Future Generation Computer Systems xxx (xxxx) xxx

Contents lists available at ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

Managing authentication and authorization in distributed science gateway middleware

Marcus A. Christie [a], Anuj Bhandar [b], Supun Nakandala [c], Suresh Marru [a,*], Eroma Abeysinghe [a], Sudhakar Pamidighantam [a], Marlon E. Pierce [a]

[a] Science Gateways Research Center, Pervasive Technology Institute, Indiana University, Bloomington, Indiana 47408, USA
[b] Intuit, USA
[c] University of California, San Diego, USA

ARTICLE INFO

Article history:
Received 19 June 2018
Received in revised form 8 July 2019
Accepted 10 July 2019
Available online xxxx

Keywords:
Identity and access management
Keycloak
Apache Airavata
Science gateways
Authentication and authorization
Cyberinfrastructure

ABSTRACT

Establishing users' identities and determining their permissions before they access research infrastructure resources are key features of science gateways. With many science gateways now relying on general purpose gateway platform services, the challenges of managing identity-derived features have expanded to include network-based authentication and authorization scenarios that connect science gateway tenants, science gateway platform middleware, and third party identity provider services, including campus identity management systems. This paper examines both architectural and implementation considerations for integrating these services. We provide a summary case study that further shows how end-to-end authentication and authorization can be provided between gateways, campus authentication systems, science gateway middleware, and campus computing resources. We conclude with observations on lifecycle management of third party components in science gateway platform services, which is an important consideration for both selection of new technologies and transitioning from older systems.

1. Introduction

Managing user identity is a critical feature for science gateways [1,2], which must provide secure and auditable access to restricted resources such as supercomputers, data sets, licensed scientific applications, and for-fee computing clouds. Science gateways must authenticate users, decide if they are authorized to access specific resources, manage expired accounts, and disable compromised accounts. The basic approach is for a gateway to provide its own user management and authentication system that is an integral part of the gateway's implementation. For example, a gateway that builds over a more general purpose framework such as Django, Drupal, or Joomla may use authentication add-ons for managing users. Gateway developers today have several additional options, building on larger trends. First is the emergence of well-supported authentication services, such as the InCommon Federation [3,4] that is used by many academic institutions. Facebook, Google, GitHub and other Web-based companies also provide free authentication services that can be integrated into online applications. OpenID Connect [5] has become a popular protocol for Web authentication; it builds over the OAuth 2 authorization protocol [6]. CILogon [7] provides a unifying authentication layer over these different providers. Thus, gateways may outsource user authentication to various services. The gateway may still choose to manage its users internally through a user store (such as an attached database or LDAP server), or it may outsource this as well; a campus-centered gateway may for example connect to a user account system (such as LDAP) managed by the campus cluster providers. The second important trend has been the emergence of science gateway platform-as-a-service offerings. These are hosted services that can serve multiple gateway tenants simultaneously. Science gateway platforms provide general purpose services such as user management, data management, and job execution, while the gateway tenant provides specific access to computational and storage resources, data and applications and user interfaces geared towards a specific user community. Gateway tenants access the gateway platform middleware through secure, network accessible APIs. Various patterns for interactions between gateway tenants and gateway middleware are reviewed in [8,9], which can be mapped to different OAuth 2 authorization grant flows. This paper considers how to implement basic interactions between a gateway tenant and multi-tenanted middleware; we use Apache Airavata middleware [10] to illustrate the main concepts, as described in

* Corresponding author.
E-mail addresses: machrist@iu.edu (M.A. Christie), anujbhan@apache.org (A. Bhandar), snakanda@eng.ucsd.edu (S. Nakandala), smarru@iu.edu (S. Marru), eabeysin@iu.edu (E. Abeysinghe), pamidigs@iu.edu (S. Pamidighantam), marpierc@iu.edu (M.E. Pierce).

https://doi.org/10.1016/j.future.2019.07.018
0167-739X/

the following section. Apache Airavata is open source software that the authors operate as part of the Science Gateways Platform as a service (SciGaP.org) project. We use Keycloak [11], an open source identity management system, to implement SciGaP's Identity and Access Management system. Keycloak can authenticate users through a number of different mechanisms, replacing our previous approach based on WSO2 Identity Server (WSOS IS) [12] that was described in [8].

