oidc-agent - Integrating OpenID Connect Tokens with the Command Line

Gabriel Zachmann[1*], Marcus Hardt[1] and Diana Gudu[1]

[1*]Scientific Computing Center, Karlsruhe Institute of Technology, Herrmann-von-Helholtz-Platz 1, Eggenstein-Leopoldshafen, 76344, Baden-Wuerttemberg, Germany.

*Corresponding author(s). E-mail(s): gabriel.zachmann@kit.edu;
Contributing authors: hardt@kit.edu; gudu@kit.edu;

Abstract

The `oidc-agent` is an OpenID Connect tool suite designed to simplify authentication processes for command-line applications and workflows that require access to resources protected by OpenID Connect. It provides a secure, but user-friendly way to manage tokens on the command-line, reducing the need for manual re-authentication. This paper presents an in-depth overview of the architecture and features of the tool suite, alongside its real-world applications. `oidc-agent` serves as a valuable tool in token based authentication workflows, particularly for applications in cloud computing, high-performance computing, and scientific research, where efficient and secure access to resources is critical.

Keywords: OpenID Connect, OIDC, tokens, token-based, AAI, IAM, AARC, WLCG, command-line

1 Introduction

OpenID Connect (OIDC) is an important key technology used in token based authentication and authorisation infrastructures. It is widely used in industry (e.g. by `Google`, `Microsoft`, `Apple`, `Amazon`, `Okta`, etc.), but also in research infrastructures (e.g. `EGI`, `GEANT`). Also, `X.509` based infrastructures, such as the grid (`WLCG`), are moving away from certificates to token-based authentication [1].

Being designed primarily for web applications, OIDC assumes user presence in most of its specified "flows". The only flows that can be non-interactive are the "resource owner credentials" and "client credential" flows, as well as the "refresh flow". For the former there are security concerns; the latter requires a previous authentication flow.

Non-web, however, describes authentication to services, typically from the command line, and potentially without a user present. Even with the user present it is cumbersome to transfer the web-browser based authentication to the command-line. Typical non-web examples include authentication of remote OIDC-protected APIs, monitoring, and development work. More recent examples are the submission of computing jobs in a token based computing infrastructure, such as WLCG, and `ssh-oidc` [2].

The widespread adoption of OIDC, also in non-web use-cases, requires robust integration with command-line environments.

2 Requirements

The main goal of `oidc-agent` [3–5] is to enable usage of OIDC tokens on the command-line to securely support "non-web" use-cases.

In all of these use-cases, the authenticating client component requires an OIDC Access Token (AT) for authenticating to a server-side component (the protected resource). Sometimes, the presence of a user cannot be assumed. In addition, several use-cases implement delegation scenarios, where ATs are used from remote services with and without direct network connection back to the initiating host.

From these scenarios we derived general requirements that helped designing `oidc-agent`:

- **API access** from the command line requires up-to-date ATs that can easily be obtained from a short shell command.
- **Multiple OpenID Providers (OPs)**: Obtaining ATs from multiple different providers is required mostly by developers that need to test and compare behaviours of different OPs, but also from users that engage in different communities. Simply switching between different OPs became a key requirement.
- **Non interactive operation** must be supported such as regularly called APIs, e.g. when monitoring services that require OIDC authentication.
- **Application support** is required since many applications struggle implementing appropriate OIDC integration on the client-side. Straightforward integration into existing applications requires easy-to-use client libraries.
- **Customisation of tokens** where applications need to specify the scope and/or audience of the obtained ATs.
- **Delegation** where the obtained ATs are forwarded to remote computers and used from there.

3 Related Work

When the development of `oidc-agent` started in 2017, we found no tool that could satisfy our requirements. Only tools for managing OIDC clients on the command line existed [6, 7]. Since they did not offer a way to obtain ATs, they were no longer considered. In hindsight, both projects ([6] and [7]) became unmaintained and did not receive any updates since six or nine years, respectively.

In the meantime, further solutions were implemented: a tool also named `oidc-agent` was implemented by `shelmangroup` [8]. It follows a similar goal and promises to provide ATs on the command line. However, it only has support for creating a client, performing the authorisation code flow (see section 5) to obtain a Refresh Token (RT), and to obtain additional tokens from then on. In comparison, our `oidc-agent` provides a wide range of additional features and improvements (see section 6). Furthermore, the shelmangroup/oidc-agent did not get any further release after its first major release in 2019.

