

IDTOKEN Authentication in the HTCondor Software Suite (HTCSS)

**European HTCondor Workshop
Sept 2021**

**Todd Tannenbaum
Center for High Throughput Computing
Department of Computer Sciences
University of Wisconsin-Madison**

Outline

- › Introduction and Motivation
 - Need for authentication in HTCSS
 - Need for yet another authentication method
- › Basic Concepts and Usage
 - For Admins: Using IDTOKENS to secure your Pool
 - For Users: Using IDTOKENS to utilize a remote Access Point
- › Advanced Topics
 - Invalidation
 - Multiple signing keys, multiple tokens
 - Token Requests
 - How does HTCSS securely present IDTOKENS

Introduction and Motivation

Need for Authentication in HTCSS

1. HTCondor services (aka daemons) authenticating to remote HTCondor services

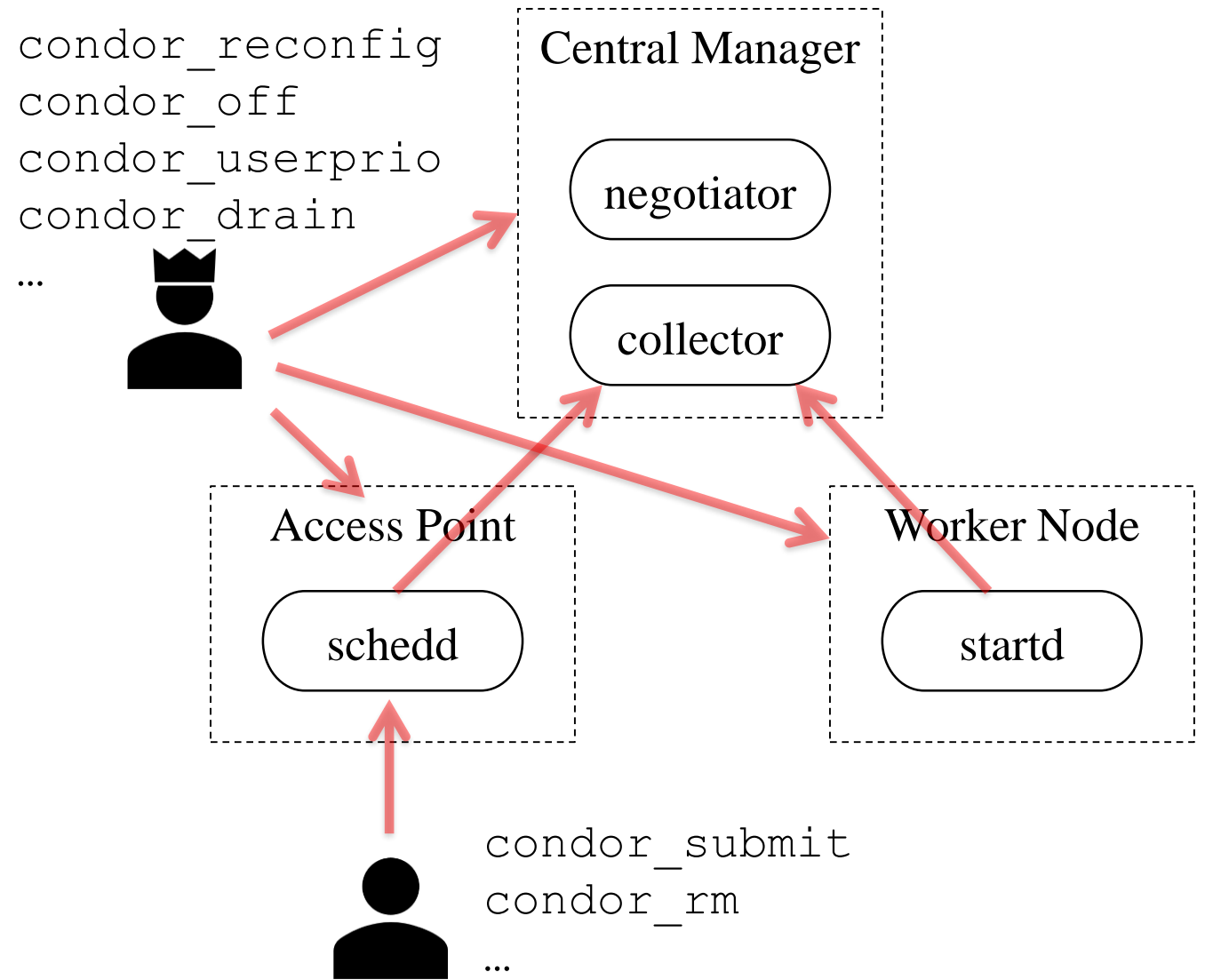
- *Only allow trusted nodes into the pool*

2. Users authenticating to an Access Point (schedd)

- *Need to know who owns which jobs*

3. Admins authenticating to an HTCondor service

- *Only allow trusted users to make administrative changes*



After Authentication comes Authorization

- › Authentication method results an identity
- › Identities are granted *authorizations* via HTCSS configuration, e.g.

```
# Processes that authenticate as user condor
# are considered services in my HTCondor Pool
ALLOW_DAEMON = condor@mysite
# Users alice and bob can submit jobs
ALLOW_WRITE = alice@mysite, bob@mysite
# root or condor can do administration
ALLOW_ADMINISTRATOR = root@mysite, condor@mysite
```

Why yet another authentication method?