2. Gateway identity management and Apache Airavata

Apache Airavata is an open source software framework that enables gateway providers to compose, manage, execute, and monitor large scale applications and workflows on distributed computing resources such as local clusters, supercomputers, computational grids, and computing clouds. Apache Airavata is middleware that runs separately from the computational resources that it serves; a single gateway can use many different backend systems to execute jobs. Science gateways access Apache Airavata through a network accessible, programming language-independent Application Programming Interface (API). Apache Airavata middleware can support multiple science gateway tenants simultaneously, with each tenant maintaining its own user store, backend resources, and internal metadata description instances for resources and available software. The authors operate a hosted version Apache Airavata and other services (such as Keycloak) needed to run a production version of the system for client gateways.

Fig. 1 provides a high level summary of Apache Airavata. Users authenticate to a gateway tenant, which in turn interacts with Apache Airavata's API server. The API server routes requests to the appropriate internal components, which provide access to data and metadata, execute applications on remote resources, manage security credentials for accessing remote resources, move data between resources, and manage notifications between components. Internal components in Airavata are distributed and typically exposed through Apache Thrift-based component programming interfaces [13,14]. Airavata security is handled in two phases. Gateway providers and power users can deploy computational resource credentials to Airavata as a one time registration step. Users authenticate and authorize with the gateway and Airavata services to use these credentials to perform computational tasks. Gateways interact with Airavata over secured API implemented with standard compliant OAuth protocols, as described in [8].

The general interaction between Web-based gateway tenants and Apache Airavata middleware is shown in Fig. 2; see [8] for a full discussion. When a user logs in (Step 1), the gateway contacts the Authorization Server (Step 1a), which redirects to a federated authentication service such as CILogon (Step 1b) which in turn redirects to institutional identity system (Step 1c). After a successful authentication (Step 1d), the Authorization Server returns an access token to the gateway. The gateway can use this access token to request services through the API server (Step 2). The API server validates the access token (Step 2a) and may make additional authorization decisions.

In addition to establishing user identity, the other core activity is securing calls between the science gateway tenant and the Apache Airavata middleware. Airavata's Application Programming Interface (API) contains methods for both users and gateway administrators. Regular gateway users, for example, set up and execute computational experiments through API calls embedded in the gateway, while administrators have access to API methods for modifying metadata about the gateway tenant's available computing resources and software. Thus before executing an API method, Apache Airavata needs to establish that the user has successfully logged in and that the user has the appropriate permissions.

The three major components in Fig. 2 (the Science Gateway, the Keycloak Authorization Server, and Apache Airavata) are not co-located and communicate over public networks, so we must establish trust between the gateway tenant, the Authorization Server and Airavata. This is done in the following layers. Briefly, we must establish secure connections between the components of Fig. 2 (preventing man-in-the-middle and similar attacks), and we must also verify the communications from the tenant that are initiated by user interactions through the gateway.

First, all communication between components uses TLS [15] for connection-level security. To make tenant registration more scalable, we do not use mutual authentication at this layer as this requires operating a Public Key Infrastructure (PKI) [16] and only covers point-to-point authentication between services. Second, the gateway tenant is registered as an OpenID Connect client with the Authorization Server and uses its client ID and secret to securely obtain an access token (see [5,6]). Third, Airavata uses the access token to call the userinfo endpoint of the Authorization Server and verifies that the username in the request matches the username returned in the userinfo response. Airavata's Security Manager figures out the userinfo endpoint based on the gateway id passed in the request. Therefore, if the access token is able to be used to successfully retrieve a response, then both the gateway id and the user's username have been verified.

3. Implementation details and related considerations

As discussed in [13], we believe that science gateway middleware should leverage best of breed third-party systems whenever possible, and gateway middleware developers should not reinvent their own versions of subsystems (such as messaging or authentication). Science gateways and science gateway platforms are integration technologies that need to provide a wide range of services, including very prominently authentication and authorization. Selecting third party technologies is thus an extremely important activity. Over time, third party software will also need to be replaced in favor of new solutions for various reasons. Thus it is also important to future-proof the integrating gateway system to effectively shield itself from this switchover between systems.

