Evolution of SSH with OpenId Connect

Dr. Marcus Hardt,[a,*] Dr. Diana Gudu,[a] Gabriel Zachmann[a] and Lukas Brocke[a]

[a] *Karlsruhe Institute of Technology,*
Herrmann-von-Helmholtz-Platz 1, 76344 Eggenstein-Leopoldshafen, Germany
E-mail: hardt@kit.edu, gudu@kit.edu, gabriel.zachmann@kit.edu, lukas.brocke@student.kit.edu

The Secure Shell Protocol (SSH) is the de-facto standard for accessing remote servers on the commandline. Use cases include remote system administration for unix administrators, git via ssh for developers, rsync via ssh for system backups, and HPC access for scientists. Unfortunately, there is no globally accepted usage pattern for federated usage yet.
The large variety of users with different backgrounds and usage profiles motivated us to develop a set of different tools for facilitating the integration with federated user identities. The main novelty of our contribution is the integration of an ssh Certificate Authority (CA) into the existing motley-cue + oidc-agent mechanism via a toolchain that we call "oinit". It simplifies the usage of ssh certificates by leveraging authorisation information via established federation mechanisms. The benefit is that – after an initial setup step – ssh may be used securely without interrupting existing flows. This allows for example the use of rsync or git via ssh.
To enable this, oinit consists of a collection of programs to enable OpenSSH login for federated identities based on certificates:

- The oinit-ca provides a REST interface to an ssh-ca at which authorised users obtain an ssh certificate for a specified host or host group. Authorisation decisions are enforced by motley-cue, the component that maps federated identities to local ones on the ssh server side. User provisioning may also be triggered at this point, e.g. via motley-cue and feudal.
- Users employ the oinit tool to add hosts to the oinit mechanism. Once established, ssh certificates will automatically be retrieved, whenever this may be necessary and stored in the ssh-agent.
- Serverside tools and configuration for enabling ssh without knowledge of local usernames ("non-username operation"), which is particularly useful in federated scenarios.

  In this paper we outline the basic solution and focus on on user interface optimisation with the integration of ssh certificates.

Source code modifications of `ssh` are prohibitive and are not necessary with the solution described in this paper.

International Symposium on Grids and Clouds (ISGC2024)
24 -29 March, 2024
Academia Sinica Computing Centre (ASGC), Institute of Physics, Academia Sinica Taipei, Taiwan

*Speaker

1. Introduction

The Secure Shell Protocol (SSH) is the most used network protocol for login and execution of commands on remote servers, with OpenSSH being the most prominent and widespread implementation [1, 2]. SSH is based on local unix accounts on the server system, the concept of federated identities evolved at a much later point in time.

While protocols, such as Kerberos, can be used with OpenSSH to make it more federated, its configuration is complex and difficult to set up between multiple realms or organisations. This is especially true in the context of federated services deployed at different research institutions in different organisational domains.

Our goal is to extend OpenSSH so that it directly supports authentication with federated identities. For this we use the established OpenID Connect protocol. Since the actual identity of the user is already conveyed via OpenID Connect, we want to remove the need to specify usernames in ssh. We call this "non-username operation" in this paper.

The goal is furthermore, to allow administrators to conveniently authorise the correct users, while the burden of user management and authentication remains at their home organisations. Administrators will rather be enabled to specify criteria which federated users need to fulfil to be authorised for using a particular ssh server.

Finally, our solution will not require any changes to the OpenSSH source code, since the high security requirements of any ssh implementation require prioritising security over novel features.

This paper is organised such that we first give an overview over the work that other groups are doing in section 2. We then describe our currently working solution in section 3 before we come to the main part of our paper in section 4. There we describe how ssh certificates work, which features we use, how we integrate this with our current solution, and how this greatly improves the usability of ssh with federated identities. Section 5 summarises and gives an outlook over future work.

1.1 Benefits of Federated Identities

Our solution relies on the efficient use of federated identities. Historically, there are two evolutionary steps between federated identities and local accounts on systems. Those are still used, but predominantly for small individually administered systems. Services that are used in larger contexts are typically connected to an organisation-wide user-management service, such as LDAP. In this first evolutionary step, identities are centrally managed for services inside an organisation, along with their attributes that may be used for authorisation at a service.

