Preliminary version; to appear in 7th IEEE/ACM International Conference on Grid Computing (Grid 2006), Sept 28-29,2006, Barcelona Spain

Flexible and Secure Logging of Grid Data Access

Weide Zhang, David Del Vecchio, Glenn Wasson, and Marty Humphrey
Department of Computer Science, University of Virginia, Charlottesville, VA 22904-4740

Abstract -- In Grid collaborations, scientists use middleware to execute computational experiments, visualize results, and securely share data on resources ranging from desktop machines to supercomputers. While there has been significant effort in authentication and authorization for these distributed infrastructures, it is still difficult to determine, post-facto, exactly what information might have been accessed, what operations might have occurred, and for what reasons. To address this problem, we have designed and implemented a secure logging infrastructure for Grid data access. We uniquely leverage and extend XACML with new capabilities so that data owners can specify logging policies and these policies can be used to engage logging mechanisms to record events of interest to the data owners. A case study based on GridFTP.NET is presented and analyzed, utilizing both local storage of log records and remote storage via SAWS, an independently developed secure audit Web service. We show that with relatively little performance overhead, data owners are provided with new flexibility for determining the post-facto conditions under which their Grid data was accessed.

1. Introduction

The technology of Grid computing has been successfully applied to leverage existing legacy systems to accomplish computing intensive tasks across different security domains. A typical scenario involves remotely executing jobs that consume a large number of data sets scattered among different domains. While today's grid systems typically meet the authorization requirements for accessing these data sets, there are also stringent requirements for auditing this data access. The data owner should be able to determine, at any point, all the possible ways his data was utilized and by whom. Such requirements are also important in the health care domain to enforce and comply with regulations, such as HIPAA [1]. Although general, system-level auditing tools such as SysLog and Windows Events Logging may be extremely useful in recording system events for later analysis, there is no way for a data owner to specify conditions in which he requires a log entry. One possible solution would be to hardcode the auditing policies into the application code; however, this approach is very inflexible and makes it difficult to accommodate the diversity of data owners and auditing policies that actually exist.

Our vision for a generic grid auditing infrastructure is to let the data owners participate directly in defining auditing policies for their data sets. These policies should let the data owner specify conditions that, if satisfied, result in logging of information about the current data access. This paper addresses not only the language used to specify auditing policy, but also the policy decision and enforcement mechanisms necessary to have a functioning system. It also discusses the use of an auditing infrastructure with legacy applications (i.e., can audit policies be specified independently of the semantics of the underlying application, can legacy applications be modified to enable the auditing of data accesses?). The target grid application we consider is GridFTP.NET, a Grid FTP tool developed by the Grid Computing Group at the University of Virginia [2].

The rest of this paper is organized as follows. Section 2 introduces related work in current auditing and logging systems and discusses the problems of these systems. Section 3 presents the overall system architecture of our general-purpose auditing infrastructure. Section 4 discusses a specific implementation of this architecture for data access auditing based on XACML and GridFTP.NET. Section 5 includes a qualitative and quantitative evaluation of auditing system to assess its successfulness in meeting the needs of data owners. In Section 6, we conclude with some future directions for this Grid auditing research.

2. Related Work

In current audit logging systems, the conditions under which events are logged are either defined by the operating system mechanisms or hard coded by the application developer. Two common system level audit logging systems such as syslog [3] on Unix/Linux system or the Event subsystem [4] on Microsoft windows are extremely useful but rely on the application developer to create auditing triggers. Similar situations occur with another useful distributed logging toolkit—NetLogger [5]. Although NetLogger is efficient and reliabile in collecting events from multiple distributed locations, it is again the application developer's responsibility to instrument an application to produce event logs utilizing the NetLogger API and send them to the NetLogger Server.

