SciTokens SSH: Token-based Authentication for Remote Login to Scientific Computing Environments

You Alex Gao
University of Illinois
yougao2@illinois.edu

Jim Basney
NCSA
jbasney@illinois.edu

Alex Withers
NCSA
alexw1@illinois.edu

ABSTRACT

SciTokens SSH is a pluggable authentication module (PAM) that uses JSON Web Tokens (JWTs) for authentication to the Secure Shell (SSH) remote login service. SciTokens SSH supports multiple token issuers with local token verification, so scientific computing providers are not forced to rely on a single OAuth server for token issuance and verification. The decentralized design for SciTokens SSH was motivated by the distributed nature of scientific computing environments, where scientists use computational resources from multiple providers, with a variety of security policies, distributed across the globe.

CCS CONCEPTS

• **Security and privacy → Authorization**.

KEYWORDS

SSH, OAuth, JWT, PAM, distributed computing

ACM Reference Format:
You Alex Gao, Jim Basney, and Alex Withers. 2020. SciTokens SSH: Token-based Authentication for Remote Login to Scientific Computing Environments. In *Practice and Experience in Advanced Research Computing (PEARC '20), July 26-30, 2020, Portland, OR, USA.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3311790.3399613

1 INTRODUCTION

Secure Shell (SSH) [14] is widely used for distributed scientific computing, enabling scientists to log in to supercomputers to compile, optimize, and run simulations, to transfer scientific data sets to/from Science DMZs, to connect with scientific computing resources through JupyterHub, OpenOnDemand, etc., and to forward network ports to securely access visualization systems, to name a few common uses. SSH supports many authentication methods, including long-lived passwords, one-time passwords, public keys, Kerberos tickets, and X.509 certificates. Each one of these SSH authentication methods has been adopted by the distributed scientific computing community over the years. More recently, OAuth [5] has become a preferred authentication (and authorization) method, due to its design as a web-native (REST) protocol, using the standard JSON Web Token (JWT) format [6].

PEARC '20, July 26-30, 2020, Portland, OR, USA

ACM ISBN 978-1-4503-6689-2/20/07…$15.00
https://doi.org/10.1145/3311790.3399613

In this article, we present the design and open source implementation of SciTokens SSH, an SSH version that uses JWTs for authentication. SciTokens SSH supports multiple token issuers with local token verification, so scientific computing providers are not forced to rely on a single OAuth server for token issuance and verification. The decentralized design for SciTokens SSH was motivated by the distributed nature of scientific computing environments, where scientists use computational resources from multiple providers, with a variety of security policies, distributed across the globe.

Our article is organized as follows. In Section 2, we present related work, including the existing software components that we used to implement SciTokens SSH. Then in Section 3, we describe our implementation and compare it to alternatives. Next, in Section 4, we present our threat model and associated security evaluation. Lastly, we present conclusions and future work in Section 5.

2 RELATED WORK

For over 20 years, GSI-OpenSSH[1] has been the de facto standard solution for remote login to scientific computing resources. GSI-OpenSSH is a modified version of OpenSSH that adds support for authentication and credential forwarding (delegation) using X.509 proxy certificates [10] as part of the Grid Security Infrastructure (GSI) [11]. GSI-OpenSSH suffers from two major drawbacks: 1) it requires use of modified SSH clients and servers, and 2) it uses "impersonation" credentials, so an exposed proxy certificate yields access to all the user's logins globally (i.e., the opposite of least privilege).

Globus Auth SSH [1] aims to address the drawbacks of GSI-OpenSSH using web-native authentication. It provides a pluggable authentication module (PAM) that accepts OAuth tokens for authentication, so it is compatible with standard SSH clients and servers that support PAM ("keyboard-interactive") authentication. Specifically, it supports opaque tokens from the Globus Auth [9] service, which enables logins from many institutional identity providers. The PAM module uses OAuth token introspection [8] to validate the opaque token, including the target account and server (audience). Thus, Globus Auth SSH addresses the GSI-OpenSSH drawbacks by 1) supporting unmodified SSH clients and servers, and 2) using per-server (least-privilege) tokens. However, Globus Auth SSH introduces a new drawback: it relies on a single OAuth token server for token issuance and verification, which is a poor match for large-scale distributed scientific computing environments. SciTokens SSH aims to address that drawback. Globus Auth SSH has recently been donated to the XSEDE project and renamed XSEDE OAuth SSH.[2]