A more recent development is `oidc-cli` by Reimann [9] with a similar feature set as shelmangroup/oidc-agent at the time of writing. However, since `oidc-cli` is relatively new and actively developed, additional features might be implemented. At the time of testing we found several limitations. For example, no options to control the requested scopes from the command line are available. The only way to obtain specific scopes or even a RT at all would require manual manipulation of the authorisation URL.

Both solutions have a simpler architecture in common. We describe the architecture of our `oidc-agent` solution in the next section. The other solutions only consist of a single binary and do not run as a daemon in the background. This simpler approach might be more appealing for light-weight use cases, though. Their largest drawback is that both store sensitive data (e.g. client credentials, RTs) without encryption.

4 Architecture

The design of `oidc-agent` [3–5] is inspired by `ssh-agent` [10]. This also includes the fact that there are multiple `oidc-*` binaries, each serving a distinct purpose. Following the principle of privilege separation there are also different privileges required by the components. In the following we introduce all components and their purpose.

- **oidc-agent:** The `oidc-agent` is the core component and lends its name to the whole toolchain. The agent runs as a daemon in the background and other components as well as third-party applications can communicate with it through a unix domain socket. The `oidc-agent` is internally split into two parts, the `oidcd` and `oidcp` part. This split is done for privilege separation and is transparent to the user. Privilege separation allows limiting a component to only the subset of operating-system privileges that are required for one component. It is used to avoid network-enabled components writing to disk or executing software, and is key in securing software.
  The `oidcd` is the daemon taking care of the OIDC communication with the OPs. It stores the configuration of loaded accounts in memory; sensitive data is encrypted and only decrypted on need.
  `oidcp` proxies requests from other parties to `oidcd`. If needed it can prompt the user via `oidc-prompt` for information (e.g. this might be needed if an AT is requested for a currently not loaded account configuration, see subsection 6.3 for details of this scenario). There are also flows in which `oidcp` might require file access, to read and update an account configuration file. We describe these cases later on.
- **oidc-prompt:** The `oidc-prompt` binary is a utility component used by `oidcp` to prompt the user for information. It is not intended to be called by the end-user. It is inspired by `ssh-askpass` but its functionality was extended over simple password prompting and includes prompting for text input, passwords, confirmation, and selection from a list (may include custom input). User prompts can also be defined using arbitrary html content.
- **oidc-gen:** In order to obtain ATs with `oidc-agent`, users first need to create a so called *account configuration*. Account configurations are named by the user, so they can be memorised easier. Generating account configurations is done through `oidc-gen`. The tool provides an interactive interface prompting the user for the necessary information. Alternatively, it is also possible to provide all information using command line arguments so that an account configuration can be created non-interactively. This way communities can also provide a command to their users to facilitate easy oidc-agent account creation for their relevant OP.
  The account configuration needs an OIDC client. To ease setup of new account configurations, `oidc-agent` comes with a number of pre-registered clients for commonly used OPs. `oidc-gen` automatically uses pre-registered clients if available; otherwise, a client is dynamically registered with the OP if they support this feature. As a last resort, clients may be manually registered with the OP.
- **oidc-add:** This tool is used to load an existing account configuration into the agent. This may be necessary after a reboot. In a desktop environment, i.e. a system with a graphical user interface, this is usually not required, since `oidc-agent` can automatically load account configurations when needed.
  `oidc-add` can also be used to unload accounts, obtain a list of all accounts configured or loaded, and to lock and unlock the agent. While locked, the agent does not accept any requests until unlocked.
- **oidc-token:** The `oidc-token` tool is used to obtain tokens from the agent. `oidc-token` uses the IPC-API through the `C` library. Other applications can easily request ATs in a similar way by utilising the library for their programming language, or – if no library exists – they can communicate directly with the unix domain socket.
  With `oidc-token` and our libraries it is possible to request tokens for a specific account configuration or for a specific OpenID Provider, which is identified by a URL. The latter is particularly useful for applications that require ATs from a specific OP but do not want to query the user about which account configuration (identified by a user-defined name) to use.
- **oidc-agent-service:** In line with `ssh-agent`, `oidc-agent` starts with a randomised socket path. This allows running multiple instances of `oidc-agent` independent of each other. This also means that there is no well-defined location where applications can connect to the agent. Applications rely on the `$OIDC_SOCK` environment variable to locate the agent.
  While integration across different terminals can also be done in other ways, it brings benefits to have a well-defined socket path. Primarily, this

is the case for restarting the agent, which might be required after an update. After a restart a randomised socket path typically changes. The `oidc-agent-service` script simplifies the integration with a system, by providing the agent-socket at a well-defined location:
`/tmp/oidc-agent-service-$UID/oidc-agent.sock`
where `$UID` is the user id. With this mechanism the agent can easily be restarted and is still available at the same location. This well-known location is also used as a default value if the `$OIDC_SOCK` environment variable is not set.
We also provide an integration into a user's `X session`. Here the agent is started at the beginning of an X session. This makes the environment variables available throughout the whole X session. For this we make use of the `oidc-agent-service` tool, so that `oidc-agent-service restart` can be used to restart the X-Session integrated agent, whenever a software upgrade requires this. Recent versions of `oidc-agent` also have the capability to automatically restart after an update.