A persistent problem in state of art audit and logging systems is their lack of dynamicity, flexibility and support for delegation of authority in both the generation and the enforcement of logging conditions. Lack of dynamicity refers to the static nature of today's auditing policies

which must instead evolve over time. Lack of flexibility refers to the fact that it is currently difficult to modify, create or delete auditing policy and is similarly difficult to incorporate enforcement mechanisms. Delegation of authority means that (authorized) application users and not application writers should be allowed to determine policy. These issues are especially important in Grid computing, where resources are shared across security domains and where it is the data owners wish to monitor accesses to their data. System level logging is typically too voluminous and too costly in terms of performance for the auditing required by many data owners. Instead, we have created a general purpose auditing infrastructure that uniquely allows *data owners* to specify auditing policy data resources, such that the system is not forced to "record everything and sort it out later". We evaluate our infrastructure through integration with GridFTP.NET.

3. System Architecture

Our auditing policy architecture leverages the fact that any data consumer (whose data accesses would be audited) must first receive authorization to access the data. Typically, this is accomplished by sending an authorization request on behalf of the consumer to an authorization decision point. In our architecture, our approach is to co-locate the auditing component with the authorization decision point, allowing us to leverage the policy engine (e.g., an XACML engine) already in place for authorization.

3.1 Basic Components

Our auditing infrastructure consists of three primary components: the auditing decision and enforcement point (ADEP), the auditing service that stores logs created by the enforcement of auditing policy, and the auditing policy administration point through which the data owner specifies auditing policies. In addition, an auditing policy language is defined.

Auditing Decision and Enforcement Point. The Auditing Decision and Enforcement Point is used to evaluate the auditing policies for the requested data and if at least one of the policies specifies a condition that is true, then the enforcement point will generate the audit record and invoke the remote auditing service to store this record. When performing policy evaluation, the auditing decision point examines the current access request to see if any it conforms to any of the policy's elements. If there is more than one policy for a data set, the auditing decision point will traverse through all the policies for that data set to see whether the request should be audited or not.

Auditing Service. The Auditing Service is a web service that accepts and stores auditing logs generated by the auditing enforcement points. Such a service should meet several requirements. First, it should guarantee that the audit records it receives were sent by a trusted party and ensure that the confidentiality and integrity of the records are preserved during storage. Second, the service needs to remain accessible under heavy network load or perhaps certain levels of denial of service attacks. Service availability is important because were the auditing service to go down, the logs for detecting unwanted data accesses will never be recorded.

Auditing Policy Administrating Point. The auditing policy administration point provides the data owner with policy management features including the ability to define, update and delete audit policies. Clearly, the data owner(s) must be properly authenticated when logging into the administration point. "Attaching" these policies to their respective data resources can be complicated by a number of factors such as the number of data resources to which a policy applies and replication or migration of data sets. Our current mechanism relies on the ADEP sending the URL of the data to be accessed to the Administration point, which then sends back the appropriate policy. Other mechanisms are being examined and are the subject of future work.

Auditing Policy. Auditing policy is the focus of our current work. The main problem we want to solve is to define an auditing policy language so that data owners can express any number of complex conditions using the language. Unlike the auditing mechanisms developed in operating systems such as Windows 2000 [4], our auditing policy language is defined specifically to audit the data accesses of interest to the data owner. Thus if auditing is more data centric it becomes less specific to the application that is accessing the data. *This leads to our contention that the auditing policy will primarily need to concern itself with four primitive data access operations—create, read, update, delete, abbreviated as "CRUD".* In any grid application, there might be a variety of operations which might contain code to consume a particular data set. However, the data owner will likely only care about whether his data was accessed by any of the four basic operations since he or she doesn't know the details of the client's application-specific operations. In our policy language, we have defined five key attributes used to specify auditing conditions: Subject, Operation, Authorization, Time and Location. The possible values for the Operation attribute are one of the four primitive "CRUD" data access operations.

3.2 XACML

XACML is an XML-based access control policy language [6]. It includes a set of complex data types, functions and combining logic to allow detailed access control policies

to be constructed and evaluated. It also defines a request/response protocol for policy enforcement points to obtain authorization decisions from policy decision points.

