Apache Airavata Security Manager: Authentication and Authorization Implementations for a Multi-Tenant eScience Framework

Supun Nakandala
Research Technologies, UITS
Indiana University, USA
snakanda@iu.edu

Hasini Gunasinghe
Computer Science Department
Purdue University, USA
huralai@purdue.edu

Suresh Marru
Research Technologies, UITS
Indiana University, USA
smarru@iu.edu

Marlon Pierce
Research Technologies, UITS
Indiana University, USA
marpierc@iu.edu

***Abstract*—eScience middleware frameworks integrating multiple virtual organizations must incorporate comprehensive user identity and access management solutions. In this paper we examine usage patterns for a multi-tenanted distributed system and map these to widely used security standards and approaches. Usability is as important a design feature as standard compliance in our considerations. While the contributions of this paper are generally applicable to the eScience community, to make the arguments concrete and pragmatic, we focus on science gateways, a class of distributed system cyberinfrastructure. Science gateways are end user environments that provide access to a wide range of academic and commercial computing and storage resources for virtual organizations. Successful gateways focus on specific scientific communities and domains, but they build on many reusable features that can be provided by general purpose hosted science gateway platform services that can support multiple science gateway tenants. Providing a security framework for identity and access management for such hosted service is a key feature, as it removes the burden for each gateway to handle its user identity management and control access to its critical resources; from the resource provider's point of view, it provides a basis for more uniform accounting and auditability. This however is a challenging problem when one considers the range of gateways (both legacy and newly created), the range of technologies used to build gateways, and the range of end user access points (Web, mobile, desktop, and programmatic API clients) that gateways provide. This paper describes our approach for providing a unified identity and access management solution for a multi-tenanted e-Science platform for science gateways with diverse Virtual Organizations (VOs) that is based on the open source Apache Airavata software. We examine three common gateway types based on where the user identity information is actually maintained (platform-maintained, gateway-maintained, and third party-maintained) and how these can be treated in a unified manner. We also examine authorization requirements for user requests in each gateway when accessing resources in Airavata API and options of policy-based API-level authorization. We describe how these are implemented for specific Apache Airavata client gateways. Our solutions for identity and access management are not specific to Apache Airavata but can be generally applied to any e-Science platform.**

***Index Terms*—science gateways, identity management, access management, distributed systems security, apache airavata**

I. Introduction

Science Gateways [1], [2], [3] [4] are user environments and supporting services that help researchers make effective, optimal, and enhanced use of a diverse set of distributed computing, storage, and related resources. Many successful gateways target specific user communities, such as bioinformatics [5], computational chemistry [6], and nanotechnology [7]. Gateways, in effect, serve as Virtual Organizations, brokering access for their users to a broad collection of resources from different, often unaffiliated resource providers including campus grids, national scale cyberinfrastructure, and commercial cloud vendors. Thus, identity management, authentication, and authorization are critical capabilities that a gateway must provide. Security management is one example of the general-purpose features that underlie many domain specific gateways. Other examples include job and workflow management, data and provenance management, and information monitoring and auditing. Thus an important architectural trend is for gateways serving specific Virtual Organizations to use hosted, general purpose gateway platform services. The goal of the Apache Airavata project [8], [9], [10], [11] is to implement and integrate many of these general purpose services into a common framework that can be run as a multi-tenanted platform service that can serve multiple gateways simultaneously. In Apache Airavata, these services are implemented by multiple components (see [10]) but are exposed through a common API and a single, logical connection point, the API Server. Here, we describe the design and implementation needed to secure the interactions between the gateway client and the Apache Airavata API Server. Figure 1 summarizes how Airavata interacts with client gateways and remote resources. All traffic between the gateways goes to Airavata's API Server, which hands off messages for internal routing. Managing identities and authorizing API invocations between the gateways and the API Server are the subjects of this paper.

Science gateway tenants to Apache Airavata services run remotely and are typically under the control of the gateway provider. Gateway clients interact with Apache Airavata through an Apache Thrift-based API. For an overview of the API, see [8]. The Thrift-defined API and data models allow Airavata to provide client SDKs in many programming languages including Java, PHP, Python and C++. These SDKs provide a programming language-neutral encoding, and mar-

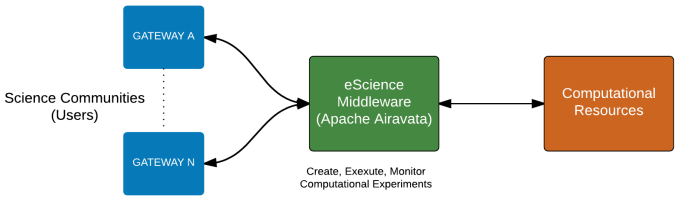

Fig. 1. Multi-tenanted eScience Middleware

shal and unmarshal over-the-network communications between the gateway and the Airavata server. Our technical challenge is to implement security features over the top of these communications. We note an additional challenge: gateways are not always Web-based but may also be desktop or native mobile applications, or embeddable clients intended to be called directly from end-user scripts. This introduces a challenge since Airavata SDKs are distributed directly to the end user's device. We cannot assume access is restricted behind a Web server. In this paper, we discuss the design and implementation of the solution for securing the Airavata API. This includes authenticating and authorizing end users into the Airavata API, based on different types of identified use cases that depend on the different types of user identity management scenarios in gateways. The primary contributions of this paper are the identification of general patterns in gateway identity management, mapping of these patterns to widely adopted OAuth2 scenarios, identifying a flexible approach to role-based access using XACML policies, and implementing these solutions. We examine the implementation using three gateways that are clients to hosted Apache Airavata services.