- **oidc-tokensh** provides a shell-wrapper for further enhanced user-friendliness. `oidc-tokensh` starts a shell and an `oidc-agent`, and loads a configuration into the agent, using `oidc-add`. This helper can be added to a `.bashrc` file, so users are prompted for a password upon login to their shell. Once running, the `oidc-tokensh` helper ensures that fresh ATs are available in a well-known location, such as `/tmp/bt_u1000` [11].
- **oidc-keychain** enables re-using oidc-agent between login sessions. It uses `oidc-agent-service` to find existing login sessions, and active `oidc-agent` instances. It will start an agent instance for the user when required.

5 OpenID Connect

In this section we provide an overview of the OpenID Connect (OIDC) protocol, its most important flows and their usage within the agent.

OIDC [12] is an authentication protocol on top of OAuth2 [13] and is based on a number of different tokens used for different purposes:

- *Access Tokens (ATs)* are used for authorisation granting access to protected resources. They are usually *bearer* tokens [14] which means that any party that possesses the token can use it to obtain access without further authentication. In practice this means that ATs can be passed around (with some limitations). `oidc-agent` makes use of this property in its core mechanism: the agent requests the AT from the OP but then passes it on to other applications to use.
  ATs are valid only for a limited time-span; usually this means that they are short-lived (at most an hour). ATs can furthermore be restricted using *scopes* and *audiences*. The *scope* of a token represents the scope of access at the resource server, while the *audience* defines at which servers the token can be used. Resource servers that are not listed in a token's audience must reject the token.
- *Refresh Tokens (RTs)* are long-lived tokens that can be used to refresh ATs, i.e. an RT is used to obtain additional ATs. Other than ATs, RTs are intended to only be used by the client to which they were issued, i.e. they cannot be passed around. Therefore, they are bound to that specific client and must be kept secret.
  In `oidc-agent`, RTs are used at the very core of our mechanism: whenever a new AT is required. In our scenario, the RT is only used internally by the agent. The RT cannot be requested by other applications and is not exposed to the user.
- *ID Tokens* are JSON Web Tokens (JWTs) [15] encoding information about the user inside the ID token. This authentication information then can be used by the requesting client.
  ID tokens are only used for passing authentication information from the OP to a client and not for authorisation. Additionally, user information is available from the OP's *userinfo endpoint* and can be queried by using an AT as authorisation.
  Therefore, we decided that only ATs should be returned by the agent and not ID tokens, in order to discourage practices where ID tokens are improperly used for authorisation. However, valid use-cases for obtaining ID tokens exist, e.g. as a development tool to see what information is available from the ID token. Therefore, `oidc-agent` supports ID tokens, but it requires an explicit confirmation by the user.

A client obtains the tokens after a successful authentication flow, where the user authenticates and authorises the client to obtain certain scopes. At the end of such a flow the client obtains a set of tokens, consisting of at least an AT and ID token, and optionally also an RT. There are different authentication flows defined.

The most common OIDC flow is the *authorisation code flow* [12, 13], which can be summarised as: the client generates an authorisation URL which includes all relevant request parameters; using this URL the user authenticates and then authorises the client. The OP redirects the user's browser back to the client returning an *authorisation code*, which the client exchanges into the requested set of tokens. This is the default flow on the web and usually all OPs support it. With `oidc-agent` we use a `localhost` redirect URL, which requires the user to do the authentication on the same machine as the agent is running. Though, there is the possibility to work around this with a "manual redirect", but this is not a desired approach.

A better approach for situations where authentication should be done on another device than the one on which the agent is running is to use the *device code flow* [16] – a flow particularly designed for input constrained devices, such as TVs. With this flow `oidc-agent` starts the flow on one machine while the user can authenticate from any other device. The agent polls the OP for a successful authentication and eventually obtains the requested set of tokens. This flow is very suitable for command line usage but not supported by all OPs.