[1] https://github.com/globus/gsi-openssh
[2] https://github.com/XSEDE/oauth-ssh

```
{
  "scope":"ssh:vt20",
  "aud":"martok.ncsa.illinois.edu",
  "iss":"https://demo.scitokens.org",
  "exp":1583855836,
  "iat":1583855236,
  "nbf":1583855236,
  "jti":"073ac358-4f07-4090-ae5f-b5c5be273269"
}
```

Figure 1: An example SciToken issued by demo.scitokens.org that enables user vt20 to log in to server martok.ncsa.illinois.edu

SciTokens [12, 13] is a JWT profile and associated open source implementation[3] of least-privilege OAuth tokens for distributed scientific computing. Instead of using opaque tokens, SciTokens uses digitally-signed, self-describing JWTs, so SciTokens-enabled services can locally verify tokens from multiple issuers, to support large scale distributed scientific computing environments. The SciTokens JWT profile has been incorporated into the WLCG Common JWT Profiles [2] for international interoperability. SciTokens SSH uses the SciTokens JWT profile and SciTokens open source libraries.

oidc-agent[4] provides a set of tools to manage OpenID Connect (OIDC) and OAuth tokens and make those tokens easily usable from the command line. It follows the ssh-agent design, so users can handle OIDC/OAuth tokens in a similar way as they do with SSH keys. While oidc-agent does not (yet) directly integrate with SSH, it provides the very useful capability of managing multiple tokens, which is needed for users who log in to multiple SSH servers.

3 IMPLEMENTATION

The OAuth SSH server package provides a PAM module that is designed to respond to a challenge prompt requested from the client and then receive an OAuth token from the client in response to that challenge. SciTokens SSH is built on XSEDE OAuth SSH. SciTokens SSH and XSEDE OAuth SSH currently support CentOS 7.

The original OAuth SSH expects an opaque Globus token and will require registering the SSH service on the Globus developers console as well as authorizing the host with their SSH client. The SciToken implementation intercepts the verification logic as soon as the PAM module receives the token from SSHD and uses the SciTokens-CPP library [3] to verify the token. SciTokens use JSON web Tokens [6] as its token format. As shown in Figure 1, two claims need to be specified by the token issuer. The first one is scope and the value of the scope field will be a string: "ssh" followed by a username for which the user of the token will be logging in as on the host. The second key is the audience which is the SSH hostname.

There are two other key differences between Globus Auth SSH and SciTokens SSH. First, the SciTokens verification process is completely independent of Globus Auth and does not require registration on the Globus developers console nor registering a fully qualified domain name with Globus. Instead, SciTokens verifies tokens directly in the PAM module using the SciTokens-CPP library. Second, there is no username mapping in SciTokens SSH. Mapping in SciTokens SSH occurs at token issuance time rather than in the SSH server.

As illustrated in Figure 2, the verification flow starts with the authorization server issuing a token to a user. After receiving a token from the authorization server, the user can use any SSH client application and cut-n-paste the token after to respond to the prompt from the host. The prompt is generate by SSHD's call to pam_authenticate(). When pam_authenticate is called, libpam reads the configuration file /etc/pam.conf and determines the modules that participate in the operation. In this case, we will have multiple modules defined for the service's operation and these modules are called a PAM stack.

In order to route SSH authentication requests to the PAM module provided by the server package pam_oauth_ssh.so, the user is required to configure PAM to use pam_oauth_ssh.so for SSHD authentication. Figure 3 illustrates an example of necessary modifications to PAM on a fresh EL7 installation. This is the PAM stack mentioned in the last paragraph. In addition to the PAM configuration, SSHD must also be configured to use PAM and ChallengeResponseAuthentication protocol in /etc/ssh/sshd_config. Notice that if ChallengeResponseAuthentication in SSHD configuration is set, the password from the client is ignored. However, if the PAM service on the host has both password and challenge-response authentication enabled, the PAM service will prompt for the user a second time if challenge-response authentication initially fails.

