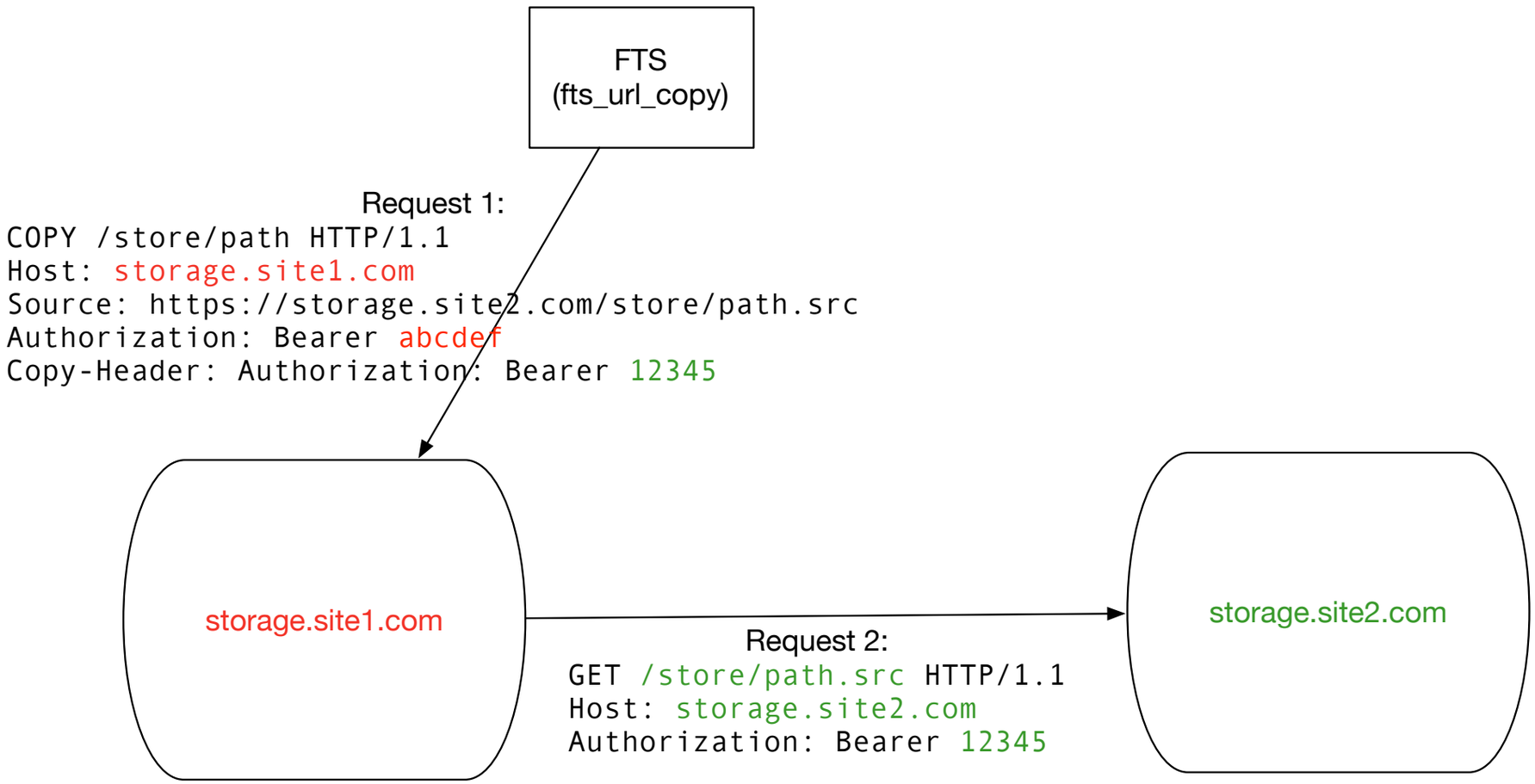
Capability-based authorization

Brian Bockelman,
WLCG Workshop, 2018

Warning: NERD ALERT

- This presentation is meant as a follow-up to the high-level overview:
    - https://indico.cern.ch/event/658060/contributions/2886775/
- The point is to provide the underlying technical details that I glossed over.
    - So, if you don't want to be reading about HTTP headers and token formats … it's a good time to start reading your email

Reminder:
Here's our picture



Here, I illustrate the case where the actual copy also goes over HTTP

HTTP Request

HTTP verb

Resource at active SE (destination)