Request/Response Protocol. The authorization decision request message contains the credential of the requested user, the intended actions, the requested resources and other environment variables. After the request is evaluated against the policies at the policy decision point, a response message will be constructed to indicate whether the access is permitted or not. This protocol is also applied to our auditing infrastructure where the auditing enforcement point will first construct a request message to the auditing decision point. After the request has been evaluated against the auditing policies, the decision point will return an auditing decision to the enforcement point indicating whether or not to log the access.

Auditing Policy Definition. The XACML policy definition language provides a variety of features that are best suited for defining access control policies. In our auditing infrastructure, we leverage XACML to define data access logging policies as well. In contrast to using XACML to construct access control policies, auditing policies are effectively defined in the "Condition" element of an XACML rule. The policies are basically a set of predicates whose truth value can be evaluated. There are also three logical operations (AND, OR, NOT) that can be applied to predicates. For example, we may want to log any data access request that involves an operation other than "read". Figure 1, which conforms to XACML 1.1 specification (with namespaces omitted or shortened), shows how this condition is expressed in XACML:

```
<Condition FunctionId="and">
  <Apply FunctionId="not">
    <Apply FunctionId="string-equal">
      <Apply FunctionId="string-one-and-only">
        <ActionAttributeDesignator
          AttributeId="action-id"
          DataType="XMLSchema#string"
          MustBePresent="false" />
      </Apply>
      <AttributeValue
        DataType="XMLSchema#string">
          read
        </AttributeValue>
    </Apply>
  </Apply>
</Condition>
```

Figure 1: Sample XACML Audit Policy

The element "Apply" indicates the proper function will be applied to the predicates inside the tags. The element "ActionAttributeDesignator" will be processed to retrieve the attribute value contained in the XACML request. This is the requested action name. The retrieved value will be compared to the "read" string specified in the following "Attribute Value" element. The comparison function specified in the outer "Apply" element has the function id of "urn:oasis:names:tc:xacml:1.0:function: string-equal". If these 2 strings are equal, then the function will return true; otherwise, it will return false. However, there is an unary logical operator "NOT" applied to the whole inner predicate expression, which will evaluate to true when the data owner's request action is not "read" and to false when the action is "read." This is the XACML formulation of conditions which can be made as complex as possible to capture the complex auditing requirements of the data owner.

3.3 System Operation

Our main focus of our auditing system is not to capture the application's internal behavior but for the data owner to trace the historical accesses of his data. So, rather than modifying the design of the grid application code to incorporate auditing functionality, we chose to co-locate the auditing functionality with the authorization decision point. So, a grid application already doing some form of authorization callout would require minimal changes to support automated logging via our auditing architecture.

Here is the basic flow of messages through our architecture, as shown in Figure 2:

1. The data consumer sends a request to access the data to the data custodian.
2. The authorization enforcement point for the data custodian' will first send an authorization request to the authorization decision point to determine whether or not the data access is permitted.
3. At the authorization decision point, the request will first be evaluated against the authorization policies to return an authorization decision.
4. Next, the authorization decision point will construct an auditing request context to be evaluated against relevant auditing policies.
5. Then, if the auditing decision evaluates to true, the authorization decision point will invoke a remote auditing service to log this specific access of the data.
6. After the logging has completed successfully, the authorization decision point will return the authorization decision to the authorization enforcement point.
7. The authorization enforcement point will return the requested data to the consumer depending on the authorization decision.

