The Development of LDAP-based Data Storage Library in the Common Data Security Architecture

Along Lin, Richard Brown
HP Laboratories Bristol
HPL-1999-125
15th October, 1999*

LDAP, CDSA, secure data storage library

In this paper, we discuss the issues on developing Data Storage Library (DL) based on Lightweight Directory Access Protocol (LDAP) for CDSA 1.2 on HP-UX 11.0. We first introduce CDSA 1.2 DL interface specification and some representative LDAP APIs, and then point out the issues on developing an LDAP-based DL. Finally, some suggestions are given.

*Internal Accession Date Only

The Development of LDAP-based Data Storage Library in the Common Data Security Architecture

Along Lin

Trusted E-Services Group
Hewlett-Packard Laboratories
Filton Road, Stoke Gifford
Bristol BS34 8QZ, U.K.
Email: alin@hplb.hpl.hp.com

Abstract

In this paper, we discuss the issues on developing Data Storage Library (DL) based on Lightweight Directory Access Protocol (LDAP) for CDSA 1.2 on HP-UX 11.0. We first introduce CDSA 1.2 DL interface specification and some representative LDAP APIs, and then analyze the issues on developing an LDAP-based DL. Finally, some suggestions are given

1. Motivations

With the support of network technologies including Java, Web, CORBA and mobile code, more network applications such as electronic commerce over the Internet have been developed. One major concern is security and privacy. Among the proposed security solutions, Intel's Common Data Security Architecture (CDSA) is a flexible framework allowing software/hardware vendors to provide add-in security modules. It supports a complete set of security services and interoperability between security applications developed on different platforms.

CDSA is an extensible, standards-based, industry-accepted framework. Since the Open Group adopted CDSA 2.0 specification in December 1997, it has been widely supported. In July 1999, HP released a CDSA 1.2 product. As part of our on-going research work, we developed a Data Storage Library (DL) based on the CDSA 1.2 and Netscape Directory Server v4 for HP-UX 11.0.

In the following, we introduce CDSA 1.2 DL interface specification first, and then analyze the issues on developing an LDAP-based DL. Finally, some suggestions for given.

2. Data Storage Library Interface

The primary purpose of a DL is to provide access to certificates, certificate revocation lists (CRLs), keys, policies and other security-related objects in a persistent data store by translating calls from the DL interface into the native interface of the data store. Stable data storage can be provided by one of the followings: commercially available database management system product, directory service, custom hardware-based storage device or native file system.

The data storage library Service Provider Interface (SPI) defines four categories of operations.

- Data storage library operation DL_Authenticate(), which is used to control access to the data storage library. A user may be required to present valid credentials to the data storage library prior to accessing any of the data stores embedded in the DL module.
- Data store operations, which operate on a data store as a single unit. These operations include opening, closing, creating, deleting, importing and exporting data stores.
- Data record operations, which operate on a single record of a data store. They include adding new data objects, deleting data objects, and retrieving data objects based on application-provided selection criteria.
- DL_PassThrough(), which is used to allow data store libraries to expose additional services beyond what is currently defined in the CSSM API.

3. LDAP-based DL Development Analysis

3.1 DL_Authenticate

```
CSSM_RETURN CSSMDLI DL_Authenticate
                (const CSSM_DL_DB_HANDLE DLDBHandle,
                 const CSSM_DB_ACCESS_TYPE_PTR AccessRequest,
                 const CSSM_USER_AUTHENTICATION_PTR UserAuthentication)
```

This function allows the caller to provide authentication credentials to the DL module at a time other than data store creation, deletion, open, import and export. AccessRequest defines the type of access to be associated with the caller. If the authentication credential

applies to access and use of a DL module in general, then the data store handle specified in the DLDBHandle must be NULL. When the authorization credential is to apply to a specific data store, the handle for that data store must be specified in the DLDBHandle pair.

Because a data store is implemented using directory service, after a user has been successfully connected to an LDAP server, access to it may require some authentication but the access mode is inappropriate for a data store.