II. PROBLEM DEFINITION

Apache Airavata is multi-tenanted science gateway middleware that can be used by different types of science gateways as a platform to create, execute and monitor jobs and workflows on a diverse range of backend computing resources. Airavata provides both user-level functions and administrator level-functions. User-level operations are organized into "experiments" that contain all the information about a job request. Administrator-level operations include managing metadata descriptions of compute resources and scientific applications. Airavata uses this metadata to generate job requests on specific target compute resources, so this is not something a typical user should modify, although it may be something some users want to view. Incorrect modifications will break the resource or application for all users of a gateway tenant. Other privileged operations include accessing the experiment metadata for all users. This is useful for debugging problems when job submission requests fail. Thus, all API calls to the Airavata server a) should be authenticated, b) should be traceable to specific user identities, and c) should only be invoked by users with the required privilege levels or roles. Furthermore, these security considerations should be applied at the gateway tenant level, and tenants should be run in virtual isolation. At this time, we do not consider issues such as the ability of certain users to access certain resources or applications, or the ability of users to share experiments. Group based sharing is an important future work, but here we focus on securing the API calls themselves and enforcing authorization at user level. Previously, the end users who interacted with Airavata API through such gateways were only authenticated and authorized at the gateway level, and no validation was done at the Airavata API level. The security model was based on the trust relationship between gateway software and Airavata middleware by restricting access to the Airavata API only from pre-validated web-based gateway clients using firewall rules, TLS mutual authentication, and similar approaches. Hence, there was no notion of user identity maintained within Airavata. This approach was reviewed with the Center for Trustworthy Scientific Cyberinfrastructure and determined to be operationally acceptable [12]. However, the approach does not scale to a large number of gateways, it does not address the issue of securing native client (desktop and mobile) access to the Airavata API, it does not enable a uniform approach to user-level tracking of API calls, and it is not satisfying from the architectural point of view. When designing a solution to address the aforementioned requirements, there are three different identity management scenarios that we must consider for gateway clients to Airavata services.

- Scenario 1: The gateway client does not have a user store and would like to depend on Airavata to provide user management features. This is most common for a new gateway that may not have completed its implementation or gone into full operations.
- Scenario 2: The gateway has a user store and in-house identity management mechanisms. In this scenario, different gateways have different preferences on the level at which they share user identity information with Airavata. This is typical of mature gateways.
- Scenario 3: The gateway does not have a dedicated user store but authenticates users into the gateway using some federated identity provider such as InCommon using mechanisms such as SAML SSO, OpenID, and OAuth.

We must be able to provide a unified identity management solution that can meet the requirements of the above use cases and provide proper Airavata API security that can be seamlessly adopted by all types of gateways including web based and native (desktop and mobile) clients. The proposed security solution should also facilitate multi-tenancy within hosted Airavata services as a first class feature.

III. SOLUTION OVERVIEW

When devising a solution, one option that we considered was to provide user authentication at the gateway layer using the authentication protocol of the gateway's choice, and use the system-to-system authentication mechanism between the gateway and the Airavata server. Examples of two of the system-to-system authentication mechanisms that can be used are mutual authentication using SSL certificates and basic authentication (i.e: Gateway credentials over TLS). The main

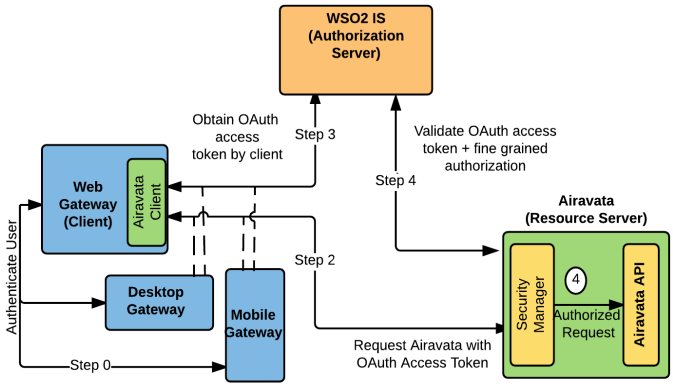

Fig. 2. High level overview of the solution

drawback of the certificate-based mutual authentication approach is that it is not scalable from the management point of view. Each gateway can have different types of applications (web, native), and ideally each of these applications should have different credentials. When the number of Airavata gateway clients increase, management of gateway credentials and PKI infrastructure is not scalable for the API server as enrollment, certificate issuance, and certificate revocation become tedious for large number of gateways [13]. Also, both of these approaches can be used only for web based gateways, where credentials or certificates can be securely maintained in a web server. They are not secure enough to be used in native clients as a potential malicious user can reverse engineer the client and gain access to the credentials/keys. One way to facilitate native clients using these approaches is to route the requests from native clients via a proxy server and establish the mutual trust between the proxy server and Airavata using gateway credentials or certificates. This requires extra effort from gateway developers and is also not an elegant solution as end user information required by Airavata needs to be passed to the Airavata API Server by the clients with every API request.

The other option that we considered and adopted is to use OAuth 2.0 based authorization delegation [14] to the gateway by the user authenticated at the gateway. The main advantage of this approach is scalability. This approach does not require any management of gateway credentials or PKI infrastructure. The OAuth access tokens can be generated by a separate dedicated authorization server and not by the API Server. This approach also benefits from wider adoption in the general "Software as a Service" and "Platform as a Service" communities to which science gateways and gateway frameworks like Airavata belong. OAuth 2.0 is the de-facto standard for access delegation. It can be used in conjunction with existing authentication protocols such as SAML 2.0 based single sign on via the OAuth 2.0 extension profiles. Even though OAuth is an authorization delegation protocol, authentication of users can be done using OpenID - Connect [15], which is an authentication protocol that runs on top of OAuth 2.0. Hence OpenID-Connect + OAuth 2.0 can be used as a comprehensive authentication and authorization protocol. The advantages of the OAuth 2.0 approach led us to implement it as the authorization delegation solution to our problem. Figure 2 illustrates the high level architecture of the solution with mappings to the standard OAuth 2.0 based authorization-delegation solution architecture. Because OAuth 2.0 and related technologies are widely adopted, we chose to integrate a third party service with our solution rather than develop it ourselves. Our solution makes use of the identity management features offered by WSO2 Identity Server (IS) [16]. WSO2 IS is a widely used open source (Apache V2 License) identity management system that provides out of the box support for many identity management standards such as OAuth/OpenID, SAML SSO, XACML and features such as multi-factor authentication, password policies, etc. It supports multi-tenancy, per-tenant user store configuration, and custom pluggable user store manager extensions, all of which are relevant to the problems discussed in this paper.