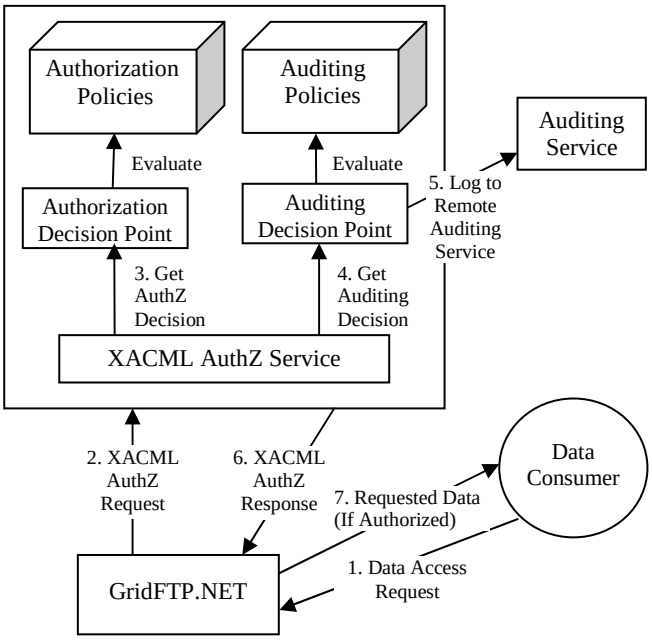

Figure 2: Overall Auditing Architecture

4. Design and Implementation

The fundamental and important requirements of generic auditing in information system security are discussed in [7]. To design and implement an effective data access auditing infrastructure requires a thorough understanding of the essential requirements of the current data grid domain. Current auditing policy requirements include logging who did what to the data and might be conditioning on other environmental variables such as the location of the data consumer, accessing time, authorization decision and also should facilitate the detection of unauthorized access.

Our target grid application is our .NET implementation of GridFTP called GridFTP.NET. GridFTP.NET acts as the data custodian which makes stored data sets available to remote consumers. It will send the data access requests to the authorization decision service for access control and logging as discussed above. We are currently investigating the application of this approach to the Globus-based GridFTP implementation.

4.1 Auditing Policy Leveraging XACML

The way we leverage XACML framework to define grid auditing policies is to allow data owners to specify each rule's target resource as one or more of his data sets and define the corresponding auditing policy in the Condition element. Since the auditing policies is evaluated on a per data access request basis, it is not necessary to specify the quantifier in the auditing policies.

The five attributes defined in our auditing policies are: Subject, Operation, Authorization, Time and Location. *Subject* identifies the data consumer who requests the access of the data. The subject can be a X.509 Distinguished Name or a role name. *Operation* is the type of data access that the data consumer invokes. As we discussed above, its value is indicates one of the "CRUD" operations. *Authorization* is a predicate condition whose value depends on the authorization decision. For example, the data owner may want to log the data accesses that were "Denied" rather than "Permitted." *Location* represents either the IP address or some Virtual Organization identifier for the data consumer. *Time* indicates the period over which the data owner wants access to his or her data sets access to be logged. For instance, the data owner might only need to audit the requests after 5pm.

4.2 Policy Administration Web Portal

Since auditing policies as represented in XACML could become quite complex and difficult to define correctly and accurately, we have automated the policy definition process by developing a web portal for the data owner to specify the policies for his data sets URL and formulate this into XACML policies. This policy administrating web portal provides an easy-to-use user interface, formulating the user specified propositional logic into XACML policy files. The portal allows the data owner to define components containing logical propositions and construct policies components these components.

For example, the user might want to specify the following policy:

((subject == X509CertDN) and
(operation name != read) or
(authorization decision == Permit) and
(location name =127.0.0.1))
and (time>5pm)

Using our web portal, the user could define "(subject= UVACG-X509CertDN and operation name! = read)" as component 1 and define "(authorization decision = Permit and location name =127.0.0.1)" as component 2 and combine them to be "((subject= UVACG-X509CertDN and operation name! = read) or (authorization decision = Permit and location name =127.0.0.1))" as component 3. And finally, component 3 will be combined with predicate "(time>5pm)" with AND operator. Thus, complex conditions can be built from through composition of simple prepositions.

4.3 XACML Auditing Evaluation and Enforcement

After the auditing policies have been defined by the data owner, it is the responsibility of the auditing decision and enforcement point to check whether a specific data access

request satisfies the auditing requirement. We utilized XACML.Net for the policy evaluation. The XACML.Net evaluation engine gets invoked twice, first to make an authorization decision and second to make an auditing decision.

