
# What you wish you knew about ... tokens!

Elvin Sindrilaru

on behalf of the **EOS team**

25.04.2023

# Outline

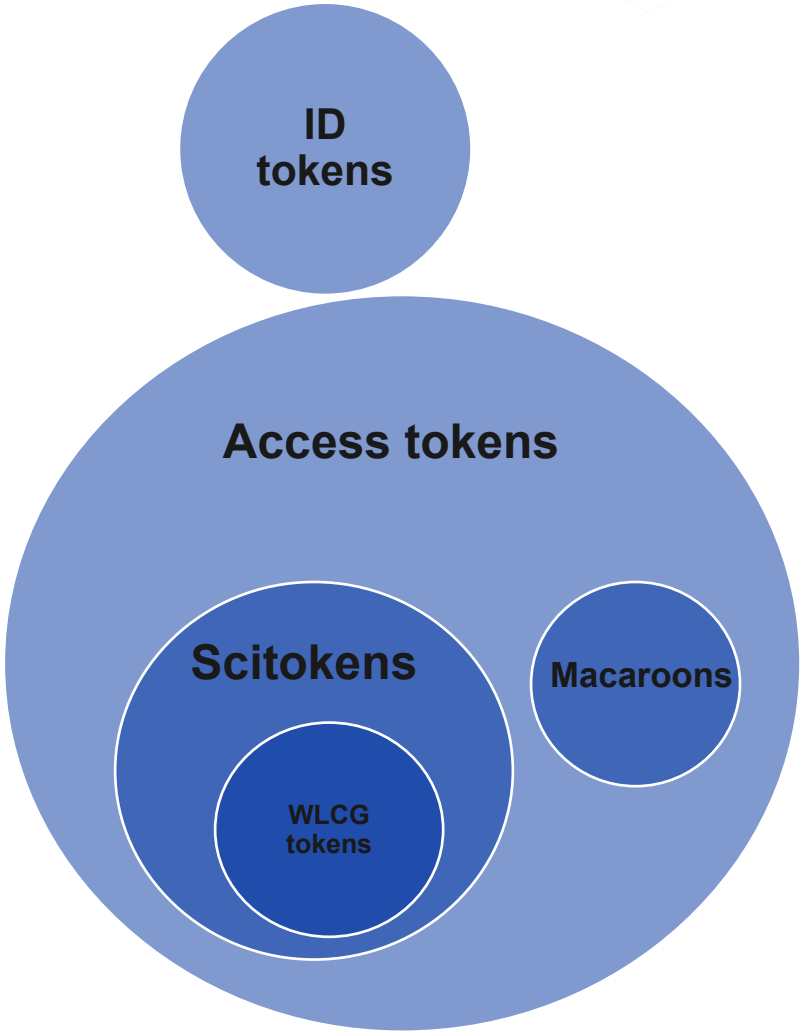
- Types of tokens and terminology
- Token architecture overview and interaction with other components
- Token support for the xroot protocol
- Token support for the HTTPS protocol

# Why we need tokens?

This talk will **not** attempt to answer any philosophical questions ...

