Guidelines on non-browser access

Published Date: 13-06-2017
Revision: 1.0

Work Package: JRA1
Document Code: AARC-JRA1.4F
Document URL: https://aarc-project.eu/wp-content/uploads/2017/03/AARC-JRA1.4F.pdf

The research leading to these results has received funding from the European Community's Horizon2020 Programme under Grant Agreement No. 653965 (AARC).

Table of Contents

1 Introduction 3

2 SSH/SFTP 4
- 2.1 GSI enabled SSH 4
- 2.2 SSH key provisioning with web portal 6

3 HTTP APIs 7
- 3.1 Accessing HTTP APIs using OIDC/OAuth2 8
- 3.2 Accessing HTTP APIs using X.509 certificates 9
- 3.3 Accessing HTTP APIs using service specific API tokens 10

4 References 10

5 Glossary 11

Table of Figures

Figure 3.1: Sequence diagram for GSI SSH. 6

Figure 3.2: Sequence diagram for key provisioning with web portal. 7

Figure 4.1: Sequence diagram for accessing HTTP APIs using OIDC/OAuth2. 9

Figure 4.2: Sequence diagram for HTTP API using X.509 certificates. 10

The research leading to these results has received funding from the European Community's Horizon2020 Programme under Grant Agreement No. 653965 (AARC).

1 Introduction

There is a need for non-browser-based federated access to resources ([AARC-DJRA1.1], requirement R14) and this type of access brings more technical problems to solve than web-browser-based access. Involving a third party (IdP) in the process of authentication and authorisation may be realised by two methods: by having the IdP issue and sign assertions prior to accessing the resource (e.g. in the form of X.509 certificates) or by querying the IdP while accessing the resource. The first method is not always feasible, or recommended, as it requires using modified software and handling of the signed assertions by the users (e.g. certificates, SAML assertions, OAuth2 token). The second method is often used for web-browser access, as features of the HTTP protocol (e.g. redirection) and client-side scripting allow its implementation. This section focuses on non-browser access to resources, which to date has often been realised using typically non-federated methods of authentication, meaning that legacy client or server software must be modified to support federated identity management.

Any writable access to resources usually require local, non-transient identities like those used by the operating system, databases or other services (e.g. iRODS). Such an identity typically requires prior provisioning (a local account must be set up) and deprovisioning if it is no longer used. During a sign-on the global identity of the user must be securely mapped to the local one. Additionally, local privileges are decided by group membership or role binding. This process may be split into two parts: token translation from the federated authentication protocol to the local authentication protocol, and translation from federated attributes to the local account name and local rights.

In all cases, the challenges involved can be grouped into the following categories:

- Account provisioning.
- Authentication:
  - Federated credentials or
  - Local credentials (still require a check if the remote credentials are up to date)
- Authorisation usually boils down to attribute mapping to local account and groups.
- Account deprovisioning means removing an account and freeing resources (e.g. data stored) that are no longer used. The analysis of this problem is outside the scope of this document.

The research leading to these results has received funding from the European Community's Horizon2020 Programme under Grant Agreement No. 653965 (AARC).

2 SSH/SFTP

SSH allows secure (encrypted) shell access to an operating system account on a remote machine over an unsecured network and SFTP allows file transfer using SSH. This technology is also used for port forwarding, proxying or restricted command execution (as used in GitHub and SVN). Typically, the authentication is based on an account name (username) and password or a pair of cryptographic keys. Additionally, if both client and server applications support Generic Security Service Application Program Interface (GSS-API)[1], another authentication mechanism may be provided. In all cases, the account on the resource server must already exist (locally or in a database store). In addition to the authentication mechanisms inside the SSH server, operating system authentication mechanisms may be used (e.g. Linux PAM).

This section explores the use of SSH/SFTP by considering the following scenario: there is a need for a service providing shell access or secure file transfer to a server. The service needs to authenticate the user and map him to a local account and groups. The mapping must be based on user attributes. The local accounts must be provisioned automatically. The solution must leverage federated access.