4.4 Auditing Infrastructure based on GridFTP.NET

GridFTP.NET supports ftp commands such as "get, put, del, mkdir, create" for data download, upload, deletion and creation, etc. Since each command a data consumer executes invokes a method in GridFTP.NET, we modified GridFTP.NET to send an XACML authorization request to an authorization decision point before executing certain commands. After the authorization decision has been obtained, the auditing decision mechanisms will be triggered by the same XACML service. Figure 3 shows a typical XACML auditing request context.

```
<Request
xmlns="urn:oasis:names:tc:xacml:1.0:context">
  <Subject SubjectCategory="userid:dn">
    <Attribute AttributeId="subject-id"
               DataType="x500Name">
      <AttributeValue>/C=US/O=University of
Virginia/OU=UVA Standard PKI User/
emailAddress=wz6y@virginia.edu/CN=Weide  Zhang
5</AttributeValue>
    </Attribute>
  </Subject>
  <Resource>
    <Attribute AttributeId="resource-id"
               DataType="XMLSchema#anyURI">
      <AttributeValue>http://shandling.
           cs.virginia.edu/dataset1.txt
      </AttributeValue>
    </Attribute>
  </Resource>
  <Action>
    <Attribute AttributeId="action-id"
               DataType="XMLSchema#string">
      <AttributeValue>read</AttributeValue>
    </Attribute>
  </Action>
  <Environment>
    <Attribute
       AttributeId="locationid:ipaddress"
       DataType="XMLSchema#string">
      <AttributeValue>
        128.143.63.137
      </AttributeValue>
    </Attribute>
    <Attribute AttributeId="timeid:id"
               DataType="XMLSchema#time">
      <AttributeValue>
        05:00:00.0000000-05:00
      </AttributeValue>
    </Attribute>
    <Attribute
       AttributeId="authorizationid:id"
       DataType="XMLSchema#string">
      <AttributeValue>
        Permit
      </AttributeValue>
    </Attribute>
  </Environment>
</Request>
```

Figure 3: Sample XACML Auditing Request Context

The Subject, Resource, Action, Location, Authorization and Time elements in the Request have the same form and meaning as those defined in the auditing policy. It should be noted that originally GridFTP.Net sent an XACML authorization request containing the action of its requested command (get, mkdir, cd, etc.). While creating an auditing request, this command will be mapped to one or more CRUD operations (create, read, etc.). Currently, the mapping is defined statically in the auditing service, but one could imagine a separate library or web service which performs such application-specific mappings of commands to primitive "CRUD" operations.

4.5 Auditing Logs and Remote Auditing Service

Since our focus is the protection of data accesses, our auditing logs will only contain the information related to the data accesses. An audit record used for our specific purposes is defined to have the following form:

Subject Identity | *Operation Name* | *Data Set URL* | *Access Time* | *Access Location* | *Authorization Decision*

This is different from the typical form for event based auditing logs. This is because we only care only about a single event: the access of grid data sets. Thus, the content of our auditing log only reflects the information: *who* did *what* to *which* data sets *when* and *where*, along with the authorization decision on a requested access.

Our auditing enforcement component will send such logging information to a remote auditing service. This auditing service simply accepts logs in the above form and stores them locally. This auditing service will categorize the logs by the data sets' URLs. Thus, if the data owner wants to carry out post analysis on these logs, the auditing service will easily find the corresponding logs for his data sets. Auditing trail post analysis is another important issue in designing a good audit log analyzer. The purpose of auditing trail analysis is to "determine vulnerabilities, establish accountability, assess damage and recover the system" [8]. The issue of audit trail analysis in our system must be left to future work.

5. Evaluation

This section offers an evaluation of our work, demonstrating how our auditing infrastructure meets the data auditing requirements in of Grid computing. In particular we analyze the power, flexibility and extensibility of our auditing policy language and system even for a diverse set of data-centric applications. Furthermore, our framework is easily integrated with the XACML authorization component for Grid resource access control.