- Example request from FTS to “active” SE:

```
COPY /store/path HTTP/1.1
Host: storage.site1.com
Source: https://storage.site2.com/store/path.src
Authorization: Bearer abcdef
Copy-Header: Authorization: Bearer 12345
```

Source URL

Token for active SE

Token for inactive SE

Indication to copy header to GET request

“Real” Request

Here’s an example request from yesterday:

```
COPY /user/uscms01/pnfs/..truncated../LoadTestDownload/LoadTest07_UCSD_B0_nI3SsXhd0iT5RF6W_286 HTTP/1.1
User-Agent: fts_url_copy/3.7.7 gfal2/2.15.0 neon/0.0.29
TE: trailers
Host: red-gridftp12.unl.edu:1094
Source: https://gftp-1.t2.ucsd.edu:1094/cms/..truncated../LoadTest07_UCSD_B0
X-Number-Of-Streams: 3
Secure-Redirection: 1
Authorization: Bearer eyJhbGciOi..truncated..NYU5gx6yrZhKpdCt2SedVocIhZsuqKNUNZcRhXj6tBjxozA
ClientInfo: job-id=dc417124-30d7-11e8-bd67-5254000b9cba;file-id=1080;retry=0
TransferHeaderAuthorization: Bearer eyJhbGci..truncated..5_-Z7XQw
RequireChecksumVerification: false
```

Let’s look at an example token

You might see this base64 mess in a header:

Header
Payload
Signature

eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCIsImtpZCI6IjYzMjUxYWFmMjM2OWQ2Njc3ZjYwZmFhZDY0ZDJjYTcwMzFjZjNlYTVkMDYxM2M5YmU3OGZlYWYyZGVkNWM0NzcifQ.eyJqdGkiOiJiOGQ1NGE2Mi1jZDMzLTRiNGItYmI2NC0xMWI4MDQyNzJmMWQiLCJzdWIiOiJjbHVuZHN0IiwiZXhwIjoxNTIxNTYxMzgyLCJpc3MiOiJodHRwczovL3NjaXRva2Vucy5vcmcvY21zIiwiaWF0IjoxNTIxNTU3NzgyLCJzY3AiOlsid3JpdGU6L3N0b3JlL3VzZXIvY2x1bmRzdCIsInJlYWQ6L3N0b3JlIl0sInJlYWQ6L3N0b3JlIl0sIm5iZiI6MTUyMTU1Nzc4Mn0.bmL-w3drGqTEyZCXt_06DTeg2cT6aWobDMQhKP00dEyk8FpO3YjqBz-ToNnmEDEeFuCRLgpwx_U14ttBulxkOw6hkOFbPGOilsLazK-6WcP2cpFkM9uuUJU_1_wLrCSkGBrFIjJwH4uzRWCIlttEAFnjy19QLCwgLsuGOBjNOdxTSh4Gp_SqjHYtMnXfI8T3q3krGCtSM39YmNwgc8oNYF7Bmq8WEP_Hqa-wr0B7qxMYb89BZQrZMjqve91klqI5Fvjd5RobA1oe5EMC9-FREInq8xCUu4TtpQAazC3UmL4vXipLUPGRXSioLGU6R74aP0Avlbc5sWvKsdeXCvea4Q

Decoded header:

{“alg":"RS256","typ":"JWT","kid":"63251aaf2369d6677f60faad64d2ca7031cf3ea5d0613c9be78feaf2ded5c477"}

Decoded payload:

'{"jti":"b8d54a62-cd33-4b4b-bb64-11b804272f1d","sub":"clundst","exp":1521561382,"iss":"https://scitokens.org/cms","iat":1521557782,"scp":**["write:/store/user/clundst","read:/store"]**,"nbf":1521557782}'

Token Verification

Decoded header: {“alg":"RS256","typ":"JWT",**"kid":"key1"**}
Decoded payload:
'{"jti":"b8d54a62-cd33-4b4b-bb64-11b804272f1d","sub":"clundst","exp":1521561382,**"iss":"https://scitokens.org/cms"**,"iat":1521557782,"scp":["write:/store/user/clundst","read:/store"],"nbf":1521557782}'

- Client library will first look at the `iss` claim in the payload.
- From this, we derive the auto-discovery URL:
  - https://scitokens.org/cms/ -> https://scitokens.org/cms/.well-known/openid-configuration
- The auto-discovery URL provides a location of the public keys for the issuer:
  ```
  {"issuer":"https://scitokens.org/dteam",
   "jwks_uri":"https://scitokens.org/dteam/oauth2/certs"}
  ```
- The JWKS URI will contain a dictionary of public keys. The client looks at the `kid` claim in the payload and uses this value to locate the corresponding public key.