3.2 DL_DbOpen

```
CSSM_DB_HANDLE CSSMDLI DL_DbOpen
                (CSSM_DL_HANDLE DLHandle,
                 const char *DbName,                    // The host name of the LDAP server
                 const CSSM_DB_ACCESS_TYPE_PTR AccessRequest,
                 const CSSM_USER_AUTHENTICATION_PTR UserAuthentication,
                 const void *OpenParameters)            // The port number of the server
```

The function opens the data store with the specified logical name under the specified access mode. If user authentication credentials are required, they must be provided. For data store opening/closing functions, a data store name is the server name associated with the corresponding directory. However, for data store deletion, it is the distinguished name for a data store to be deleted.

As pointed out in 3.1, access type may be inappropriate for an LDAP-based data store.

3.3 DL_DbClose

```
CSSM_RETURN CSSMDLI DL_DbClose( CSSM_DL_DB_HANDLE DLDBHandle )
```

This function closes an open data store. After the user is unbound from the LDAP server, any access to it will not be allowed.

3.4 DL_DbDelete

```
CSSM_RETURN CSSMDLI DL_DbDelete
                (CSSM_DL_HANDLE DLHandle,
                 const char *DbName,
                 const CSSM_USER_AUTHENTIACTION_PTR UserAuthentication)
```

This function deletes all records from the specified data store and removes all state information associated with that data store. Please be aware that the data store name in an LDAP-based DL is the distinguished name of an entry in a directory.

3.5 DL_DataInsert

```
CSSM_DB_UNIQUE_RECORD_PTR CSSMDLI DL_DataInsert
                (CSSM_DL_DB_HANDLE DLDBHandle,
                 CSSM_DB_RECORDTYPE RecordType,
                 const CSSM_DB_RECORD_ATTRIBUTE_DATA_PTR Attributes,
                 const CSSM_DATA_PTR Data)
```

This function creates a new persistent data record of the specified type by inserting it into the specified data store. The values contained in the new data record are specified by the *Attributes* and the *Data*. The attribute value list contains zero or more attribute values. The *Data* is an opaque object to be stored in the new record.

In an LDAP-based DL, because the low level details of LDAP implementation are unavailable, the unique identifier for a record is represented by its distinguished name in a directory.

3.6 DL_DataDelete

```
CSSM_RETURN CSSMDLI DL_DataDelete
                (CSSM_DL_DB_HANDLE DLDBHandle,
                 CSSM_DB_RECORDTYPE RecordType,
                 const CSSM_DB_UNIQUE_RECORD_PTR UniqueRecordIdentifier)
```

This function removes the data record specified by the unique record identifier from the specified data store. In an LDAP-based DL, the distinguished name of an entry is used to replace the unique record identifier.

3.7 DL_DataGetFirst

```
CSSM_DB_UNIQUE_RECORD_PTR CSSMDLI DL_DataGetFirst
                (CSSM_DL_DB_HANDLE DLDBHandle,
                 const CSSM_QUERY_PTR Query,
                 CSSM_HANDLE_PTR ResultsHandle,
                 CSSM_BOOL *EndOfDataStore,
                 CSSM_DB_RECORD_ATTRIBUTE_DATA_PTR Attributes,
                 CSSM_DATA_PTR Data)
```

This function retrieves the first data record in the data store that matches the selection criteria. The selection criteria (including selection predicate and comparison values) are specified in the Query structure. This function returns the first record satisfying the query in the list of Attributes and the opaque Data object. It also returns a results handle to be used when retrieving subsequent records satisfying the query. Finally, this function

returns a unique record identifier associated with the retrieved record. This structure can be used in future references to the retrieved data record. If no records satisfied the query, the EndOfdataStore flag will be CSSM_TRUE and the function will return NULL.

However, an LDAP-based search is based on the base distinguished name of the entry that serves as the starting point of the search. It is difficult to get the information from the current API specification.

3.8 DL_DataGetNext

```
CSSM_DB_UNIQUE_RECORD_PTR CSSMDLI DL_DataGetNext
                (CSSM_DL_DB_HANDLE DLDBHandle,
                 CSSM_HANDLE ResultsHandle,
                 CSSM_BOOL *EndOfDataStore,
                 CSSM_DB_RECORD_ATTRIBUTE_DATA_PTR Attributes,
                 CSSM_DATA_PTR Data)
```

This function returns the next data record referenced by the ResultsHandle. The ResultsHandle references a set of records selected by an invocation of the DL_DataGetFirst function. The record values are returned in the Attributes and Data parameters. The function also returns a unique record identifier for the returned record. If no other records satisfied the query, the EndOfdataStore flag will be CSSM_TRUE and the function will return NULL.

