Placement Tokens

Capability-Based Authorization for Job Placement

Mátyás Selmeci

HTC25 — June 2nd, 2025

CHTC HTCondor Software Suite PATh

Introduction

- Experimental work in the area of authorization

- Making it easier to grant, audit, and revoke access to the job placement services (submit, remove, examine, modify jobs and lists of jobs) that an Access Point provides

Motivation

- Consider a classroom environment
- 30 domain science students for a semester-long class
- Need access to HTC to do their work
- Not familiar or comfortable with the command line
- Don't need shell access on a Unix machine
- Don't need yet another username/password to remember

Goals

- Grant students access to the AP's *services* for the duration of the class
- Revoke the access at the end of the class
- Not require an account on the AP with privileges they won't use

Current state

- Placement does not require logging in to the AP (remote placement/"remote submit") but has limitations
- Job "ownership" in the queue is linked to OS user
- A user can't place jobs without their own OS account at the AP
- AP admin must:
  - Provision an account when a new user joins
  - Deprovision the account when the user leaves
  - Create and give the user a credential for remote placement

Improvements to make

"AP User" is already a first-class concept in HTCondor — HTCondor has its own user database, but the user still needed to be linked to an existing OS account with the same name

1. Unlink the AP User from the OS account
   - Create "generic accounts" to own processes and files for jobs in the queue
   - Dynamically associate AP Users with accounts as needed (like "slot users")
2. Give the user a way to obtain credentials by themselves

Improvements to make

"AP User" is already a first-class concept in HTCondor — HTCondor has its own user database, but the user still needed to be linked to an existing OS account with the same name

1. Unlink the AP User from the OS account
   - Create "generic accounts" to own processes and files for jobs in the queue
   - Dynamically associate AP Users with accounts as needed (like "slot users")

2. Give the user a way to obtain credentials by themselves

Self-service credentials? Let's consider why and how

Who needs authorization?

- The user needs authorization?

~~Who~~ What needs authorization?

- ~~The user needs authorization?~~

- The user's program needs authorization?

~~Who~~ What needs authorization?

- ~~The user needs authorization?~~
- ~~The user's program needs authorization?~~
- Some of the user's programs need authorization?

~~Who~~ What needs authorization?

- ~~The user needs authorization?~~
- ~~The user's program needs authorization?~~
- ~~Some of the user's programs need authorization?~~
- Some of the user's programs need some kind of authorization

- Not all programs need access to the same things
- Not all programs should be given access to the same things
- Least Privilege: Give programs only the access they need to do their jobs
- Already a model for that: Capabilities

Capabilities

- A Capability is some object ("token") that provides the bearer the authorization to perform a certain set of actions on a certain resource
- Capabilities can be copied and delegated — user to program, program to program, user to user
- Holding the Capability should be sufficient to grant authorization for the actions — the identity of the bearer should not matter

Early attempt: IDTOKENs

- HTCondor already had a form of token — IDTOKENs — but they are not real capabilities
- IDTOKENs contain the identity of the bearer; that identity is checked against HTCondor's access controls when the token is used
  - e.g. the token's "subject" must be in ALLOW_WRITE and cannot be in DENY_WRITE
- IDTOKEN "scopes" can be used to further restrict the bearer's access but it's still the identity that determines access
- Token creation not recorded (except in text log)

From IDTOKENs to Placement Tokens

- Let's improve on this: make a "Placement Token" that behaves more like a pure capability
- Once issued, the ID of the bearer ("subject") will *not* matter for authorization
- The ID will not get checked against ALLOW_WRITE et al. If the token says the bearer can write, they can write
- Need more care when issuing tokens — and keeping track of them

From IDTOKENs to Placement Tokens

- Create a dedicated daemon for making placement tokens: the PlacementD

- Create a table of user names with what authorizations they should be able to acquire

- Create a database for keeping track of the created capabilities

From IDTOKENs to Placement Tokens

- Only the PlacementD will make placement tokens
- The PlacementD will not create a token for a user that is not in its table
- The PlacementD will not create a token for a user whose access has expired (in the table)
- The PlacementD will not create a token with a privilege that the user is not listed as being allowed to have (in the table)

From IDTOKENs to Placement Tokens

- The PlacementD will record in its database who requested the capability, what AP user it's for, what authorizations it has, when it will expire, etc.

