Single Sign On for UNICORE command line clients

Krzysztof Benedyczak

ICM, Warsaw University

Current status of UNICORE access

- Legacy certificates still fully supported
  - nice on home workstation, especially when loaded into a browser
- With help of Unity username & password authentication is possible
- as well as federated login to UNICORE portal.
- Unity also solves delegation issue: it generates it on user's behalf, so chained Grid workflows can be executed.

The gap

- SSO works only for client runtime duration. After restart authentication must be repeated.
  - Not a problem for URC, portal and UCC in the shell mode.
  - UCC in non-shell mode is problematic, yet more useful.
- Same problem occurs with agent machines where real credentials can't be uploaded due to security policies.
- Old Proxy Certificates were tackling this issue...

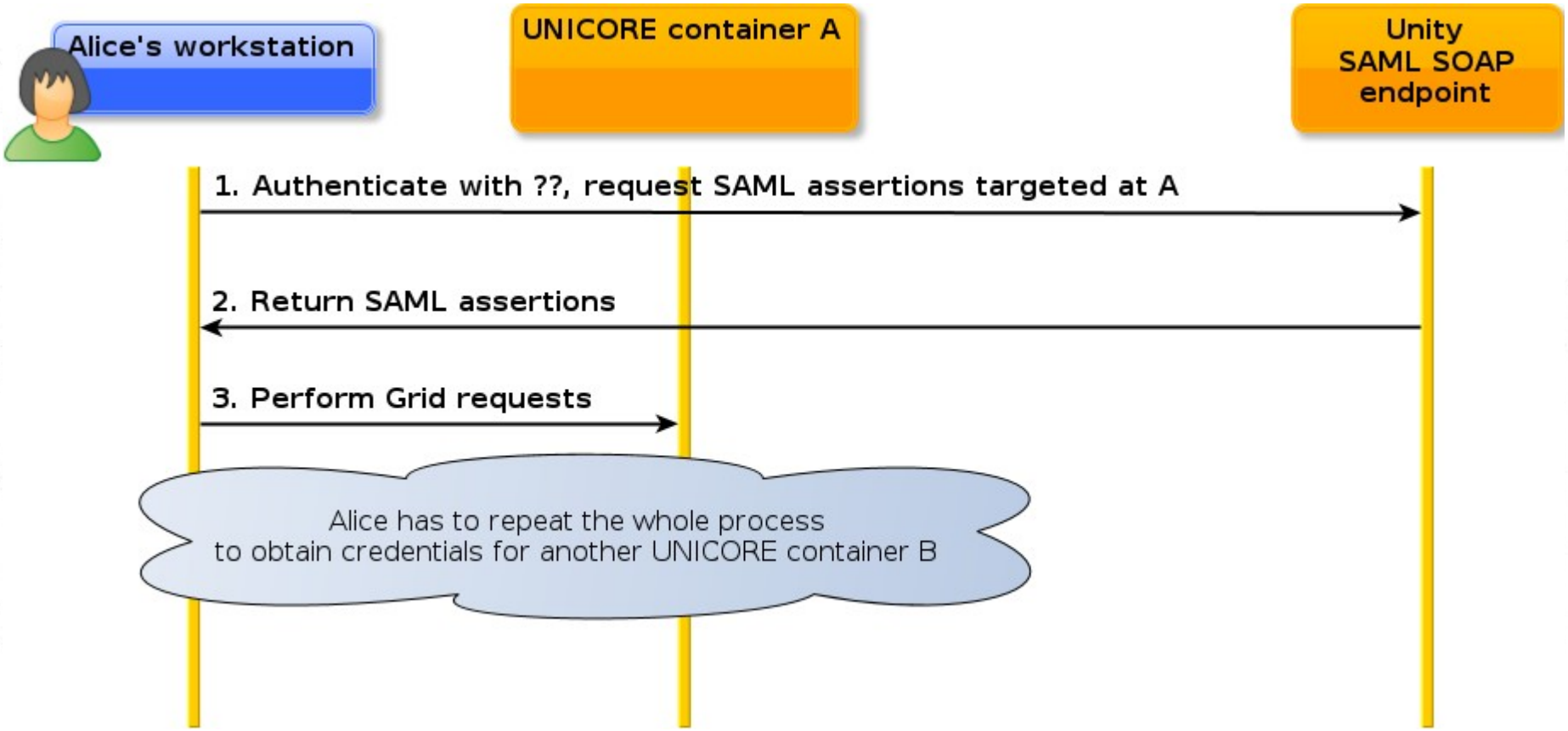
Technical perspective of the problem

- There are two parts of the problem to be solved.
- To access a server, the client has to authenticate itself:
  - with SAML authN assertion or by using certificate&PK
- Additionally trust delegation must be sent:
  - either signed with PK or received from Unity
  - *actually not always required but often is*

Is certificate enough?

- If a certificate is available, a trust delegation can be generated.
- Using the certificate directly in automated scenario requires either:
  - storing PK without password, or
  - storing password in a text file
- *Insecure, violates CA policies, can't be used with federations, no go...*

Access with help of Unity



Access with help of Unity

- When using Unity trust delegation is generated by Unity on user's behalf.
  - validity is sufficient, typically 2 weeks or so.
- However SAML authentication assertion is:
  - targeted at particular receiver (server container)
  - very short lived (range of minutes)
- *Hacking SAML authentication is theoretically possible but controversial.*

Security sessions to rescue?