Details of the interactions illustrated in Figure 2 are as follows. Numbers correspond to the labels of interactions in the figure.

0) User is authenticated to Airavata. How this is done with respect to aforementioned three different scenarios is described in the next section.
1) OAuth token is obtained from the WSO2 IS to access the Airavata API on behalf of the authenticated user.
2) Request to Airavata (depending on what actions user wants to perform) are sent along with the obtained OAuth token.
3) Each request sent to the Airavata API Server is authorized by the Security Manager before it hits the API. The Security Manager acts as a client within Airavata to various WSO2 IS services. First the attached OAuth token is validated and user's profile information is retrieved. Then the request is authorized using eXtensible Access Control Markup Language (XACML) [17] using the authorization policy which is pre-defined by the gateway administrator (details to follow).
4) Only the authorized requests are allowed to reach Airavata API.

Authorization of subsequent requests from the same user will be handled by using caching of authorization decisions in order to avoid the requirement of contacting the WSO2 IS for every call to increase performance (see below). The above solution supports multi-tenancy; when the user identity is exchanged, it includes the tenant domain that the user belongs to so that the authentication and authorization is performed with respect to that tenant domain in WSO2 IS. The same client side logic to authenticate a user and obtain an OAuth token can be implemented in web, desktop and mobile clients using the recommended grant types available in the OAuth 2.0 specification. The next section elaborates more on this high level solution and how it can be adopted to previously mentioned three identity management use cases using the features provided in the WSO2 IS.

IV. SOLUTION IN DETAIL

A. Authenticating User & Obtaining an OAuth 2.0 Access Token

User authentication and obtaining an access token by the client application (Step 0 and Step 1 in Figure 2) are the only steps in the high level solution that differs among the three scenarios that we identified. Again, we note that we use OpenID-Connect extensions to the OAuth2 authorization protocol for authentication. The OAuth 2.0 specifies several ways (grant types) to obtain an access token by the client app. Each of these grant types serves a different purpose and is used in a different way. Depending on the type of application is being used, the appropriate grant type should be used to obtain access tokens. The grant types are (1) Authorization Code grant type, (2) Implicit grant type, (3) Resource Owner Password grant type, (4) Client Credential grant type, and (5) Refresh Code grant type. We evaluate each and discuss their applicability to our scenarios.

The Authorization Code grant type is the recommended grant type to be used if the client application is capable of spawning a web browser to redirect the user to the authorization server's authentication page. The user will sign in to the authorization server with their credentials and authorize the gateway application to obtain an access token for the user. This authorization is a one-time-only application authorization. If the authorization server authorizes the request, the user will be redirected back to the client with a token (called the authorization code) in the query string (e.g. https://client.com/redirect?code=XYZ123), which the client application will capture and exchange for an access token in the background. In order to use this grant type, the application itself should already be trusted by the authorization server through some application registration process. The Implicit grant type is very similar to the Authentication Code grant type. The difference is that for an Implicit grant type, the access token is directly returned to the client application after the user signs in. Implicit grants are for use by clients that are not capable of keeping the client application's own credentials secret. An example is a thick browser client that uses JavaScript to directly access the Apache Airavata API methods rather than going through an intermediate Web server.

If the user (resource owner) already trusts the client application to provide his/her own credentials, the Authorization Code grant type is not needed. Resource Owner Password grant type is an alternative grant type that can be used to simplify the flow in this case. This grant type is useful in scenarios where the client application cannot spawn a web browser and the user trusts the client, such as gateway provided desktop and mobile client applications. In this grant type instead of redirecting the user to the authorization server's authentication page, the client application itself obtains the user credentials and then sends these to the authorization server along with the client's own credentials. If the authentication is successful then the client will be issued an access token.

The Client Credential grant type is similar to the Resource Owner Password grant except only the client application's credentials are used to authenticate a request for an access token. This grant type should only be used by trusted clients. This grant is suitable for machine-to-machine authentication where there is no user interaction or user authorization required. In this grant type we do not need to use OpenID-Connect to authenticate users as there is no user involvement.

Authorization servers that support Refresh Code grant types will also issue a refresh token when they return an access token to a client application. When the access token expires, the client can use the refresh token to retrieve a new access token with the same permissions. Now the client has to maintain the state of each token. This can be done by either periodically using the active refresh token or by using the refresh token to acquire a new token after an access failure.

B. Science Gateway Client Scenarios

In the following discussion, we explain how the high-level solution described in the previous section is adapted to address three different gateway identity management scenarios and which of the specific OAuth 2.0 grant types discussed above should be used to obtain access tokens from the authorization server. All solutions involve the introduction of a Security Manager component within Apache Airavata (see Figure 2) that acts as a client to various WSO2 IS services.

1) The gateway doesn't have a user store and would like to depend on Airavata to provide user management features : The gateway application makes use of the Airavata Client SDK to create users, which in turn invokes the User Admin API of the WSO2 IS. When the users are authenticated to the gateway, their credentials are validated against those stored in the WSO2 IS user store. The gateway uses Airavata's client SDK to authenticate users and obtain OAuth access tokens using OpenID-Connect. All the client applications are provided by the gateway itself and no user generated or third party applications need to connect to Airavata. Thus all gateway client applications can be considered as trusted clients, and therefore instead of Authorization Code grant type we use the Resource Owner Password grant type to obtain access tokens in both web and native clients. If the web application is a thick browser based application (browser applications that directly interact with Airavata API through a JavaScript SDK) where client credentials can not be securely maintained, the Implicit grant type should be used and privileges of access token obtained by this grant type should be highly restricted.