An overview of the auth stack for the login service: Auth specifies the module providing two aspects of authenticating the user. First, establish the user's identity by prompting the user for a password, or a token in this case. Second, grant privileges through credential granting properties. In order to determine the result of this stack, the result codes of the individual modules require an integration process. Modules in the stack are executed in order as specified and each result is integrated into the final result using the module's control flag. For example, the first control flag of pam_sepermit.so is set to required. That means a success in meeting pam_sepermit.so's requirement is necessary and a failure will cause the overall result to fail and the login request to be declined. pam_oauth_ssh.so has a relatively more complicated control flag (success,maxtries,new_authtok_reqd,default) which corresponds to the return code from pam_oauth_ssh.so and they are all PAM error codes. For example, maxtries is corresponding to PAM_MAXTRIES as defined as "One or more of the authentication modules has reached its limit of tries authenticating the user. Do not try again." The die, done, and ignore settings, on the other hand, are the action to take after receiving a code. For example, die tells PAM to immediately return to the application and indicates that the return code specifies the module failing. Followed by pam_oauth_ssh.so, the uid >= 1000 quiet_success option restricts the login request to non-system accounts. (In CentOS SYS_UID_MAX, the upper limit of IDs that can be used for the creation of system users is usually set to 999.) pam_sepermit.so "allows or denies login depending on SELinux enforcement state".

[3] https://github.com/scitokens
[4] https://github.com/indigo-dc/oidc-agent

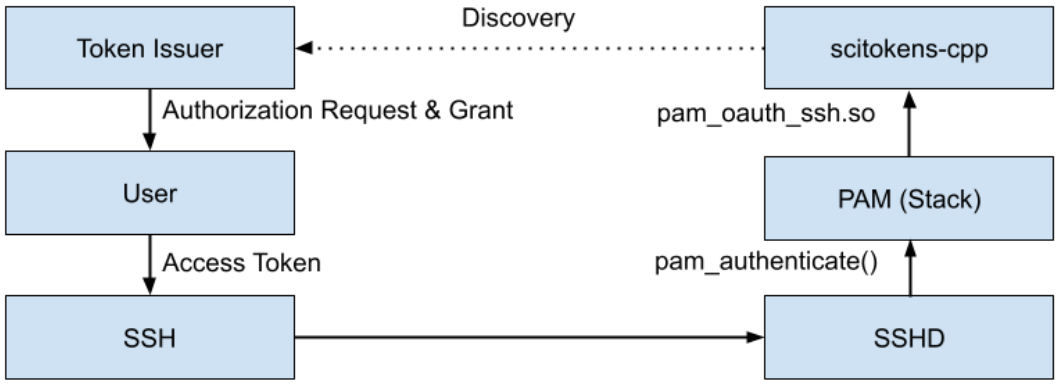

Figure 2: Verification flow

```
auth   required    pam_sepermit.so
auth   required    pam_env.so
auth   [success=done maxtries=die new_authtok_reqd=done default=ignore] pam_oauth_ssh.so
auth   requisite   pam_succeed_if.so uid >= 1000 quiet_success
auth   required    pam_deny.so
```

Figure 3: example PAM modification for OAuth SSH(Auth Stack), (/etc/pam.d/sshd)

pam_sepermit module returns PAM_IGNORE for users not matching any entry in the config file, and pam_env.so is used to set/unset the environment variables.

The pam_oauth_ssh.so module executes the SciTokens verification process. The verification function first checks if the token is a valid Scitoken [12, 13]. To verify the digital signature on the token, the verification function uses the OAuth discovery protocol to obtain metadata for the token issuer, including public signing keys [7]. The OAuth issuer metadata is cached, and the token issuer does not need to be contacted for each authentication. In addition to validating the digital signature, the verification function also checks other standard JWT criteria such as having an expiration time no later than the current time.

Three other conditions are enforced. First, the token must be issued by an issuer listed in the configuration which will be discussed below. Second, the username that the user is trying to log in as must be the same as the one in the scope field in the token. Third, the audience of the token must match the hostname. Notice that the second check is not the only user matching check, since as mentioned above, before pam_oauth_ssh.so. is executed, pam_sepermit.so will attempt to match user against list of usernames in /etc/security/sepermit.conf.