The above scenario may be affected by some use-case-specific constraints, e.g. policies, already-existing services, allowed protocols and software, etc.

Two possible solutions are described below: GSI-enabled SSH, and SSH key provisioning with web portal. The limitations previously mentioned should be kept in mind when selecting the appropriate one.

2.1 GSI enabled SSH

The Grid Security Infrastructure (GSI) is a specification for secure communication between software components, where authentication is based on PKI and credential delegation. GSI-OpenSSH is a modified version of OpenSSH that adds support for GSI, providing a single sign-on remote login (gsi-ssh) and file transfer services (gsi-scp and gsi-sftp). This solution requires both modified client software and modified server software. The users must have an X.509 certificate, so that the solution fits specific use cases.

[1] Note that this support may be limited depending on the SSH implementation, e.g. OpenSSH supports only Kerberos GSS-API and the “gssapi-with-mic” authentication method, which is not sufficient for GSI, described in Section **Error! Reference source not found.**.

The research leading to these results has received funding from the European Community's Horizon2020 Programme under Grant Agreement No. 653965 (AARC).

PKI allows authentication to be delegated to an external entity. Typically, the user obtains a long-lived certificate signed by a CA related to their institution or community and uses it to create a short-lived proxy certificate used for the authentication. Another possible approach is to obtain a short-lived certificate from an online CA[2] that uses another method of authentication.

In order to obtain a (short-lived) certificate from an online CA on the command line, several possibilities are now available. Until recently, the only options were for the user to manually export a credential from the web browser or download directly from the online CA. The user would then need to manually put the credential in the right place. Recently, using the enhanced client protocol (ECP) interface of the CILogon service, the Open Science Grid and LIGO have started using a SAML-ECP-based command-line tool to automatically download and handle the credential[3]. The approach reduces the certificate + key to an opaque token, much like Kerberos[4].

The certificates may hold attributes signed by some attribute authority (e.g. VOMS) to be used while mapping to a local account. The mapping and account provisioning mechanism is pluggable in GSI and one, commonly used, is LCMAPS, which maps users to accounts from pre-created pools using VOMS attributes.

[2] See the AARC CILogon TTS pilot [AARC-CILogon] and the RCauth.eu service [RCauth] as a reference.

[3] For a description of the CILogon ECP profile and the tool that uses it, cigetcert, see [CILogon ECP] and [CIGETCERT].

[4] An alternative approach, intended to circumvent the problem of limited ECP availability and to be implemented by the RCauth, is to use SSH-key authentication to retrieve and automatically handle a short-lived credential from a MyProxy server, where the key is previously uploaded to a web portal (see Section 2.2).

The research leading to these results has received funding from the European Community's Horizon2020 Programme under Grant Agreement No. 653965 (AARC).