2) The gateway has a user-store and its own, in-house identity management mechanisms: Here we consider three solutions to address this scenario based on the differences in preference of the gateway to share user identity information with Airavata. In the first category, the gateway will not share any information about the end user's identity with Airavata and will use Airavata only as a job execution engine. Therefore the gateway client will obtain an OAuth access token by authenticating to the WSO2 IS. The grant type that is used in this case is the Client Credential grant type. After retrieving

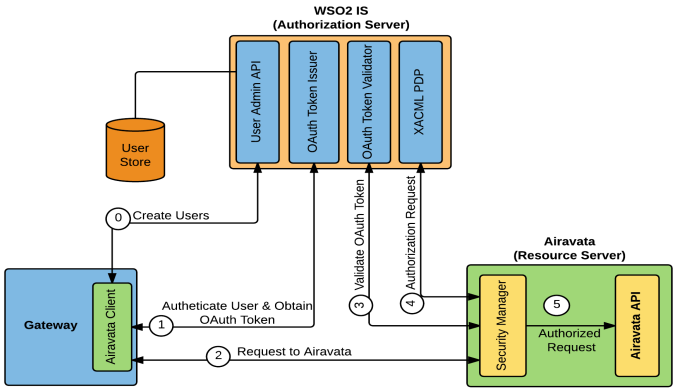

Fig. 3. Gateway using Airavata provided user store

an access token, end user requests that are sent to Airavata attach this token (see Figure 4). From Airavata's point of view all experiments will be run by one community user account. In this case the types of gateway applications that can be supported are limited only to web based gateways where the client credentials can be securely maintained in the web server. As native clients cannot securely maintain client credentials, the requests from those clients should be sent through an intermediary proxy server that can securely maintain the client credentials.

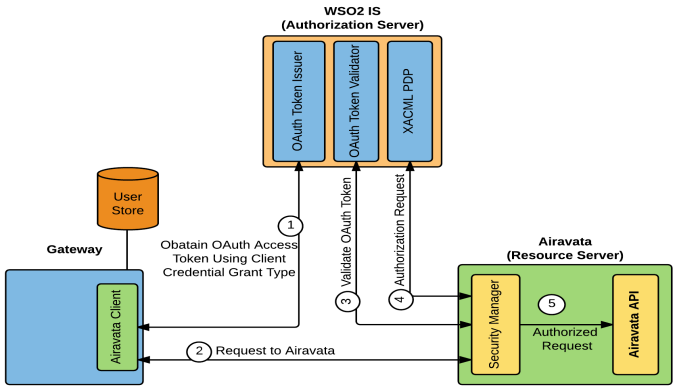

Fig. 4. Gateway does not share any user identity information with Airavata

In the second category, the gateway is willing to share user identity information but does not allow Airavata to connect to the gateway's user store. User accounts need to be provisioned with the identity information required by Airavata. In this case the gateway makes use of the identity provisioning client in the Airavata client SDK to provision user accounts to the identity manager of Airavata (i.e WSO2 IS) at the time of user creation in the gateway. Already created user accounts in the gateway user store can be provisioned through a bootstrapping phase. Invoking the Airavata SDK for user provisioning will in turn invoke the System for Cross-domain Identity Management (SCIM) [18] endpoint in WSO2 IS for identity provisioning. This enables the execution flow of accessing the Airavata API to be the same as in scenario 1 (see Figure 5). In this category user information will be duplicated in both the gateway user store and the WSO2 IS user store. It is the responsibility of the gateway to maintain them in coherence. After the user accounts have been provisioned, user authentication, access token retrieval and the rest of the execution is same as in Scenario 1.

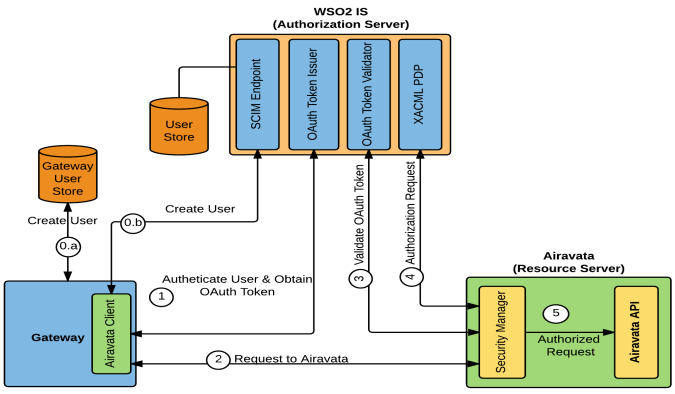

Fig. 5. Gateway which provision user accounts to the Airavata provided user store

In the third category, the gateway is willing to share user identity information. The gateway allows Airavata to connect to its organizational user store in read-only mode. In this case, the identity manager of Airavata (i.e WSO2 IS), is connected to the gateway's organizational user store through the user store manager extension provided by the WSO2 IS. This enables the user authentication, access token retrieval, and the rest of the execution flow of accessing the Airavata API to be the same as in scenario 1 (see Figure 6). The WSO2 IS distribution by default provides user store manager extensions that can be used to integrate Active Directory and LDAP based user stores. Custom user store manager extensions can be written to integrate any custom user store.

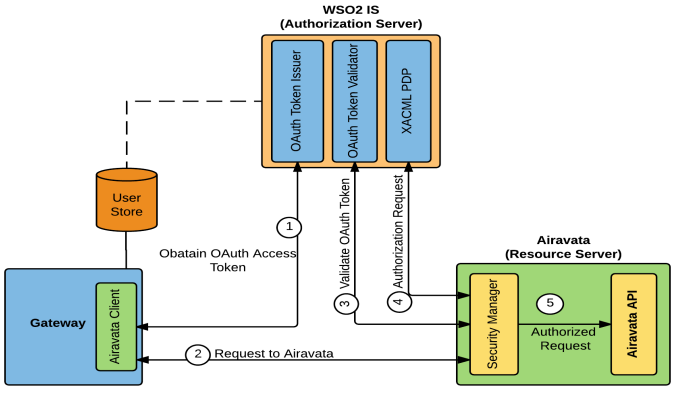

Fig. 6. Gateway which allows Airavata to have read only access to the organizational user store

3) The science gateway does not have a user store but instead authenticates users into the gateway using a third party federated identity provider. In this scenario the gateway becomes a relying party, and it needs to authenticate users to Airavata through the federated authentication mechanism

that is being used in the gateway community: To solve this issue, we used the inbound authentication configuration feature in WSO2 IS. Using this feature, it is possible to plug external federated authentication providers as inbound authentication providers to a user store (see Figure 7). Out of the box, WSO2 IS provides support for OpenID, SAML, OAuth2/OpenID-Connect, WS Federation, Google, Yahoo, Microsoft and Facebook federated authentication providers. For a federated authentication protocol not in the previous list, it is possible to plug in a custom inbound authentication provider.