- › HTCSS can perform authentication via many different methods, but they all had shortcomings or complications...
 - **FS** (filesystem) : Cannot work over the network, since the server challenges the client to create a file with proper ownership in /tmp
 - **POOL** (pool password): Only for daemons, not for tools / users.
 - **SSL, GSI, SCITOKENS, KERBEROS, MUNGE**: Requires significant setup work and/or installations from third-party for tools/services
- › Wanted a solution that is *self-contained*, works over the *network*, and works for *daemons or users using tools*
 - Thereby suitable for a "secure by default" installation

BASIC CONCEPTS AND USAGE

The IDTOKEN

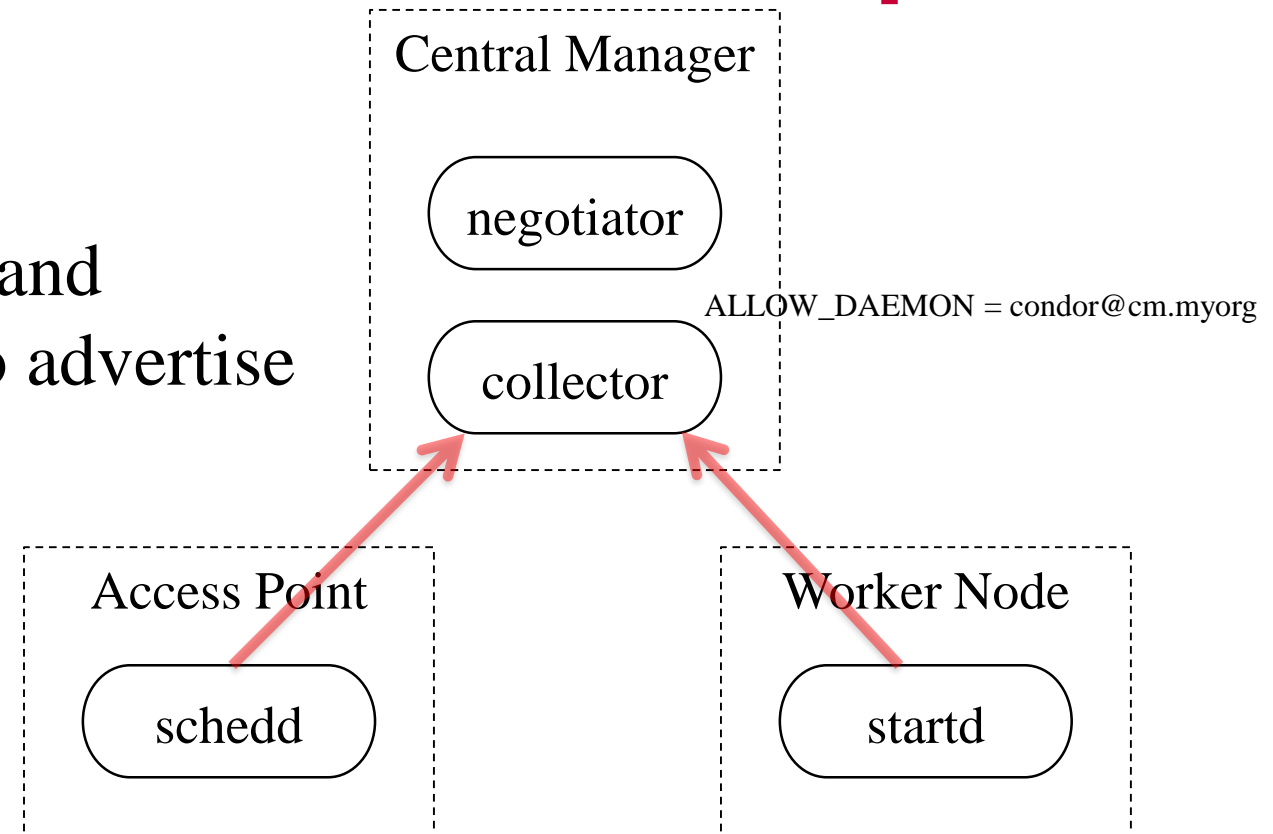
› An IDTOKEN contains:

- An **Identity**. Also
 - *Issuer, Unique ID, Issued date, possibly an Expiration date.*
 - *Authorization limits.* If present, these reduce the authorizations configured at the server – it does NOT add authorizations.
 - All signed with a **Digital Signature** (using a secret **signing key** stored on the server) to prove authenticity
 - Serialized out as an alphanumeric string and stored in a file.
- ## › An IDTOKEN is **always presented by a client to a server**.
- The server must have access to the same secret key that was used to sign the client's token.

```
$ condor_token_list
Header: {"alg":"HS256","kid":"POOL"}
Payload: {
issued date → "iat": 1588474719,
issuer → "iss": "pool.example.com",
unique id → "jti": "c760c2af193a1fd4e40bc9c53c96ee7c",
identity → "sub": "alice@pool.example.com"
}
```

