Token Taxonomy

Brian Bockelman
European HTCondor Workshop 2023
Tokens: A Primer
Authorization and Credentials

• A credential is a document detailing an issued identity or an authorization.
  • Think: passport, driver’s license, diploma.

• In computing, we typically use credentials as part of the **authorization** to utilize a resource.

• Two common approaches to auth’z:
  • Authentication and identity mapping: credential establishes *who you are* and then mapped to a local identity with enumerated authorizations.
  • Capabilities: credential is an assertion of *what you can do*.

**Capabilities are the right way to go on distributed systems!**
Bearer Tokens and JWT

- Capability-based credentials are often implemented as **bearer tokens**.
  - To establish authorization, one must establish “proof of possession” to the remote side.
  - For X.509 credentials, this is done by signing a statement with the private key bound to the credential.
  - For tokens, we prove possession by sending the token itself to the remote side.
    - **What are the tradeoffs here?**
- Whoever bears the token is assumed to have the authorization – hence the name!
Bearer Tokens and JWT

- An **opaque** bearer token is a random sequence of bytes.
  - Quite secure but typically difficult to coordinate in a distributed system with many independent actors.

- A **JSON Web Token (JWT)** is a bearer token using a specific JSON-based format:
  - Often signed by a public key.
  - Allows arbitrary key-value pairs to be asserted.
  - Sequence of RFCs defining the semantics of certain keys (subject, expiration time, issuer, unique identifier).
Building Authorization Schemes with JWTs

• Think of a JWT as a format and a toolbox for building an authorization system.
• To build an authorization system, there needs to be common agreement on how to interpret the contents of the JWT and common semantics.
• We must therefore build a profile describing how the system work.
  • Analogy: X.509 vs GSI
  • Analogy: Grammar vs Language

Note: An authorization profile is both a superset and subset of JWTs. It restricts what’s considered a valid token plus adds rules for interpretation.
HTCSS and Tokens
Authorization vs Management

HTCSS uses tokens in two contexts:

• **Authorization**: Providing access to a HTCSS component / daemon.

• **Credential Management**: Managing credentials on behalf of a user, typically meant for using credentials within jobs.
HTCSS supports two different profiles for token-based authorization:

- **IDTOKENS**: HTCSS’s “native” token format for establishing identity.
- **SCITOKENS**: Authorization based on signed JWTs from an independent issuer.
Parts of an IDTOKEN

An IDTOKEN is a token signed with a symmetric key held by the remote daemon. It has the following parts:

- **Identifier**: This is the “HTCondor identity”
- **Issuer**: The trust domain – where the token is expected to be honored.
- **Signing key name**: The name of the key used to sign the token. The server must have this key to accept the token!
- **Restrictions on use**:  
  - “Not before” and “expiration” times.  
  - A list of authorizations for the token (intersected with the authorizations the HTCondor identity already has!).

```json
{
    "iat": 1588710496,
    "iss": "flock.opensciencegrid.org",
    "scope": "condor:/READ condor:/ADMINISTRATOR",
    "sub": "osg-admin@flock.opensciencegrid.org"
}
```

**Discussion Topics:**

- Why symmetric signatures?
- What’s important about the “trust domain”?
IDTOKEN Authentication

• The IDTOKEN authentication protocol is based on a shared secret verification protocol (AKEP2).

1. Client sends the public part of the token to the server and a client nonce.
2. Server uses its symmetric key to compute the token signature.
   • Now both sides can derive a shared secret based on the token signature!
3. Server responds with a server nonce and the hash of the token + client nonce + shared secret.
4. Client verifies the server response and sends its hash of the token + server nonce + shared secret.

**To succeed:** Client needs the signed token, server needs the signing key.

**Note:** at no point are secrets sent over the network! Public contents of the token are sent in the clear.
SCITOKENS

• Like IDTOKENS, SCITOKENS builds on JWT.
  • Uses libSciTokens from the scitokens-cpp project. Any token conforming to a supported profile can be used (misnomer: not limited to SciTokens. WLCG tokens are great!).

• Signatures are public key based. Daemon does not need the private key to verify the token’s validity.
  • Good when the daemon is independent from the signing entity.
  • Daemon looks up the signing key based on the issuer URL.

