Feeding TOFU to a Condor: Trust and Authorization Changes in HTCSS

MORGRIDGE
INSTITUTE FOR RESEARCH
RESEARCH COMPUTING

FEARLESS SCIENCE

Well, what did you expect?

Slide from HTCondor Week 2020

"Forget what you know about HTCondor security. We changed it"

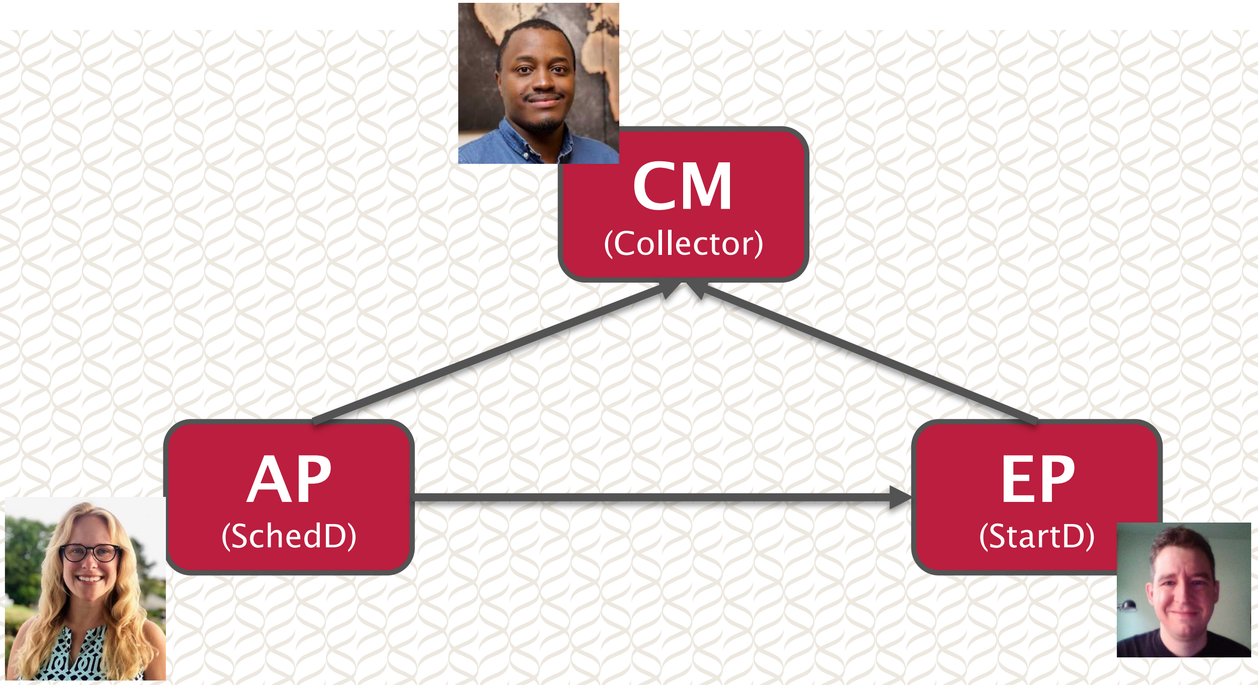
MORGRIDGE
INSTITUTE FOR RESEARCH
CORE COMPUTATION

We Kept Working in 2021!

FEARLESS SCIENCE

For more introductory material, see last year's HTCondor Week talk on security.