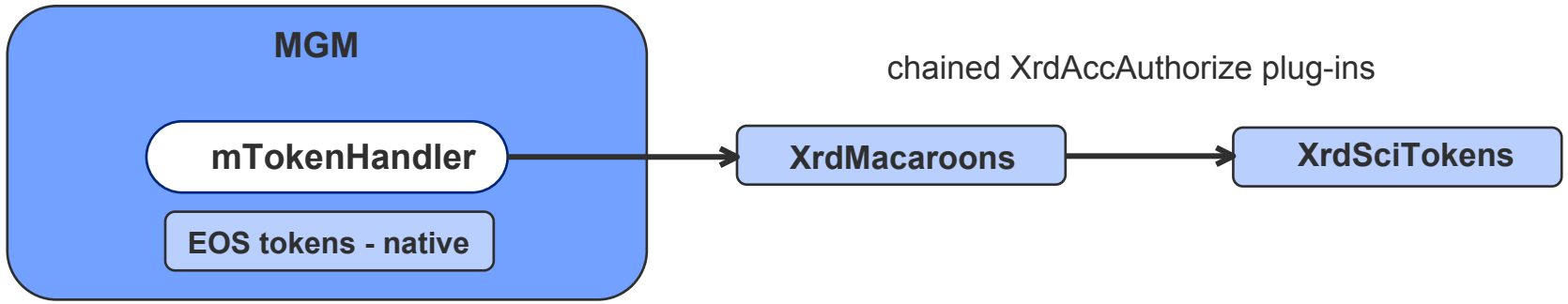
# Types of tokens and terminology

- ***Bearer Token:*** "*a string representing an access authorization issued to the client*"
- **Types of (bearer) tokens:**
  - **ID tokens** (Open ID Connect) - **who someone is**
    - ***Must not*** be used to make requests to the resource server
  - **Access tokens** (OAuth 2.0) - **what someone is allowed to do**
    - **Should** be used only to make requests to the resource server
    - Different formats from simple hex string to JSON Web Tokens (JWT)
    - **JWT** - way to encode claims in a JSON document that is then signed
- **Advantages**
  - Simple to use for API requests
  - Don't require cryptographic signing of each request
- **Disadvantages**
  - Communication channel needs to be encrypted
  - Anyone getting access to a token can use it
  - In general **can not be revoked**