• Otherwise, many similar concepts – subjects, issuers, restrictions.
  • Since it’s not a “native” credential, the subject/groups are mapped to a local HTCondor identifier.
SCITOKENS Authentication

• Since the daemon doesn’t have the signing key, we don’t have a shared secret.

• Instead, we bootstrap the secure channel by establishing a TLS connection from client to server.
  • **NOTE**: Implies the server has a host certificate, shares common CAs with client.

• Once established, the client sends the entire token to the server.
  • Server can then verify the token then authorize the client.
  • **Food for thought**: what’s the impact of a malicious server?
Token Arcana

We try to have a complete ecosystem around tokens. Examples:

- **condor_token_request**: Securely request any arbitrary token.
  - *Approval* of such a request is left to the admin. Great for bootstrapping authentication when you have a way to communicate with the user out-of-band. No copy/pasting tokens in your email!

- **condor_token_fetch**: Returns a fresh IDTOKEN equivalent to the current security session.

- **condor_scitoken_exchange**: Returns a fresh IDTOKEN equivalent to the mapped input SciToken.
  - Note: SCITOKENS auth requires a TLS cert for the remote daemon. IDTOKENS does not!
Credential Management
An HTCSS AP can manage a user’s **credential wallet**.

- Each user has their own wallet.
- The types of available credentials are configured by the administrator.
- A job specifies the credentials it needs!
- HTCondor ensures the token is available & up-to-date on the EP.
Credentials – the User View

Each job specifies a list of credential services required to execute:

• The credential services list is managed by the AP administrator.

• Some services can generate multiple credentials; these additional credentials are referred to by “handles”.
  • This allows the user to specify more fine-grained authorizations than the default.

condor_submit interprets this list, potentially asking the user for additional information to generate the credential.

• Depending on the credential service implementation, this information may come via the CLI (Kerberos) or a link to complete the generation in a browser (OAuth2).
WARNING: We’re bad at naming things

• We call the condor_submit configurations for credential management “oauth” :(  
  • In fact, only one of four commonly used plugins use oauth!

---

**use_oauth_services = <list of credential service names>**

A comma-separated list of credential-providing service names for which the job should be provided credentials for the job execution environment. The credential service providers must be configured by the pool admin.

**<credential_service_name>_oauth_permissions[_<handle>] = <scope>**

A string containing the scope(s) that should be requested for the credential named **<credential_service_name>[_<handle>]**, where `<handle>` is optionally provided to differentiate between multiple credentials from the same credential service provider.

**<credential_service_name>_oauth_resource[_<handle>] = <resource>**

A string containing the resource (or “audience”) that should be requested for the credential named **<credential_service_name>[_<handle>]**, where `<handle>` is optionally provided to differentiate between multiple credentials from the same credential service provider.

---

**Work-in-progress:**

Significant cleanup of terminology and abstractions are desired for HTCSS 23.x! For example:

• You can’t ask the AP the list of services.
• You can’t enumerate the available handles or their definitions.
• You can’t fetch your own credentials.
• You can’t BYOC: Bring Your Own Credential.
• The “API” to interact is largely condor_submit
Credentials - the Admin View

Credentials in the wallet are **opaque**. The AP doesn’t assume a specific type or format!

• Hence, each credential type must be serviced by a daemon implementing a “credmon” interface.
  • Several daemons are shipped with HTCSS ... but you’re encouraged to write your own if needed!
  • There are two credmon interfaces – Oauth and Kerberos. Each AP can only have one of each.

• The CredD daemon provides the API for credential management.
  • Must be running for users to utilize the wallet!
We’re bad at naming things, Redux

• Want to store a credential?
  Use condor_store_cred

• Want to delete a credential?
  Use condor_store_cred

• Want to query a credential?
  Use condor_store_cred

• Want to list your credentials?
  Too bad!
Credential Management Data Model

- Credmon 1 (type: OAuth2)
- Credmon 2 (type: KRB5)

Only one credmon per type is permitted

- Service A
- Service B
- Service C

Handles are optional

Admin View

User/Job View
Our Complete Taxonomy

Tokens in HTCSS

Authorization

IDTOKENS

SCITOKENS

Credential Management

OAuth2

Local Issuer

Vault

Kerberos
Questions?

This project is supported by the National Science Foundation under Cooperative Agreements OAC-2030508 and OAC-2114989. Any opinions, findings, conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.