Web App
Access to EOS using OAUTH2

Andreas-Joachim Peters
CERN IT Storage Group

Web App Access

In front-end web-services today (e.g. SWAN/CERNBOX Web) we provide the involved service gateways with **privileges to either create kerberos token** or to have **sudo access** to act on behalf of the user. This approach works well, but with regard to security considerations **violates the least privilege model.**

Web Applications on the other hand typically use single-sign-on mechanisms and **CERN is transitioning the SSO infrastructure** to be based on KeyCloak (OAUTH2 tokens).

Web App Access

In a joint effort we worked on the realisation of JIRA OS-9604 - allowing to use a single sign-on token for EOS access via **eos**xd.

The **initial idea** exporting the SSO token as an environment variable **failed**, because tokens are relatively short-lived and need to be refreshed. In LINUX environment variables are inherited on fork e.g. it is impossible for a parent process to change the environment variable of inheriting children.

This lead us to implement what was implemented already decades ago:
a token file

OAuth2 Token Support in EOS

eosxd has *(thanks to the modular design of the security infrastructure from Georgios)* all the logic to support token files.

eosxd picks up the token file either from a default location `/tmp/oauthtk_$UID` or from a location set via an environment variable `OAUTH2_TOKEN`=[FILE:]<tokenfile>

OAuth2 Token Transport

XRootD4 has request encryption on the wire. Therefor the token is transferred as an endorsement in an **sss** authentication.

To provide **sss** we use a *nobody* key e.g. the key alone authenticates as user nobody, but if an endorsement is present, the contained oauth2 token is resolved at the allowed AuthResolver service, which provides which CERN user is authenticated by the SSO token.

Since every MGM call calls an ID mapping function, which translates embedded tokens, we currently cache these token by default for 15 minutes in the MGM, before re-checking them again agains the AuthResolver service.

OAuth2 Token Transport

```
[root@ajp auth]# ./oinit apeters
Password:
info: created token file /tmp/oauthtk_0

[root@ajp auth]# cat /eos/ajp/proc/whoami
Virtual Identity: uid=100755 (2,99,100755) gid=1338 (1338,99,4) [authz:oauth2]
host=ajp.cern.ch domain=cern.ch geo-location=ajp key=<oauth2> fullname='Andreas Joachim
Peters' email='andreas.joachim.peters@cern.ch'

[root@ajp auth]# rm /tmp/oauthtk_0
rm: remove regular file '/tmp/oauthtk_0'? y
[root@ajp auth]# cat /eos/ajp/proc/whoami
Virtual Identity: uid=99 (99) gid=99 (99) [authz:unix] host=ajp.cern.ch domain=cern.ch
geo-location=ajp
```

Outlook

These mechanism **can be used for SWAN and CERNBOX** web front-ends and provide a stream-
lined and more secure authorisation framework.

Using the **concept of scopes**, the usability (abuse) of the token can be even more restricted in the future!

Ideally **XRootD could provide a native token-exchange mechanism** / API in XRootD5. The currently used backdoor via sss authentication+endorsement has the advantage that the token is only exchanged once (which makes sense if you grant accessto a subtree and not single files).