If the federated authentication protocol supports retrieval of the user's identity attributes from the identity provider (such as SAML, OpenID), a user account is created in WSO2 IS with this identity information using just-in-time provisioning. Now when a user tries to authenticate to WSO2 IS, the user can select the configured identity provider and login to that provider. Once the user is authenticated via the federated identity management protocol, an OAuth access token is generated in WSO2 IS that has access to the Airavata API, and the rest of the flow of execution continues in the same way as in other cases. Since federated authenticators most of the time support only web-based access, a web based OAuth 2.0 grant type such as Authorization Code grant or in the case of thick web clients Implicit grant type should be used. Native clients are now required to spawn an embedded browser for user authentication and extract the access token.

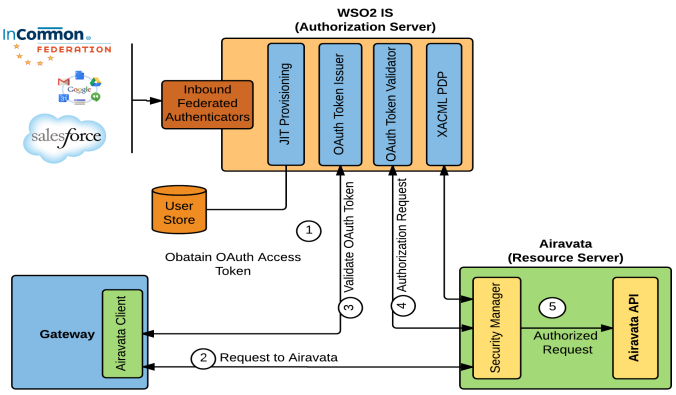

Fig. 7. Gateways which want to use federated identity provider.

C. Role Based Fine Grained User Authorization using XACML

All the requests that are sent to the Airavata API Server go through Security Manager before API methods are invoked in the API Server. As per Step 3 of the high level solution overview (Figure 2), the Security Manager first validates the OAuth token attached to the request. This guarantees that the request comes from an authenticated user who has obtained a valid OAuth access token. In addition to validating that the request is coming from an authorized user, we also validate that the user has privileges to access the given API method. Through our reviews of existing gateways, we found that each gateway has its own user role hierarchy. Based on this user role hierarchy, the privileges of each user role will be different. For example, in a typical gateway there will be gateway admins who administer all gateway related configurations such as application and computer resource registrations. Regular gateway users will only need to access API methods for managing computational experiments. Based on the roles assigned to the user, privileges should be restricted when accessing the Airavata API. Instead of creating a global user role hierarchy and set of predefined user privileges for all gateways, we chose the XACML (eXtensible Access Control Markup Language) [17] feature available in the WSO2 IS. XACML is the de-facto standard for fine grained, policy-based access control. This approach allows different gateway tenants to have different, customized policies. Policy-based access control is more flexible than backing the authorization logic into the code of Security Manager. Gateways might have different authorization rules based on their tenant in Airavata. The implementation is based on the fully fledged implementation of XACML reference architecture in WSO2 IS. Three main components of XACML reference architecture that are used in this solution are Policy Administration Point (PAP), Policy Deployment Point (PDP), and Policy Enforcement Point (PEP) (see Figure 8).

- PAP facilitates defining, updating and publishing the authorization policies. Gateways use this feature directly. Figure 8 illustrates the involvement of PAP for defining the authorization policies by the gateway administrator. The gateway makes use of the Airavata client for this, which in turn invokes the XACML PAP API of WSO2 IS. XACML roles can also be defined within the WSO2 IS user interface.
- PEP is where the authorization is actually enforced on the API requests. Airavata's Security Manager handles these operations. Upon intercepting a request sent to Airavata API, the PEP forms a XACML authorization request including the information related to the current API request (such as user identity, name of the API function or resource that was requested), and then it sends that XACML authorization request to the PDP.
- PDP is the XACML policy engine of WSO2 IS. The PDP evaluates the authorization request sent by the PEP against the policy defined by the gateway administrator and returns the authorization decision back to the PEP. Based on this decision, the PEP (in Security Manager) allows or denies the API request that it intercepted.

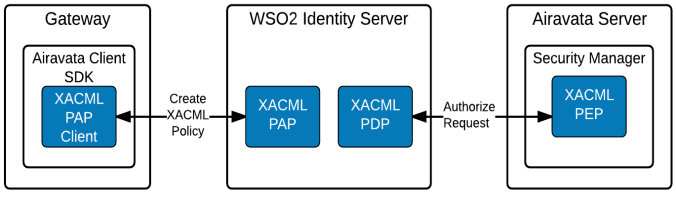

Fig. 8. Using XACML for role based fine grained access control

Using XACML we can define role based access control rules per gateway, with each gateway as a tenant in WSO2 IS. The rules are defined in XML; WSO2 IS provides graphical tools can be used to create these rules. Figure 9 shows a set of sample rules that is used to define XACML rules for the "Gateway User" role.

```
<Rule Effect="Permit" RuleId="GATEWAY-USER-PERMIT">
    <Target>
        <AnyOf>
            <Match>
                <AttributeValue>^(?:(?!
                        /airavata/getAPIVersion|
                        /airavata/createProject|
                        /airavata/updateProject|
                        /airavata/getProject|
                        /airavata/createExperiment|
                        /airavata/updateExperiment|
                        /airavata/getExperiment|
                        /airavata/launchExperiment|
                        .....
                        .....
                        .....
                        ).)*$\r?\n?
                </AttributeValue>
            </Match>
        </AnyOf>
    </Target>
    <Condition>
        <Apply>
            <AttributeValue>"GATEWAY-USER"</AttributeValue>
            <AttributeDesignator AttributeId="http://wso2.org/claims/role"
                Category="urn:oasis:names:tc:xacml:1.0:subject-category:access-subject"/>
        </Apply>
    </Condition>
</Rule>
```