5.1 Policy Language for Expressing Auditing Requirements

Expressiveness for specifying auditing conditions. Our auditing policy language allows for attribute-based condition specification. Data access attributes could include the identity of the data consumer, the location where the data consumer comes from, the time when the data consumer accesses the data, the exact operation the data consumer invokes for this access and whether this request is authorized or not. The data owner can specify conditions in terms of any number of such attributes, joined together by conjunction, disjunction, and negation: AND, OR, NOT for the data owner to specify conditions. In this sense, almost anything expressible as an attribute can be used to make an auditing decision. Auditing criteria can be arbitrarily complex. For example, logging might only be triggered based only on the data consumer's location when the access time is between 8 am to 5 am, but based on both the data owner's location and identity when the access time is after 5 pm. In short, we believe data owners can specify logically complex propositions or conditions for their data sets.

Extensibility for auditing policy conditions. Since XACML policies can also accept custom defined attribute Identifications, we can extend our policies conditions to incorporate attribute types beyond the five attributes we described above. To support an addition attribute in auditing policies, we simply create a unique attribute id and use that id directly in the XACML policy file. When the engine is invoked for auditing policy evaluation, it simply extracts the attribute value from the XACML request context according to the attribute identification specified in the auditing policies. If the request context contains a matching value for that attribute identification, it will compare the value in the request element to the value in the auditing policies. Note that this is much simpler than extending the XACML policy specification to incorporate new internal logical functions or XML data types, but only works for simple attributes. If the attribute contains a complex data type, the data owner has to extend the support of his specific data type by changing the XACML implementation.

An example use of attribute extensibility might be to define attributes for Location, such as IP address, Virtual Organization name or domain name. Similarly, a subject identity might either be an X.509 Distinguished name or the Role one assumes in a specific collaboration. To incorporate such variety different attribute ids are used for each type.

5.2 System Flexibility and Scalability

We claim our auditing infrastructure is more flexible than traditional auditing infrastructure. By using an XACML policy construction and evaluation engine, we have an auditing infrastructure which makes it easy for the data owner to manage, update or delete his or her data auditing policies. The policy is dynamic component of the auditing infrastructure and it changes whenever the data owner wants to.

Another flexibility benefit of our system is that our auditing policies are independent of the variety of Grid applications' data access function signatures. From our above discussion of the use of "CRUD", we identified "CRUD" as the primitive data access operation and any other application specific operation is assumed to be composed of one or more "CRUD" operations. Data owners are free to define policies simply in terms of "CRUD," provided that there exists a reasonable mapping from the original data access request (may contain action name as the complete access function name, such as GridFTP.NET commands) to "CRUD" operations. Such a mapping table could be defined either statically or dynamically depending on the Grid application and the desired flexibility.

As for system scalability, we think our auditing system will work quite well, even for large Virtual Organizations. Since the auditing infrastructure only deals with the access requests and evaluates those requests with the auditing policies, this invocation is usually made by the authorization service. The auditing decision and enforcement point can be distributed and replicated in order to cope with large amounts of data access requests with in a large virtual organization.

5.3 Security and Trust Management

There are several security and trust management issues we identified in our auditing infrastructure. There are security requirements associated with auditing trail protection and the trust management between auditing enforcement and the remote auditing service, between the auditing service and the auditing analyzer. Although some of the issues have been identified as areas for future work a few are worth discussing here.

For example, there are data flow encryption and signing requirements for auditing requests sent to the auditing decision point. Also, when the auditing logs are generated and sent by the auditing enforcement point to the remote auditing service, they should be signed as well. When the auditing service receives the auditing logs and preserves them in local storage resource, the audit trails need to be protected against unauthorized access, modification, or deletion. This is also a basic requirement for protecting auditing trails identified by [7][8].

5.4 Ease of Use and Installation

We believe that with or web-based policy definition interface, our auditing system should be fairly easy for data owners to use. The primary obstacle may be that data owners are not experienced in defining logging policies in terms of attributes and conditions.