- UNICORE security stack supports *security sessions* concept
- In response to the first, fully authenticated and authorized client's request, a server returns session identifier.
- Client can subsequently use this identifier instead of pushing the AA data again.
  - Improves performance
- Enhancing the mechanism to store the session ids on disk is possible.
- *Unfortunately for each connected service container we need to send AA data again…*

Problem summary

- SAML authentication assertion is the root of our problem.
- Usage of certificates doesn't help.
- Client<->Server security sessions won't help too.
- We need something easy to use, as easy as proxies are (after you have one generated!).

Proposed approach

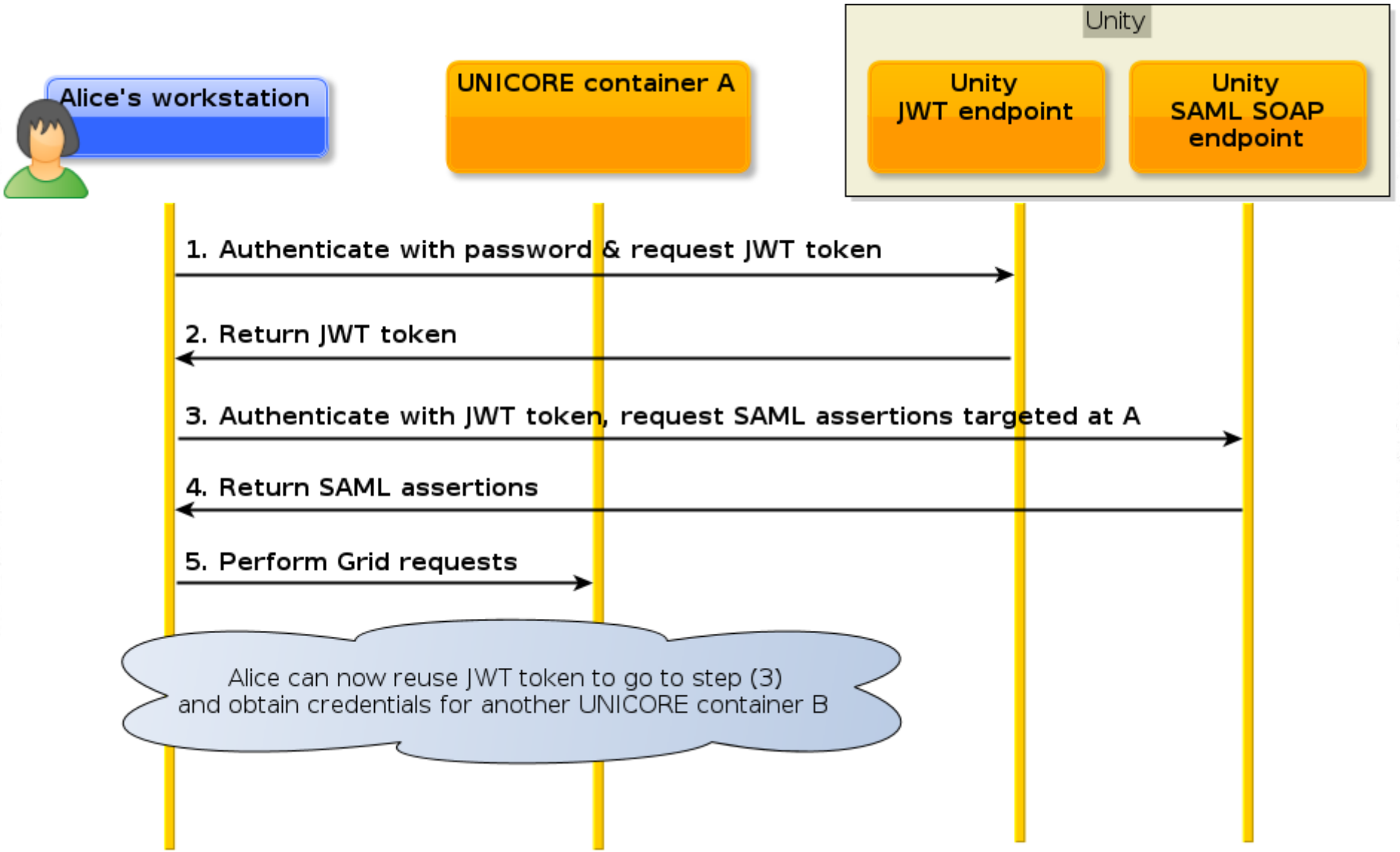
- Session between Unity and client
  - used only when requesting assertions from Unity
  - token stored on disk
  - protected with FS rights

JWT endpoint and authN in Unity

- Unity already supports such mechanism
- There is a JWT endpoint allowing to generate tokens
  - Token can be also refreshed and revoked
  - Endpoint is trivial, RESTful
- Token is a signed, self contained JSON
- Unity also provides JWT authenticator, which can be used for both REST and SOAP endpoints.

```
{
    "sub":"c6789770-f587-4936-99df-d57c5d8a68e6",
    "aud":"https:\/\/localhost:53456#testr",
    "iss":"https:\/\/localhost:53456",
    "exp":1441292091,
    "iat":1441292089,
    "jti":"c97cf800-0259-44c0-8d0e-bdb0446c9fb8"
}
```

Complete scenario



Missing parts to be implemented

- UCC (or secutils-cxf?) would need to have simple JWT support
- Commands to revoke and refresh token
- UCC authentication method using JWT
  - How to cleanly implement this? In fact we have two anthN mechanisms here, Unity then JWT
- Currently JWT validity is only controlled in endpoint's configuration.
  - If needed it can be enhanced in Unity so client can control the validity in an allowable range.

Summary

- From user perspective: single authentication to Unity
- Token can be copied/reused/refreshed and revoked.
- No or minimal Unity modifications.