Using Hashicorp Vault with HTCondor for Oauth Credentials in Jobs

Dave Dykstra, dwd@fnal.gov

HTCondor Week
24 May 2022

Fermilab

Why switch to tokens at all?

- The primary reason to switch to tokens is that X.509 proxy certificates were never used outside of the HTC/grid community
  - They were invented by Globus, and Globus has abandoned support for the libraries. OSG and a few others took up support in the Grid Community Toolkit but OSG has stopped support
  - X.509 user proxy certificates depended on support at the SSL/TLS layer that is only rarely used
- Oauth2/OpenID Connect (OIDC) JSON Web Tokens (JWTs) are in widespread use, and are potentially more secure because they enable much more fine-grained control
  - There are a lot of existing tools that we can use with them, although we also often need some customization
    - Scitokens are JWTs with a community-standard profile of the claims in the JWTs
  - They're easier to implement because they are sent at a higher layer, i.e. http Authorization header
  - Fine grained control does make them more complicated to use, however
- This talk is focused on tokens obtained by end users to access storage, including sending them with jobs through HTCondor
- Note: X.509 host certificates are not going away, and they are an essential component to securely verifying JWTs over https

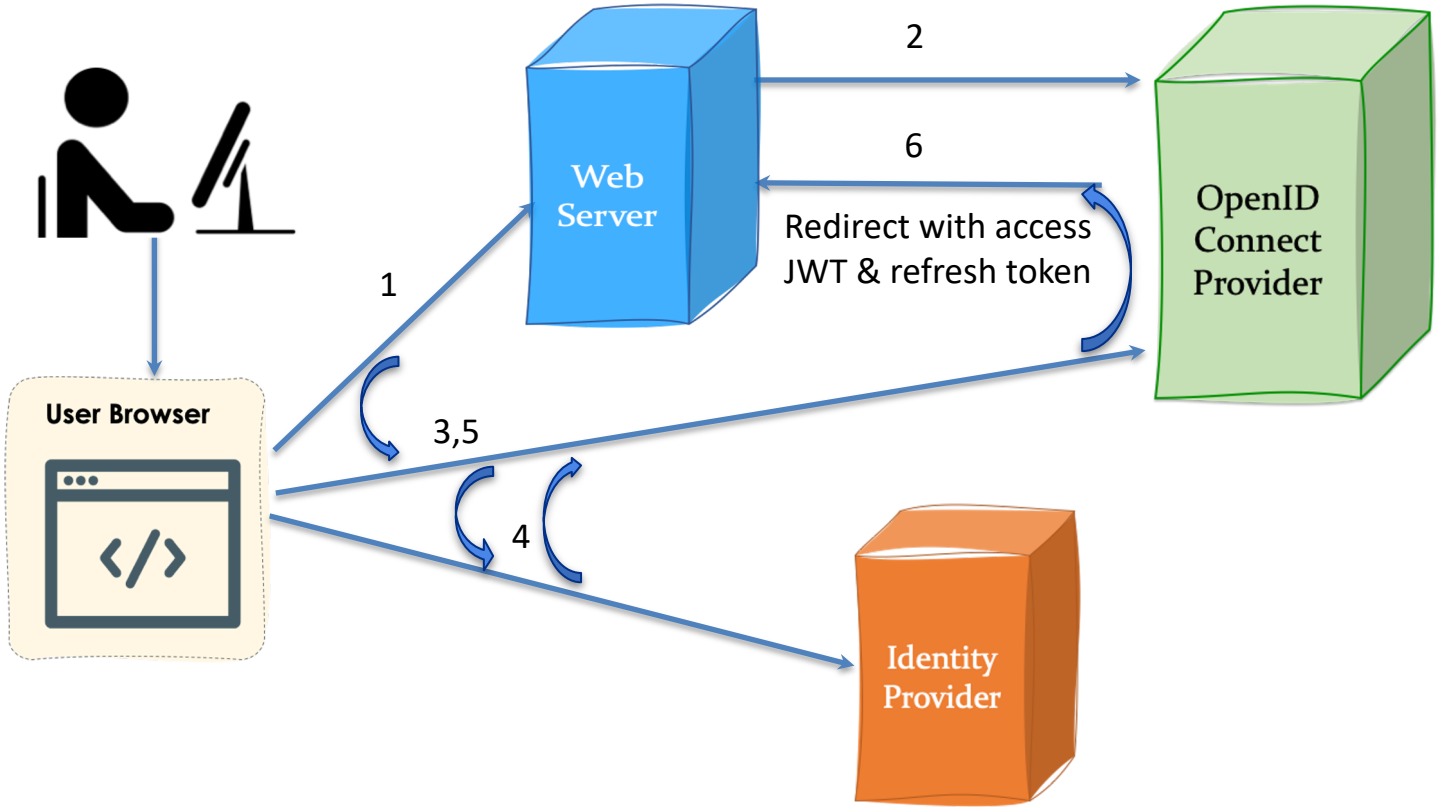
Prior HTCondor solutions

- HTCondor had a couple of solutions of its own, but they each have limitations
  - The “local token issuer” solution, where HTCondor issues it own tokens without Oauth2, doesn’t scale to many submission points, and only supports a fixed set of JWT scopes
  - The “Oauth2 credentials” solution, where HTCondor is an oauth2 client, requires web browser authentication before most job submissions, and doesn’t help with non-condor use cases
- We wanted to minimize the number of web browser interactions and be able to use the same credentials both inside and outside of HTCondor
  - We wanted the multiple end user case to be as easy to use as possible

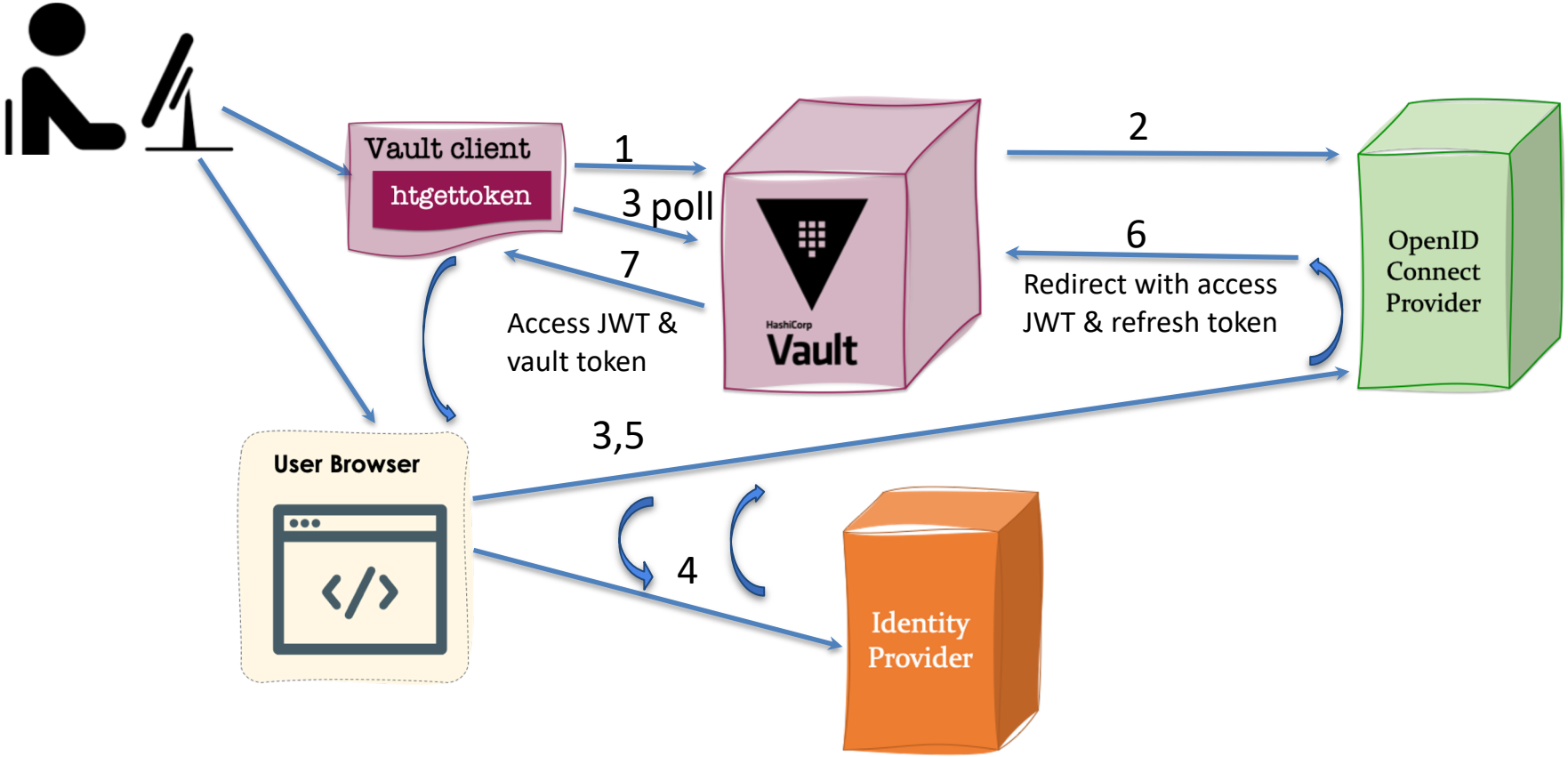
Vault with htgettoken (independent of HTCondor)

