EUROPEAN MIDDLEWARE INITIATIVE

Common Authentication Library - C Part: Usage examples

CESNET caNl Team

EMI is partially funded by the European Commission under Grant Agreement RI-261611

Main Features

- Largely based on existing code L&B, VOMS
- Core API provides basic support for
  - communication mgmt channel establishment & message protection
  - Generic with no SSL/X.509 dependency
  - Easier portability outside PKI world

Main Features II

- Support for X.509 and SSL provided by another layer of API
  - separate header file
  - Setting SSL specifics for connections
  - Certificate and proxy mgmt
  - generating CSRs, signing proxies,
  - ...

Examples

- Sources of examples are available in cvs repository at http://glite.cvs.cern.ch/cgi-bin/glite.cgi/e...
- Built packages of examples in EMI 2.
- Sample client, server – use Main API.
- Delegation and New Proxy Initialization – Main + Certificate API.

Context

- The Main API uses contexts of types *canl_ctx* and *canl_io_handler*.
- **canl_ctx** is global context and must be initialized before other API calls.
- **canl_io_handler** holds information about each particular connection. All created contexts of this type are independent on each other.

Error Codes and Messages

- The return type of most of the API functions is **canl_err_code**.
- Unless specified otherwise, zero return value means success, non-zero failure.
- **canl_ctx** stores details of all errors which has occurred since context initialization, in human readable format.
- To obtain error description one may use **canl_get_error_message()**:

```
printf("%s\n", canl_get_error_message (my_ctx));
```

C-S Connection – Client

- **Include necessary header file:**

  # include <canl.h>

- **Initialize context and set parameters:**

  canl_ctx my_ctx;

  canl_io_handler my_io_h;

  my_ctx = canl_create_ctx( );

  canl_create_io_handler(my_ctx, &my_io_h);

  canl_ctx_set_ssl_cred(my_ctx, cert, key, NULL, NULL);

C-S Connection – Client II

- **Connect to the server:**

  canl_io_connect(my_ctx, my_io_h, p_server, NULL, port, NULL, 0, &timeout);

- **Send something then read the response**

  n_bytes = canl_io_write(my_ctx, my_io_h, buf, buf_len, &timeout);

  n_bytes = canl_io_read(my_ctx, my_io_h, buf, sizeof(buf), &timeout);

C-S Connection – Client III

- **Free the allocated memory:**

  canl_io_destroy(my_ctx, my_io_h);

  canl_free_ctx(my_ctx);

C-S Connection – Server

- The server side is pretty similar to the client except **canl_io_accept()** function is used instead of **canl_io_connect()**:

  canl_io_accept(canl_ctx cc, canl_io_handler io, int fd, struct sockaddr s_addr, int flags, canl_principal *peer, struct timeval *timeout)

- The server also needs to create and manage connection sockets (i.e. socket(), bind(), listen() functions).

“grid-proxy-init” Example I

- **Include necessary header files:**

  # include <canl.h>

  # include <canl_cred.h>

- **caNl contexts variables:**

  canl_cred signer = NULL ;

  canl_cred proxy = NULL ;

  canl_ctx ctx = NULL ;

“grid-proxy-init” Example II

- **Initialize context:**

  ctx = canl_create_ctx();

  canl_cred_new(ctx, &proxy);

- **Create a certificate request with a new key-pair.**

  canl_cred_new_req (ctx, proxy, bits);

- **(Optional) Set cert. creation parameters**

  canl_cred_set_lifetime(ctx, proxy, lifetime);

  canl_cred_set_cert _type(ctx, proxy, CANL_RFC);

  canl_cred_set_extension(ctx, proxy, x509_ext);

“grid-proxy-init” Example III

- **Load the signing credentials**

  canl_cred_new(ctx, &signer);

  canl_cred_load_cert_file(ctx, signer, user_cert);

  canl_cred_load_priv_key_file(ctx, signer, user_key, NULL, NULL);

- **Create the new proxy certificate**

  canl_cred_sign_proxy(ctx, signer, proxy);

- **And store it in a file**

  canl_cred_save_proxyfile(ctx, proxy, output);

“grid-proxy-init” Example IV

- **Free the allocated memory:**

  canl_cred_free (ctx, signer);

  canl_cred_free(ctx, proxy);

  canl_free_ctx (ctx);

Credentials Delegation

- Compared to example of proxy initialization, delegation example may also find useful these functions to save requests and certificates into proper openssl structures:

  canl_cred_get_req(canl_ctx, canl_cred, X509_REQ **)

  canl_cred_get_cert(canl_ctx, canl_cred, X509 **)

  canl_cred_get_chain(canl_ctx, canl_cred, STACK_OF(X509) **)

- These functions has **load** counterparts:

  canl_cred_set_req(canl_ctx, canl_cred, const X509_REQ *)

  ...

  ...

Possible utilization in VOMS

- AC acquisition
  - “normal” SSL/TLS connection between the client and the server
- AC manipulating
  - Generating digital signature (VOMS server)
  - Checking VOMS signatures (RPs)
    - API to be provided

More Information

- API reference and examples: CANL C Developer's Guide available at http://egee.cesnet.cz/cvsweb/S...

Thank you for your attention.