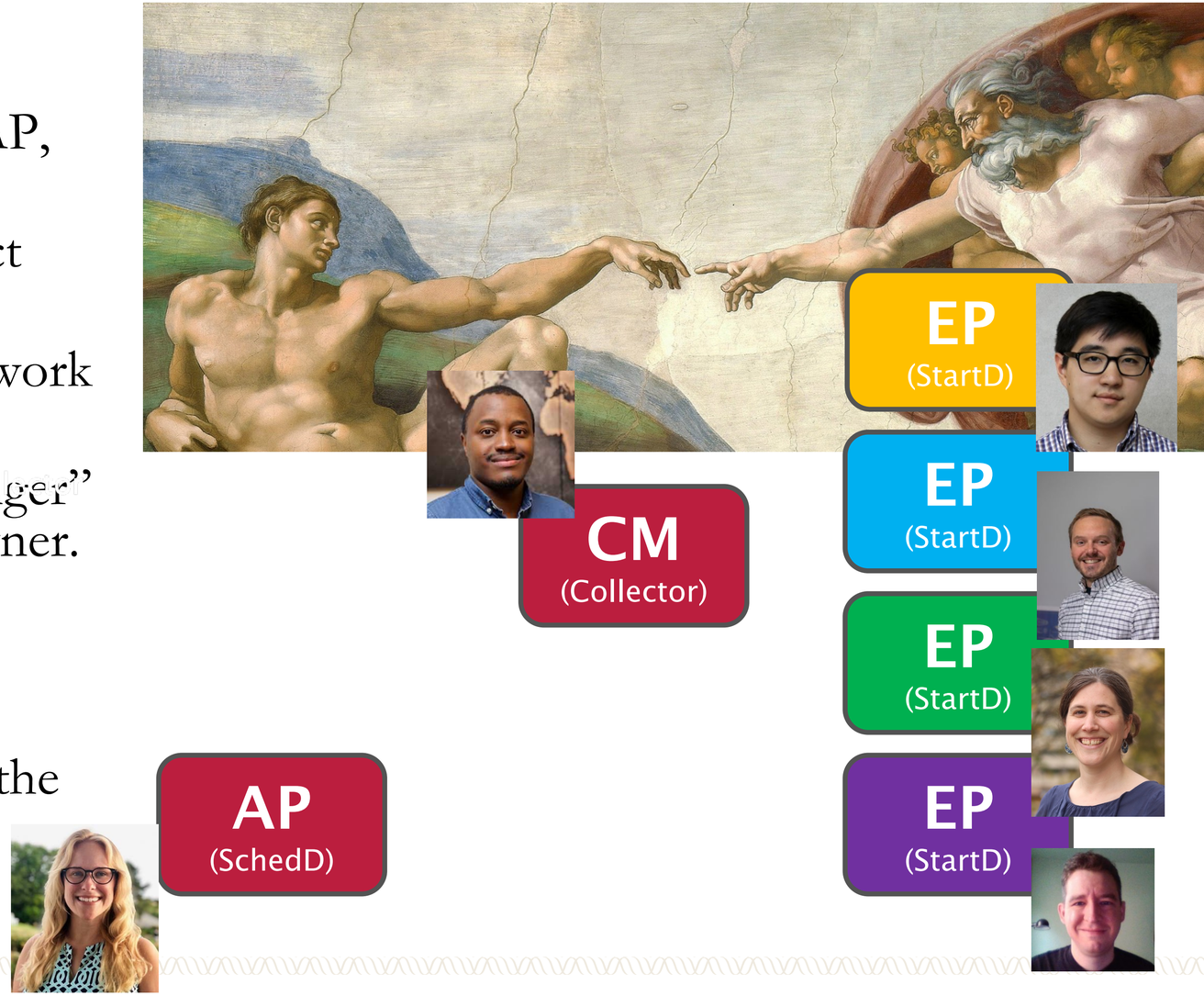
Trust Models for HTCondor



In the beginning …

Trust model has separate identities for the user, AP, EP, and CM.

- Each of these were unique entities with distinct concerns.
  - HTCondor provides a robust policy framework so each entity could
  - Strongly influenced by the "desktop scavenger" model – the EP owner was the desktop owner.
- Also required HTCondor to have a complex authentication and authorization model:
  - Each daemon had its only identity and the admin configures a set of policies on what the identity is permitted to do.

