DIRAC Security Infrastructure

A. Casajús Ramo[*]
R. Graciani Díaz[†]
Universitat de Barcelona, Departament d'Estructura i Constituents de la Matèria

Abstract

DIRAC is the LHCb Workload and Data Management System. Based on a service-oriented architecture, it enables generic distributed computing with lightweight Agents and Clients for job execution and data transfers. DIRAC implements a client-server architecture exposing server methods through **XML** **R**emote **P**rocedure **C**all (XML-RPC) protocol. DIRAC is mostly coded in python.

DIRAC security infrastructure has been designed to be a completely generic XML-RPC transport over a SSL tunnel. This new security layer is able to handle standard X509 certificates as well as grid-proxies to authenticate both sides of the connection. Server and client authentication relies over OpenSSL and pyOpenSSL, but to be able to handle grid proxies some modifications have been added to those libraries.

DIRAC security infrastructure handles authentication and authorization as well as provides extended capabilities like secure connection tunnelling and file transfer.

Using this new security infrastructure all LHCb users can safely make use of all the computing resources available on the Grid through DIRAC.

SECURITY INFRASTRUCTURE DESIGN

As most grid middleware, DIRAC security is based on digital certificates and grid proxies. Both certificates and proxies are signed by a trusted **C**ertification **A**uthority (CA). Whenever a grid certificate has to be authenticated, the CA signature carried by the certificate is verified against the trusted ones. Grid proxies are special self-signed time resticted user certificate and key kept together.

DIRAC security infrastructure is based on three principles:

1. **Authentication** to ensure that both client and server are identified by each other.

2. **Authorization** to restrict access to server functionality according to the identity of clients.

3. **Logging** to allow traceability of client requests and server actions.

When a connection is established not only the client is authenticated by the server, but also the server must be authenticated by the client. Authenticating both sides of the connection is the first step to perform a secure connection. The server knows who is performing the query and the client knows that the connection is being established with the right server. After this initial identification, and following the Secure Socket Layer (SSL) protocol, the client-server communication is encrypted.

At this moment, no one can stand in the middle of the connection without being noticed. Once the authentication has taken place, the user gets identified by the **D**istinguished **N**ame (DN) included in the certificate or the proxy.

Once the mutual authentication has been done, the client sends a query to the server and the authorization takes place. In the XML-RPC protocol, a query is a call to a server exported method with certain parameters. At this point, the server decides, based on its configuration, if the identified user is allowed to access the requested method. Authorization is the second step to perform a secure connection.

If the user is authorized, the server executes the method and returns the result. Both the query and the returned value are logged together with the identity of the user and a timestamp. All data sent through the connection is encrypted to prevent third parties from getting any data. Logging information is extremely important to ensure that all actions can be traced back. Logging is the third step to perform a fully secure and traceable implementation.

DISET TRANSPORT

The initial DIRAC versions did not have any security since its use was reduced to a limited number of production managers responsible for the Monte Carlo simulation activites at LHCb sites. In order to make use of DIRAC in a distributed GRID environment it was necesary to add a security layer in its architecture. Therefore, the **DI**RAC **Se**cure **T**ransport (DISET) was born.

DISET is an extension of the **h**yper**t**ext **t**ransfer **p**rotocol over **s**sl(HTTPS) supporting X509 certificates and grid proxies. It also enhances some of the native python XML-RPC capabilities.

DISET hides all nasty SSL code, authentication and first level of authorization mechanisms, under simple python calls. Developers just use DISET objects as if they were the native python XML-RPC objects.

DIRAC developers can use DISET to perform XML-RPC queries over secure and insecure connections depending on the protocol set (http/https/diset) in the **U**niversal **R**esource **L**ocator (URL). When performing a query over

[*] adria@ecm.ub.es
[†] graciani@ecm.ub.es

an insecure connection, native python's *xmlrpclib* module is used. But, when the query is made over a secure connection, DISET relies on *pyOpenSSL* and *OpenSSL* to handle all cryptographic authentication algorithms, as seen in Fig. 1.

Furthermore DISET includes an authorization mechanism that controls access at the level of each method based on user groups, and provides the autheticated user information to the developer of the method handler in case a finer grain control is necesary.