Identity management in Apache Airavata provides a concrete example. In previous work [8], we implemented the architecture illustrated in Fig. 2 using WSO2 IS to manage authentication and authorization. We were motivated however to find a replacement service when we encountered technical difficulties integrating with CILogon. In theory, WSO2 IS supports external identity providers, but in practice we encountered difficulties configuring the PKI trust store so that it would accept the signing Certificate Authority (CA) of CILogon's SSL certificate.

We identified an alternative technology as a replacement. Keycloak offers a feature set very similar to WSO2 IS. For example, Keycloak provides a user store and administrative functions for administering users, including user roles. Keycloak also provides some interesting new features. For example, Keycloak also supports identity federation for identity providers that support OpenID Connect or SAML.

Another interesting feature is integration with CILogon's administration API. The CILogon service provides an OpenID Connect service including a self-service ability to register clients and OpenID Connect endpoints for retrieving user information for further use. Once an administrator registers a web portal with CILogon, the client ID and client secret can be used to setup and configure a CILogon identity provider in Keycloak. This enables new gateway tenants to set up CILogon integration without requiring any manual steps by SciGaP administrators.

Please cite this article as: M.A. Christie, A. Bhandar, S. Nakandala et al., Managing authentication and authorization in distributed science gateway middleware, Future Generation Computer Systems (2019), https://doi.org/10.1016/j.future.2019.07.018.

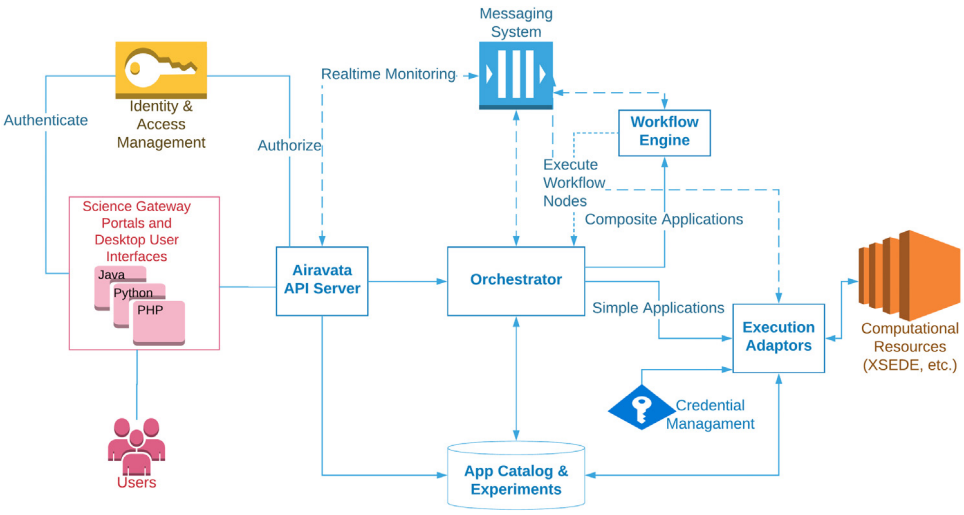

Fig. 1. Apache Airavata's conceptual framework.