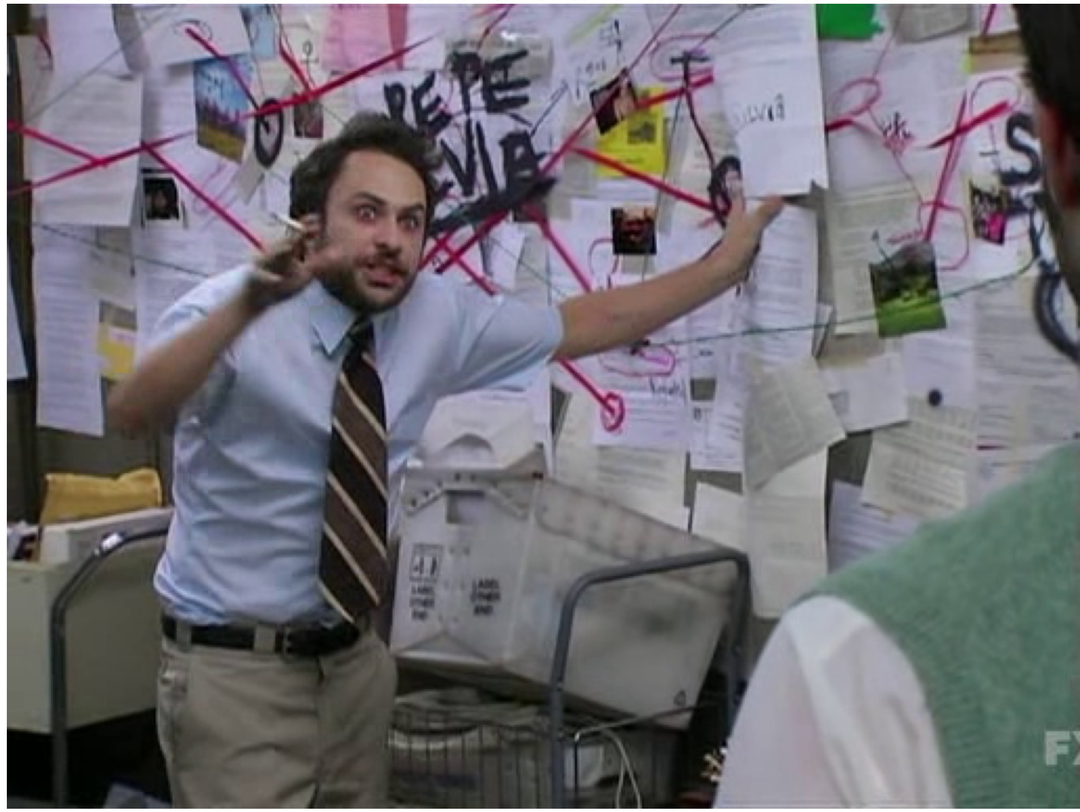


Authentication and Trust in HTCondor

Setting up trust in classic HTCondor in 7 easy steps:

- Decide on a trusted third party. Generate a root certificate representing that third party.
- Distribute the publickey to all your hosts (somehow) and configure AUTH_SSL_SERVER_CAFILE and AUTH_SSL_SERVER_CAFILE.
- Have each of the daemon owners generate a certificate request with their legal (passport) name and send the request to the trusted third party.
  - Have that person verify the identity against the user's passport then sign the request.
- Install the new cert and key onto the server.
- Configure SSL in the daemons by setting AUTH_SSL_SERVER_CERTFILE, AUTH_SSL_CLIENT_CERTFILE, AUTH_SSL_SERVER_KEYFILE, and AUTH_SSL_CLIENT_KEYFILE.
- Map the certificate subjects to a HTCondor identity in the mapfile.
- Setup the access control lists (ALLOW_READ, ALLOW_WRITE, etc) to give permissions to the correct individual.

Easy, right?



Depicted here: the lone sysadmin after spending a week in the HTCondor manual's chapter on security configuration.

A crack in the trust framework (~2009)

A first crack in the trust model occurred with the introduction of the lovably-named knob 'SEC_ENABLE_MATCH_PASSWORD_AUTHENTICATION'.

- When set, the StartD would generate a random string of characters and send it to the collector.
  - Any entity who could present this random string – **a capability** – was permitted to claim the StartD and run jobs.