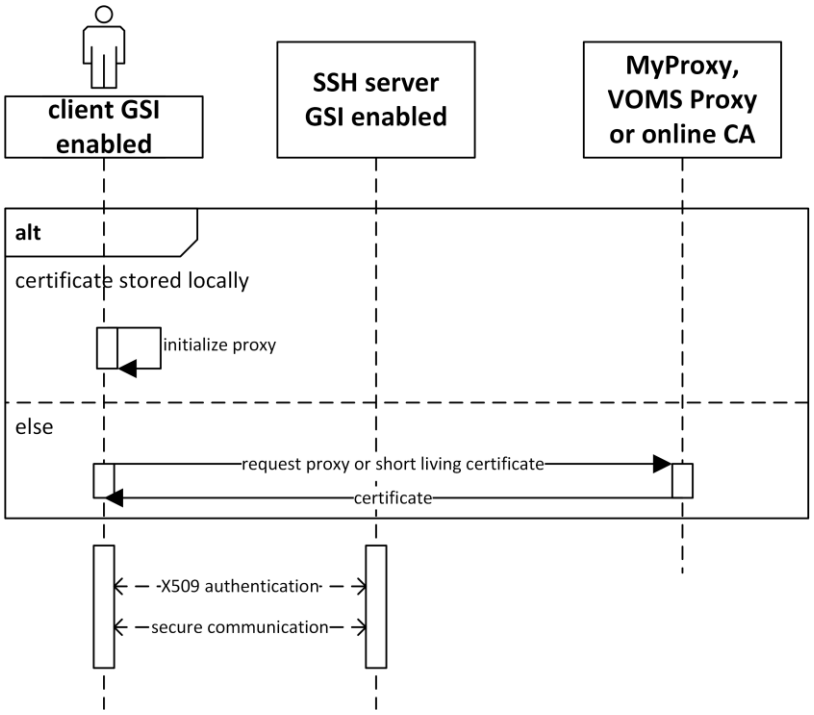

Figure 2.1: Sequence diagram for GSI SSH.

2.2 SSH key provisioning with web portal

This approach requires an additional step (logging into the web portal) to be taken by the user prior to accessing the non-browser resource, in particular an SSH resource. The flow is as follows (summarised in Figure 2.2):

- The user logs into a dedicated web portal that performs authentication and authorisation. It may leverage HTTP features and the existing variety of implementations of authentication methods for web solutions.
- The user uploads or generates credentials to be used while accessing the non-web resource. Typically, a public cryptographic key is uploaded or a pair of keys is generated. In addition, an account name on the resource is generated or selected by the user. Mapping of the user identity to that name must be persistent over multiple logins.
- The service provisions or updates an account on the resource and maps the credentials in the background. This is achieved by, for example, modifying an LDAP that provides accounts on the resource or by using an API to a VM hypervisor to set the public key in a VM. Additionally, group or VO membership on the portal may be mapped to local groups on the resource in order to achieve fine-grained authorisation.
- The user accesses the resource using the configured credentials.

The research leading to these results has received funding from the European Community's Horizon2020 Programme under Grant Agreement No. 653965 (AARC).

- Access to the resource is enabled while the user has a valid login session opened in the portal. The access can be locked, for example by removing the public key from the LDAP or the VM. Note that this is not account deprovisioning.