For legacy reasons `oidc-agent` also supports the *resource owner password credentials flow*, also known as the *password flow*: the user passes their credentials to the agent which directly authenticates with the OP. Since user credentials are passed to a third-party application (the agent), it is highly discouraged to use this flow and most OP do not support it anymore.

Any of these flows – depending on the support on the OP side – can be used to obtain an initial AT and the RT which is needed by the agent to obtain additional ATs. The initial authorisation flow generally only needs to be done once - depending on OP policies the RT might expire and re-authentication will be necessary. Once the agent has obtained an RT, it can use it in the *refresh flow* to obtain additional ATs.

By using the widely supported flows, such as the authorisation code flow and the refresh flow, `oidc-agent` can be used with any specification compliant OP.

`oidc-agent` makes use of a number of other flows to improve security and user experience. We use PKCE [17] whenever it is supported by the OP for both public and confidential clients as is recommended by the OAuth working group [18]. Also discovery [19, 20], dynamic client registration [21, 22], and token revocation [23] are supported.

Support for the *client credentials flow* [13], *token exchange flow* [24], as well as for *OpenID Federations* [25] were considered but we found including those would not be in line with our vision for `oidc-agent`. Especially, OpenID federations would enable the agent to be used with an extensive range of OPs without the need to register clients for all of them. However, the design of `oidc-agent` and OpenID federations are incompatible without the usage of an external service. It is still possible to use `oidc-agent` with all the OPs in the federation by utilising an OpenID Federation enabled third-party, e.g. a proxy or mytoken server.

6 Features

In this section we describe the key features of `oidc-agent` that make it particularly useful for scientific communities. Many of the features have been developed in collaboration with or on request of our users.

6.1 Security

Security is one of the prime design goals. Consequently, `oidc-agent` stores account configurations always in an encrypted way. It is not possible to use `oidc-agent` with account configurations stored in plain text.

By default the user is prompted for an encryption password when the account configuration is created, an encryption key is derived from the password and the account configuration is encrypted through the `libsodium` library [26] with the `XSalsa20` stream cipher. This requires the user to provide the encryption password whenever an account configuration is loaded (and also

when it needs to be updated). Additional methods for providing the encryption password are supported: it is possible to provide the password via an environment variable or via a file, or to specify a command that returns the password. For security reasons, passing the encryption password directly as a command-line option is not supported. Users are responsible for choosing the most suitable method for passing the password, considering their security requirements and their need for non-interactivity.

As an alternative to password-based encryption, `oidc-agent` also supports gpg-based encryption. In this mode the user provides a gpg-key id and the account configuration is encrypted for the associated key using `gpg-agent`. Depending on the user's `gpg-agent` configuration, this can considerably reduce the number of password prompts (`oidc-agent` does not prompt the user for a password in this mode, however `gpg-agent` might). In particular for cases where `oidc-agent` needs to update the account configuration file regularly, e.g. because the RT changes (see below), this is very useful: since with public-key methods the encryption can be done with only the public key, the user will not be prompted for any password regardless of its `gpg-agent` configuration. It is possible to enable gpg-based encryption in the configuration file for all (new) account configurations.

Encryption is also applied in memory. All configurations loaded into the agent include sensitive data – i.e. client credentials, (in the case of the *resource owner password credentials flow* also user credentials), and tokens. This sensitive data resides in encrypted memory whenever possible. It is only stored in its plain form while being processed.

The agent can also be locked: in this state the agent will decline all requests, until unlocked again with the user provided locking password. While locked this locking password is used to additionally encrypt all stored configurations.

Consequently, we also use a custom `free` function which overwrites the allocated memory before freeing it. This approach helps ensuring that no sensitive data remains in locations where attackers could potentially access it.

Since communication with the agent is done using a unix domain socket, access control is delegated to the file system. By default, any application running as the same user as the agent can communicate with the agent using its socket. `oidc-agent` provides an option for extending this to applications running as other users by utilising the file system's group mechanism. The socket can be created such that it is accessible by a unix group, in which case all applications that run as a group member may access the agent. Furthermore, it is possible to restrict access to the agent, e.g. by requiring confirmation from the user before tokens are returned. This can be enabled for all configurations stored in the agent or for individual account configurations.

Kupsch and Miller [27] describe different attacks and problems that can arise when opening a file (such as the socket). In line with their proposed safe implementation, the agent checks if the socket path is trustworthy before creating it.

6.2 Token Configuration

When requesting an AT from the agent, certain properties can be specified, including a minimum required (remaining) lifetime for the token. Usually, the AT is cached in the agent and if a token is requested multiple times while it is still valid, the same token is returned. To ensure that the token is valid long enough – and will not expire immediately after it is obtained, the minimum validity duration can be specified. It is also possible to request a fresh AT.