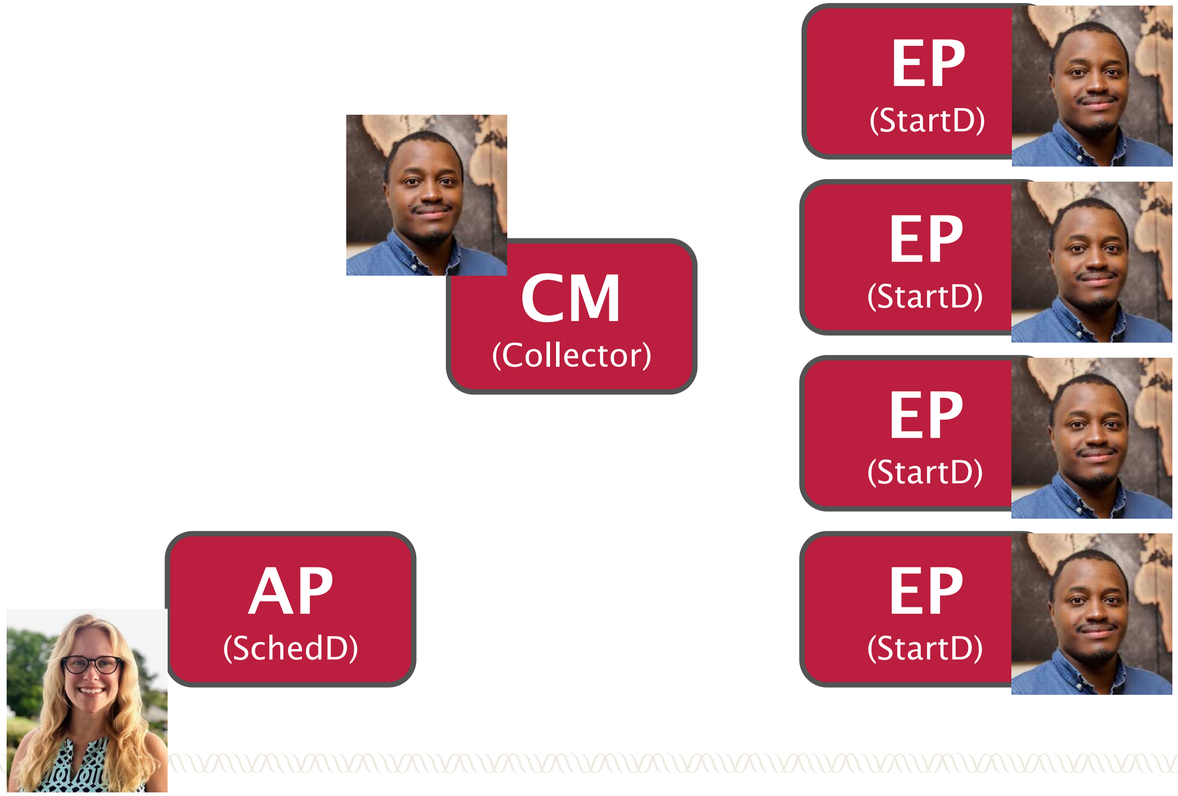
What's the change?

- The StartD does not authenticate the remote entity with jobs. It trusts (and authorizes) anyone who is trusted by the collector.
- Similarly, the SchedD does not authenticate the StartD. There is transitive trust – the SchedD trusts the StartD through the mutual collector.

What makes this work? A change in ownership…

You _can_ still use HTCondor as a desktop scavenger. However, it's far more common to see HTCondor as a batch system.

- One individual owns / operates a pool of resources. These resources are far more homogeneous – especially policy-wise.
- **If you trust the resource pool (CM), then you trust all the resources (EP).**
  - In 2019, we introduced the **TRUST_DOMAIN**: a set of resources under the control of a single administrator.



2019: IDTOKENS – a built-in authentication infrastructure

The IDTOKENS authentication method introduced in 2019 had a few new concepts:

- The trust domain provided a **namespace for identity**.
  - Signing keys are associated with a trust domain and are all-powerful.
  - Simply drop a key in place and you can create tokens.
  - No global namespaces!
- **HTCondor itself could create credentials**.
  - No need to rely on external third-party services (Kerberos, SSL, GSI).
  - **Bootstrapping**: An anonymous user can request an identity and an administrator can approve it out-of-band.
- Authorization can be limited in the credential itself.