3.9 DL_PassThrough

This function allows applications to call data storage library module-specific operations that have been exported. Such operations may include queries or services that are specific to the domain represented by a DL module.

We implemented about 20 operations specific to LDAP-based DL.

3.10 Unsupported APIs in LDAP_based DL

3.10.1 DL_DbCreate

In LDAP-based DL, we assume that a directory has been created. Entries of different schemas can be added into the directory without having to create a data store later. Therefore, this function is not supported.

3.10.2 DL_DbImport
3.10.3 DL_DbExport

The above two functions are not directly supported by LDAP APIs even though they are supported by Netscape Console.

3.10.4 DL_DbSetRecordParsingFunctions
3.10.5 DL_DbGetRecordParsingFunctions

Because the details on LDAP service are not available, the above two functions are not supported.

3.10.6 DL_GetDbNameFromHandle

Because the semantics of a data store name is inconsistent in LDAP-based DL, this function is not supported.

3.10.7 DL_DataAbortQuery
3.10.8 DL_FreeUniqueRecord

4. Netscape LDAP v4 API Specification

In the following, a set of representative LDAP APIs is given. They represent the main categories of LDAP APIs. If an LDAP API has two forms, only its synchronous version is given. For simplicity, LDAPControl parameters can be ignored and assigned NULL.

4.1 ldap_init

LDAP *ldap_init (const char *host, int port)

Before you can connect to an LDAP server, you need to initialize an LDAP session. If we plan to connect to the LDAP server over a Secure Sockets Layer (SSL), the procedure for initializing an LDAP session differs. The returned connection handle will be needed to all LDAP API functions for connecting, authenticating, and performing LDAP operations on a server.

The parameters host and port specify the host name and port number of the LDAP server.

4.2 ldap_simple_bind_s

```
int ldap_simple_bind_s(  LDAP *ld,
                         const char *who,
                         const char *password)
```

LDAP v2 severs typically require clients to bind before any operations can be performed. The purpose is to provide information about the client's LDAP version, authentication method and the client's credentials to be used for authentication.

4.3 ldap_unbind_s

```
int ldap_unbind_s( LDAP *ld )
```

When all necessary LDAP operations are performed, the connection to the LDAP server needs to be closed.

4.4 ldap_search_ext_s

```
int ldap_search_ext_s(
        LDAP *ld,
        const char *base,
        int scope,
        const char *filter,
        char **attrs,
        int attrsonly,
        LDAPControl **serverctrls,
        LDAPControl **clientctrls,
        struct timeval *timeoutp,
        int sizelimit,
        LDAPMessage **res )
```

This function sends an LDAP search request to the server. Here, DN means Distinguished Name.

base specifies the starting point in the directory, or the base DN (an entry where to start searching).
scope specifies the scope of the search. You can narrow the scope of the search to the base DN, entries at one level under the base DN, or entries at all levels under the base DN.
filter specifies a search filter.
attrs and *attrsonly* specify the type of information that you want to return (which attributes you want to retrieve) and whether you want to retrieve only the attribute type or the attribute type and its values.
serverctrls and *clientctrls* specify the LDAP v3 controls associated with this search operation.
timeoutp and *sizelimit* specify the search constraints that you want applied to the search.
res is the returned result.

4.5 ldap_add_ext_s

```
int ldap_add_ext_s(
        LDAP *ld,
        const char *dn,
        LDAPMod **mods
        LDAPControl **serverctrls,
        LDAPControl **clientctrls)
```

dn specifies the DN for the new entry. For example, dn=”uid=alin, ou=people, o=hpl.hp.com”.
mods specifies an array of LDAPMod structures to represent the attributes in the entry. Each LDAPMod structure is used to specify the name and values of each attribute, which is described as follows.

```
typedef struct ldapmod {
    int mod_op;
    char *mod_type;
    union {
        char **modv_strvals;
        struct berval **modv_bvals;
    } mod_vals;
#define mod_values mod_vals.modv_strvals
#define mod_bvalues mod_vals.modv_bvals
} LDAPMod;
```

mod_op specifies the operation to be performed on the attributes and the types of data specified as the attributes values. It can be one of the following values:

- LDAP_MOD_ADD adds a value to the attribute.
- LDAP_MOD_DELETE removes the value from the attribute.
- LDAP_MOD_REPLACE replaces all existing values of the attribute.

mod_type specifies the attribute type you want to add, delete, or replace the values of.
mod_values is a pointer to a NULL-terminated array of string values for the attribute.
mod_bvalues is a pointer to a NULL-terminated array of *berval* structures for the attribute.