For SciTokens SSH, two additional options are required in the server-side configuration file (/etc/oauth_ssh/globus-ssh.conf): authentication method and issuers. Both SciTokens and Globus Auth tokens may be accepted by the same SSH server. The example in Figure 4 shows a configuration that allows both SciTokens issued by demo.scitokens.org and valid Globus tokens.

```
auth_method globus_auth scitokens
issuers https://demo.scitokens.org
```

Figure 4: An example OAuth SSH server configuration that accepts both Globus Auth tokens and SciTokens

The authentication method option tells PAM to either consider the input token as a Globus token or Scitokens. If both "globus_auth" and "scitokens" are present in the auth_method option field, the order of the setting is respected. Two attempts will be tried based on the user-supplied order.

XSEDE OAuth SSH supplies a wrapper script that will authenticate the user once OAuth SSH is authorized to access a host. However, in the SciTokens implementation, users will need to manually enter the token they receive.

In the following section, some potential security risks will be discussed. In order to protect the token, it will be mentioned that the user needs to keep the access tokens in transient memory and should never attempt to write tokens to disk.

4 EVALUATION

Here we evaluate the security of SciTokens SSH and present a threat model of SciTokens SSH using the OAuth 2.0 Threat Model and Security Considerations [4].

- Eavesdropping access tokens: an attacker could attempt to obtain an access token when transported between the SSH

client and server. As a countermeasure we rely on the SSH protocol to encrypt the access token via an SSH public key encrypted channel [14]. Additionally it is recommended that a short lifetime is used for the access token.

- Replay of resource server requests: an attacker could attempt to replay resource server requests to access or modify data. Again, the SSH protocol provides transport security measures as a countermeasure to this threat.
- Guessing of access tokens: an attacker could attempt to guess access tokens based on knowledge gained from other access tokens. As a countermeasure to this threat all SciToken tokens are signed JSON Web Tokens [6]. The WLCG Common JWT Profiles provides token lifetime guidance to ensure token lifetimes are set appropriately to mitigate against this and other threats [2].
- Counterfeit SSH server to phish access tokens: attackers could attempt to imitate an SSH server to which the user is attempting to access with an access token. A countermeasure to this threat exists in the SSH protocol itself in verification of SSH host keys. However, there is still a risk if a host has never been connected to and the user does not validate the host keys. An additional option would be to create a wrapper script to check the "audience" of the token. This would also protect against users pasting in or using the wrong access token. Note that this protection would not work with opaque tokens.
- Abuse of token by resource server: a legitimate resource server could use an access token to access other resource servers. For example, if a token's scope is broad then a stolen token from one compromised could be used on other resource servers. To counteract this threat the token's scope should be narrowly defined. Additionally, the audience field dictates specific resource server to be used, further narrowing the scope.
- Leakage of tokens via log files: access tokens may be logged and thus leaked for attackers to harvest. As a countermeasure, the SciTokens SSH implementation is very conservative as to what information is logged with enough information to allow administrators and user to debug issues with their connection.

Other security considerations:

- SciTokens does not encrypt its tokens and the tokens do contain user data. SciTokens relies on the SSH protocol to keep the token encrypted in transit.
- SciTokens is an assertion-based token design and adopts the JavaScript Object Notation Web Token (JWT) [6].
- In order to further protect the access tokens, user as required to copy and paste the access token into the input field when using SciTokens SSH. This keeps the access tokens in transient memory and they're never written to disk.

5 CONCLUSIONS AND FUTURE WORK

In conclusion, SciTokens SSH provides a remote login service for distributed scientific computing that supports multiple token issuers with local token verification, so scientific computing providers are not forced to rely on a single OAuth server for token issuance and verification. SciTokens SSH is a modification to Globus Auth SSH (a.k.a. XSEDE OAuth SSH) that adds support for SciTokens JWTs alongside the existing support for opaque Globus Auth tokens. We have submitted the SciTokens SSH modifications in a pull request to the XSEDE OAuth SSH project on GitHub. Future work for SciTokens SSH includes packaging and documentation to enable broader use, as well as integration with oidc-agent for client-side management of multiple tokens. Visit https://scitokens.org/ for the latest information about SciToken SSH.

ACKNOWLEDGMENTS

This material is based upon work supported by the National Science Foundation under Grant No. 1738962.