- Hashicorp Vault
  - Popular open source general purpose secure secret store server
  - Very flexible plugin architecture and client/server REST/JSON API, and secrets are stored like in a filesystem
  - Has existing OIDC and Kerberos plugins
    - Needed some extensions, submitted as pull requests
  - Behaves as an Oauth2/OIDC client
    - Integrates well with both Indigo IAM and CILogon OIDC Providers, at least
  - Manages access with its own tokens (“vault tokens”)
  - We use it to store long-lived refresh tokens for many users
- htgettoken
  - Relatively simple custom python command line Vault client to automate the flows
  - Initially authenticates via OIDC & a web browser
  - Long life (~1 month, renewable) refresh token stays in Vault, limited life (~1 week) Vault token and even shorter life (~3 hour) access JWT both stored unencrypted in local files
  - Follows WLCG Bearer Token Discovery standard for local filename
  - Uses Vault token to get bearer tokens, or renews Vault access with Kerberos or ssh

Normal federated OIDC flow



htgettoken with Vault initial OIDC flow



Capability sets, issuers, and roles

- JSON Web Tokens can be tailored to minimum privilege by use of “capability” scopes with access limits (and also specific audiences)
- The knowledge of what scopes are allowed per user is maintained by the OIDC Provider, aka the token issuer
  - Does not need to be known by OIDC clients
- We configure Vault to request scope wlcg.capabilityset:/group which the token issuer translates into a set of capability scopes
  - Groups correspond to VOs and roles within those VOs
  - Vault configuration is done per issuer, with one VO per issuer, and each role maps to a wlcg.capabilityset, for example:

```
  htgettoken -a htvault.fnal.gov –i dune –r production
=> https://cilogon.org/dune, wlcg.capabilityset:/dunepro
```

htgettoken normal operation summary

- Given a vault server address and issuer name and optionally a role, htgettoken always gets an access token and stores it in a file
  - By default in ${XDG_RUNTIME_DIR:-/tmp}/bt_u$(id -u) according to WLCG Bearer Token Discovery
- The first time it uses OIDC authentication and additionally gets two more files
  - A vault token stored by default in /tmp/vt_u$(id –u)
  - The “credkey” stored under $HOME/.config/htgettoken defining part of the storage path in vault for the issuer and role
    - Comes from the token issuer based on who authenticated in the web browser
- If credkey exists but the vault token doesn’t work (e.g. vault token expired or for wrong issuer or role), htgettoken attempts Kerberos authentication to get new vault token
  - If no kerberos credentials available or attempt fails, but ssh-agent is available, htgettoken attempts ssh authentication for the new vault token
- And htgettoken has a lot of options for tailoring its operation

Example with htgettoken, initial flow

```
$ env|grep HTG
HTGETTOKENOPTS=--web-open-command=xdg-open --nossh
$ htgettoken -v -a vault.ligo.org -i ligo
Attempting OIDC authentication with https://vault.ligo.org:8200

Complete the authentication at:
    https://cilogon.org/device/?user_code=QZ3-X99-3KG
Running 'xdg-open' on the URL
Waiting for response in web browser
Storing vault token in /tmp/vt_u3382
Saving credkey to /home/dwd/.config/htgettoken/credkey-ligo-default: david.dykstra
Saving refresh token to https://vault.ligo.org:8200
  at path secret/oauth/creds/ligo/david.dykstra:default
Getting bearer token from https://vault.ligo.org:8200
  at path secret/oauth/creds/ligo/david.dykstra:default
Storing bearer token in /run/user/3382/bt_u3382
```