4.6 ldap_modify_ext_s

```
int ldap_modify_ext_s(
        LDAP *ld,
        const char *dn,
        LDAPMod **mods,
        LDAPControl **serverctrls,
        LDAPControl **clientctrls)
```

dn specifies the DN for the entry to be modified.
mods specifies an array of LDAPMod structures to represent the attributes and values that you are adding, replacing, or removing.

4.7 ldap_delete_ext_s

```
int ldap_delete_ext_s(
        LDAP *ld,
        const char *dn,
        LDAPControl **serverctrls,
        LDAPControl **clientctrls)
```

dn specifies the DN for the entry to be deleted.

5. Problems with Developing DL APIs in CDSA 1.2

After developing LDAP-based DL, we realize there are some problems with developing an LDAP-based DL.

- The unique record identifier of a data store record is inappropriate for LDAP-based DL. This is because it is hard for a DL developer to obtain the information about the LDAP implementation. As a user of LDAP directory service, LDAP-based DL developer does not know about the indexing details on entries in a directory.
- If a data store is built as the server associated with a directory, it is hard to implement DL APIs for data store creation and deletion. However, if we assume a directory has been built and a data store is a branch of the directory, it is difficult to implement DL APIs for data store opening and closing based on LDAP directory service. Therefore, the semantics of a data store name is inconsistent when all of these data store operations need to be implemented. The ideal solution is that a data store is implemented as a directory. However, there do not exist LDAP APIs for creating/deleting/opening/closing the directory.
- Because the operations on entries in a directory are based on LDAP APIs, it is both bizarre and inefficient to map the standard data structures for DL APIs to the parameters to LDAP APIs, which are usually string-based. Generally speaking, LDAP APIs are much easier to use than DL APIs.
- The LDAP search is based on both the Distinguished Name (DN) of the starting point and the scope of the search in the directory. However, DL search API contains neither the base DN nor the scope of the search. The information about these parameters has to be included in the values of other parameters to the corresponding DL API, which is rather bizarre.

- Because data in an LDAP directory is organized hierarchically and only end entries can be deleted, it is expensive to remove a branch of the directory by recursively deleting all of its entry nodes.
- LDAP directory service supports multiple values for an entry attribute, which is not allowed in standard DL API. That is, if a user wants to retrieve some entries having multiple attribute values, the user will be unable to do that based on the current standard CDSA DL APIs.

6. Conclusion

It is a little bizarre to develop an LDAP-based DL. There are several issues. First, the mapping from a data store to a directory is not easy. Second, the format and semantics of standard DL APIs are based on low-level implementation details whereas LDAP directory service is at high level. Third, the main purpose of an LDAP directory service is for retrieving information on entries whereas DL APIs are for more general purpose. Because there is little difference between CDSA 1.2 DL and CDSA 2.0 DL, these issues remain same when developing Netscape directory server based DL.

One benefit of an LDAP-based DL is that certificate-based authentication is provided in Netscape directory server v4. Because an LDAP-based DL can retrieve entries efficiently and support distributed data stores, it is ideal for storing digital certificates and policies.

To support the DL development using directory service, more LDAP APIs will be needed to implement data store operations such as creation/deletion and opening/closing.

In our current prototype implementation, we assume the following points.

- A directory has been established.
- A data store is associated with either the directory server or the DN of an entry in a directory depending on which DL API is dealt with.
- The DN of an entry is mapped to the unique record identifier in CDSA 1.2 DL specification.
- The record types in a data store are mapped to the attribute values by introducing an extra attribute of *recordType* into the schema in directory service.
- Searching in LDAP-based DL is performed through the whole branch in a directory rooted by a DN.

References

[1] *Common Data Security Architecture Specification*, Intel Architecture Labs, 1997.
[2] *CSSM Application Programming Interface*, Intel Architecture Labs, 1997.
[3] *CSSM Data Storage Library Interface Specification*, Intel Architecture Labs, 1997.