condor — bbockelm@hcc-briantest7:~ — ssh hcc-briantest7.unl.edu — 80×8

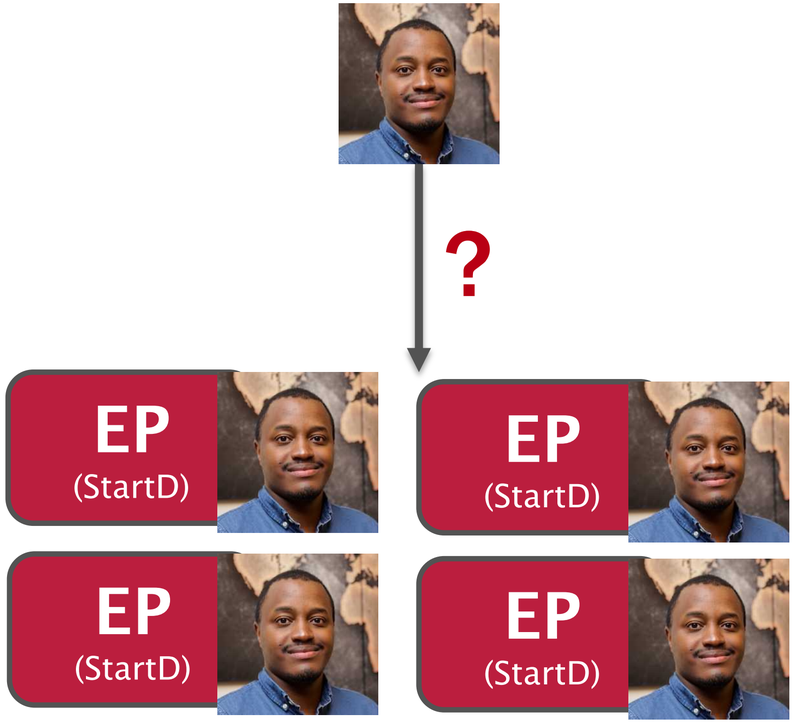
```
[bbockelm@hcc-briantest7 ~]$ condor_token_request -pool cm-1.ospool.osg-htc.org
-identity bbockelm@flock.opensciencegrid.org
Token request enqueued.  Ask an administrator to please approve request 6361875.
```

IDTOKENS - Limitations

IDTOKENS are based on a symmetric signature. **To verify a token is valid, you need the signing key**.

1. What happens when the admin wants to send a command to a remote EP?
   - The EP needs the signing key to verify the admin's IDTOKEN – but now you're distributing the 'keys to the kingdom' to every worker node!
2. The token only validates the client; the server does not have a distinct identity.
3. There is no concept of a secure, anonymous session. To use IDTOKENS, you need an IDTOKEN.
   - Bad for remote reads of a condor daemon.
   - **To bootstrap/request an IDTOKEN, you can't use the IDTOKENS authentication!**

What would happen if the only person who could validate your driver's license was the state DMV and not the grocery store clerk?



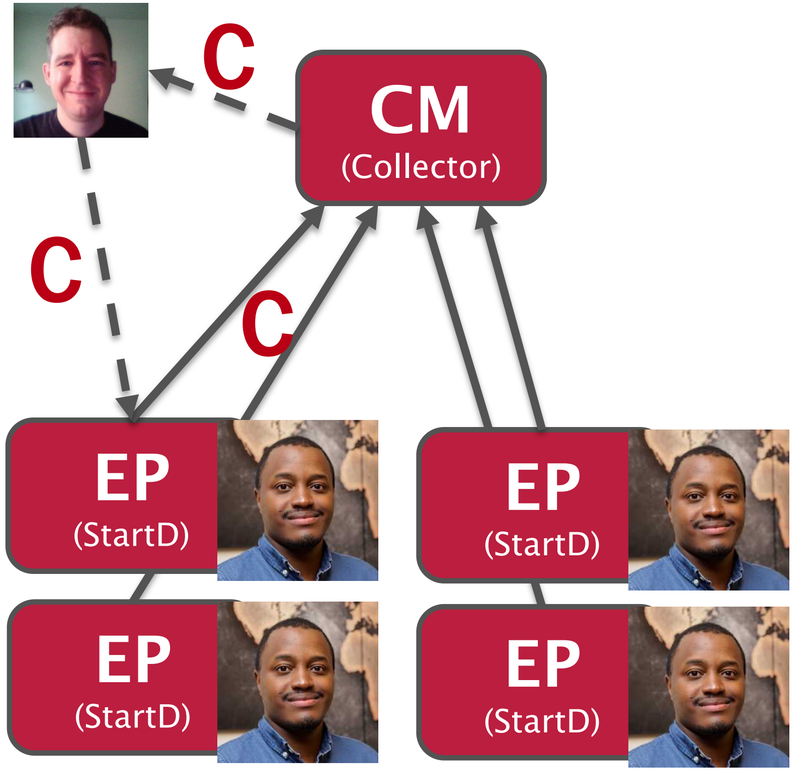
‘Fixing’ IDTOKENS, part I

One ‘fix’ is a change in the trust model. HTCondor 9.9.0 will introduce **remote administration capabilities**.