Fig. 9. Sample XACML rules for Gateway User role

D. Securing the Inter Component Communication

It is critical to secure the communication between (1) Gateways and WSO2 IS, (2) Gateways and Airavata and (3) Airavata and WSO2 IS, as illustrated in Figure 2. Communication between the Airavata client at the gateway and the WSO2 IS involves requests that invoke the critical administration services of WSO2 IS, such as User Admin API, XACML PAP API, and the OAuth 2.0 token issuer endpoint. The requests invoking WSO2 IS administrative services are authenticated with the IS tenant admin-credentials and are sent over TLS, as per the default settings of WSO2 IS. All the OAuth 2.0 endpoints in IS are exposed as HTTPS endpoints by default. The same is true for communication between the Security Manager and the gateway client and the WSO2 IS. All the Airavata API calls from the Gateway to the Airavata API are made over TLS and require an OAuth access token as a mandatory parameter. The communication between Security Manager and WSO2 IS for validating access tokens and XACML policies are authenticated with the corresponding IS tenant admin-credentials and should be sent over TLS.

V. IMPLEMENTATION AND EVALUATION

Every Apache Airavata API method definition was changed to incorporate an additional mandatory field nameed AuthzToken, which contains the OAuth access token and the Airavata tenant ID information that is set at the gateway client layer. In the implementation to process this token, we added the aforementioned Security Manager (See [19]) to the Airavata API Server. The Security Manager is the cornerstone of Airavata's security implementation. This new component is designed to be pluggable so that we can change it if required without affecting the rest of Airavata. The main function of the Security Manager is to intercept all API requests and validate them from the separately hosted WSO2 Identity Server, which acts as the Authorization Server for Airavata. For intercepting the API requests the Security Manager uses annotations supported by the Google Guice library [20], which are added to every API method in API server. As mentioned in the solution section, the authorization validation at the Security Manager is a two step process. First the OAuth access token is validated by calling the OAuth token validator endpoint in WSO2 IS, and then the authorization policy is enforced by invoking XACML PEP in WSO2 IS. When invoking both of these endpoints, the requests need to be authenticated using HTTP basic authentication by providing the respective tenant's admin credentials in WSO2 IS. Therefore, Airavata needs to know and store the respective credentials for IS tenants. For this we used the credential store server within Airavata [21]. Gateway administrators can store their IS tenant credentials by updating the GatewayProfile object in Airavata. OAuth 2.0 mandates sending the access tokens over TLS. Therefore, the communication shown in Step 2 of Figure 2 happens over TLS. With this feature of the solution, Airavata is shipped with a default key store (airavata.jks) that is found in the bin directory of the server distribution. It should be replaced with the organization's key store in a production deployment. Communication between the Airavata server and WSO2 IS also happens over TLS. The client-truststore.jks in the bin directory of the server distribution contains the public certificates of both Airavata and WSO2 IS, which is used in the client side code which invokes Airavata and WSO2 IS. Ideally, every Airavata API request should be validated against the WSO2 IS as shown in Step 3 in Figure 3. However, we found that this imposes a significant performance overhead (about 350 ms) on the Airavata API method latency. Figure 10 shows latency for some of the commonly used API methods. Results for other API methods are comparable. To overcome this issue we added an authorization cache module into the Security Manager that caches the authorization decisions for each access token-tenant-API method combination after successful validation from WSO2 IS. Subsequent API requests to the API Server from the same user are checked in the authorization cache instead of directly invoking the IS validation endpoints. The caching duration of each authorization decision is set to the remaining valid duration of the corresponding OAuth access token, which can be obtained from the OAuth token validation response when invoking IS for validation. This OAuth token validation response contains various user attributes such as username and email that are also cached and made available during the execution of the API methods by using the thread local object. After enabling the authorization cache, the security enforcement overhead was drastically reduced. The performance tests were done for Scenario 1 (see Figure 3) deployment using MySQL backend as the tenant user store in WSO2 IS. Gateway client, Airavata server, and WSO2 IS server were running on three different machines within the same local area network.

We have implemented the designs described here to support several science gateway tenants to Apache Airavata. The

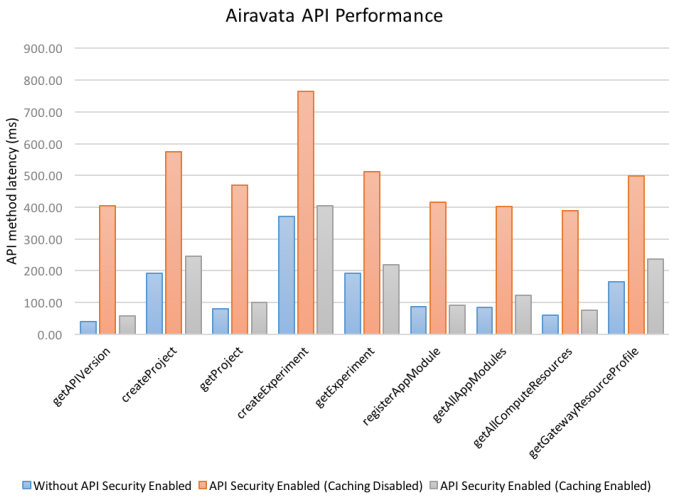

Fig. 10. API method latencies

SEAGrid science gateway [22] is undergoing a technology refresh and wanted to use Apache Airavata-provided identity management; hence it is an example of a Scenario 1 tenant. SEAGrid is using a MySQL database as the user store in WSO2 IS and provides a web based gateway and a desktop client. Both of these clients use the Resource Owner Password grant type in OAuth 2.0 specification to authenticate users and retrieve access tokens as both are trusted clients. SEAGrid has four user roles in its role hierarchy: a) gateway-admin, b) read-only-gateway-admin, c) gateway-user and d) pending-user. The gateway-admin role is the superuser in SEAGrid and has privileges such as the ability to add, modify and delete resources and applications, the ability to manage users, and access to gateway-wide administrative dashboards. The read-only-gateway-admin role can view the gateway configuration parameters and dashboards but cannot make any changes. The gateway-user role is assigned to regular users, and the pending-user role is assigned to new users who are not yet approved to use the gateway. These authorization policies are enforced in SEAGrid using XACML as described in the previous section.