We speak of "Federated Identities" when multiple organisations expose this identity information for consumption by a large set of external services. This allows services that trust a set of organisations to rely on identity information managed by external parties. The basis on which this trust is established are policies. Such policies were developed over the last two decades and promoted within eduGAIN [3] and the AARC Policy Development Kit [4]. Within the federated identity ecosystem, users are presented with a list of Home-Organisations to choose from on the login screen of a service.

Large enterprises, such as Google, Microsoft, or Meta also provide the functionality that allows services to "login with google" or similar. This does not follow the federated identity concept, but is merely an external access to the one central identity system of the respective company.

In the scientific domain, login via the Home Organisation via eduGAIN is recently being complemented with so called Community-AAIs [5] that add authorisation information to the set of attributes that describe a user's identity. These attributes will often include information about role or group within the Community, how well an identity was vetted, as well as include information from a user's Home Organisation.

1.2 Unix and the commandline

Using these attributes at the services to determine whether or not a given user is authorised to use that service is well established for web-based services. Authentication on the commandline, however, requires slightly different approaches and tools which are out of scope of this contribution. Here we focus on including unix-based services from the server side perspective. In this scenario, the federated identity of a user needs to be mapped to a local unix user account. In this paper we describe how this is done, how and what we improve on the user-friendliness of the approach.

1.3 SSH with OpenID Connect

At the time of writing, two protocols for federated identities are used in our domain: The Security Assertion Markup Language SAML [6], and OpenID Connect (OIDC) [7]. For our implementation we chose OIDC, because it is newer, supported by many industrial implementations, and better supported on the commandline than SAML.

2. Related Work

Multiple groups in the scientific context work on the goal of using OpenID Connect (OIDC) for ssh. The different approaches are characterised by the technological approaches for their solution.

Moonshot [20] was the first effort to implement ssh with federated identities. It used RADIUS to establish a trusted channel via their trust router to a SAML based Home Organisation. The underlying protocol is ABFAB [21]. Since the required uptake from infrastructures was slow and not enough use-case demand materialised, moonshot is apparently discontinued.

Another option is the so called "smart shell", in which the user shell on the remote machine is replaced with a tool that handles the OIDC authentication and authorisation steps. When successful, the smartshell calls the original shell for the user, who is then logged in. While smart shell can work in both ways, non-interactively and interactively, the latter is taken by the solution of AWI [8] and DAASI International [9], and by the SURF solution [17]. The interaction is used to authenticate the user. This interrupts the flow of the ssh login and can hardly be used for non-interactive use-cases such as scp or within scripts. The non-interactive is used by the DEIC and the KIT solution.

Another approach is based on the Linux Pluggable Authentication Module subsystem (PAM). Here, a PAM module is provided on the server side. It checks authorisation before control is handed on to the ssh daemon. Due to the enhanced security measures taken by ssh, the PAM module is not able to change the username during the authentication process. The PAM module may be used to display a QR-Code to the user, that leads her into a web browser based login via the device code flow. This is implemented by the STFC-Solution [15], which is based on a fork of a solution originally developed by CESNET [18]. An alternative implementation by PSNC [19] is prompting the user for an OIDC Access Token, which it verifies with either an OIDC Provider (OP) or another external service. The main difference is that the user is not necessarily required to interrupt the ssh login by visiting a web login page first, provided the user can obtain an Access Token easily. The openSSH client limits the length of the Access Tokens that can be used to 1023 bytes.

SSH-Certificates are used in a solution implemented at DEIC [16]. They use a web browser based login flow that generates a user-specific code for a short unix commandline, that the user copies on to the machine from which the ssh client is to be used. The command retrieves the ssh certificate for the authenticated user. After that non-interactive login is available.

Each of the presented solutions focuses on the ssh login process and depends on a corresponding unix user to exist. Earlier work was based on X.509 certificates. This approach – `gsi-ssh` – was implemented in 2008 at NCSA [13]. This approach was the first one to translate the identity conveyed in the X.509 user certificate into a Unix user account. The implementation was based on patching the original openSSH source code. Unfortunately, these changes were never accepted by the openSSH maintainers due to prioritising security over features.