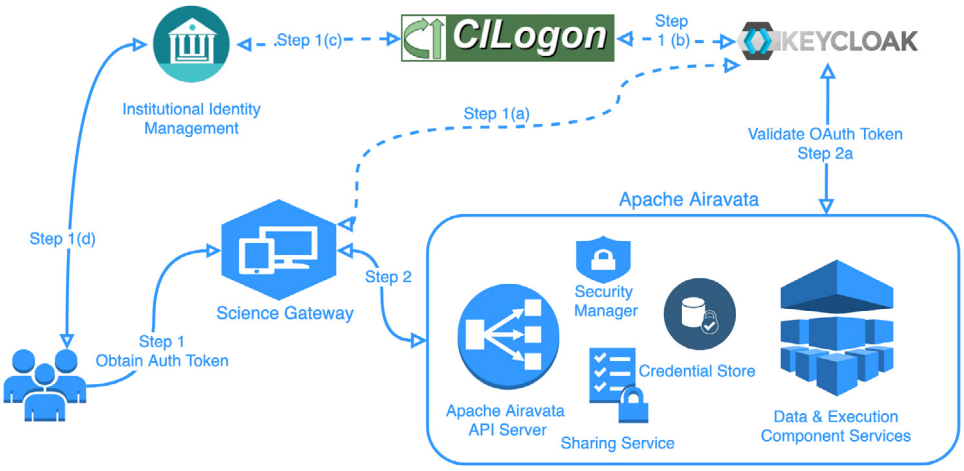

Fig. 2. Gateway-middleware integration using OpenID Connect and OAuth2.

Users of a gateway tenant who authenticate via CILogon are automatically provisioned in Keycloak's user store. Once the user is authenticated during the signin process, Keycloak redirects back to the web portal with an authorization code. The web portal can then use the authorization code to get an access token and retrieve the user's profile from Keycloak. This user profile (first name, last name, email address) will match the details that Keycloak itself retrieves from CILogon.

Users are assigned to one or more roles to grant them access to different subsets of Apache Airavata API methods. Keycloak exposes a REST API that allows a gateway administrator to manage a user's roles. This functionality is exposed in the web portal to users with the admin role. Admins are able to manage the roles assigned to a user. Typically new users are assigned to a role which has no API access and a decision must be made by an admin user as to which role(s) to assign the user.

User authorization occurs in the Apache Airavata API server, which securely brokers requests to other Airavata services. When the gateway tenant calls an API server method, it passes the user's access token. The API server first makes a call to Keycloak to verify the access token is valid. If the access token is valid a second call is made to Keycloak to get a list of roles that are assigned to the user. The API server has a list of API methods that are accessible to each role. The request is authorized if the user has a role that can access the given API method.

Another design goal we had was to use Keycloak as a backend service so that the user is never exposed to the Keycloak user interfaces. The reason for not exposing the user to the Keycloak user interface is simply to avoid needing to build user trust in this additional authentication service and thus avoid user confusion.

When a user is doing a standard username–password login, this is accomplished by using a Resource Owner Password Credentials grant flow by which the web portal directly submits the username and password to Keycloak and gets a code that can be exchanged for an access token. When a user is logging in through CILogon, the web portal redirects to Keycloak with a special query parameter indicating to Keycloak which identity provider to redirect to for the user's authentication. Thus, the web portal redirects to Keycloak, which immediately redirects to CILogon, and the user never sees the Keycloak login page.

Based on our past experience with third party identity providers like WSO2 IS, a major design goal with the Keycloak integration was to create abstractions for all the functionalities needed by Airavata for managing users and their roles. Calls to the Keycloak REST API are made indirectly through defined interfaces, providing a layer of separation between the core Airavata code and the Keycloak API. This design will shield Airavata from creating provider-specific dependencies and enable easy replaceability. As a part of this initiative, we developed interfaces for Tenant Management, client configuration and user management.

Please cite this article as: M.A. Christie, A. Bhandar, S. Nakandala et al., Managing authentication and authorization in distributed science gateway middleware, Future Generation Computer Systems (2019), https://doi.org/10.1016/j.future.2019.07.018.

Table 1
API methods for managing new tenant requests.

Method name	Description
addTenant	Creates a new realm in Keycloak and creates default realm level roles for the realm
createTenantAdminAccount	Creates an admin account for the realm. This is a special utility user account that is used to make REST API calls to Keycloak for this realm. (The REST API calls to create this admin account are made using an admin account of the master realm.)

Table 2
Client configuration API method.

Method name	Description
configureClient	Creates a default client to be used by the web portal. This also adds allowed redirect URIs for this client, to match the domain name of the science gateway when the tenant was created.

Table 3
API methods for user management.