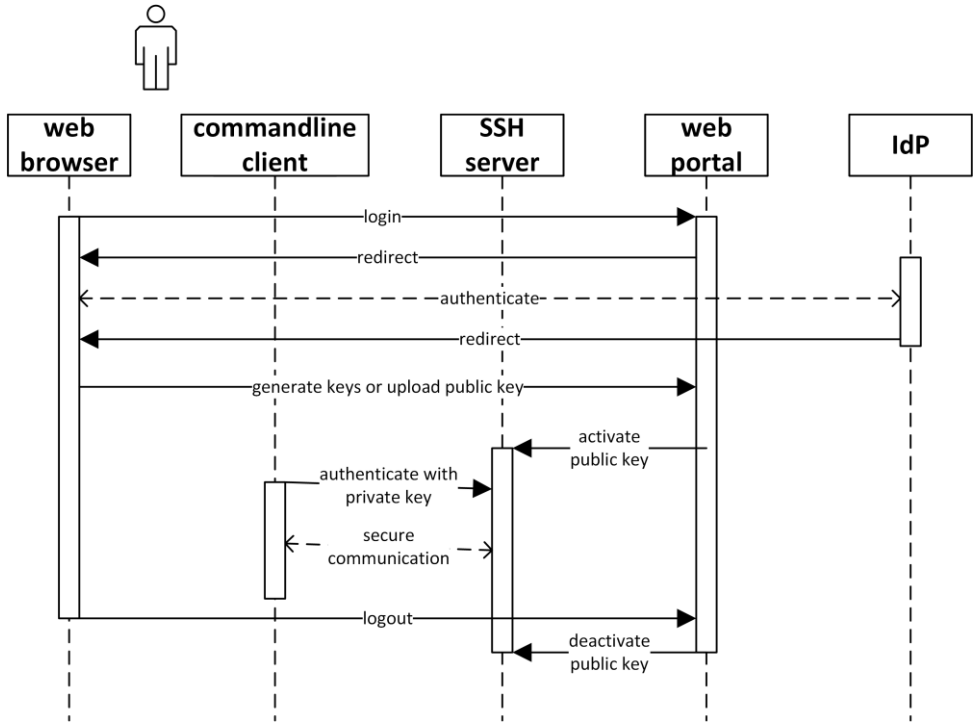

Figure 2.2: Sequence diagram for key provisioning with web portal.

The AARC project has successfully piloted two solutions that follow this approach: COmanage SSH key management [AARC-CO-SSH] and WaTTS TTS SSH pilot [AARC-WTS-SSH].

3 HTTP APIs

HTTP APIs allow the implementation of applications that access services with the HTTP protocol. These applications may be run from a web browser or standalone, without a GUI or even in batch mode. Non-browser applications have limited ability to involve a third party in the authentication and authorisation process (using HTTP redirect) and to interact with the user.

This section explores the use of HTTP APIs by considering the following scenario: there is a service providing an HTTP API to let the clients access and manipulate certain resources (e.g. IaaS cloud or cloud storage). A user community (UC) wants to use the service from a specific application (e.g. batch script or office software) and also wants to leverage federated access. The HTTP API of the service needs to authenticate the user and

The research leading to these results has received funding from the European Community's Horizon2020 Programme under Grant Agreement No. 653965 (AARC).

take authorisation decisions based on the user's attributes (e.g. grant access to some VMs or files for users and groups).

The HTTP APIs support a variety of methods for authentication and authorisation. Note that the above scenario may have two variants: (a) the service already exists and the methods cannot be changed, and (b) the service is in the design phase and a preferred method can be selected.

The following technologies may be considered:

- OIDC/OAuth2.
- X.509 certificates.

On the other hand, the user identity and attributes taken from the user community may be provided by a different technology. Typically, for the above scenario it might be:

- SAML IdP.
- X.509 certificate.
- OAuth2 Authorisation Server (usually guest identities).

Since the AAI technologies and domains used by the service and identity or attribute provider may be different, token translation may be necessary. Two possible solutions are described below.

3.1 Accessing HTTP APIs using OIDC/OAuth2

The solution is based on the following flow (summarised in Figure 3.1):

- The user goes to a token generation service (i.e. part of the token translation service) using their browser.
- To access the service the user has to authenticate at their home IdP (standard browser-based flow).
- After successful authentication/authorisation the service can generate an API token for the user, i.e. an OIDC access token.
- The OIDC access token can be used by command-line clients to authenticate against HTTP APIs that support OIDC/OAuth2. The token must be manually copied from the browser and pasted into the command line.
- The HTTP API service can use the OIDC access token to retrieve user information from the OIDC provider component of the proxy service.

The research leading to these results has received funding from the European Community's Horizon2020 Programme under Grant Agreement No. 653965 (AARC).