Examples with valid Vault token and with Kerberos

```
$ htgettoken -v -a vault.ligo.org -i ligo
Credkey from /home/dwd/.config/htgettoken/credkey-ligo-default: david.dykstra
Attempting to get bearer token from https://vault.ligo.org:8200
  using vault token from /tmp/vt_u3382
  at path secret/oauth/creds/ligo/david.dykstra:default
Storing bearer token in /run/user/3382/bt_u3382
$ rm -f /tmp/vt_$(id -u)
$ htgettoken -v -a vault.ligo.org -i ligo
Credkey from /home/dwd/.config/htgettoken/credkey-ligo-default: david.dykstra
Initializing kerberos client for host@vault.ligo.org
Negotiating kerberos with https://vault.ligo.org:8200
  at path auth/kerberos-ligo_default
Attempting to get bearer token from https://vault.ligo.org:8200
  at path secret/oauth/creds/ligo/david.dykstra:default
Storing vault token in /tmp/vt_u3382
Storing bearer token in /run/user/3382/bt_u3382
```

Example decode

```
$ httokendecode -H
{
  "sub": "david.dykstra@ligo.org",
  "aud": "ANY",
  "ver": "scitoken:2.0",
  "nbf": "Mon Mar 14 15:24:07 CDT 2022",
  "scope": "read:/frames read:/DQSegDB query:/DQSegDB",
  "iss": "https://cilogon.org/ligo",
  "exp": "Mon Mar 14 15:39:12 CDT 2022",
  "iat": "Mon Mar 14 15:24:12 CDT 2022",
  "jti":
"https://cilogon.org/oauth2/62b3e7866521a5ce9b6570bef50d630f?type=accessToken&
ts=1647289451660&version=v2.0&lifetime=900000",
  "cid": "cilogon:/client_id/caltech/ligo/prod"
}
```

