SciTokens and Credential Management

Zach Miller zmiller@cs.wisc.edu
Jason Patton jpatton@cs.wisc.edu
HTCondor Week 2019

This material is based upon work supported by the National Science Foundation under Grant No. 1738962. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.
The SciTokens project, started July 2017, aims to:

- Introduce a *capabilities-based authorization infrastructure* for distributed scientific computing,
- Provide a *reference platform*, combining CILogon, HTCondor, CVMFS, and XRootD, and
- Implement specific use cases to help our science stakeholders (LIGO and LSST) better achieve their scientific aims.
At the core of today’s grid security infrastructure is the concept of *identity* and *impersonation*.

- A grid certificate provides you with a globally-recognized identification.
- The grid proxy allows a third party to impersonate you, (ideally) on your behalf.
- The remote service maps your identity to some set of locally-defined authorizations.

We believe this approach is fundamentally wrong because it exposes too much global state: identity and policy should be kept locally!
Capability-based Authorization

• We want to change the infrastructure to focus on *capabilities*!

• The tokens passed to the remote service describe what authorizations the bearer has.

• For traceability purposes, there may be an identifier that allows tracing of the token bearer back to an identity.

• Identifier ≠ identity. It may be privacy-preserving, requiring the issuer (VO) to provide help in mapping.

• Example: “The bearer of this piece of paper is entitled to write into /data/zmiller".
If GSI took over the world, an attacker could use a stolen grid proxy to make withdrawals from your bank account.

With capabilities, a stolen token only gets you access to a specific authorization ("stageout to /data/zmiller at Wisconsin").

SciTokens is following the principle of least privilege for distributed scientific computing.
• Integrating an OAuth2 client on the HTCondor submit host
• Enhancing HTCondor to manage token refresh and delivery to jobs
• Enhancing data services (e.g. Xrootd) to allow read/writes using tokens instead of grid proxies
• Runs under the condor_master like all other HTCondor daemons

• Manages credentials stored in a special “credential directory” with restricted permissions. Regular users cannot read or write within this directory, but the CredD can.
• Has two “modes”
  • Kerberos mode, which I talked about last year
  • OAuth mode, which I am talking about now

• Currently the two modes cannot coexist due to different conventions for layouts of the credential directory

• Future work includes merging these modes so both can be used at the same time
• In the old “Kerberos Mode”, the CredD would only hold one credential per user.

• The CredD in “OAuth Mode” can now hold multiple credentials per user

• I’m skipping the internals for this talk and focusing more on the higher-level concepts, but please come talk to me if you are curious or have questions.
• Okay… back to OAuth mode!
• The CredD in “OAuth Mode” can now hold multiple credentials per user
• These can be tokens from different services:
  • scitokens
  • box.com
• There can be different scopes (permissions) for the same service:
  • scitokens_uw_read_zmiller
  • scitokens_uw_write_jpatton
• The user defines the tokens they need and the names (handles) and scopes in their submit file

• Jason will describe and demo that later…
  • w00t! DEMO! 😊
• The CredD itself deals with the secure storage and retrieval of the credentials

• It does NOT know or understand the contents of the credentials – they are opaque to the CredD

• Another component is in charge of understanding and manipulating OAuth tokens: the CredMon
• Responsible for obtaining tokens by talking to the various services

• Monitors the existing tokens and knows how to refresh them

• Receives signals from the CredD when there is potentially new work for it do do
Architecture

Job Submission
- condor_submit
- condor_schedd
- condor_credd
- condor_shadow

Job Execution
- condor_startd
- condor_starter
- User’s job

Data Access
- Data Server
  (XRootD)

Token Server
- Identity Provider

Policy DB

User

= refresh tokens
= access tokens
• User specifies in their submit file what credentials they need.

• Run `condor_submit`:

Hello, zmiller.
Please visit:
https://baphomet.cs.wisc.edu/key/f40740d...34a0eebac1

• User does so and follows directions
  • That’s Jason’s demo and I’m not going to steal his thunder!
• User specifies in their submit file what credentials they need.
• Run `condor_submit`:

    Submitting job(s).
    1 job(s) submitted to cluster 39033.

• This time it worked because `condor_submit` checked with the CredD and all the tokens were present. Thus, the job can now run!
• Job matches and starts running
• After the sandbox directory is created, but BEFORE any files are transferred, the condor_starter calls back to the condor_shadow to fetch tokens
• Only the tokens for THAT job are sent
• Only the ACCESS tokens are sent
• HTCondor ensures the communication channel is encrypted, or it refuses to send the tokens.
The access tokens are placed into the job sandbox in the `.condor_creds` directory.

Environment variable within the job `_CONDOR_CREDS` points to the full path for that directory.

Tokens are refreshed periodically while job continues running.