Public Key Verification

- From the public key, the client implementation goes back to the base64-encoded version of the token and verifies the public key matches the token contents and signature.

- Current signature algorithms include RSA and EC.

  - Provide reasonable levels of security, but EC signatures are smaller and cost less CPU time to verify.

Decoded header:
{“alg":"RS256","typ":"JWT",**"kid":"key1"**}

JWKS URI contents


```
{
    "keys": [
        {
            "alg": "RS256",
            "e": "AQAB",
            "kid": "key1",
            "kty": "RSA",
            "n":
"oj5UxvZGgQU2UHGdO2ViR6zkilHjTSFdTA_Jtb1KPKmqr3I7W
r2QbF-9qwunaLoMpal2ZFJTkbMweiFiq-
duhzcKI1JuaNkUJJpd6BGXVoszn31KHlVkUxd739FYyKLArUnn
6Dn6rfT6QppBQPz_7t1LN-PIs-lO50nEsiDPHhb7GI0XucajA9
            "use": "sig"
        }
    ]
}
```


Public Key

Token Validation

Decoded header: {“alg":"RS256","typ":"JWT",**"kid":"key1"**}
Decoded payload:
'{"jti":"b8d54a62-cd33-4b4b-bb64-11b804272f1d","sub":"clundst","**exp**":1521561382,"iss":"https://scitokens.org/cms","iat":1521557782,"scp":["write:/store/user/clundst","read:/store"],"**nbf**":1521557782}'

- Great, our token has a valid signature! What next?
  - Look at the “expiration time” (`exp`) and “not before” (`nbf`) to make sure token is valid.
  - Log the token ID (`jti`) and subject (`sub`) to the audit log.
- Next, we look at the scope claim (`scp`) to determine what the token is allowed to do.

Token Authorization

- Authorizations are composed of an operation and a resource.
- Example: “**write**:**/store/user/clundst**”
    - **write**: allows one to change data (write) at a resource.
    - **/store/user/clundst**: resource operation applies to relative to VO’s base path.
- Authorizations are relative to a VO’s storage area.
- Example:
    - VO base storage path: **`https://example.unl.edu/pnfs/cms`**
    - Authorization: **`write:/store/user/clundst`**
    - Then, bearer can write into: **`https://example.unl.edu/pnfs/cms/store/user/clundst`**

Note: for our demonstrator we use this particular profile; working on standardizing this across WLCG within the Authorization Task Force!

Implementing on Storage - Xrootd

- We have two plugins for XRootD:
  - **WebDAV third-party-copy support**. Hooks into the COPY request and utilizes libcurl to perform HTTPS transfers; supports multiple TCP streams for transfers.
  - SciTokens (JWT) authorization. Extracts the `Authorization` header and translates the token authorizations to the XRootD internal framework.

Map issuer to storage area

[Issuer CMS]
issuer = https://scitokens.org/cms
base_path = /user/uscms01/pnfs/unl.edu/da
default_user = cmsprod

[Issuer dteam]
issuer = https://scitokens.org/dteam
base_path = /dropfiles/dteam
default_user = dteam

Map issuer to Unix ID for writing to actual filesystem. (Useful for hybrid configurations)

Transfer Layer

- Great, these tokens are wonderful. How do we get them?
    - Right now, we cheat: when the FTS transfer *starts*, FTS will use the submitter's X509 proxy to contact the token issuer (as provided by the file metadata).
        - Based on the DN + VOMS attributes, token issuer applies a simple set of rules and determines the available authorizations. Token is returned.
        - If a JWT isn't available, then FTS contacts the *storage system* to acquire a macaroon (a different bearer token approach).
        - If neither are available, then FTS will fallback to GridSite-based delegation.
- This is useful for the bootstrapping phase (X509 not used for transfers), but still has X509 for interaction with FTS. Improved native OIDC support is needed in FTS to remove this last piece.

Interested?
Come aboard!

- There are many places I skipped:
    - How are tokens generated? How does this software come into existence (or does it already exist!)?
    - Long term, which OAuth2 workflow is used?
- Join us at https://groups.google.com/forum/#!forum/wlcg-http-transfer
    - We need additional sites to transfer with and people to drop code into additional storage elements.