Another client is a USDA-funded science gateway, currently under development, for accessing bioinformatics applications on USDA-provided resources. Unlike SEAGrid, this client wanted to integrate the user management with the existing USDA user store so that the users can use their existing credentials to access the gateway. Thus this falls to the Category 3 of Scenario 2 in the above mentioned taxonomy. The existing USDA user store is an LDAP server and hence it was straightforward to integrate with IS using its already provided LDAP user store manager extension. This gateway has only a web based portal, so a Resource Owner Password grant type was used for authentication and access token retrieval. This gateway also has a similar role hierarchy to SEAGrid and uses a similar authorization policy.

The UltraScan science gateway [23] is a longstanding client for Apache Airavata services; this integration predates the work described in this paper. Ultrascan has its own user store and wanted to use Airavata only as a job execution management engine and does not currently share any user information with Airavata. Thus, this falls to the Category 1 in Scenario 2 in our taxonomy. As described in the solution section, Client Credential grant type is used to retrieve an access token that is used to submit API requests to Airavata. Although UltraScan provides native clients, supporting them through this model was not an issue as the requests from those clients were sent via an intermediary proxy server that comes from the previous architecture. This intermediary server is capable of securely maintaining gateway client's credentials.

VI. RELATED WORK

Science gateway security, especially the aspect of user credential management for individual gateways, is a well studied area. In the early stages, the most widely adopted approach was to assign per user credentials for computing resources. Whenever a gateway user wanted to submit a job to a remote compute resource, the gateway middleware had to use that particular user's credentials on the remote host for authentication. Per user grid certificates, MyProxy and various variations of MyProxy service such as OAuth for MyProxy [24], CILogon for MyProxy and the MyProxy Gateway were the most popular implementations of this approach [25]. Other implementations of this approach such as Security Assertion Markup Language based Single-Sign-On (SAML SSO) implementation in MoSGrid science gateway can be found in literature [26]. As the number of science gateway users grew, the community soon came to realize that the per user credential mechanism does not scale well. To avoid unscalable manual overhead of new user approval, user credential issuing and credential revocation, the science gateway Authentication, Authorization, Auditing and Accounting (AAAA) model [27] was designed; XSEDE uses a simplified version of this model in production.

The main idea of the AAAA model was to adopt community user accounts and avoid using per user credentials to authenticate to the remote resources. This significantly reduces the effort required for user management tasks in compute resources as now there are a limited number of science gateway community accounts. This approach outsources gateway user management tasks to the gateway layer itself. In this model, multiple gateway users share the same community account to submit jobs to compute resources. Apache Airavata and its client gateways use this model, and thus it is very important that Airavata implement and practice proper user AAAA. Within Airavata, community credential management is done by the credential store component [21].

In this paper we describe how gateway user identity management can be implemented in Airavata as a generic feature that can facilitate diverse use cases of different gateways. Based on our use cases, the Center for Trustworthy Cyberinfrastructure (CTSC) has evaluated various authentication and authorization schemes such as username-password, kerberos, X509, API keys and OAuth, and has suggested that OAuth 2.0

is both viable and secure method to implement authentication and authorization in multi-tenanted services like Airavata [21].

Globus Nexus [28] is a very similar system to the WSO2 IS that is used in our solution. It is a "platform as a service" application that provides user identity management, profile management and group management features. Its identity management capabilities allow users to create a unique Globus identity that can be associated with federated external identities from campus identity providers (e.g CILogon), computing resource identity providers (XSEDE accounts using MyProxy OAuth), and commercial identity providers such as Google. This Globus Identity can be consumed by subscribing applications using an OAuth-based workflow to create a Single-Sign-On environment. Globus Nexus provides group management capabilities, and groups can be used to enforce authorization policies. The main difference between Globus Nexus and WSO2 IS is that Nexus can be only integrated with federated identity providers whereas WSO2 IS can be integrated with variety of user store such as LDAP, Active Directory and custom RDBMS based stores using the user store manager extensions provided. As of now Globus Nexus lacks support for System For Cross Domain Identity Management (SCIM) that is needed for some of the gateway scenarios that we identified. Globus Nexus also lacks support for XACML, which we found useful for providing per-gateway policies. It is possible that these scenarios could be implemented with Globus Nexus by mapping them to groups. Finally, Globus Nexus is a hosted service, whereas WSO2 IS is downloadable software that is also available as a for-fee service. This difference introduces tradeoffs that gateways and gateway platform service providers should consider.

VII. CONCLUSIONS AND FUTURE WORK

The most significant advance in gateway architectures over the last several years is the use of hosted, general purpose gateway platform services, typically for task and workflow execution and data management, that interact with multiple gateway clients, or tenants. This paper examines the over-the-wire access patterns that exist between a wide range of gateway clients and multi-tenanted platform services like Apache Airavata. These patterns were then mapped to OAuth 2 grant types. We also examined fine-grained role based authorization requirements for gateways, mapping the role based access control to XACML policies. Finally, we described an implementation that can support all the scenarios that we identified. We summarized three current clients to hosted Apache Airavata platform services and mapped these to specific scenarios. The implementation code described here is open source and available through Apache Airavata's Git repository under the Apache Software License, version 2; see airavata.apache.org.

ACKNOWLEDGMENT

This work was supported by NSF award #1339774 "Collaborative Research: SI2-SSI: Open Gateway Computing Environments Science Gateways Platform as a Service (OGCE SciGaP)". S. N. and H. G. were supported by Google Summer of Code. We thank the Center for Trustworthy Scientific Cyberinfrastructure (NSF awards #1234408 and #1547272) for their extensive consultations on this work, which is summarized at http://trustedci.org/scigap/. S. N. and H. G. made equal contributions to this paper and should be considered co-principal authors.

REFERENCES

[1] K. A. Lawrence, M. Zentner, N. Wilkins-Diehr, J. A. Wernert, M. Pierce, S. Marru, and S. Michael, "Science gateways today and tomorrow: positive perspectives of nearly 5000 members of the research community," *Concurrency and Computation: Practice and Experience*, vol. 27, no. 16, pp. 4252–4268, 2015.