Tokens are cleaned up automatically when job exits since they are in the job sandbox.

```
my_prog --token=$_CONDOR_CREDS/scitokens.use
```
• Get a certificate for their submit server
• Configure box.com
  • You need a developer account
  • Create a new app
• Register your submit server
• Configure HTCondor
• This will appear in more and complete detail on the HTCondor Wiki:
  • https://htcondor-wiki.cs.wisc.edu/index.cgi/wiki
• One fairly straightforward way to get a certificate is by using the Let’s Encrypt service and certbot

• https://letsencrypt.org/
• Create a custom box.com app that uses OAuth
Configuration

Configure the authentication and permissions for your app to begin using the Box APIs. Check out our Getting Started Guide for a walkthrough of these settings.

OAuth 2.0 Credentials

Credentials for using OAuth 2.0 as your Authentication type.

Client ID

wluxsxho2c4vabn3xs681h0c0fznwu

Client Secret

***************

OAuth 2.0 Redirect URI

The redirect URI is the URL within your application that will receive OAuth 2.0 credentials.

Redirect URI

https://baphomet.cs.wisc.edu/return/box

Save Changes
Example configuration for the submit machine to interface with box.com

# Box.com client
BOX_CLIENT_ID = wluxtsxho2c4vabn3xs6n81h0c0fznwu
BOX_CLIENT_SECRET_FILE = /etc/condor/.secrets/box
BOX_RETURN_URL_SUFFIX = /return/box
BOX_AUTHORIZATION_URL =
    https://account.box.com/api/oauth2/authorize
BOX_TOKEN_URL = https://api.box.com/oauth2/token
BOX_USER_URL = https://api.box.com/2.0/users/me
• Many details were glossed over
SciTokens Credmon and Job Submission
• Two parts:
  • Credential Monitor
  • Web app (Python Flask framework)

• Currently supports:
  • OAuth2-style tokens (including SciTokens)
  • Locally issued SciTokens (i.e. issue-on-submit)

• Separate package from HTCondor
  • Near future: `yum install python-scitokens-credmon`
SciTokens Credmon – OAuth2 Support

Web app
Gathers initial tokens

1. Reads "key" file with user’s and OAuth2 providers’ info.
Web app
Gathers initial tokens

1. Reads “key” file with user’s and OAuth2 providers’ info.
2. Sends user to OAuth2 providers for authentication and authorization.
Web app

Gathers initial tokens

1. Reads "key" file with user’s and OAuth2 providers’ info.
2. Sends user to OAuth2 providers for authentication and authorization.
3. Stores refresh and access tokens in credential directory.
Credential Monitor

*Keeps active tokens refreshed*

1. Scans credential directory for valid refresh tokens.
Credential Monitor

*Keeps active tokens refreshed*

1. Scans credential directory for valid refresh tokens.
2. Refreshes corresponding access tokens.
Credential Monitor

Keeps active tokens refreshed

1. Scans credential directory for valid refresh tokens.
2. Refreshes corresponding access tokens.
3. Writes `CREDMON_COMPLETE` (watched by CredD).
Submitting OAuth2 Jobs
OAuth2 Submit Syntax

- `use_oauth_services = <service1, service2, ...>`
  - **REQUIRED** list of requested OAuth2 service providers, which must match (case-insensitive) the provider names in the HTCondor config.

Minimal example - Single provider with no required scopes or resources:

```
executable = transfer_my_box_file.py
arguments = htcondor/testfile.txt

use_oauth_services = box

queue

$_CONDOR_CREDS/box.use
```
OAuth2 Submit Syntax

- `use_oauth_services = <service1, service2, ...>`
  - REQUIRED list of requested OAuth2 service providers, which must match (case-insensitive) the provider names in the HTCondor config.

- `<SERVICE>_oauth_permissions[_<HANDLE>] = <scope1, scope2, ...>`
  - List of requested token scopes. OPTIONAL IF the OAuth2 service provider does not require a scope. The user can provide a handle to give a unique name to the token.

- `<SERVICE>_oauth_resource[_<HANDLE>] = <resource>`
  - The resource that the token should request permissions for. OPTIONAL IF the OAuth2 provider does not require a resource (a.k.a. audience) to be defined.

Note that service providers are defined by the admin in the config and handles are user-defined (optional).
Multiple scopes and resources:

```sh
eexecutable = compute_stats
arguments = --in=https://mironlab.wisc.edu/shared/rawdata.zip
              --out=https://jpatton.wisc.edu/home/jpatton/analysis.txt

use_oauth_services = uwtokens

uwtokens_oauth_permissions_read = read:/shared
uwtokens_oauth_resource_read = https://mironlab.wisc.edu/

uwtokens_oauth_permissions_write = write:/home/jpatton
uwtokens_oauth_resource_write = https://jpatton.wisc.edu/
```

queue

```sh
$_CONDOR_CREDS/uwtokens_read.use
$_CONDOR_CREDS/uwtokens_write.use
```
Live Demo

SCI TOKENS
Thank You!

jpatton@cs.wisc.edu
zmiller@cs.wisc.edu