3. SSH with OpenID Connect

Our solution, which we call `ssh-oidc` is composed of a set of different components. Each component addresses one specific task, so that individual components may be combined to support a larger range of use-cases. In this section we shortly describe the available components that compose the existing solution.

One key aspect that differentiates our solution from others is that we use the information contained in the federated identity to identify the corresponding unix account on the server system, just like the `gsi-ssh` does.

For this we rely on additional software components, that we developed ourselves, and in collaboration with external partners. In this paper we outline the basic solution and focus on on user interface optimisation with the integration of ssh-certificates.

3.1 PAM Module

We use a patched version of the PSNC pam module [19]. This enables us to verify the Access Token with motley cue, instead of the actual OP that issued the token. This allows us to use any information that motley-cue can verify in the password field. This is important in cases in which the Access Token is longer than 1023 byte.

3.2 oidc-agent

To authenticate to services with OIDC Access Tokens, we need OIDC Access Tokens. `oidc-agent` provides a convenient way to obtain them. Once configured, `oidc-agent` can obtain fresh Access Tokens without any interruption of a users workflow. In principle, it would be possible to use web-services that expose Access Tokens to users and copy-paste them whenever needed. But due to the short life time using `oidc-agent` is simply more convenient.

`oidc-agent` [10] is a command-line tool to obtain Access Tokens. It follows the concept of `ssh-agent`, but instead of `ssh-key` material it stores an OIDC Refresh Token and the associated `client_id` and `client_secret`, encrypted with a passphrase on the user's client computer.

`oidc-agent` supports multiple account configurations, which correspond to different OIDC Provider (OP) configurations, and hence to different identities of the user. The identities may be referred to with a `shortname`, or using the full issuer URL.

3.3 motley cue

`motley-cue` [11] is a daemon that runs on or nearby the ssh server. Nearby means, that motley cue runs in a context that allows it to access the system's user database. Depending on the use-case read-only or read-write access may be required.

`motley-cue` exposes its functionality through an OIDC-protected REST interface. A client authenticates with the Access Token of the user that needs to log in to the ssh server. Details are described at [11]. The features of `motley-cue` used in the context of this paper are:

- Ensure authorisation of the user that provided the Access Token.
- Provide information to existing mapping of a federated identity to a local unix account.
- Optional: Provision an account for the authorised user.
- Verify the Access Token, used by the pam module.
- Support One Time Passwords (OTP) in case of Access Tokens longer than the password field.

With this setup ssh does already work, if the user knows the remote username, and has an Access Token (<1023 byte) to paste into the ssh client when prompted "Access Token".

3.4 mccli

The `motley-cue` command line interface "`mccli`" is a client side tool to automate several steps that the user would otherwise need to do manually. `mccli` uses the REST interface of `motley-cue` to identify whether or not the user will be authorised to use the ssh service.

In a first step, `mccli` will contact `motley-cue` with the Access Token of the user. If the user is authorised, `mccli` will obtain the username on which the user will be mapped. If the user does not exist, `mccli` can contact `motley-cue` to trigger the provisioning of the user account, if that is supported by the system. In case the Access Token is larger than 1023 bytes, `mccli` will also obtain an OTP for the ssh session that it will initiate in the 2nd step. In the second step, `mccli` calls ssh and uses `pexpect` [12] to place the Access Token (or OTP) into the ssh client, when prompted for `Access Token`, thus enabling a non-interactive ssh login. `mccli` supports the ssh commands `ssh`, `scp`, and `sftp` and various options for contacting the correct instance of `motley-cue`. For details, see [11].

A typical use of mccli is shown below. It would – just as with ordinary `ssh` – result in a shell on the remote system.

```
mccli <protocol> <ssh-host> --oidc <oidc-agent configuration>
mccli ssh ssh-oidc-demo.data.kit.edu --oidc egi
```

The drawbacks of the `mccli` approach are that users have to install two components on their systems: `oidc-agent` and `mccli`. In addition, only selected ssh-based commands are supported. Advanced usage such as piping `stdin/stdout` through ssh are not supported by `mccli`.

We address this in two ways that we describe in subsection 3.5 and in section 4

3.5 Web ssh

To reduce the barrier of entry, we implemented the steps done by `mccli` in `javascript` so that it can be used on a webpage. To obtain the required Access Token, the webpage would redirect the user for login to her OpenID Provider (OP). After the authorization-code-flow of OIDC, the Access Token is available to the javascript code of `mccli` in the browser and can be used for a web shell. This way, installing the `oidc-agent` is not required either. The user only needs a web browser.