Installation of the auditing infrastructure is quite simple. The web portal is developed using ASP.NET based on XACML.NET library. The auditing service is an ordinary .NET web service. The setup, in general, is straightforward.

5.5 Performance and Quantitative Analysis

In order to gain a better understanding of how our system performs, we test the performance of our architecture on a 3.2 GHz Intel Pentium 4 with 512MB of RAM running Windows XP SP2. The key components include the GridFTP.NET server, the XACML authorization service, XACML auditing decision and enforcement component and remote logging service. The XACML authorization service and the XACML auditing decision and enforcement component are co-located. In order to get a more comprehensive view of our system performance, we utilize 3 different logging services to actually store the log generated by the auditing enforcement point. These 3 logging services are: 1. Generic logging service 2. Syslog service [3] 3. SAWS [10]. The generic logging service is actually a .NET web service accepting logs from auditing decision point and writing these logs into a file named for the URL of the requested resource. Syslog is a widely used logging protocol originally defined for Unix systems. In our test environment, we used a Windows implementation of Syslog called Kiwi Syslog [11]. Syslog service uses network sockets to communicate with clients and receive log strings. SAWS is a PKI-based secure audit web service developed by Computing Laboratory of University of Kent. Besides supporting remote logging, SAWS also provides more security characteristics such as the use of SSL to do mutual authentication with the SAWS service and the signing and encryption process to support the confidentiality and integrity of the log saved in local disk. In our experiment, SAWS is configured to have 0 debug level and has 100 maximum log entries to save in a file.

When running our tests, we spawn a client/data consumer that talks to a GridFTP.NET server and passes it commands to download a file from a remote site. This will trigger the authorization/auditing requests and responses as described in Section 2.3. We calculated the round trip time starting from when the GridFTP.NET server sends the XACML authorization request to the point when the XACML authorization response is returned to the GridFTP.NET: the total cost of authorization and logging. We tested several different auditing policies with increasing complexity to see how the complexity of XACML policies will affect the performance of the XACML evaluation engine and accordingly affect the performance of our auditing decision and enforcement architecture. The literal meaning of each policy is given below starting from the simplest to the most complex one:

Policy 1: (*subject == X509CertDN)*
Policy 2: *(subject == X509CertDN) and*
(operation name != read)
Policy 3: *(subject == X509CertDN) and*
(operation name != read) and
(authorization decision == Permit) and
(location name !=127.0.0.1)
Policy 4: *(subject == X509CertDN) and*
(operation name != read) and
(authorization decision == Permit) and
(location name !=127.0.0.1)) $\times 5$

Policy 4 contains 5 times the number of conditions for policy 3. Thus, the XACML evaluation engine has 5 times the number of conditions to evaluate since there are no logical short circuits. To round out the performance comparison, we also tested the time for a simple Gridmap file lookup for authorization without auditing policy evaluation and remote logging. This will give a measure of the additional cost imposed by our infrastructure. Each result is an average of 1000 trials:

Table 1: Durations to download a file from a remote site.

	Grid Map	XACML Auth	XAMCL Auth (generic)	XACML Auth (syslog)	XACML Auth (SAWS)
Policy 1	0.47 ms	74.93 ms	111.73 ms	113.96 ms	119.39 ms
Policy 2	0.47 ms	74.93 ms	115.79 ms	122.16 ms	128.82 ms
Policy 3	0.47 ms	74.93 ms	119.89 ms	130.03 ms	157.72 ms
Policy 4	0.47 ms	74.93 ms	162.61 ms	166.17 ms	189.75 ms

As shown in Table 1, doing a Gridmap lookup costs the least amount of time since it does a simple text file lookup. The second column is the round trip time solely for the XACML authorization process. As we compare the time difference between column 2,3,4,5, we can see that it takes about twice as long to do XACML authorization and logging as it does to just do logging. The increasing complexity of auditing policy does not incur much additional evaluation cost and even for policy 4 which has 5 times as many conditions as policy 3. Another interesting result is the efficiency of SAWS as a logging service. Although it provides logging encryption and signing and requires mutual authentication, it actually

has comparable performance with other logging services which provide less in the way of security. As for the current working structure of SAWS, it actually receives log messages and stores them in a queue in memory, the client can then return -- so this is the round trip time we collected in the table; inside the SAWS server, another task is then doing the signing and encryption and saving the log file jobs.