[2] S. Gesing and N. Wilkins-Diehr, "Science gateway workshops 2014 special issue conference publications," *Concurrency and Computation: Practice and Experience*, vol. 27, no. 16, pp. 4247–4251, 2015.

[3] N. Wilkins-Diehr, S. Gesing, and T. Kiss, "Science gateway workshops 2013 special issue conference publications," *Concurrency and Computation: Practice and Experience*, vol. 27, no. 2, pp. 253–257, 2015.

[4] P. Kacsuk, *Science Gateways for Distributed Computing Infrastructures*. Springer, 2014.

[5] E. Afgan, J. Goecks, D. Baker, N. Coraor, A. Nekrutenko, J. Taylor, G. Team *et al.*, "Galaxy: A gateway to tools in e-science," in *Guide to e-Science*. Springer, 2011, pp. 145–177.

[6] R. Dooley, G. Allen, and S. Pamidighantam, "Computational chemistry grid: Production cyberinfrastructure for computational chemistry," in *Proceedings of the 13th Annual Mardi Gras Conference*, 2005, p. 83.

[7] G. Klimeck, M. McLennan, S. P. Brophy, G. B. Adams III, and M. S. Lundstrom, "nanohub. org: Advancing education and research in nanotechnology," *Computing in Science & Engineering*, vol. 10, no. 5, pp. 17–23, 2008.

[8] M. Pierce, S. Marru, B. Demeler, R. Singh, and G. Gorbet, "The apache airavata application programming interface: overview and evaluation with the ultrascan science gateway," in *Proceedings of the 9th Gateway Computing Environments Workshop*. IEEE Press, 2014, pp. 25–29.

[9] S. Marru, L. Gunathilake, C. Herath, P. Tangchaisin, M. Pierce, C. Mattmann, R. Singh, T. Gunarathne, E. Chinthaka, R. Gardler *et al.*, "Apache airavata: a framework for distributed applications and computational workflows," in *Proceedings of the 2011 ACM workshop on Gateway computing environments*. ACM, 2011, pp. 21–28.

[10] S. Marru, M. Pierce, S. Pamidighantam, and C. Wimalasena, "Apache airavata as a laboratory: architecture and case study for component-based gateway middleware," in *Proceedings of the 1st Workshop on The Science of Cyberinfrastructure: Research, Experience, Applications and Models*. ACM, 2015, pp. 19–26.

[11] Apache, "Apache airavata," http://airavata.apache.org/.

[12] R. Heiland, J. Basney, and V. Welch, "Suggested security practices for scigap: A preliminary report," http://hdl.handle.net/2022/20811.

[13] P. Gutmann, "Pki: it's not dead, just resting," *Computer*, vol. 35, no. 8, pp. 41–49, 2002.

[14] D. Hardt, "The oauth 2.0 authorization framework," 2012.

[15] N. Sakimura, J. Bradley, M. Jones, B. de Medeiros, and C. Mortimore, "Openid connect core 1.0," *The OpenID Foundation*, p. S3, 2014.

[16] W. Inc, "Wso2 identity server: The first enterprise identity bus," http://wso2.com/products/identity-server/.

[17] S. Godik, A. Anderson, B. Parducci, P. Humenn, and S. Vajjhala, "Oasis extensible access control 2 markup language (xacml) 3," Tech. rep., OASIS, Tech. Rep., 2002.

[18] K. Grizzle, E. Wahlstroem, C. Mortimore, and P. Hunt, "System for cross-domain identity management: Core schema," *System*, 2015.

[19] A. Airavata, "Apache airavata security implementation," https://github.com/apache/airavata/tree/master/airavata-api/airavata-api-server/src/main/java/org/apache/airavata/api/server/security.

[20] R. Vanbrabant, *Google Guice: agile lightweight dependency injection framework*. Apress, 2008.

[21] T. A. Kanewala, S. Marru, J. Basney, and M. Pierce, "A credential store for multi-tenant science gateways," in *Cluster, Cloud and Grid Computing (CCGrid), 2014 14th IEEE/ACM International Symposium on*. IEEE, 2014, pp. 445–454.

[22] Y. Fan, S. Pamidighantam, and W. Smith, "Incorporating job predictions into the seagrid science gateway," in *Proceedings of the 2014 Annual Conference on Extreme Science and Engineering Discovery Environment*. ACM, 2014, p. 57.

[23] B. Demeler and G. E. Gorbet, "Analytical ultracentrifugation data analysis with ultrascan-iii," in *Analytical Ultracentrifugation*. Springer, 2016, pp. 119–143.

[24] J. Basney, R. Dooley, J. Gaynor, S. Marru, and M. Pierce, "Distributed web security for science gateways," in *Proceedings of the 2011 ACM workshop on Gateway computing environments*. ACM, 2011, pp. 13–20.

[25] J. Basney, J. Gaynor, S. Marru, M. Pierce, T. A. Kanewala, R. Dooley, and J. Stubbs, "Integrating science gateways with xsede security: A survey of credential management approaches," in *Proceedings of the 2014 Annual Conference on Extreme Science and Engineering Discovery Environment*. ACM, 2014, p. 58.

[26] S. Gesing, R. Grunzke, J. Krüger, G. Birkenheuer, M. Wewior, P. Schäfer, B. Schuller, J. Schuster, S. Herres-Pawlis, S. Breuers *et al.*, "A single sign-on infrastructure for science gateways on a use case for structural bioinformatics," *Journal of Grid Computing*, vol. 10, no. 4, pp. 769–790, 2012.

[27] J. Basney, V. Welch, and N. Wilkins-Diehr, "Teragrid science gateway aaaa model: implementation and lessons learned," in *Proceedings of the 2010 TeraGrid Conference*. ACM, 2010, p. 2.

[28] K. Chard, M. Lidman, B. McCollam, J. Bryan, R. Ananthakrishnan, S. Tuecke, and I. Foster, "Globus nexus: A platform-as-a-service provider of research identity, profile, and group management," *Future Generation Computer Systems*, vol. 56, pp. 571–583, 2016.