WLCG Bearer Token Discovery

Authored by the WLCG AuthZ Working Group

Proposal

Client tools that rely on a bearer token for authenticating themselves need a mechanism for receiving the tokens from their environment. While the browser is a monolithic user agent (and can internally manage tokens), the terminal environment involves a number of independently-developed tools; the environment needs a way to communicate the token to be used to Unix processes. To the best of our knowledge, there's no previously defined standard about how a Unix tool should discover a token from its environment.

If a tool needs to authenticate with a token and does not have out-of-band WLCG Bearer Token Discovery knowledge on which token to use, the following steps to discover a token MUST be taken in sequence (where `$ID` below is taken as the process's effective user ID):

1. If the `BEARER_TOKEN` environment variable is set, then the value is taken to be the token contents.
2. If the `BEARER_TOKEN_FILE` environment variable is set, then its value is interpreted as a filename. The contents of the specified file are taken to be the token contents.
3. If the `XDG_RUNTIME_DIR` environment variable is set*, then take the token from the contents of `$XDG_RUNTIME_DIR/bt_u$ID` **.
4. Otherwise, take the token from `/tmp/bt_u$ID`.

If a potential token is found at a step, then the discovery implementation MUST strip all whitespace on the left and right sides of the string (we define whitespace the same way as the C99 `isspace` function: space, form-feed (`\f`), newline (`\n`), carriage return (`\r`), horizontal tab (`\t`), and vertical tab (`\v`)). Upon finding a valid token according to section 2.1 of RFC6750, the discovery procedure MUST terminate and return this token. Upon finding an empty token, the discovery implementation should continue with the next step. Upon finding an invalid token, the implementation SHOULD stop and return an error.

Once a valid token, `$TOKEN`, is discovered, if it is used to authenticate an HTTP request, the tool MUST use it in accordance with RFC6750, for example in the Authorization header as follows:

```
Authorization: Bearer $TOKEN
```

High-level tools that need to simultaneously support bearer tokens for multiple purposes (e.g. multiple VOs) MAY set `$BEARER_TOKEN_FILE` using the patterns of steps 3 and 4 with filenames having an added hyphen and purpose name appended to the filename. For example, a toolset named fife, keeping one token per VO, may choose the following name for user 1221 and VO CMS:

```
/tmp/bt_u1221-fife-CMS
```

The `purpose name` is deliberately left undefined and intended for use by the tool implementer. These high-level tools SHOULD consider potential filename collisions with other tools when implementing a naming scheme. When executing lower-level tools, the high-level tool SHOULD set the `$BEARER_TOKEN_FILE` to the desired file. Tools SHOULD NOT inspect multiple tokens to try to determine which one to use based on content.

---

* https://specifications.freedesktop.org/basedir-spec/basedir-spec-latest.html

** This additional location is intended to provide improved security for shared login environments as `$XDG_RUNTIME_DIR` is defined to be user-specific as opposed to a system-wide directory.