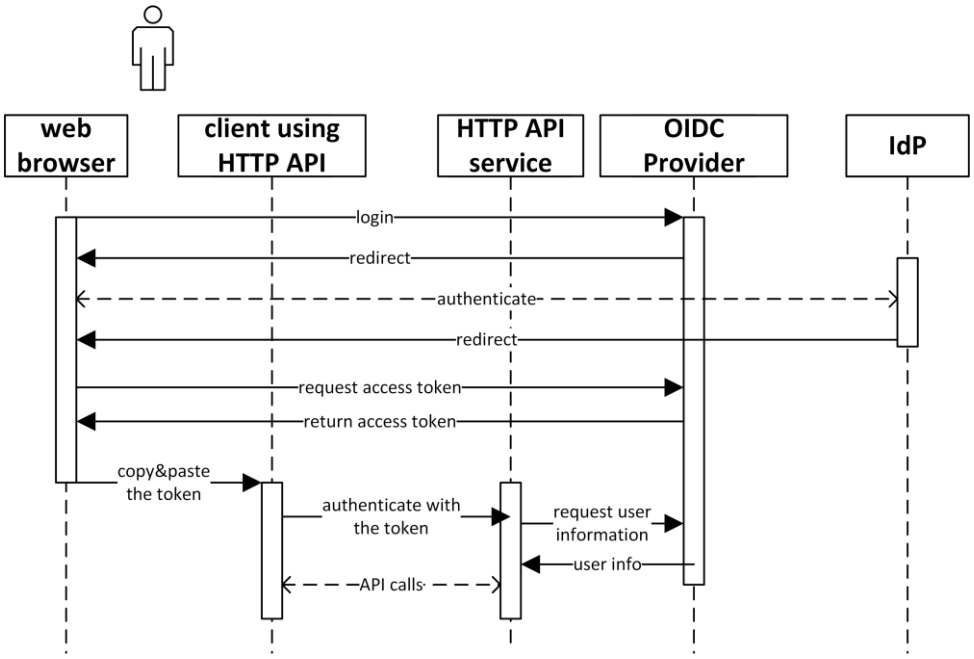

Figure 3.1: Sequence diagram for accessing HTTP APIs using OIDC/OAuth2.

3.2 Accessing HTTP APIs using X.509 certificates

If the user already has an X.509 certificate and may use a derived proxy in the API, neither token translation nor the interaction is required. If he/she does not, then X.509 short-lived certificates can be retrieved by a token translation service accessible via a web browser.

This API can also be used in delegation scenarios, where the first service to be accessed (for example, a web portal) needs to access other services on behalf of the end user. Such delegation is done using RFC3820 proxy certificates. An example of such a situation is third-party transfer copies between GridFTP-based storage elements all handled from within a web portal, where the portal has obtained an X.509 (proxy) certificate for the user via a portal delegation scenario such as that used in the CILogon-based AARC pilot.

The research leading to these results has received funding from the European Community's Horizon2020 Programme under Grant Agreement No. 653965 (AARC).