Method name	Description
createUser	This is used to create a local user account. The user account is created in a disabled state and only enabled when the email address is verified (see enableUserAccount).
enableUserAccount	Enables the local user account when email address is verified.
isUserAccountEnabled	Check if account is enabled or not
resetUserPassword	Updates the Keycloak User's Credential record with a new password
findUser	Search for Keycloak users by username or email address
updateUserProfile	Updates the Keycloak User's profile record (called when the user updates their first name or last name, for example)
addRoleToUser	Adds a realm role to the user
removeRoleFromUser	Removes a realm role from the user
getUsersWithRole	Gets a list of all users with the given realm role
getUserRoles	Gets a list of roles for the user with the given username

Each of these interfaces will also help automate the process of creating a new gateway without the need of an administrator to manually configure the new tenant identity provider's admin console.

For tenant management we introduced the interface methods shown in Table 1.

For client configuration we introduced the interface method shown in Table 2.

For user management we introduced the interface methods shown in Table 3.

4. Case study: LDAP integration for end-to-end authentication

A larger security problem for science gateways is end-to-end authentication. Apache Airavata-based gateways and many others support remote executions of computational jobs on supercomputers. In the XSEDE model [17], these jobs run under community accounts. The user typically does not have an account on the target resource and may not know it is even being used. The gateway or gateway middleware is accountable for proper usage and reporting of the community account. However, this model is not adopted widely outside XSEDE, and many resource providers (such as campus clusters) require users to use their own accounts through the gateway in order to comply with university policies.

In our architectural approach, as illustrated in Fig. 1, there are two authentication and authorization steps that are decoupled: (a) the authentication of a user of a gateway and his or her subsequent authorization to use an Apache Airavata API method, and (b) the authentication and authorization between the Apache Airavata middleware and the target computing resource.

Apache Airavata servers run independently of any particular computing resource and can allow users to execute software on multiple resources; communications use SSH and SCP and build on the JSch libraries [18] to optimize these connections. Access to remote resources is done with public–private key pairs. Airavata's Credential Store [19] creates and manages the key pairs, and the public key can be exported to be installed on the target resource. This step is manual and does not scale well when community accounts cannot be used.

We describe here an approach that we are using for one gateway tenant, the Indiana University Cyberinfrastructure Gateway, which links a user's Web authentication (through IU's Central Authentication Service via CILogon) with access to IU clusters over SSH via keys managed by an LDAP server maintained by IU's Research Technologies group. This approach provides an end-to-end integration in which the user's gateway authentication is tied directly to subsequent authentications to the end resource, and it may be of general interest and a model that can be used by other resource providers.

In Airavata's metadata schema [20], an SSHAccountProvisioner can be configured for each compute resource used by a gateway. The SSHAccountProvisioner interface defines methods for checking if a user has an account on that compute resource as well as methods for creating an account or installing an SSH key on the compute resource.

For the IU Cyberinfrastructure Gateway we implemented an SSHAccountProvisioner that will update an entry for the user with an SSH public key in the compute cluster's LDAP server. The compute cluster's SSHD daemon is configured to authenticate SSH connections against public keys in the cluster's LDAP server (in addition to the usual method of comparing against the user's authorized_keys file). The SSH key is generated by Airavata's Credential Store service on behalf of the user and the public part is stored in the LDAP server. This SSHAccountProvisioner is not able to create the cluster account directly, but it can query the LDAP server to see if the user has an account. This information is then used in the gateway to inform the user if they need to request an account and to provide the user details and where to request an account (a web form must be submitted on a separate IU web application to request a cluster account).

The IU Cyberinfrastructure Gateway only allows users with an IU account to log into the gateway. Authentication is performed by CILogon which redirects the user's browser to IU's Central Authentication Service page. Airavata connects securely to the compute cluster's LDAP repository using TLS and a username and password, credentials that are also stored in Airavata's Credential Store. The LDAP repository has a firewall but access between the Airavata servers and the LDAP repository was opened up to allow this communication.

5. Related work

Science gateway security has been examined in [2,17]. These works are the basis of the current paper but do not consider the details of multi-tenanted science gateway middleware.

Our previous work [8] considered a larger number of gateway-middleware integration use cases. The basic case, depicted in Fig. 2, assumes the user store is attached to the Authorization server. Other cases include user stores attached to the gateway and user stores provided by external services (such as an

Please cite this article as: M.A. Christie, A. Bhandar, S. Nakandala et al., Managing authentication and authorization in distributed science gateway middleware, Future Generation Computer Systems (2019), https://doi.org/10.1016/j.future.2019.07.018.