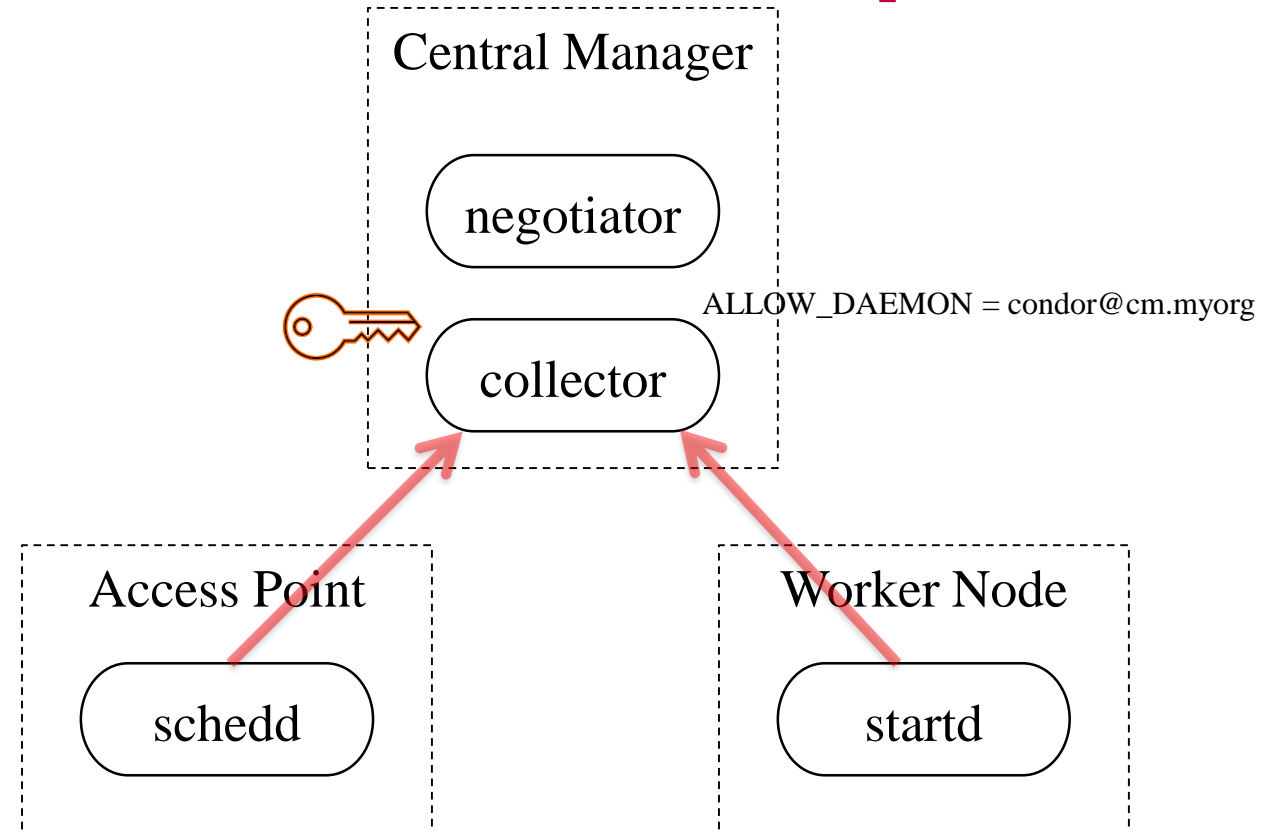

Example 1: IDTOKENS to secure a pool

The key to a secure pool is to only allow trusted (authenticated and authorized) startds and schedds to advertise into the collector.



