, GA, USA.
 - [28]. WSRF.NET. <http://www.ws-rf.net>
 - [29]. UVa Campus Grid. <https://uvacg.cs.virginia.edu/>