LDAP server maintained by a department). Ref. [8] also examined other OAuth2 grant flow cases such as those that occur when integrating desktop interfaces.

The Globus Auth [21] service is a software-as-a-service system that implements many of the same capabilities as Keycloak and WSO2 IS. Globus Auth additionally provides support for groups and is integrated with other Globus services such as file transfer. Keycloak and WSOS IS are open source software that can be used by gateways and middleware operators to offer identity management services. It is possible to integrate these with Globus Auth as well by making Globus Auth a trusted identity provider. This would enable an Apache Airavata-based gateway to use Keycloak-based identity management services and Globus file transfer services.

With cloud platform services proliferating through enterprises, similar challenges to the ones described here for gateway tenants accessing platform services exist within industry, and solutions exist as well. Cloud providers such as Amazon have bundled solutions, and a few companies have dedicated standalone solutions similar to the ones described in this paper. Auth0 [22] and Okta [23] are the leaders in the commercial sector. While these services are good for out of box solutions, they do not necessarily cater to academic needs such as integrating with campus identity management systems. Moreover, these services are very expensive [24,25] and are proprietary.

Our primary consideration in this paper is authentication and its use in enforcing API level authorization between a gateway tenant and a multi-tenanted science gateway middleware like Apache Airavata. However, there are many other considerations beyond API access. For example, a user may or may not have access to a particular computing resource or piece of software, which involves the same API methods but not the same parameters. This type of authorization goes beyond the semantics of API calls. We enforce these with group-based authorization based on the Apache Airavata Sharing Service [26].

6. Conclusions

This effort touches on two larger challenges faced by science gateways. First, gateways no longer need to implement all required capabilities themselves. We refer to this as the build versus buy decision, in which a gateway development team decides if they should use a third party piece of software or service or if they should build what they need from scratch. The case under consideration, identity management, has matured significantly over the last decade, and there are a number of high quality solutions that a gateway can choose from. The case for integrating third party identity management and related software and services is especially strong, given the importance of cybersecurity to the gateway client, the middleware provider, and the resource providers.

Although the buy option (that is, use a third party software or service) has many advantages over the build option over the long term, it is inevitable that the gateway will need to replace a solution over time. In our case, WSO2 IS worked well for our initial use cases, and we put it successfully into production, but it failed to support new requirements that we did not initially consider. Furthermore, even though WSO2 IS is open source software, we realized that the modifications needed to implement our use case were too much of a burden to implement and maintain ourselves, and we furthermore needed to trust WSO2's open source governance model to make sure any changes we made would be integrated into the main code base. Creating a branch of an open source project that only we maintain is not a sustainable option.

Replacing WSO2 IS with Keycloak, even though the products have similar capabilities, was complicated by the use of WSO2 IS-specific code within our reference implementation gateway. Keycloak also does not support XACML, which we used in WSO2 IS to make access control decisions on API access. Instead the API server uses Keycloak's REST API to apply a simple role based access control as described above. This complication could have been ameliorated by providing more wrapping coding to isolate implementation details, which we have now done as described in Section 3.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgment

This work is supported by NSF, USA Award #1339774.