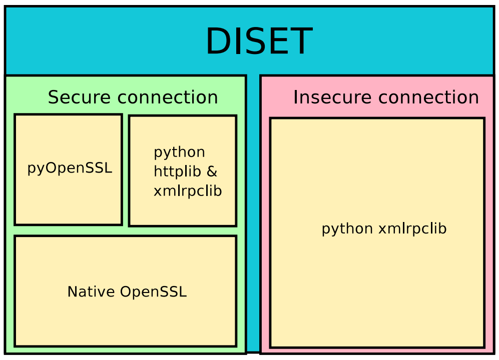

Figure 1: DISET dependencies

DISET third party dependencies are:

- *OpenSSL* is an open source full-featured toolkit implementing **S**ecure **S**ockets **L**ayer (SSL v2/3) and **T**ransport **L**ayer **S**ecurity (TLS). *OpenSSL* is able to handle authentication of X509 certificates, but is unable to authenticate grid proxies.

- *pyOpenSSL* is a python module encapsulating some of the native *OpenSSL* functionalities.

pyOpenSSL did not have all needed functionalities so some modifications were made:

- Some missing *OpenSSL* methods were added.

- An external authentication callback for *OpenSSL* was added. This new callback is able to recognize grid proxies and certificates. Calling a *pyOpenSSL* python method makes this callback the one *OpenSSL* will use to authenticate.

DISET usage

DISET provides DIRAC developers with a secure XML-RPC client and server framework. DISET client is used just as python native XML-RPC client. Furthemore, DISET client will establish a secure or an insecure connection based on the protocol (http/https/diset) specified in the given URL. Provided that a grid proxy and CA public keys and **C**ertification **R**evocation **L**ists (CRL) are available at predefined standard locations, no configuration is needed on the client. Appropriated configuration parameters are available to change these defaults.

DISET provides a framework to ease developing XML-RPC servers. Developers just need to code the server's request handler. Everything else is handled by DISET.

Server handler must inherit from *DISETRequestHandler* and automatically all methods beginning with *"export_"* will be accessible for clients.

Once the server handler is coded, it needs to be attached to a server object to be able to handle queries. DISET provides a secure (*DISETSecureServer*) and a insecure server object (*DISETInsecureServer*) both sharing the same interface. Making a server secure just depends on the server object used.

Authentication and authorization scheme

When establishing a connection using DISET, both sides must authenticate. To do so, DISET uses the SSL handshake provided by *OpenSSL*. The client sends it's certificate or proxy certificate to the server. If the certificate valid time has not expired, the cetificate is checked against all CA's the server knows until one CA verifies the certificate. Once a CA ackowledges the certificate, it's checked against that CA's CRL to see if it has been revoked by the CA. If it hasn't been revoked, the certificate gets verified and the user is authenticated. Then, the server sends it's certificate to the client. The certificate is checked against all CA's the client knows to authenticate the server, the CA's CRL and certificate time validity. When both sides get authenticated the actual communication can start. If any part fails to verify the other, the connection is closed.

The authentication mechanism is handled automatically by DISET.

In order to increase performance, DISET uses SSL sessions to reduce the number of SSL authentications needed. When a user gets authenticated for the first time, a SSL session is established so further queries by the same client will just use the previous session instead of having to get authenticated again. SSL sessions are handed internally by *OpenSSL* and they have a lifetime. Whenever the lifetime expires, the client needs to get authenticated again and another session is generated. SSL sessions lifetime is a *OpenSSL* parameter defined through the server configuration.

DISET has two levels of authorization:

- The first level of authorization allows or denies clients to call each method. This level of authorizations is based in groups. LHCb defines groups of users in the DIRAC **C**onfiguration **S**ervice (CS) and the server configuration defines which groups are authorized to execute each of the exported methods. A given user may belong to more than one group but when making a query the user must specify which group he wants to use for this call. If the user belongs to the specified group and this group is allowed to execute the requested method the DISET handler will call the appropriated method. Otherwise, the query will be denied returning a message to the user. This level of authorization is provided automatically by DISET and **every** query performed over a secure connection must pass it.

- The second level of authorization is finer grained but optional. The developer of the server may program further authorization inside the method making use of the authenticated user's DN and group that are provided by DISET.

DISET PORTALS

DISET provides a connection tunnelling mechanism. Instead of clients connecting directly to servers, a tunnelling server can be placed in the middle so the client can connect to it and its request tunnelled to the server. This tunnelling service is called DISET portals.