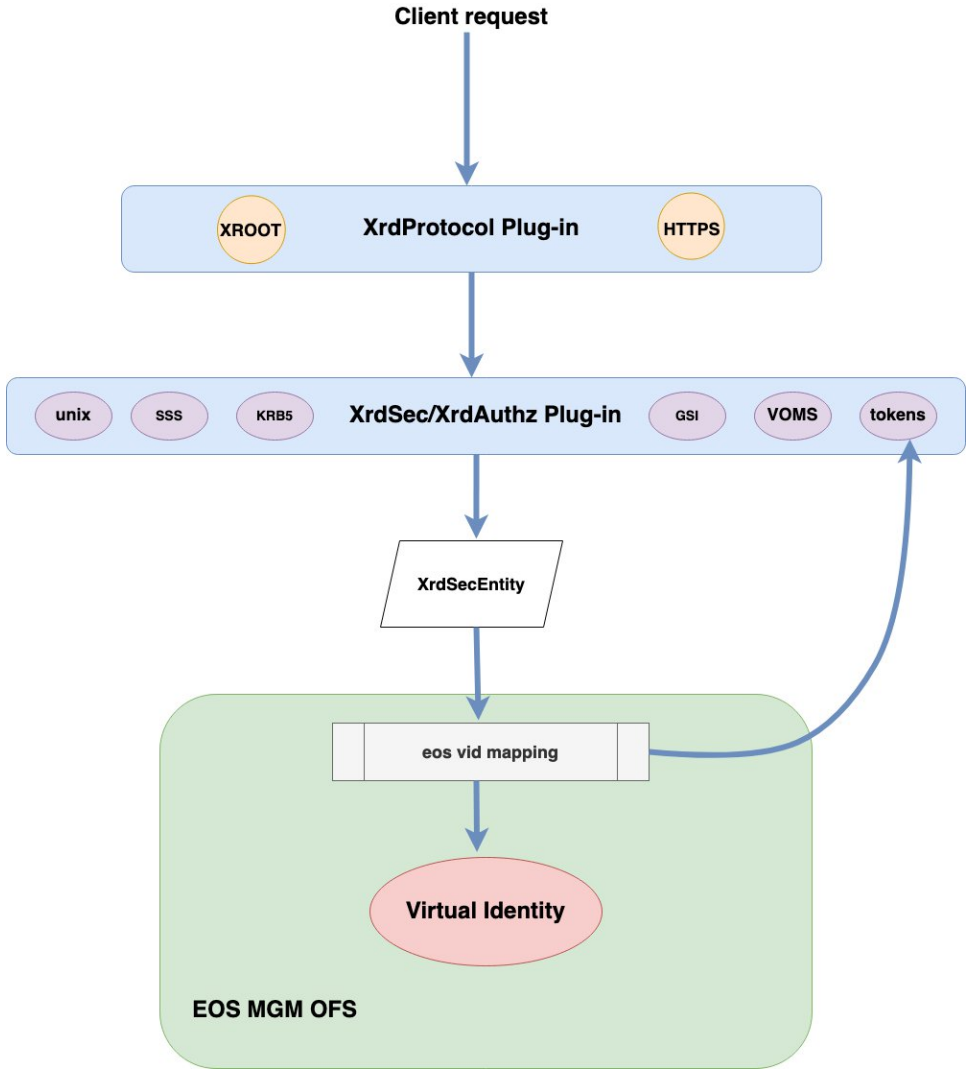


# (Tokens) Architecture overview

- **EOS** relies on the **XRootD authorization framework**
- **Token support** is implemented as an **XrdAccAuthorize** plug-in
- Invoking an XRootD authz plugin will populate the **XrdSecEntity** object
  - bound to the **lifetime of the TCP connection**
- **XrdSecEntity.name** field is (in general) used to extract the local **uid/gid** mapping
- Various types of **authentication** can be **enabled/disabled selectively**
  - i.e. `eos vid enable gsi; eos vid enable https; eos vid enable ztn;`



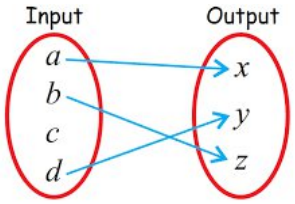
# (Auth) Architecture overview



# (Auth) Architecture overview

**XrdSecEntity → eos vid mapping → eos::common::VirtualIdentity**

```
class XrdSecEntity
{
public:
         char    prot[XrdSecPROTOIDSIZE]; //!< Auth protocol  used (e.g. krb5)
         char    prox[XrdSecPROTOIDSIZE]; //!< Auth extractor used (e.g. xrdvoms)
         char   *name;                    //!< Entity's name
         char   *host;                    //!< Entity's host name dnr dependent
         char   *vorg;                    //!< Entity's virtual organization(s)
         char   *role;                    //!< Entity's role(s)
         char   *grps;                    //!< Entity's group name(s)
         char   *caps;                    //!< Entity's capabilities
         char   *endorsements;            //!< Protocol specific endorsements
         char   *moninfo;                 //!< Information for monitoring
         char   *creds;                   //!< Raw entity credentials or cert
         int     credslen;                //!< Length of the 'creds' data
unsigned int     ueid;                    //!< Unique ID of entity instance
XrdNetAddrInfo  *addrInfo;                //!< Entity's connection details
const    char   *tident;                  //!< Trace identifier always preset
const    char   *pident;                  //!< Trace identifier (originator)
         void   *sessvar;                 //!< Plugin settable storage pointer,
                                          //!< now deprecated. Use settable
                                          //!< attribute objects instead.
         uid_t   uid;                     //!< Unix uid or 0 if none
         gid_t   gid;                     //!< Unix gid or 0 if none

         void   *future[3];               //!< Reserved for future expansion

XrdSecEntityAttr *eaAPI;                  //!< non-const API to attributes
}
```




```
//------------------------------------------------------------------------------
//! Struct defining the virtual identity of a client e.g.
//! their memberships and authentication information
//------------------------------------------------------------------------------
struct VirtualIdentity {
  uid_t uid;
  gid_t gid;
  std::string uid_string;
  std::string gid_string;
  std::set<uid_t> allowed_uids;
  std::set<gid_t> allowed_gids;
  XrdOucString tident;
  XrdOucString name;
  XrdOucString prot;
  std::string host;
  std::string domain;
  std::string grps;
  std::string role;
  std::string dn;
  std::string geolocation;
  std::string app;
  std::string key;
  ....
}
```