References

[1] M.E. Pierce, M.A. Miller, E.H. Brookes, W. Wong, L.Y. Afgan, E.S. Gesing, M. Dahan, S. Marru, T. Walker, Towards a science gateway reference architecture, in: Proceedings of the 10th International Workshop on Science Gateways, IWSG, 2018.
[2] J. Basney, V. Welch, Science gateway security recommendations, in: Cluster Computing (CLUSTER), 2013 IEEE International Conference on, IEEE, 2013, pp. 1–3.
[3] InCommon Federation (Accessed June 18, 2018). URL https://www.incommon.org/.
[4] W. Barnett, V. Welch, A. Walsh, C.A. Stewart, A roadmap for using nsf cyberinfrastructure with incommon, Tech. rep. (2011).
[5] N. Sakimura, J. Bradley, M. Jones, B. de Medeiros, C. Mortimore, Openid connect core 1.0 incorporating errata set 1, The OpenID Foundation, specification, 2014.
[6] D. Hardt, The oauth 2.0 authorization framework, 2012.
[7] J. Basney, T. Fleury, J. Gaynor, Cilogon: A federated x. 509 certification authority for cyberinfrastructure logon, Concurr. Comput.: Pract. Exper. 26 (13) (2014) 2225–2239.
[8] S. Nakandala, H. Gunasinghe, S. Marru, M. Pierce, Apache airavata security manager: Authentication and authorization implementations for a multi-tenant escience framework, in: E-Science (E-Science), 2016 IEEE 12th International Conference on, IEEE, 2016, pp. 287–292.
[9] R. Heiland, S. Koranda, S. Marru, M. Pierce, V. Welch, Authentication and authorization considerations for a multi-tenant service, in: Proceedings of the 1st Workshop on the Science of Cyberinfrastructure: Research, Experience, Applications and Models, ACM, 2015, pp. 29–35.
[10] 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.
[11] Keycloak (Accessed June 18, 2018). URL http://www.keycloak.org/.
[12] WSO2 Identity Server (Accessed June 18, 2018). URL http://wso2.com/identity-and-access-management.
[13] M.E. Pierce, S. Marru, L. Gunathilake, D.K. Wijeratne, R. Singh, C. Wimalasena, S. Ratnayaka, S. Pamidighantam, Apache airavata: design and directions of a science gateway framework, Concurr. Comput.: Pract. Exper. 27 (16) (2015) 4282–4291.
[14] S. Marru, M. Pierce, S. Pamidighantam, 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.
[15] T. Dierks, E. Rescorla, The transport layer security (tls) protocol version 1.2, Tech. rep. (2008).
[16] R. Housley, W. Ford, W. Polk, D. Solo, Internet x. 509 public key infrastructure certificate and crl profile, Tech. rep. (1998).
[17] V. Welch, J. Barlow, J. Basney, D. Marcusiu, N. Wilkins-Diehr, A aaaa model to support science gateways with community accounts, Concurr. Comput.: Pract. Exper. 19 (6) (2007) 893–904.
[18] JSch (Accessed June 18, 2018). URL http://www.jcraft.com/jsch/.

[19] T.A. Kanewala, S. Marru, J. Basney, 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.
[20] Apache Airavata API Data Model Definitions (Accessed June 18, 2018). URL https://github.com/apache/airavata/tree/master/thrift-interface-descriptions/data-models.
[21] S. Tuecke, R. Ananthakrishnan, K. Chard, M. Lidman, B. McCollam, S. Rosen, I. Foster, Globus auth: A research identity and access management platform, in: E-Science (E-Science), 2016 IEEE 12th International Conference on, IEEE, 2016, pp. 203–212.
[22] auth0 (Accessed June 18, 2018). URL https://auth0.com/.
[23] Okta (Accessed June 18, 2018). URL https://www.okta.com/.
[24] Auth0 Pricing (Accessed June 18, 2018). URL https://auth0.com/pricing.
[25] Okta Pricing (Accessed June 18, 2018). URL https://www.okta.com/pricing/.
[26] S. Nakandala, S. Marru, M. Piece, S. Pamidighantam, K. Yoshimoto, T. Schwartz, S. Sivagnanam, A. Majumdar, M.A. Miller, Apache airavata sharing service: A tool for enabling user collaboration in science gateways, in: Proceedings of the Practice and Experience in Advanced Research Computing 2017 on Sustainability, Success and Impact, ACM, 2017, p. 20.

Suresh is Deputy Director of the Science Gateway Research Center at Indiana University and a nominated Member of the Apache Software Foundation and vice-president of the Apache Airavata project. His research interest is to advance the deep and wide boundaries of computational and data sciences.

Please cite this article as: M.A. Christie, A. Bhandar, S. Nakandala et al., Managing authentication and authorization in distributed science gateway middleware, Future Generation Computer Systems (2019), https://doi.org/10.1016/j.future.2019.07.018.