A demo version of this is available at https://ssh-oidc-web.vm.fedcloud.eu

4. OINIT: SSH certificates with OIDC

`oinit` is our effort to integrate `ssh certificates` with `motley-cue`, in order to further improve the usability of the solution. In addition, `oinit` allows servers to not use our PAM module `pam-ssh-oidc`, which may be required in some secured environments, such as HPC.

`SSH certificates` provide a different mechanism for authenticating to an `ssh` server. `OINIT` consists of components for the `ssh client`, the `ssh server`, and one for the `ssh CA`.

4.1 SSH certificates

`SSH certificates` are implemented in `ssh` since 2010, hence they are available in the large majority of `ssh` implementations. These certificates are conceptually similar to `X.509 certificates`, but are less complex and implemented in a different way. The ssh CA signs `user` and `host` certificates, so that hosts and users can trust one another. Users can log in to hosts that trust the CA that signed the users' certificate. The username(s) that may be used are encoded into the certificate, using the `Principals` option. Just as with `ssh keys`, the commands that may be executed with a certain certificate may be limited with the `force-command` option.

SSH certificates support most features that are known from ssh keys, but in addition, do support a validity period, so that they can expire.

4.2 oinit CA

For generating `ssh certificates`, and for integration with `motley-cue`, we implemented the `oinit CA`. It is accessible via an OIDC protected REST interface. Users authenticate with their Access Token, which is verified with the issuing OP. This Access Token is used by `oinit CA` to check the authorisation, and to query the unix username of the user from `motley cue`. The username is encoded into the certificate, which is then returned to the user. Only users authorised by `motley cue` will obtain an `ssh certificate`.

To support non-username operation, we make use of the special server-side user "`oinit`", and encode the valid username in the "`force-command`" option of the certificate. For details see [14].

4.3 SSH server

On the server side we install the dedicated user "`oinit`". That user can only execute the single command "`oinit-switch`", which switches to the user context for which the certificate was issued. This step is required for non-username operation, and might be skipped otherwise. The `oinit-switch` command obtains the unix system privileges of changing the user UID via `pam su`.

4.4 SSH client

Since our solution works without any change to existing `ssh` commandlines, we need a way for `ssh clients` to distinguish between `oinit`-enabled `ssh servers` and traditional ones. We also need a way to tell the `ssh` command, which `ssh CA` to use for which `ssh server`. This is what the `oinit` command does:

```
oinit [add|delete|list] <host>[:port] [ca]
```

To add the host `ssh-oidc-demo.data.kit.edu` to the `oinit` capable hosts, using the CA `https://ssh-ca.data.kit.edu`, this command would be used:

```
oinit add ssh-oidc-demo.data.kit.edu https://ssh-ca.data.kit.edu
```

This will adapt a user's `.ssh/config` file to support the `oinit` lookup. It will furthermore store the `ssh server` `oinit CA` mapping in `.ssh/oinit`.

The first time a user logs in to an `oinit`-enabled ssh server, she will be presented with a list of OIDC Providers (OPs) supported by the server to choose the identity with which she wishes to log in. In case `oidc-agent` is used, available configurations will be highlighted. (For simplified usage, if only one `oidc-agent` configuration is present, this one will be used in the future.) Once the identity is chosen, `oinit` will obtain an `ssh certificate`, store it in the `ssh agent` and return control to the `ssh` client, which will then establish the connection.

```
$ ssh ssh-server.edu

[1] https://aai.egi.eu/auth/realms/egi (Accounts: egi)
[2] https://accounts.google.com
[3] https://login.helmholtz.de/oauth2
[4] https://wlcg.cloud.cnaf.infn.it
? Please select a provider to use [1-8]: 1
  Received a certificate which is valid until 2024-03-08 14:04:20
```

5. Summary

We presented our solutions as the evolution of our ecosystem around commandline-based authentication, mapping of federated identities to Unix accounts and the integration of OpenId Connect mechanisms. The initial solution requires adaptation of the user's workflow and two client side-tools to be installed. We presented two improvements. First, a web-based approach was introduced, which can be used from any web-browser without the installation of any tool locally.