- The EP will generate a capability and send it to the CM.
- The CM will <u>only</u> send this capability to individuals it considers as being an ‘administrator’.
- The individual sends the capability to the EP to be authorized to perform admin actions (shutdown, fetch logs).

Important trust model changes:

- The EP trusts the CM completely to authorize individuals on its behalf.
- The EP never learns the identity of the administrator. No authentication, no identity mapping.



We did not ‘fix’ IDTOKENS but simply made HTCSS code reflect the most common trust model – the EP ‘belongs’ to the pool and trusts it as an admin.

Fixing IDTOKENS, part II

To 'fix' bootstrapping, we've always pointed people at SSL authentication:

- For SSL, anonymous clients are extremely common – most common way to use HTTPS!
- **Just what we need**: SSL provides an encrypted, integrity-checked channel for anonymous clients to request tokens.

What's the problem? **See Slide 4: SSL is a mess to setup!**

How do we fix things?

Change the trust model!

TOFU mode

TOFU = Trust on First Use

Most uses of SSL rely on global purveyors of trust – certificate authorities – whose business is being a trustworthy source of global identity.

- Expensive, often complex, and strongly oriented toward identifying hosts on the public internet. **Is this the only to establish trust?**

Of course not! We are all quite familiar (and likely comfortable) with SSH's 'trust on first use' model.

- The first time you use a service, you are asked if you'd like to trust it. Trusting the identity is an 'exercise left to the user'. Some users are diligent; some are not.
- This identity is saved locally; subsequent requests are compared against the 'first use' key.

Let's apply TOFU to SSL!

Server side

- On startup, if one is not present, the CM will generate a self-signed CA certificate and key.
  - This is now a CA for the trust domain.
- If a host cert is not present, generate one from the CA.
- That's it! Behaves like a normal SSL authentication.

Client side

- The client will proceed with a SSL handshake.
- On a CA verification failure, the TOFU logic will engage:
  - The user will be asked if they'd like to trust the unknown CA and be shown a fingerprint.
  - If there's no terminal (i.e., the client is a daemon), question is answered based on config file (default: don't trust).
- If yes, the authentication continues, and the key is written into a known_hosts file.

TOFU in Action

bbockelm — bbockelm@hcc-briantest7:~/projects/condor-build — ssh hcc-briantest7.unl.edu — 108×7

```
[bbockelm@hcc-briantest7 condor-build]$ condor_q -all
The remote host hcc-briantest7.unl.edu presented an untrusted CA certificate with the following fingerprint:
SHA-256: 781b:1d:1:ca:b:f7:ab:b6:e4:a3:31:80:ae:28:9d:b0:a9:ee:1b:c1:63:8b:62:29:83:1f:e7:88:29:75:6:
Subject: /O=condor/CN=hcc-briantest7.unl.edu
Would you like to trust this server for current and future communications?
Please type 'yes' or 'no':
```

bbockelm — bbockelm@hcc-briantest7:~/projects/condor-build — ssh hcc-briantest7.unl.edu — 108×8

```
[bbockelm@hcc-briantest7 condor-build]$ cat ~/.condor/known_hosts
hcc-briantest7.unl.edu SSL MIIBvjCCAWSgAwIBAgIJAJRheVnN5ZDyMAoGCCqGSM49BAMCMDIxDzANBgNVBAoMBmNvbmRvcjEfMB0GA
1UEAwwWaGNjLWJyaWFudGVzdDcudW5sLmVkdTAeFw0yMTA1MTcxOTQ3MjRaFw0zMTA1MTUxOTQ3MjNaMDIxDzANBgNVBAoMBmNvbmRvcjEfM
B0GA1UEAwwWaGNjLWJyaWFudGVzdDcudW5sLmVkdTBZMBMGByqGSM49AgEGCCqGSM49AwEHA0IABPN7qu+qdsfP6WR++UucrZYvMhssre8jv
gWsnPBdzCYU/EqHYp+wri/aAKyDrLM5R1lWX44jSykgIpTOCLJUS/ajYzBhMB0GA1UdDgQWBBRBPe8Ga9Q7X3F198fWBSg6VT1DZDAfBgNVH
SMEGDAWgBRBPe8Ga9Q7X3F198fWBSg6VT1DZDAPBgNVHRMBAf8EBTADAQH/MA4GA1UdDwEB/wQEAwICBDAKBggqhkjOPQQDAgNIADBFAiARf
W+suELxSzSdi9u20hFs/aSXpd+gwJ6Ne8jjG+y/2AIhAO6f3ff9nnYRmesFbvt1lv+LosOMbeiUdVoaKFOGIyuJ
[bbockelm@hcc-briantest7 condor-build]$
```