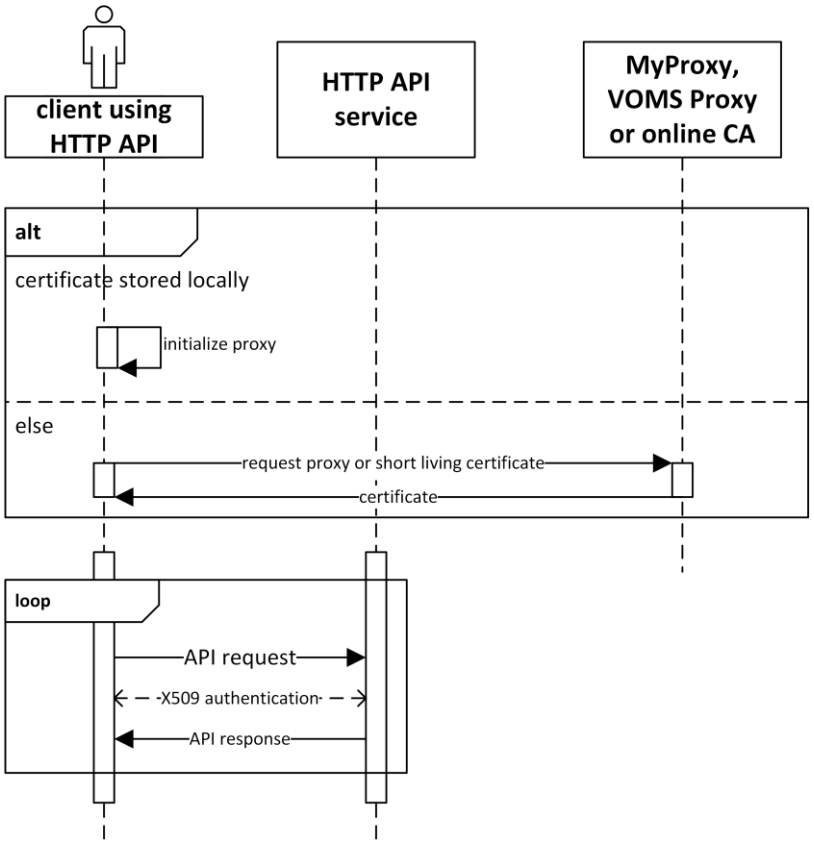

Figure 3.2: Sequence diagram for HTTP API using X.509 certificates.

3.3 Accessing HTTP APIs using service specific API tokens

This type of solution is used by online services, e.g. Github [GITHUB-1], Google [GOOGLE-1] and Apple [APPLE-1]. They can provide access to an API which otherwise would require user interaction, e.g. via a browser or by a second factor authentication. They are used e.g. as a password via a basic authentication header. To obtain them, the user should visit the web interface, where they can be generated and subsequently copied from. A token will typically have certain rights attached to it, which can be adjusted or revoked from the same web interface. As such, these tokens are very similar to OAuth token and the flow is much like the one described in Section 3.1 for OIDC/OAuth2.

4 References

[AARC-DJRA1.1] DJRA1.1: Analysis of user community and service provider requirements https://aarc-project.eu/wp-content/uploads/2015/10/AARC-DJRA1.1.pdf

The research leading to these results has received funding from the European Community's Horizon2020 Programme under Grant Agreement No. 653965 (AARC).

[AARC-CILogon]	AARC wiki page: CILogon TTS pilot https://wiki.geant.org/display/AARC/CIlogon+TTS+pilot
[AARC-CO-SSH]	AARC wiki page: COmanage OpenConext ssh access https://wiki.geant.org/display/AARC/COmanage+OpenConext+ssh+access
[AARC-WTS-SSH]	AARC wiki page: WaTTS SSH plugin – SSH access using OIDC login https://wiki.geant.org/display/AARC/WaTTS+SSH+plugin+-+SSH+access+using+OIDC+login
[APPLE-1]	Apple: Using app-specific passwords https://support.apple.com/en-gb/HT204397
[CIGETCERT]	Description of cigetcert tool https://cdcvs.fnal.gov/redmine/projects/fermitools/wiki/cigetcert
[CILogonECP]	Description of CILogon ECP profile http://www.cilogon.org/ecp
[GITHUB-1]	GitHub web interface: Creating a personal access token for the command line https://help.github.com/articles/creating-an-access-token-for-command-line-use/
[GOOGLE-1]	Google: Sign in using App Passwords https://support.google.com/accounts/answer/185833?hl=en
[RCauth]	RCauth website http://rcauth.eu

5 Glossary

API	Application Program Interface
CA	Certification Authority
ECP	Enhanced Client or Proxy
GSI	Grid Security Infrastructure
HTTP	Hypertext Transfer Protocol
IdP	Identity Provider
iRODS	Integrated Rule-Oriented Data System
LCMAPS	Local Credential Mapping Service
LDAP	Lightweight Directory Access Protocol
LIGO	Laser Interferometer Gravitational-Wave Observatory
OIDC	OpenID Connect
PAM	Pluggable Authentication Module
SFTP	SSH/Secure File Transfer Protocol
SSH	Secure SHell
SVN	Subversion
VM	Virtual Machine
VO	Virtual Organization
VOMS	Virtual Organization Membership Service

The research leading to these results has received funding from the European Community's Horizon2020 Programme under Grant Agreement No. 653965 (AARC).