Additionally, it is possible to request ATs with certain *scopes* or *audiences*, as long as these are a subset of the scopes and audiences configured for that account configuration (usually the audiences are not restricted). To request ATs with specific audiences from an OP, the agent can use the mechanism described in RFC 8707 [28] or a proprietary one, supported by the `INDIGO IAM` service [29], an OP implementation, which is used by – among others – WLCG.

6.3 Autoload

`oidc-agent` also has auto-load support: this means that applications are not required to be concerned with whether an account configuration is loaded or not, they only need to request a token identified by an account name or by an OP URL. Everything else is handled by `oidc-agent`. When applications request a token for an OP

instead of an account configuration, the agent first determines which account configuration should be loaded (users can define the default account for each OP if they have multiple accounts for an OP). When the agent knows which account should be loaded, it reads the account configuration file and needs to decrypt it. If the file is encrypted with a password the agent asks the user through `oidc-prompt` for the password. In case of gpg-based encryption, the decryption is done via `gpg-agent`. After the configuration is decrypted it is loaded into the agent and an AT is obtained from the OP which is then returned to the requesting application. For the requesting application all of this is transparent.

The auto-load feature can be well-combined with an "auto-unload", i.e. the agent can be configured to keep configurations loaded only for a limited time, after that time the account configuration will be automatically un-loaded.

6.4 Auto generate

`oidc-agent` does not only support auto-load, but also *autogen*, i.e. when an application requests an AT for an OP for which currently no account configuration exists, the agent can trigger the creation of a new account, requiring only minimal user input. After the user logged in and authorised the agent, the agent returns an AT to the requesting application. Similar to auto-load, the requesting application does not notice that the configuration first had to be created – except for a delay in response time.

6.5 Rotating Refresh Tokens

Some OPs use *rotating refresh tokens*, this means that an RT can only be used once. Each time when an RT is used it is revoked and a new one is issued. This requires the agent to support an RT update and therefore, the agent needs to be able to update the encrypted account configuration file on disk. With a gpg-based encryption this can be done without user interaction, since public-key encryption is used. For a symmetric password-based encryption the agent needs the encryption password to update the file. This may require prompting the user. To avoid this, alternative methods for obtaining the password (password file, environment variable, or command) are supported. The agent may also store the password in memory when the account configuration is initially loaded, allowing it to update the account configuration file as needed.

6.6 Custom OpenID Provider configurations

Customising the agent's behaviour for different OPs can be done via the `issuer.config` file. This can be used to pass OP specific configurations, for example to set the default account for an OP as mentioned previously. It may also be very useful to automatically keep the encryption password stored in memory for those OPs that use RT rotation. oidc-agent ships a default `issuer.config` with this feature automatically enabled for all those OPs that are known to support RT rotation.

`oidc-agent` supports several standardised OIDC flows which are usable with a wide range of OPs. However, some OPs may require specific non-standard request parameters, others support additional parameters for extended behaviour. To support these specific extensions, `oidc-agent` supports custom request parameters, which can be configured in the `custom_parameters.config` file. This configuration file allows to specify which custom parameters should be included in each request type when using a particular OP or account configuration.

Although RTs are *long-lived* tokens, their specific lifetime is defined by the OP. I.e. RTs may expire eventually, in an often unpredictable time period. In this case a new RT has to be obtained by triggering a new OIDC authorisation flow, which requires user interaction. To avoid creating a new account configuration, the `--reauthenticate` option of `oidc-gen` simply triggers a re-authentication and updates the RT in the existing account configuration. When `oidc-agent` finds an expired or otherwise invalid RT during a token request it will automatically trigger the re-authentication.

6.7 Agent forwarding

By utilising the socket's file property and `ssh` features it is possible to forward the agent's socket to a remote host – following the concept of ssh-forwarding. It's also possible to forward the socket over multiple hops. Agent-forwarding allows users to obtain OIDC tokens on remote machines from

their local agent. Still, agent-forwarding has some limitations in use-cases where ATs are required on multiple remote machines over an extended period of time. In particular, if network connectivity to the host running the agent, or user interaction cannot be guaranteed.

Such use-cases typically involve long-running compute jobs in distributed environments. The increasing demand for supporting such use-cases has led to the development of the client-server based tool `mytoken` [30]. This service provides `mytoken` tokens to users that can be used to obtain ATs from multiple machines over an extended period of time in a secure way. Those mytokens can also be used and managed with `oidc-agent`.