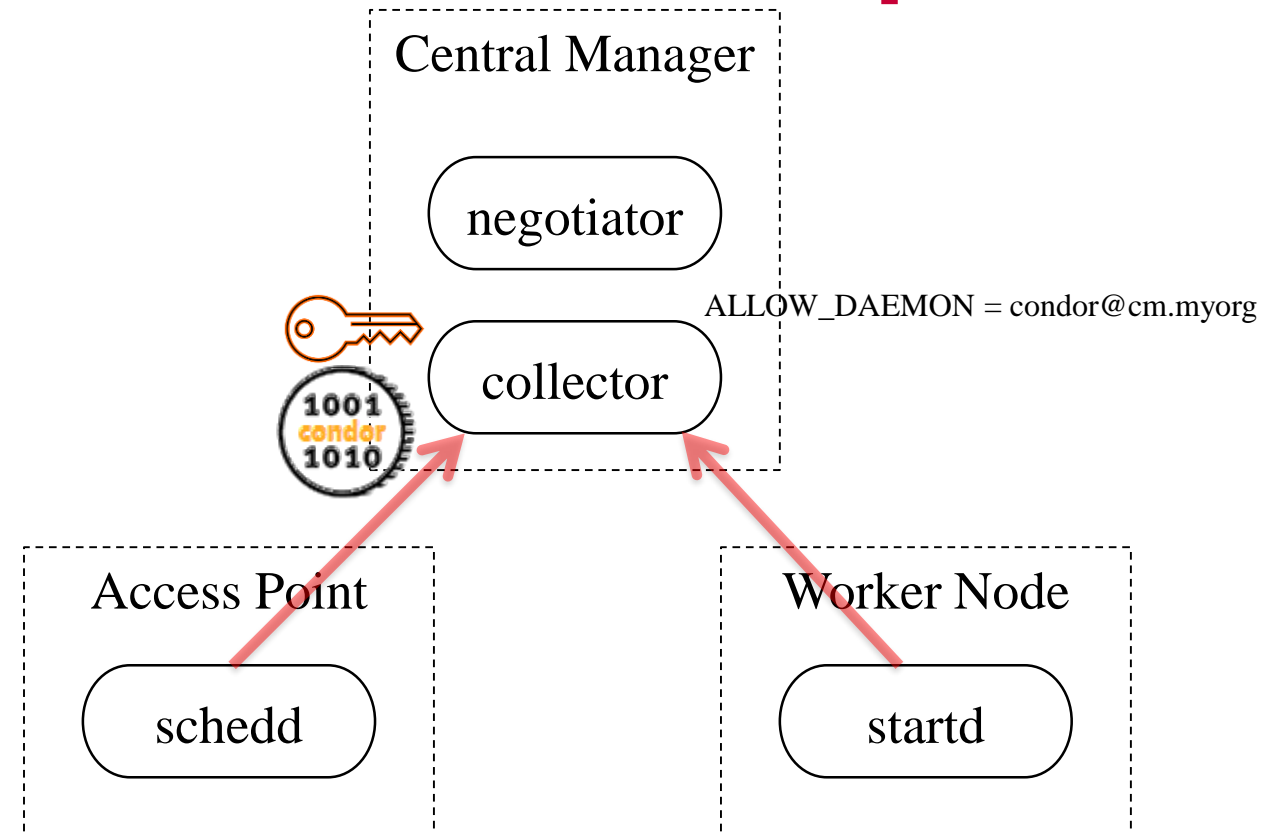
Example 1: IDTOKENS to secure a pool

- › 1. Create a signing key file on the central manager
 - A random key will be created by default at collector startup, or explicitly create with tool:
`condor_store_cred add -c`
 - Signing key is stored by default in file
`/etc/condor/passwords.d/POOL`



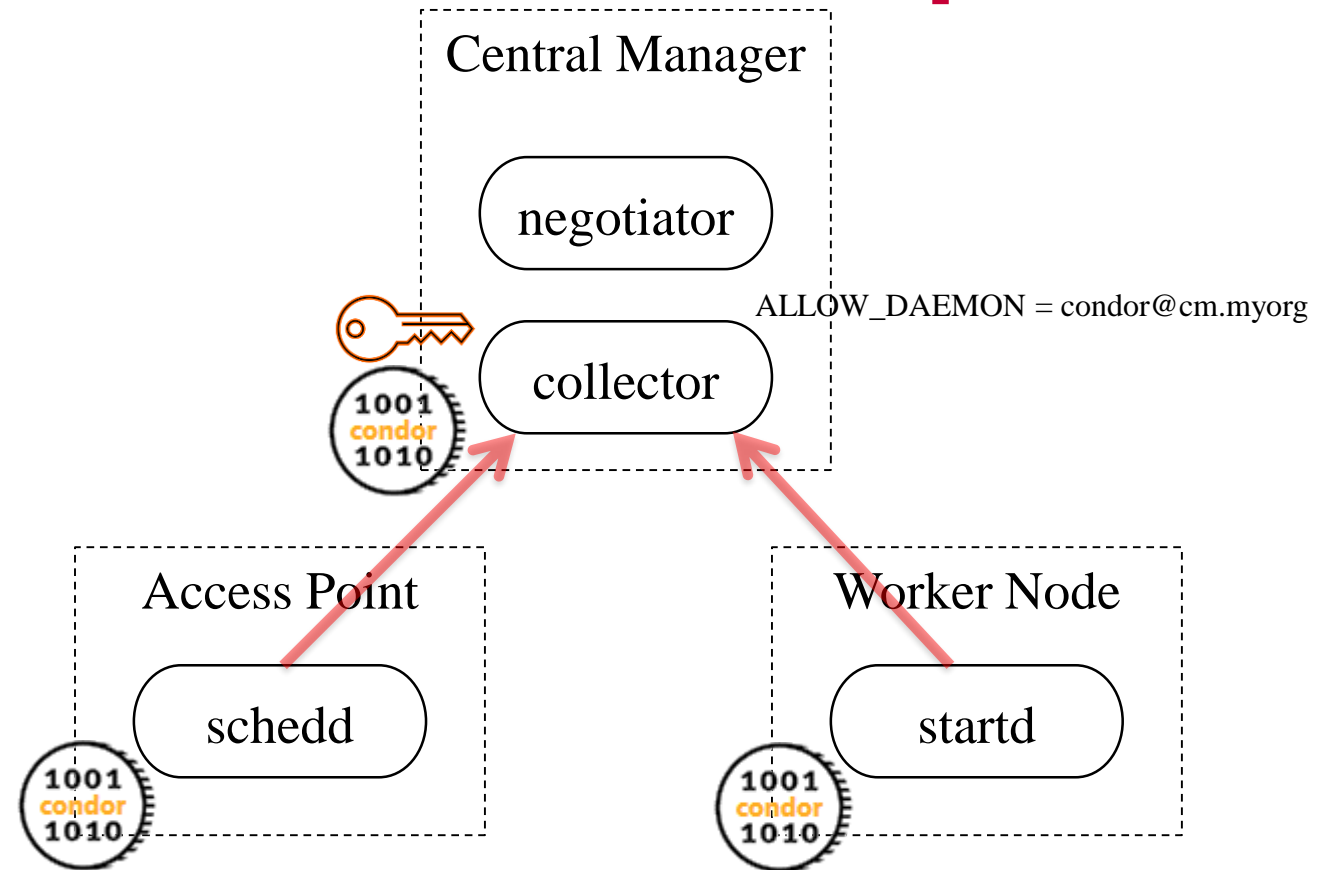
Example 1: IDTOKENS to secure a pool

- › Create a signing key file on the central manager
- › Create an IDTOKEN file with identity "condor" on central manager
 - Use tool
`condor_token_create -identity condor@cm.myorg`