# Token authz library configuration

- Token authz libraries are **shared and used similarly across protocols** (xrootd/https)
  - They are **chained** in the order of their declaration
  - First one to **recognize the token** type handles it
- MGM configuration file can also include **(token) library specific configuration directives**
- All token mapping and processing functionality grouped inside the **IdMap** method

```
// /etc/xrd.cf.mgm
...
# Token authz libraries supported (chained)
mgmofs.macaroonslib libXrdMacaroons.so libXrdAccSciTokens.so
# Token library specific config directive
scitokens.trace all
macaroons.secretkey /etc/eos.macaroon.secret
macaroons.trace all
...
```

# Tokens support over xroot protocol

- Relies on the **ztn** client validation protocol
  - **Initial token** used only to establish a **secure connection**
  - Subsequent requests on the same TCP connection **can use different tokens**
- Each new requests goes through the **identity mapping process**
- **MGM configuration in `/etc/xrd.cf.mgm`**
  - Enable **ztn** support for the XRootD framework
- **Enable VID ztn** mapping in EOS
  - `eos vid enable ztn`
  - Without this the Virtual Identity will remain **nobody** i.e. `uid=99 gid=99`
- Supplied token is decoded and interpreted by the **configured token library(ies)**

```
# ...
# ztn authentication
sec.protocol ztn
sec.protbind * only ztn sss krb5 unix
# ...
```

# Tokens support over HTTPS protocol

- Relies on HTTPS to provide the necessary **encrypted** communication channel
- Similarly to xrootd access:
  - Each new requests goes through the **identity mapping process**
- **MGM configuration in `/etc/xrd.cf.mgm`**
  - Enable **https** support for the XRootD framework
  - Supports also **plain/proxy GRID certificates**

```
// /etc/xrd.cf.mgm
...
xrd.protocol XrdHttp:9000 libXrdHttp.so
# TLS config shared across both https and xroots protocols
xrd.tls /etc/grid-security/daemon/hostcert.pem /etc/grid-security/daemon/hostkey.pem
xrd.tlsca certdir /etc/grid-security/certificates/
# EOS mandatory HTTP handler
http.exthandler EosMgmHttp libEosMgmHttp.so
...
```

- **Enable VID https** mapping in EOS
  - `eos vid enable https`
  - Without this the Virtual Identity will remain **`nobody`** i.e. **`uid=99 gid=99`**
- Token handling identical to the **ztn** scenario

# XrdSciTokens configuration and use

- **SciTokens** supported by **libXrdSciTokens.so** that comes by default with **XRootD**
- Requires direct interaction with a **IAM (Identity & Access Management) Provider**
- Configuration file for SciTokens: ***/etc/xrootd/scitokens.cfg***
- Several ways of doing authorization:
  - **scope-based** - when a certain path is authorized
  - **group-based** - group info is copied to the XRootD internal credentials object (XrdSecEntity)