Final destination is resolved by the DISET portal making use of the path in the URL provided by the client. Different paths are assigned to different final servers. The portal connects to the final server and forwards the request, waits for the response and hands it back to the client as if it was the real server. Tunnelling proceeds as seen in Fig. 2.

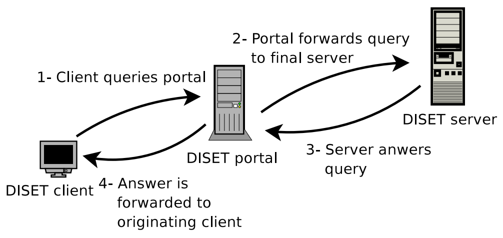
2- Portal forwards query to final server
1- Client queries portal
DISET server
DISET portal
3- Server anwers query
DISET client
4- Answer is forwarded to originating client

Figure 2: DISET portal

There are two types of DISET portals:

- **Secure portals** perform the authentication of the client and can thus tunnel connections to both secure and insecure servers. In the first case the portal authenticates itself with the server, and the full connection is secure, while in the second case the connection to the final server is not authenticated. Secure DISET portals are programmed in python.

- **Insecure portals** can only tunnel connections to insecure final servers because no authentication takes place. Insecure portals are also available in *PHP* using a web server.

For insecure connections DISET clients and servers do not know if there's a portal between them. The portal behaves as a server for the originating client and as a client for the final server. For secure connections, the DISET client will not see any difference between connecting to a server directly or via a portal. On the other hand, DISET secure servers need a list of trusted portals. When tunnelling the connection the portal presents its own certificate to the final server and, once authenticated it will trust the client identification done by the portal. From then on, the authorization takes place at the server with the forwarded user's DN and group presented by the portal.

Advantages of DISET Portals

DISET portals are single points of entry for all services. So just one point of access must be authorized to receive incoming connections.

Portals can be used to achieve load balancing. Each query can be tunnelled to a different server using simple balancing algorithms.

When using secure connections, the number of SSL authentications can be reduced by using SSL sessions intensively. By placing a DISET portal, final servers will only receive connections from portals so the number of queries per session will be drastically increased. Originating clients will only connect to portals so, instead of having to do one handshake per server, just one will be made for all servers tunneled through the same portal. The overall number of SSL authentications is minimized by placing a DISET secure portal.

DISET TRANSFERS

DISET includes a fast and easy to use mechanism to transfer files over secure and insecure channels. Transfer petitions use standard XML-RPC protocol to send transfer and authentication data, and a binary protocol to send file data and transfer confirmation, all over the same connection. XML-RPC transfer request and server response are handled as other DISET methods but, instead of closing the connection, the socket remains open and binary transfer takes place. Changing protocols is handled internally by DISET so the developer only has to program request handler's logic. When transferring through a secure connection the same authentication and authorization mechanisms as other DISET methods can be applied.

DISET servers can serve both XML-RPC queries and transfer queries. Server developers just need to code some special callbacks to allow file transfers. All the underlying protocols and data handling is done by DISET.

SUMMARY AND OUTLOOK

DISET provides an easy to use framework for developing grid enabled secure services using XML-RPC. It also eases deployment of services by minimizing dependencies to just *pyOpenSSL* and *OpenSSL*, and they can be shipped together with the service.

Developers just need to code servers logic. Connections are handled by the DISET server objects. Turning a insecure interface into a secure one depends on the DISET server object used. And by placing a DISET portal multiple servers can be accessed from just one host, that tunels the connection to the final server.

DISET also enhances XML-RPC capabilities by being able to transfer files over the XML-RPC connection in binary format.

Currently, work is on going to make use of voms-proxies which, apart from the user DN, include a user requested group and role signed by an authorized voms server. These groups are going to be matched to the existing groups currently defined in the DIRAC Configuration Service, thus making the system more dynamic.

REFERENCES

[1] Tsaregorodtsev, A. et al., DIRAC, the LHCb Data Production and Distributed Analysis system, CHEP06, Mumbai, India.

[2] Paterson, S. and Tsaregorodtsev, A., DIRAC Infrastructure for Distributed Analysis, CHEP06, Mumbai, India.