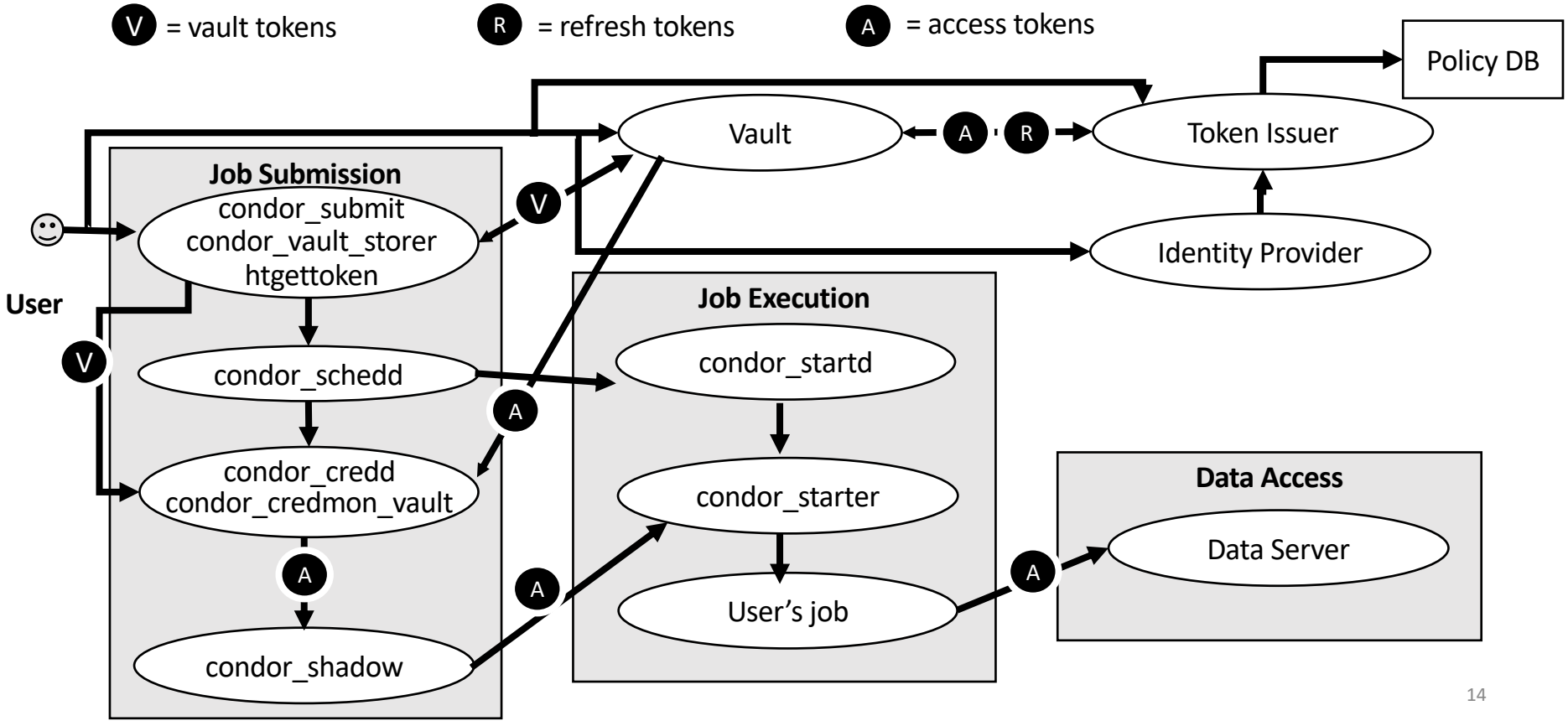
htvault-config configuration package

- Package for configuring Vault for use with htgettoken
  - Automates all the installation and setup of Vault
  - Configuration done through simple, flexible yaml files
  - Includes a modified Hashicorp plugin, an added puppetlabs plugin, and another plugin for ssh-agent support
  - Supports an option of using 3 servers for high availability using a builtin Vault capability
  - Available in OSG yum distribution along with vault and htgettoken

HTCondor+Vault integration

- htgettoken and Vault have been integrated into HTCondor
  - condor_submit can be configured to automatically invoke htgettoken as needed and store a vault token in credd
    - Vault token used by condor_credmon_vault to get new short-lived access tokens pushed to jobs
  - Submit file specifies issuer, optional role, and optionally can choose reduced audience and/or scopes
    - May obtain more than one token for a job
    - Based on HTCondor's previous implementation of Oauth2 credential support
  - In HTCondor 9.0 releases and above

Token flow with HTCondor and Vault



HTCondor configuration

- System admin:
  - Install condor-credmon-vault rpm and set for example:

```
SEC_CREDENTIAL_GETTOKEN_OPTS = -a htvault.fnal.gov
```

- User submit file for example:

```
use_oauth_services = dune
dune_oauth_permissions = storage.read:/dune #optional
dune_oauth_resource = https://dcache.fnal.gov #optional
```

- Service names may include role, such as `dune_production`
- Handles may be appended to store multiple variations for each service:

```
dune_oauth_permissions_readonly = storage.read:/dune
dune_oauth_permissions_write = storage.create:/dune/users/dwd/data
```

- All tokens end up in `$_CONDOR_CREDS`

Support for “robot” (unattended) operation

- Important for tasks such as production job submission
- Vault administrator can create indefinitely renewable vault tokens
  - Could be automated by a web service
- htgettoken & htvault-config also support use of robot Kerberos credentials to get new vault tokens
  - Robot Kerberos credentials are long lived, stored unencrypted
  - Principals are in the form “user/purpose/machine.name”
    - “user” can also be a group login, for example “dunepro”
  - User (or authorized user for a group) does OIDC authentication once but specifies htgettoken --`credkey` option matching Kerberos principal to store refresh token in subpath under the user’s Vault secrets path
    - The same htgettoken command can be used with robot Kerberos credentials
- Can also use ssh-agent with authorized keys to get new vault tokens
  - Although haven’t yet worked out how to manage the keys

Conclusions

- Getting credentials almost as hidden as they can be
  - Users with Kerberos or ssh-agent only need to approve on web browser once
- Configuration is managed by server operators, very little necessary for users unless they want to “down-scope” their tokens
- All protocols are in common industry use
- JWTs are better supported and more secure than X.509 proxies
  - Can be much more purpose-specific
- Tools all open source, generally available

Links

- Bearer token discovery:
  - https://github.com/WLCG-AuthZ-WG/bearer-token-discovery
- WLCG JWT profile
  - https://github.com/WLCG-AuthZ-WG/common-jwt-profile
- Vault & plugins
  - https://www.vaultproject.io/
  - https://github.com/hashicorp/vault-plugin-auth-jwt
  - https://github.com/puppetlabs/vault-plugin-secrets-oauthapp
  - https://github.com/42wim/vault-plugin-auth-ssh
- htvault-config: https://github.com/fermitools/htvault-config
- htgettoken: https://github.com/fermitools/htgettoken
- HTcondor docs: https://htcondor.readthedocs.io/en/latest/search.html?q=vault