6. Conclusion and Future Work

Our auditing infrastructure is a unique approach for data owners to participate in the definition of auditing policies for their data sets. A requirement that becomes crucial for the dynamic collaboration between entities from heterogeneous domains in Virtual Organizations. Our auditing infrastructure provides a flexible way for data owners to dynamically define and manage auditing policies for different data access requests from different Grid applications. When data consumers want to access some data generated by data owners residing in different security domains, the requests will be evaluated against the auditing policies for that domain. If the auditing policy conditions are satisfied, the auditing enforcement point will generate an audit trail and invoke an auditing service to preserve that log. We believe our infrastructure meets the data access auditing requirements proposed by the current Grid community. There are also some important future work such as the design and implementation of an auditing trail analyzer to detect system vulnerabilities and the further application of our auditing infrastructure to other grid services such as GRAM and performance evaluation of our auditing infrastructure within large Virtual Organizations.

References

[1] United States Department of Health and Human Services Office for Civil Rights-- “Health Insurance Portability and Accountability Act”, 1996. [Online]. Available: http://cms.hhs.gov/hipaa/hipaa1/default.asp

[2] J. Feng, L. Cui, G. Wasson, and M. Humphrey. Toward Seamless Grid Data Access: Design and Implementation of GridFTP on .NET. Proceedings of the *2005 Grid Workshop (Associated with Supercomputing 2005)*. Nov 13-14, 2005. Seattle, WA.

[3] BSD Sys Protocol, RFC 3164 [Online]. Available: http://www.faqs.org/rfcs/rfc3164.html

[4] D.Bird “Auditing on Windows 2000”, 2000. [Online]. Available: http://www.enterprisenetworkingplanet.com/netsecur/article.php/623821

[5] D. Gunter, B. Tierney, B. Crowley, M. Holding, and J. Lee. NetLogger: A Toolkit for Distributed System Performance Analysis. In Proceedings of the IEEE Mascots. 1997

[6] Oasis Access Control TC, “XACML 2.0 Specification”. 2005. [Online]. Available: http://docs.oasis-open.org/xacml/2.0/access_control-xacml-2.0-core-spec-os.pdf

[7] DSS Industrial Security, “Information System Security”, November 2005. [Online]. Available: http://www.dss.mil/isec/change_ch8.htm

[8] R.Gopalakrishna, “Audit Trails”, April 2000. [Online]. Available: http://homes.cerias.purdue.edu/~rgk/at.html

[9] Sun XACML Implementation, “Sun’s XACML Implementation: Programmer’s Guide for Version 1.2”. [Online]. Available: http://sunxacml.sourceforge.net/guide.html

[10] W.Xu, D.Chadwick, S.Otenko, A PKI-Based Secure Audit Web Service Proceeding (499) Communication, Network, and Information Security, 2005

[11] Kiwi Syslog Daemon, “Kiwi Syslog Daemon Online Help”. [Online]. Available: http://www.kiwisyslog.com/info_syslog.htm

[12] M. Humphrey, M. Thompson, and K.R. Jackson, “Security for Grids” Proceedings of the IEEE (Special Issue on Grid Computing), vol 93, No. 3, March 2005.

[13] Raman, V. and et.al. “Data Access and Management Services on Grid”. http://www.cs.man.ac.uk/grid-db/documents.html . 2002.

[14] M.Milani, J.Brown “Some Security Considerations for Service Grids”. [Online] Available: www.johnhagel.com/paper_securitygrid.pdf