TOFU in Action – known_hosts file

The past certificates observed for a user are kept in ~/.condor/known_hosts; the format is 'SSH inspired'

- One line per remote host.
- Line records the triplet of "name, method, pubkey".
  - Method is important: if we authenticate via IDTOKENS originally, do not count this daemon as 'first use' for SSL!
  - Similarly, do NOT trust a host that previously authenticated fine with SSL.
- Lines prefixed with "!" are <u>not</u> trusted.
  - Want to change your mind later? Just remove the "!"

Do NOT trust this key!

bbockelm — bbockelm@hcc-briantest7:~/projects/condor-build — ssh hcc-briantest7.unl.edu — 108×8

```
[bbockelm@hcc-briantest7 condor-build]$ cat ~/.condor/known_hosts
!hcc-briantest7.unl.edu SSL MIIBvjCCAWSgAwIBAgIJAJRheVnN5ZDyMAoGCCqGSM49BAMCMDIxDzANBgNVBAoMBmNvbmRvcjEfMB0G
A1UEAwwWaGNjLWJyaWFudGVzdDcudW5sLmVkdTAeFw0yMTA1MTcxOTQ3MjRaFw0zMTA1MTUxOTQ3MjNaMDIxDzANBgNVBAoMBmNvbmRvcjEf
MB0GA1UEAwwWaGNjLWJyaWFudGVzdDcudW5sLmVkdTBZMBMGByqGSM49AgEGCCqGSM49AwEHA0IABPN7qu+qdsfP6WR++UucrZYvMhssre8j
vgWsnPBdzCYU/EqHYp+wri/aAKyDrLM5R1lWX44jSykgIpTOCLJUS/ajYzBhMB0GA1UdDgQWBBRBPe8Ga9Q7X3F198fWBSg6VT1DZDAfBgNV
HSMEGDAWgBRBPe8Ga9Q7X3F198fWBSg6VT1DZDAPBgNVHRMBAf8EBTADAQH/MA4GA1UdDwEB/wQEAwICBDAKBggqhkjOPQQDAgNIADBFAiAR
fW+suELxSzSdi9u20hFs/aSXpd+gwJ6Ne8jjG+y/2AIhAO6f3ff9nnYRmesFbvt1lv+LosOMbeiUdVoaKFOGIyuJ
[bbockelm@hcc-briantest7 condor-build]$
```

Beyond TOFU

Why is TOFU powerful?

TOFU provides us with a new trust model, one well-understood (and likely accepted) by anyone who uses SSH.

- It provides us with a **simple way** to establish an anonymous, encrypted, integrity-checked mechanism to talk to a remote HTCondor daemon.
  - Hopefully a first step in removing less-secure CLAIMTOBE and ANONYMOUS authentication from the default configuration.
  - It simplifies bootstrapping of IDTOKENS auth – perhaps we'll see it in get_htcondor soon?

Reduces the complexity of using SSL!

- SSL is a prerequisite for using SCITOKENS authorization.

Hope to see this code land in 9.10!

New Technology is Great – But more important is the TRUST MODEL

Note changes in technology and trust model go hand-in-hand:

- The original model was each daemon established a full, direct trust relationship (authentication, identity mapping, authorization). Made sense when each daemon was bespoke.
- Over the past two years, we've evolved so the CM is the central source of trust: if you establish trust with the collector, you can use the pool.
- Simplifying the trust relationship – and deploying new technology to match it – has enabled us to tackle the complexity of securing HTCSS.

What's next?

- One side-effect of TOFU mode is it establishes a public key with a trust domain.
  - I want to leverage the fact we can now verify credentials without having access to signing keys…

Stay Tuned for 2023!

MORGRIDGE
INSTITUTE FOR RESEARCH
RESEARCH COMPUTING

morgridge.org

This project is supported by National Science Foundation under Cooperative Agreement OAC-2030508. 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.

FEARLESS SCIENCE