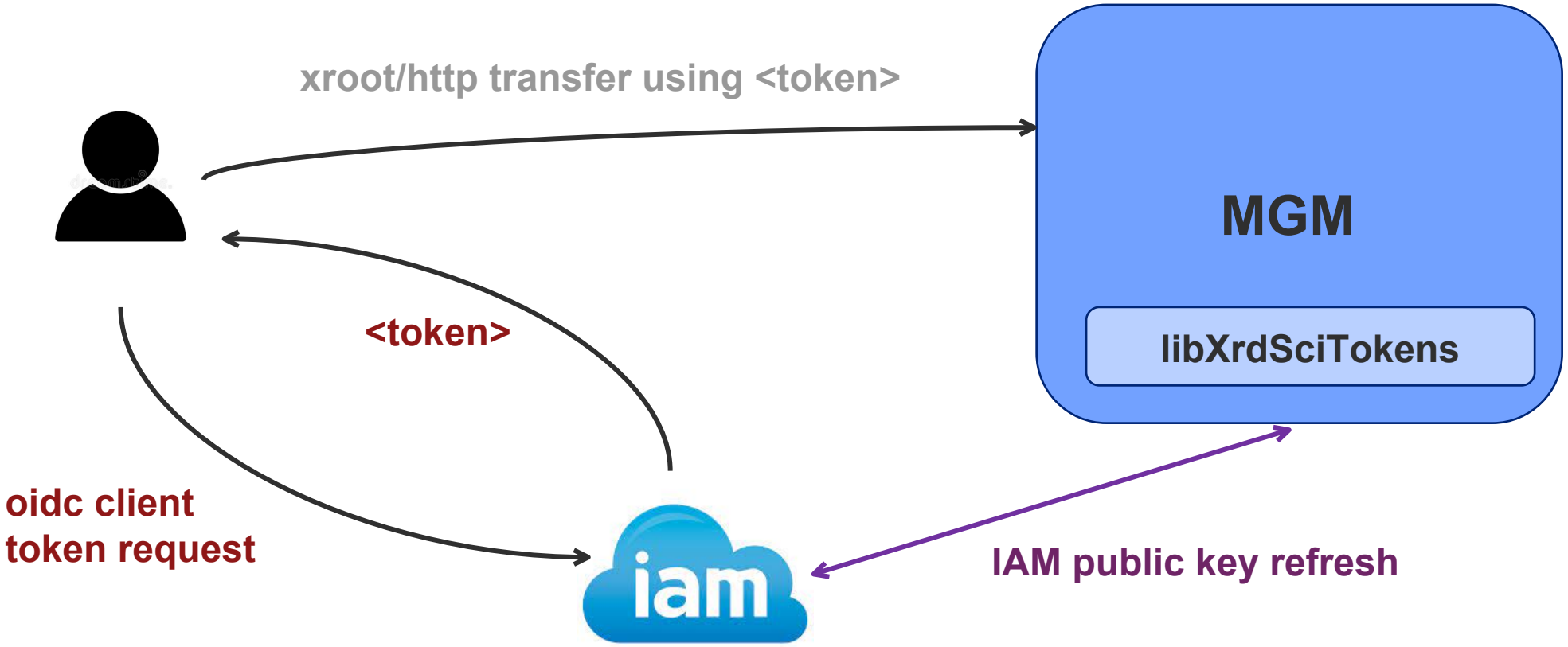
```
[Global]
audience = https://wlcg.cern.ch/jwt/v1/any,https://elvin-dev01.cern.ch

[Issuer OSG-Connect]
issuer = https://wlcg.cloud.cnaf.infn.it/
base_path = /
map_subject = False
default_user = esindril
name_mapfile=/etc/xrootd/mapfile.json
```

# XrdSciTokens name-map functionality

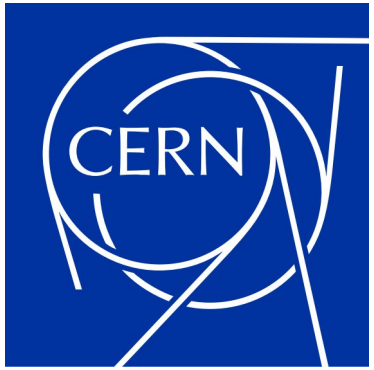
- **Storage systems** need to associate a **local username** to incoming requests
- **SciTokens** provide a **"gridmap-like" functionality** to perform identity mapping
  - can be enabled by specifying the **namemap-file** directive
  - **allows fine-grained control** over the identity mapping

```
[ {"sub": "1234-567-89ab", "path": "/home/esindril", "result": "esindril"},
  {"group": "/it/test", "path": "/it", "result": "ittest", comment="Only for tests"},
  {"group": "/it", "result": "it"},
]
```

# XrdSciTokens interactions



# Thank you! Questions? Comments?




home.cern