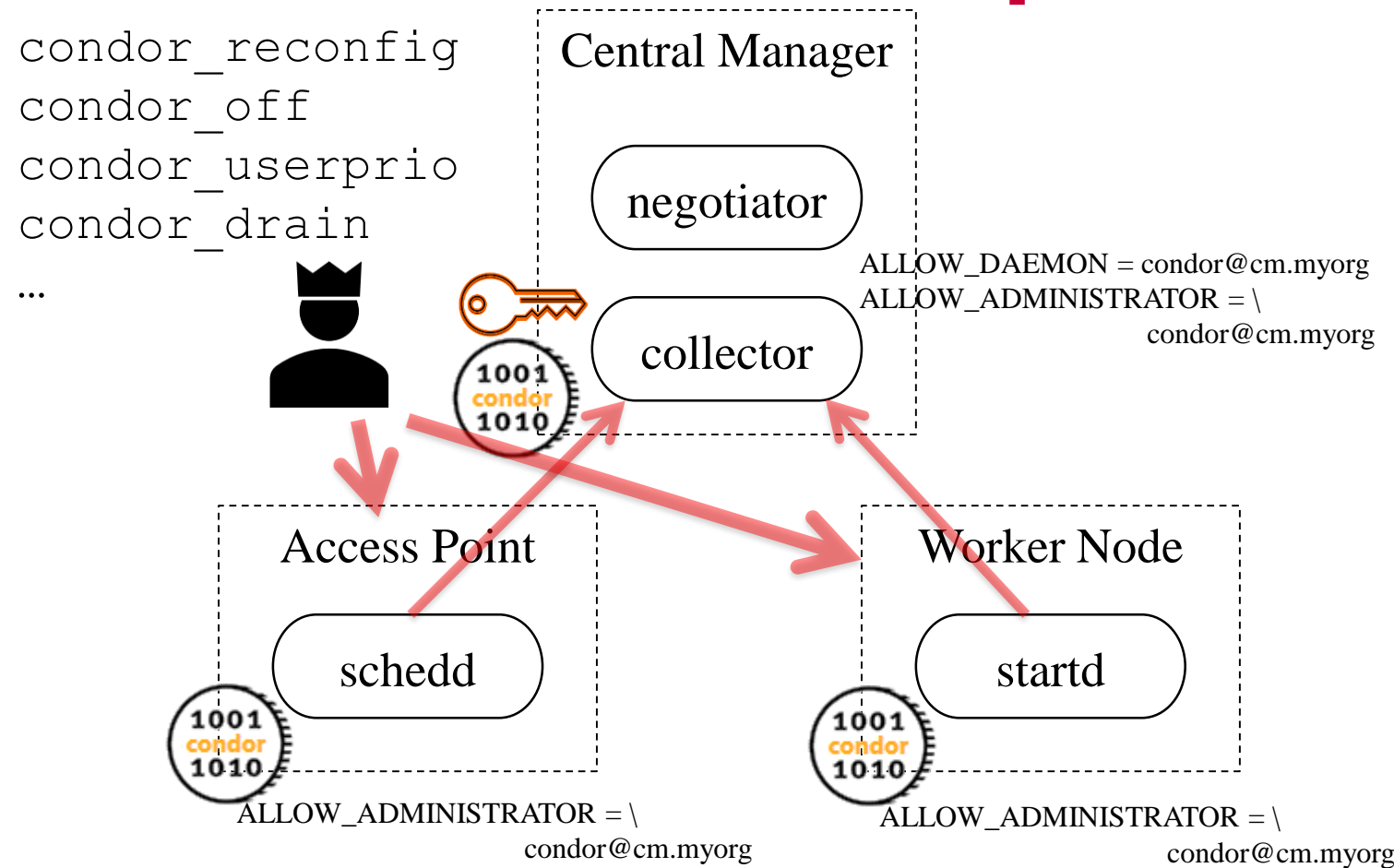
Example 1: IDTOKENS to secure a pool

- › Create a signing key file on the central manager
- › Create an IDTOKEN file with identity "condor" on central manager
 - Use tool
`condor_token_create -identity condor@cm.myorg`
- › Copy this IDTOKEN file to each trusted server you want to join your pool
 - Place file into directory
`/etc/condor/tokens.d`