Second, our ssh-certificate based solution `oinit` was presented. A lightweight online-CA which was integrated with `motley-cue` for authorisation, with `ssh client` for user convenience and with `ssh server` for security.

5.1 Future Work

Future work will involve further improvements on the client side, e.g. to simplify the selection of the user identity. Also, the mapping of federated to unix accounts will be interfaced with the account linking service ALISE [22].

Furthermore, packaging of the `oinit` solution can be improved, e.g. to work better with the nginx configuration of `motley-cue` and the `oinit ca`. Packages will be supported on most Linux distributions, Mac and Windows will be supported at a later date.

Finally it is planned to conduct a commercial security audit of all components of the ecosystem.

References

[1] 7th Zero. Results: SSH Statistics Gathering Project. https://7thzero.com/blog/ssh-statistics-gathering-project-2017-results. (Online; accessed 04-September-2023). 2017.

[2] Oliver Gasser, Ralph Holz, and Georg Carle. "A deeper understanding of SSH: Results from Internet-wide scans". In: 2014 IEEE Network Operations and Manage-ment Symposium (NOMS). 2014, pp. 1–9. doi: 10.1109/NOMS.2014.6838249.

[3] The EduGAIN homepage https://edugain.org (Online; accessed 15-April-2024)

[4] The AARC Policy Developent KIT https://aarc-community.org/policies/policy-development-kit (Online; accessed 15-April-2024).

[5] The AARC Blueprint Architecture 2019 version https://zenodo.org/doi/10.5281/zenodo.3672784

[6] Security Assertion Markup Language (SAML) https://saml.xml.org/saml-specifications (Online; accessed 15-April-2024)

[7] OpenID Connect (OIDC) https://openid.net/developers/how-connect-works (Online; accessed 15-April-2024)

[8] Alfred Wegener Institut für Polar und Meeresforschung https://www.awi.de/ (Online; accessed 15-April-2024)

[9] DAASI International GmbH https://daasi.de/en/ (Online; accessed 15-April-2024)

[10] oidc-agent https://zenodo.org/doi/10.5281/zenodo.4966816

[11] motley-cue https://zenodo.org/doi/10.5281/zenodo.7346725

[12] Pexpect Online Documentation (Online; accessed 16-April-2024) https://pexpect.readthedocs.io/en/stable/

[13] GSI SSH (Online; accessed 15-April-2024) http://grid.ncsa.illinois.edu/ssh/

[14] Lukas Brocke, Certificate-based OpenSSH for Federated Identities https://doi.org/10.5445/IR/1000165236

[15] Jens Jenssen, Thomas Dack. "PAM module for OAuth 2.0 Device flow" (Online; accessed 17-April-2024) https://github.com/stfc/pam_oauth2_device

[16] Mads Freek Petersen, Mikkel Hald, Tangui Coulouarn. "SSH Certificates in a Federated World" (Online; accessed 17-April-2024) https://github.com/wayf-dk/ssh-certs-in-a-federated-world

[17] Martin van Es, Nield van Dijk, Floris Fokkinga. "Federated authentication for non-web-based research services" (Online; accessed 17-April-2024) https://communities.surf.nl/en/oplossingen-voor-online-onderzoeksomgevingen/article/federated-authentication-for-non-web-based https://github.com/SURFscz/pam-weblogin

[18] Jarosław Surkont, Michal Prochazka. "PAM module for OAuth 2.0 Device Authorization Grant" (Online; accessed 17-April-2024) https://github.com/ICS-MU/pam_oauth2_device

[19] Pawel Wolniwiecz, Damian Kaliszan. "Pam SSH for Pracelab" (Online; accessed 17-April-2024) https://git.man.poznan.pl/stash/scm/pracelab/pam.git

[20] Moonshot Wiki (Online; accessed 17-April-2024) https://moonshot-wiki.atlassian.net

[21] Application Bridging for Federated Access Beyond web (abfab) (Online; accessed 17-April-2024) https://datatracker.ietf.org/wg/abfab/documents/

[22] Account LInking SErvice – ALISE (Online; accessed 17-April-2024) https://github.com/m-team-kit/alise