REFERENCES

[1] Jason Alt, Rachana Ananthakrishnan, Kyle Chard, Ryan Chard, Ian Foster, Lee Liming, and Steve Tuecke. 2020. OAuth SSH with Globus Auth. In *Proceedings of the Practice and Experience in Advanced Research Computing* (Portland, OR, USA) *(PEARC '20)*. ACM, New York, NY, USA, 12. https://doi.org/10.1145/3311790.3396658

[2] Mine Altunay, Brian Bockelman, Andrea Ceccanti, Linda Cornwall, Matt Crawford, David Crooks, Thomas Dack, David Dykstra, David Groep, Ioannis Igoumenos, Michel Jouvin, Oliver Keeble, David Kelsey, Mario Lassnig, Nicolas Liampotis, Maarten Litmaath, Andrew McNab, Paul Millar, Mischa Sallé, Hannah Short, Jeny Teheran, and Romain Wartel. 2019. *WLCG Common JWT Profiles*. https://doi.org/10.5281/zenodo.3460258

[3] Brian Bockelman and Derek Weitzel. 2019. scitokens/scitokens-cpp (Version v0.3.0). https://doi.org/10.5281/zenodo.2656677

[4] T. Lodderstedt (Ed.), M. McGloin, and P. Hunt. 2013. *OAuth 2.0 Threat Model and Security Considerations*. RFC 6819. https://doi.org/10.17487/RFC6819

[5] D. Hardt. 2012. *The OAuth 2.0 Authorization Framework*. RFC 6749. https://doi.org/10.17487/RFC6749

[6] M. Jones, J. Bradley, and N. Sakimura. 2015. *JSON Web Token (JWT)*. RFC 7519. https://doi.org/10.17487/RFC7519

[7] M. Jones, N. Sakimura, and J. Bradley. 2018. *OAuth 2.0 Authorization Server Metadata*. RFC 8414. https://doi.org/10.17487/RFC8414

[8] J. Richer. 2015. *OAuth 2.0 Token Introspection*. RFC 7662. https://doi.org/10.17487/RFC7662

[9] S. Tuecke, R. Ananthakrishnan, K. Chard, M. Lidman, B. McCollam, S. Rosen, and I. Foster. 2016. Globus Auth: A research identity and access management platform. In *2016 IEEE 12th International Conference on e-Science (e-Science)*. 203–212. https://doi.org/10.1109/eScience.2016.7870901

[10] S. Tuecke, V. Welch, D. Engert, L. Pearlman, and M. Thompson. 2004. *Internet X.509 Public Key Infrastructure (PKI) Proxy Certificate Profile*. RFC 3820. https://doi.org/10.17487/RFC3820

[11] V. Welch, F. Siebenlist, I. Foster, J. Bresnahan, K. Czajkowski, J. Gawor, C. Kesselman, S. Meder, L. Pearlman, and S. Tuecke. 2003. Security for Grid services. In *High Performance Distributed Computing, 2003. Proceedings. 12th IEEE International Symposium on*. 48–57. https://doi.org/10.1109/HPDC.2003.1210015

[12] Alex Withers, Brian Bockelman, Derek Weitzel, Duncan Brown, Jeff Gaynor, Jim Basney, Todd Tannenbaum, and Zach Miller. 2018. SciTokens: Capability-Based Secure Access to Remote Scientific Data. In *Proceedings of Practice and Experience on Advanced Research Computing* (Pittsburgh, PA, USA) *(PEARC '18)*. ACM, New York, NY, USA, Article 24, 8 pages. https://doi.org/10.1145/3219104.3219135

[13] Alex Withers, Brian Bockelman, Derek Weitzel, Duncan Brown, Jason Patton, Jeff Gaynor, Jim Basney, Todd Tannenbaum, You Alex Gao, and Zach Miller. 2019. SciTokens: Demonstrating Capability-Based Access to Remote Scientific Data using HTCondor. In *Proceedings of the Practice and Experience in Advanced Research Computing* (Chicago, IL, USA) *(PEARC '19)*. ACM, New York, NY, USA, Article 118, 4 pages. https://doi.org/10.1145/3332186.3333258

[14] T. Ylonen and C. Lonvick (Ed.). 2006. *The Secure Shell (SSH) Authentication Protocol*. RFC 4252. https://doi.org/10.17487/RFC4252