Example 1: IDTOKENS to secure a pool

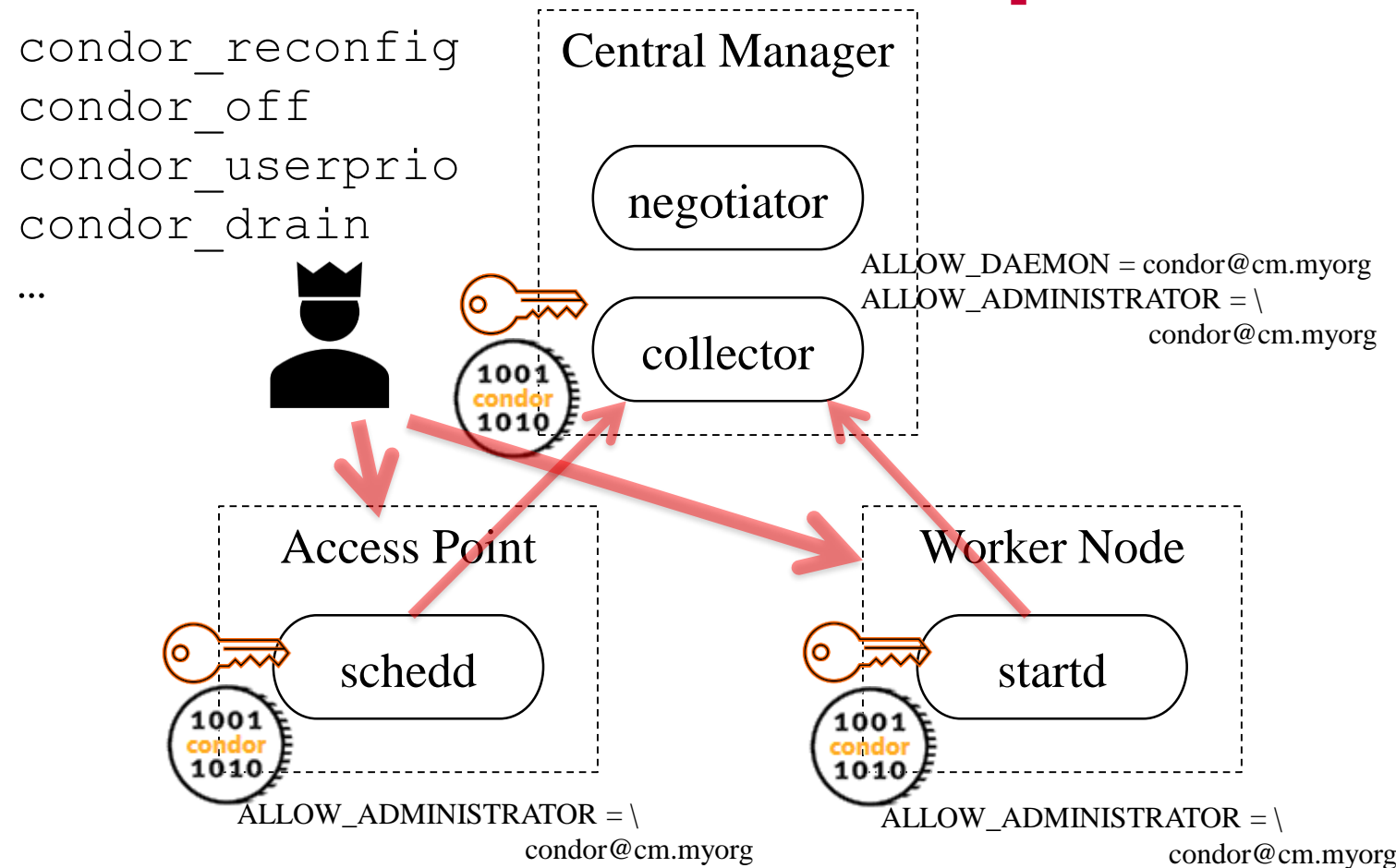
- > What about securing administrator commands?
- > Say we want root on the central manager to be able to issue admin commands to remote access points and worker nodes.... ?
 - Hint: It wont work – the access point and worker nodes cannot validate the token (no signing key)



Example 1: IDTOKENS to secure a pool

- › What about securing administrator commands?
- › Say we want root on the central manager to be able to issue admin commands to remote access points and worker nodes.... ?

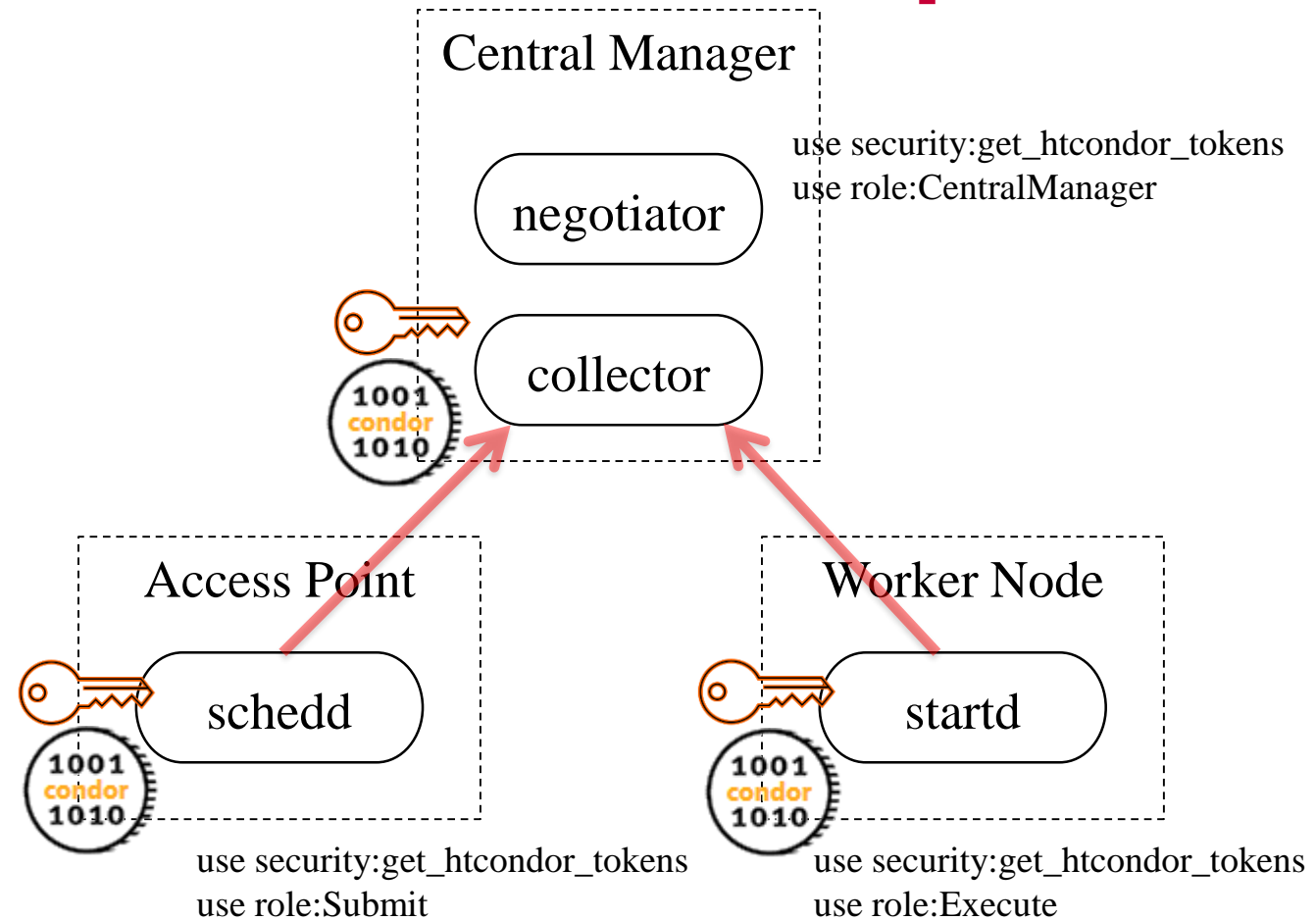
- Hint: It wont work – the access point and worker nodes cannot validate the token (no signing key)
- A Solution: Place the signing key on all trusted nodes in your pool! Voila!