7 Usage of oidc-agent

7.1 General Use-Cases

For the usage of `oidc-agent` we see two primary use-cases:

1. The user wants to use the AT directly on the command line, e.g. when calling a REST API.
2. The user wants to use another application (cli or gui) that needs an AT.

The first use-case can easily be solved through `oidc-token`. An example `curl` command could look like this:

```
$ curl -H "Authorization: Bearer
↪  $(oidc-token <account>)"
↪  https://example.com/api
```

or even shorter:

```
$ curl -H "$(oidc-token --auth-header
↪  <account>)" https://example.com/api
```

In the second use-case there are different approaches to pass the token. Which one to choose greatly depends on the application support. The ideal approach would be that the application directly communicates with the agent and the user only has to specify which account configuration or OP should be used – if at all. To ease the integration for applications, libraries for different programming languages exist. We provide an oidc-agent library for the `C`, `go`, and `python` programming languages. There is also an independent library for `rust` [31].

For applications that have other mechanisms in place to obtain the AT, `oidc-token` or `oidc-tokensh` can be utilised to pass the token via command, command-line option, environment variable, or file.

7.2 `oidc-agent` in Practice

In this section we describe how `oidc-agent` is used in general and practice.

`oidc-agent` is actively being used by a variety of different communities for different tasks. Even though we are not aware of all usage scenarios, we list the ones available to us here for reference:

- Youtube downloader app [32] from the commandline requires valid google access tokens.
- Job submission in WLCG. We are aware of advanced use cases for longer running jobs, that are solved in different ways. One way is our tool mytoken [30].
- SSH with OIDC [33] makes use of `oidc-agent`. Clients either need to provide ATs on the commandline or may use the `mccli` commandline, which uses the oidc-agent client library for obtaining valid access tokens.
- On windows, putty users may use `oidc-agent` for obtaining ATs for authenticating to OIDC-enabled ssh servers.
- Our ssh online-ca – `oinit` [34] – returns ssh-certificates to users that authenticated via OIDC on the commandline.
- At the Japanese National Institute for Informatics `oidc-agent` is being used for authenticating to APIs and services of the Japanese HPC infrastructure.
- The unicore [35] HPC middleware is token based, and relies on OIDC. Its client documentation describes `oidc-agent` for authenticating from the commandline.
- Similarly, in the grid world, the Scandinavian ARC middleware is token based and describes the use of `oidc-agent` in its documentation [36].
- The Fenix HPX Research Infrastructure documents how to use oidc-agent with their infrastructure [37].
- The rclone data transfer tool uses `oidc-agent` for authenticating to servers that support OIDC authentication, such as webdav via OIDC in dCache [38].
- We have collaborated with users from further HPC sites (CESGA, NII) and the Grid

(Fermilab) to improve their use cases with `oidc-agent`.

8 Availability/Distribution

`oidc-agent` is available under the MIT license, the source code is available at GitHub [3], and we are open for contributions in all forms including feedback, feature requests, pull requests, and packaging. We provide support for users through our mailing list[1], as well as our documentation page [5].

We build packaged release and pre-release versions for a wide number of different debian- and rpm-based distributions. Those are provided at http://repo.data.kit.edu.

Furthermore, `oidc-agent` has been included into the official repositories of different distributions. For `debian` `oidc-agent` is included since Debian 12, but is also included in the backports for Debian 11. Unfortunately, the version in the `debian` repositories has not yet been updated to version 5 at the time of writing; however this is currently actively worked on.

For `rpm`-based distributions `oidc-agent` is officially included in EPEL since version 7 and `Fedora` since version 37.

Other distributions including `oidc-agent` are `Gentoo`, `LiGurOS`, the Nix-packaging system, and the `Arch Linux` user repositories.

For non-Linux systems we provide `oidc-agent` for MacOS via `homebrew`. Some smaller user-experience enhancements might not be supported on the MacOS version and the service integration is technically different from Linux, but usage and experience is in line with the Linux version. For `Windows` we provide an installer that installs an `oidc-agent` implementation based on `MSYS2`, but it is also possible to use `oidc-agent` in the Windows-Subsystem for Linux (WSL2).

9 Summary

`oidc-agent` is a tool suite for managing OpenID Connect tokens on the command line. It supports many different OIDC flows and therefore can be used with any OIDC-compliant OpenID Provider. This certainly is one of the reasons why `oidc-agent` has gained broad adoption and has become the de facto standard solution in research and education communities for managing OIDC tokens on the command line. Its strength lies in secure and user-friendly access to tokens, while still offering a wide range of advanced features.