- The SchedD will read from the database when determining whether to allow an action

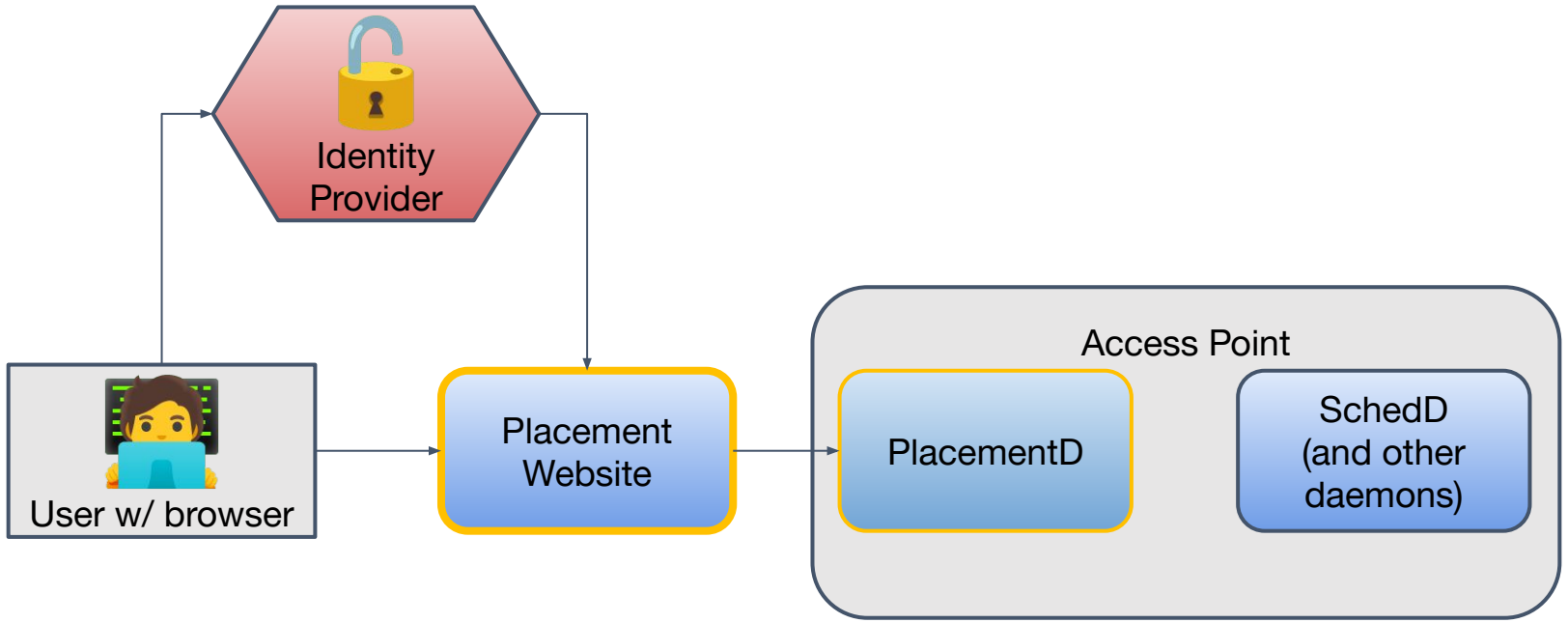
- Removing the token's entry from the database invalidates the token — no token without a record

Self-Service Capabilities

- AP admin does not know which programs the user will run
- AP admin does not know which programs will need which permissions
- User is in a better position to know these things, so user should be able to obtain authorizations by themselves

Placement Website

- Web frontend for the PlacementD
- User logs in via their campus Identity Provider (Single Sign-On)
- Identity Provider gives the website a name for the user



CHTC HTCondor Software Suite PATh

Table-Based Access Control

The PlacementD's table is keyed by the username given by the identity provider.

User name from Identity Provider	AP User	Authorizations	Expiration Date
mselmeci@wisc.edu	matyas	READ, WRITE, INSTRUCTOR	2038-01-18

Controlling who is allowed to get what capability is sufficient for controlling access.

Table-Based Access Control

To handle our class of 30, appending the class list to the table with the appropriate authorizations and expirations is enough.

No need to create accounts (we can rely on the campus ID provider), no need to deprovision afterward (the PlacementD will not give them tokens lasting past their expiration date)

User name from Identity Provider	AP User	Authorizations	Expiration Date
mselmeci@wisc.edu	matyas	READ, WRITE, INSTRUCTOR	2038-01-18
steve@wisc.edu	student1	READ, WRITE	2025-12-31
alice@wisc.edu	student2	READ, WRITE	2025-12-31
...	...	...	...

Table-Based Access Control

Moving away from knob-based access control to table-based access control opens the door for custom, more granular authorization.

User name from Identity Provider	AP User	Authorizations	Expiration Date
mselmeci@wisc.edu	matyas	READ, WRITE, INSTRUCTOR	2038-01-18
steve@wisc.edu	student1	READ, WRITE	2025-12-31
alice@wisc.edu	student2	READ, WRITE	2025-12-31
...	...	...	...

Summary

- A self-service method of obtaining access to AP services eases the burden of admins and students for the classroom / instructional use case
- Safely allowing that requires a more structured approach to credential management
- Capabilities (as implemented with Placement Tokens) lets us take access control out of config files and put it into tables
- The PlacementD lets us control and track creation of capabilities, and its web frontend provides a user-friendly method of obtaining capabilities

CHTC HTCondor Software Suite PATh

Acknowledgements and thanks

- Todd Tannenbaum
- Jaime Frey
- Miron Livny

This work is supported by NSF under Cooperative Agreement OAC-2030508 as part of the PATh Project. 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 NSF.

CHTC HTCondor Software Suite PATh