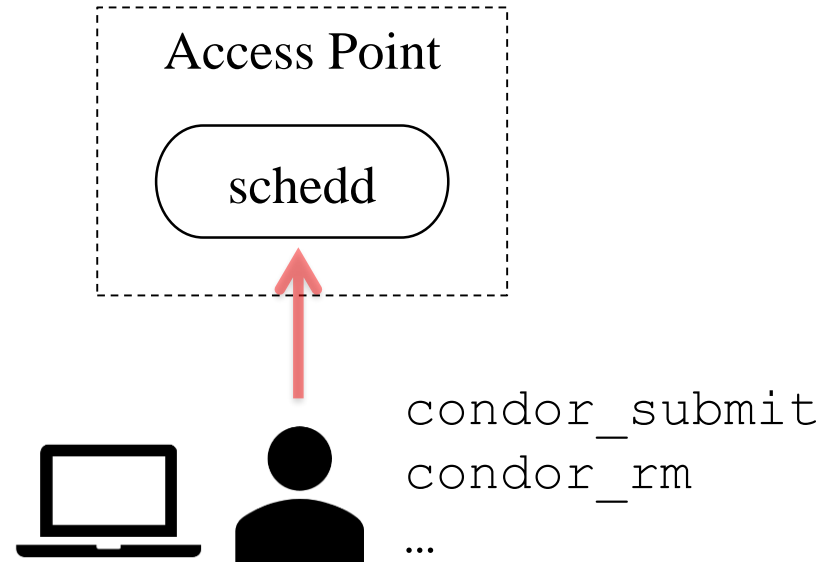
Example 1: IDTOKENS to secure a pool

- › This diagram depicts how a pool is configured for security after installing via the *get_htcondor* tool!
- › You can learn a lot by inspecting output from

```
curl -fsSL get.htcondor.org | bash -s -- --execute cm.org  
and  
condor_config_val use security:get_htcondor_tokens
```



Example 2: Using a remote Access Point

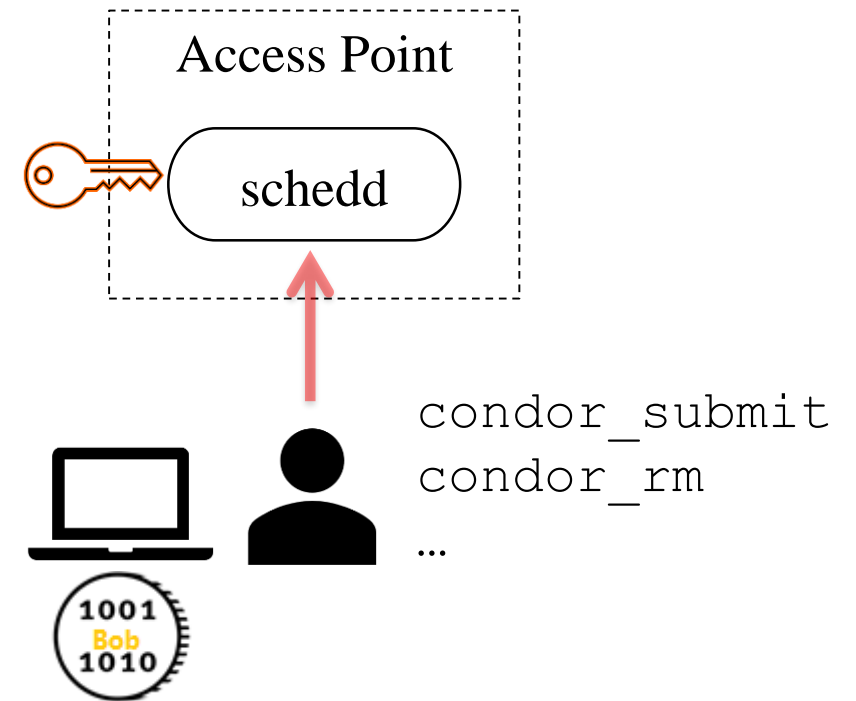


Bob is a normal user (no root access)
Bob can ssh into an access point and submit.
But he wants to submit from his laptop...