[1] https://www.lists.kit.edu/sympa/subscribe/oidc-agent-user

Declarations

Funding

No funds, grants, or other support was received.

Competing Interests

The authors have no competing interests to declare that are relevant to the content of this article.

References

[1] Dack, Thomas, Agostini, Federica, Basney, Jim, Cornwall, Linda, De Stefano Jr, John Steven, Dykstra, Dave, Giacomini, Francesco, Litmaath, Maarten, Miccoli, Roberta, Sallé, Mischa, Short, Hannah, Vianello, Enrico: Wlcg transition from x.509 to tokens. status, plans, and timeline. EPJ Web of Conf. **295**, 04054 (2024) https://doi.org/10.1051/epjconf/202429504054

[2] Gudu, D., Hardt, M.: motley-cue. https://doi.org/10.5281/zenodo.7353878

[3] Zachmann, G., Hardt, M., Dykstra, D., Marschke, L., Orviz, P., Duma, D.C., Lenk, M., Freyermuth, O., Burr, C., Çayoğlu, U., Burgey, L., Traylen, S., Ellert, M., Bockelman, B.P., Eve, Grenier, B., Wegh, B., edquist, Dehtyarov, D., Kelly, J., Millar, P., Risco, S., Saleeba, T., Vokac, P.: indigo-dc/oidc-agent (2025). https://github.com/indigo-dc/oidc-agent

[4] Zachmann, G., Hardt, M., Dykstra, D., Marschke, L., Orviz, P., Duma, D.C., Lenk, M., Freyermuth, O., Burr, C., Çayoğlu, U., Burgey, L., Traylen, S., Ellert, M., Bockelman, B.P., Eve, Grenier, B., Wegh, B., edquist, Dehtyarov, D., Kelly, J., Millar, P., Risco, S., Saleeba, T., Vokac, P.:

indigo-dc/oidc-agent (2025). https://doi.org/10.5281/zenodo.4966816

[5] Zachmann, G., Hardt, M.: Oidc-agent: A Set of Tools to Manage OpenID Connect Tokens. (2025). https://indigo-dc.gitbook.io/oidc-agent

[6] Skokan, F., Gonçalves, G., Pospisil, J.: panva/openid-client-cli. GitHub Repository (2018). https://github.com/panva/openid-client-cli

[7] Goering, B.: gobengo/oidc-cli. GitHub Repository (2015). https://github.com/gobengo/oidc-cli

[8] Sjöström, L., Mjönes, J.: shelmangroup/oidc-agent. GitHub Repository (2019). https://github.com/shelmangroup/oidc-agent

[9] Reimann, J., Fuller, J.: ctron/oidc-cli. GitHub Repository (2024). https://github.com/ctron/oidc-cli

[10] OpenSSH: OpenSSH. https://www.openssh.com Accessed 2024-07-29

[11] Bockelman, B., Ceccanti, A., Dack, T., Dykstra, D., Litmaath, M., Sallé, M., Short, H.: Wlcg token usage and discovery. EPJ Web of Conferences **251**, 02028 (2021) https://doi.org/10.1051/epjconf/202125102028

[12] Sakimura, N., Bradley, J., Jones, M., Medeiros, B., Mortimore, C.: Openid connect core 1.0 incorporating errata set 2. Technical report, OpenID Foundation (December 2023). https://openid.net/specs/openid-connect-core-1_0.html

[13] Hardt, D.: The oauth 2.0 authorization framework. RFC 6749, RFC Editor (October 2012). http://www.rfc-editor.org/rfc/rfc6749.txt

[14] Jones, M., Hardt, D.: The oauth 2.0 authorization framework: Bearer token usage. RFC 6750, RFC Editor (October 2012). http://www.rfc-editor.org/rfc/rfc6750.txt

[15] Jones, M., Bradley, J., Sakimura, N.: Json web token (jwt). RFC 7519, RFC Editor (May 2015). http://www.rfc-editor.org/rfc/rfc7519.txt

[16] Denniss, W., Bradley, J., Jones, M., Tschofenig, H.: Oauth 2.0 device authorization grant. RFC 8628, RFC Editor (August 2019). http://www.rfc-editor.org/rfc/rfc8628.txt

[17] Sakimura, N., Bradley, J., Agarwal, N.: Proof key for code exchange by oauth public clients. RFC 7636, RFC Editor (September 2015). http://www.rfc-editor.org/rfc/rfc7636.txt

[18] Lodderstedt, T., Bradley, J., Labunets, A., Fett, D.: Oauth 2.0 security best current practice. Internet-Draft draft-ietf-oauth-security-topics-29, IETF Secretariat (June 2024)

[19] Sakimura, N., Bradley, J., Jones, M., Jay, E.: Openid connect discovery 1.0 incorporating errata set 2. Technical report, OpenID Foundation (December 2023). https://openid.net/specs/openid-connect-discovery-1_0.html

[20] Jones, M., Sakimura, N., Bradley, J.: Oauth 2.0 authorization server metadata. RFC 8414, RFC Editor (June 2018). http://www.rfc-editor.org/rfc/rfc8414.txt

[21] Sakimura, N., Bradley, J., Jones, M.: Openid connect dynamic client registration 1.0 incorporating errata set 2. Technical report, OpenID Foundation (December 2023). https://openid.net/specs/openid-connect-registration-1_0.html

[22] Richer, J., Jones, M., Bradley, J., Machulak, M., Hunt, P.: Oauth 2.0 dynamic client registration protocol. RFC 7591, RFC Editor (July 2015). http://www.rfc-editor.org/rfc/rfc7591.txt

[23] Lodderstedt, T., Dronia, S., Scurtescu, M.: Oauth 2.0 token revocation. RFC 7009, RFC Editor (August 2013). http://www.rfc-editor.org/rfc/rfc7009.txt

[24] Jones, M., Nadalin, A., Campbell, B., Bradley, J., Mortimore, C.: Oauth 2.0 token

exchange. RFC 8693, RFC Editor (January 2020). http://www.rfc-editor.org/rfc/rfc8693.txt

[25] Hedberg, R., Jones, M.B., Solberg, A.A., Bradley, J., Marco, G.D., Dzhuvinov, V.: Openid federation 1.0 - draft 36. Technical report, OpenID Foundation (May 2024). https://openid.net/specs/openid-federation-1_0.html

[26] libsodium: libsodium (2024). https://doc.libsodium.org/

[27] Kupsch, J., Miller, B.: How to open a file and not get hacked, pp. 1196–1203 (2008). https://doi.org/10.1109/ARES.2008.53

[28] Campbell, B., Bradley, J., Tschofenig, H.: Resource indicators for oauth 2.0. RFC 8707, RFC Editor (February 2020). http://www.rfc-editor.org/rfc/rfc8707.txt

[29] INFN: INDIGO IAM (2024). https://indigo-iam.github.io/

[30] Zachmann, G.: mytoken - openid connect tokens for long-term authorization. Master's thesis, Karlsruher Institut für Technologie (KIT) (2021). https://doi.org/10.5445/IR/1000134712 . 46.12.02; LK 01

[31] Nastał, W.: Nithe14/oidc-agent-rs (2025). https://github.com/Nithe14/oidc-agent-rs

[32] Sergey, M., Hagemeister, P., Remi, T., Andan, P., Ferrandiz Marquinez, J., Yan, C.-H., Only, B., Heidel, T.-O., Valsorda, F., Garcia, R., Djnz, C., Sawicki, S., Miscore, L., Dev, S.: YT-DLP - a feature-rich command-line audio/video downloader. https://github.com/yt-dlp/yt-dlp

[33] Gudu, D., Hardt, M., Brocke, L., Zachmann, G.: Enabling secure shell access with openid connect. Computing and Software for Big Science **9**(1) (2025) https://doi.org/10.1007/s41781-025-00136-5

[34] Hardt, M., Gudu, D., Zachmann, G., Brocke, L.: Evolution of SSH with OpenId Connect. PoS **ISGC2024**, 023 (2024) https://doi.org/10.22323/1.458.0023

[35] Erwin, D.W., Snelling, D.F.: Unicore: A grid computing environment. In: Euro-Par 2001 Parallel Processing: 7th International Euro-Par Conference Manchester, UK, August 28–31, 2001 Proceedings 7, pp. 825–834 (2001). Springer

[36] NorduGrid Collaboration: ARC support for OIDC (2021). https://www.nordugrid.org/arc/arc6/misc/oidc_tokens.html

[37] Long, S., Pleiter, D., Patrascoiu, M., Padrin, C., Carpene, M., More, S., Carpio, M.: Integrating fts in the fenix hpc infrastructure. In: EPJ Web of Conferences, vol. 295, p. 01037 (2024). EDP Sciences

[38] Ernst, M., Fuhrmann, P., Gasthuber, M., Mkrtchyan, T., Waldman, C.: dCache, a distributed storage data caching system (2001)