Example 2: Using a remote Access Point

- › Step 1: Bob does a ssh login to his access point
 - He cannot use `condor_token_create`; only root can read signing keys in `/etc/condor/passwords.d` ... so instead...
- › Step 2: Bob creates an IDTOKEN with identity "Bob" via `condor_token_fetch` tool
 - `condor_token_fetch` authenticates to the schedd (via FS, filesystem auth), asks the schedd to create an IDTOKEN on behalf of the user's identity. Resulting IDTOKEN identity is identical to authenticated identity.
- › Step 3: Bob copies output from `condor_token_fetch` to his laptop, storing it in a file in directory `~/.condor/tokens.d`. Bob can now access the remote schedd as "Bob".

Voila!



Summary of Commands

- `condor_store_cred add -c` : Command to store a signing key.
- `condor_token_create` : Allows anyone who can read a signing key (usually just root) to create and sign an IDTOKEN with any given identity. Example with attenuation (auth limits, expiration):

```
$ sudo condor_token_create \  
-identity brian.bockelman@collector.example.com \  
-lifetime 3600 \  
-authz READ -authz WRITE
```

- `condor_token_fetch` : Authenticate with a daemon and create an IDTOKEN on behalf of the user's identity.
- `condor_token_list` : display properties of available IDTOKENS by scanning IDTOKEN directories

Summary of Default Pathnames

- `/etc/condor/passwords.d/` : Directory containing signing keys. Default signing key is in a file named "POOL" in this directory. Only readable/writeable by root.
- `/etc/condor/tokens.d` : Directory containing IDTOKEN files used by process *with* root access (HTCSS daemons, administrators with sudo). Only readable/writeable by root.
- `~/.condor/tokens.d` : Directory in a user's home directory containing IDTOKEN files used by a process *without* root access (unprivileged users). Only readable/writeable by that user.

All default path locations can be changed via configuration

ADVANCED TOPICS (for further discussion?)

Suggested Advanced Topics for Discussion

- › Revocation? How?
 - You could remove authorization of the identity, e.g. in the config put
DENY_WRITE = todd@cm.my.org
 - A classad constraint expression that can conditionally refuse tokens based on any attribute, such as identity, date range when issued, serial number.
 - See the IDTOKENS section of the HTCSS Manual: <https://tinyurl.com/ygqsc94j>
 - You can remove the signing key file, which effectively invalidates all tokens signed with that key
- › Multiple tokens
 - If the tokens.d directory has multiple files, the first token file that was issued from the server's domain and signed by a key file still present on the server is selected.
- › How does HTCSS *securely* present IDTOKENS to a remote server?
 - The signing key is used as a shared secret to initiate the AKEP2 protocol.
- › IDTOKENS are follow the JSON Web Token (JWT) standard. Why do I care?
- › Token Requests via condor_token_request ...

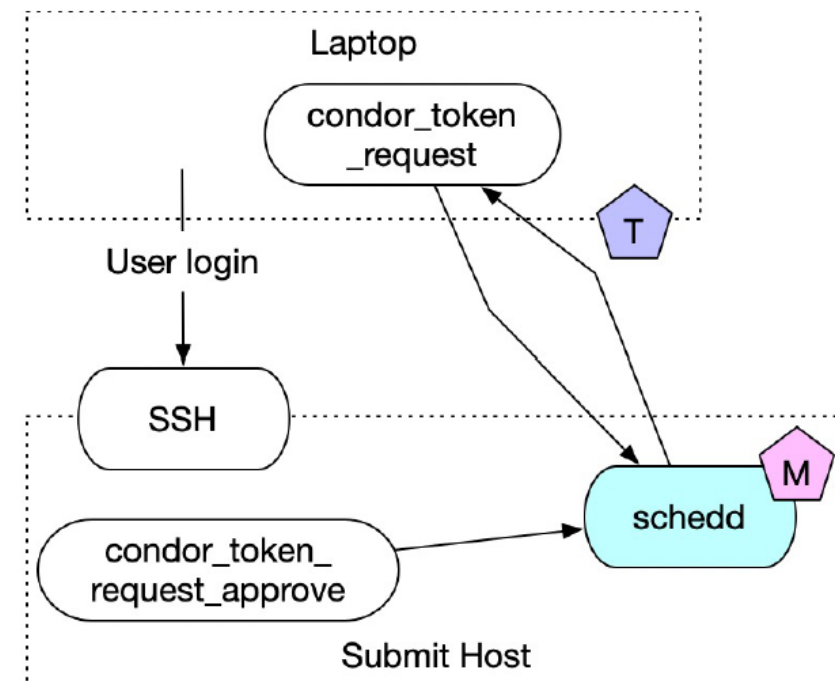


Requesting an IDTOKEN

Want to get an IDTOKEN on a machine without authenticating?

- **condor_token_request** allows an anonymous user to request a token for an arbitrary identity X.
 - The token request can be approved either by an admin or a user authenticated as X.
 - Anyone can ask. Few can approve!
- **Use case:** I have an SSH login on a schedd and want to start submitting jobs from my laptop.
 - **Solution:** Request a token from my laptop; login to the submit host and approve the request.
- **DO NOT COPY/PASTE TOKENS.** Instead, use **condor_token_request!**

The startd, master, and schedd will automatically request tokens from the collector if authentication fails.



Gotcha: to work, the client needs to trust the server –

typically, this implies SSL authentication (which is tricky to setup). Look forward to new tricks in 9.1.x...



Thank You!



Follow us on Twitter!
<https://twitter.com/HTCondor>



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

