Sample Workload Distribution Code for Luna 7 FM

APPLICATION GUIDE

Contents

Description

This document is an application guide for sample code that demonstrates a load-balancing model that can work with FMs across multiple physical HSM devices.

Audience

This document is targeted at application developers that have implemented a Luna FM (Functionality Module) and now want to implement load-balancing logic in their client side application. This logic should allow for the distribution of FM calls to multiple HSM devices per some pre-configuration, manage those devices and initiate fail-over in the event that an HSM becomes unavailable. Note that for this document, the terms slot and partition are analogous as well as the terms HSM device and adapter.

The Current Luna 7 High Availability Feature

The current Luna 7 client library provides a transparent high availability (HA) feature that allows for the distribution of PKCS 11 function calls across multiple Luna HSM devices. The HA logic maintains a table of virtual session and object handles that map to the handles on the physical devices. The physical handles are substituted for the virtual handles in the function call once the device is selected to perform the operation. It is not possible however, to apply this logic to functional modules (FMs) as the API is proprietary, however a list of HSM adapters, and their current active/failure status, can be managed in the application and FM functions calls distributed to these devices in a simple round-robin fashion.

The Workload Distribution (WLD) Model

Overview

The WLD model provides basic load balancing of “MD” calls to multiple FMs. It also manages a list of available HSM devices and based upon their availability, designates them as either being active or non-active. It should be clear however, that this implementation does not provide a totally transparent and automated HA capability as that which exists in the Luna client software. There is a requirement for the calling functions to cooperate with the underlying logic to take full advantage of the load balancing and failover capabilities. This will be outlined in the next two sections.

There are two modes of the WLD model, depending on the specific FM use case: 1) there is some interaction with internal HSM firmware, either by the client side application, or within the FM itself, or 2) all crypto functions and key management/storage functions are performed in the FM.

Goals and Limitations

One goal of the WLD capability is that it should work with all existing HSM configuration options including High Availability. If HA is configured however, the following is required:

1. The physical PKCS#11 slots must be visible to the client (no HA_Only).
2. The HA virtual slot cannot appear in the slot list.

To the extent that WLD is not a fully transparent HA, the developer should be aware of the following:

1. Object handles are only valid within the session that they are retrieved (i.e. C_FindObject). This is because the session is bound to the physical HSM and the object handle, even for the same key, may be different on different devices. The Sample Pseudo-code section illustrates how to structure code to best manage this limitation.
2. Although the WLD logic will examine the return-code for the message sent to the FM, and invalidate any device that is not responding, this code will be returned to the calling function in order for the transaction or operation to be replayed.

API

WLD_SLOT_LIST

The WLD_SLOT_LIST environment variable defines a list of PKCS11 slots that in turn identifies the HSMs or adapters upon which they exist. The slot list may be overridden with a new list passed in via the InitializeWLD function call. A slot list must always exist even if the FM does not access a PKCS 11 object as part of its function. This is because an HSM device only exists for the client in the context of a partition that has been assigned to that client. Thus, at least one HSM partition (or slot) must be created on each HSM device (or adapter) that will execute the FM code.

WLD_RV InitializeWLD(uint32 *pSlotList, uint32 numSlots);

Use this function to initialize the WLD capability. An optional slot list that replaces the one defined by the WLD_SLOT_LIST environment variable may be passed in with this function. The *pSlotList pointer should be set to NULL if a new slot list is not required.

WLD_RV GetWLDSlotID(uint32_t *pSlotID, uint32_t *pEmbeddedSlotID);

Use this function to obtain a WLD slot from the configured slot list. The slot will be chosen in a round-robin least busy fashion from the slot list, and will be bound to an FM installed on the same physical HSM adapter. The application can then open an authenticated session in order to perform either a client side or FM PKCS#11 operation. If this is being done, remember to pass the embedded slot ID to the FM via the FM command block.

MD_RV SendWLDMessageToFM(uint32 slotID,
uint16 fmNumber,
MD_Buffer_t *pReq,
uint32 timeout,
MD_Buffer_t *pResp,
uint32 *pReceivedLen,
uint32 *pFMStatus);

This function is essentially a wrapper around the MD_SendReceive FM message dispatch function. If an error occurs at the MD level however, the adapter, and any PKCS#11 slots associated with this physical device will be marked as “unavailable”. At this time, the sample code does not include any automatic recovery of a failed device. If the failure has been rectified, it will be necessary to re-execute the InitializeWLD command, although an application restart may be advisable.

It is important to note that it is not necessary to pass in a valid slotID if the application does not care which HSM adapter is used to perform the function. That is, the FM does not require access to an HSM (or PKCS 11) object. If this is the case, the application should set the slotID parameter to WLD_NO_SLOT_ID. Moreover, it is not necessary to call the GetWLDSlotID function, as the SendWLDMessageToFM function will select the next best available slot.

Sample Pseudo Code

MODE 1 – this sample addresses the scenario where the FM uses a PKCS#11 (HSM firmware) object to decrypt some data being passed into the FM. The application must first find the object handle (from the label) and then will pass the handle to the FM. It is assumed that the InitializeWLD() function has been called previously.

```
APP_ERR PerformFMFunction(....)
{
        APP_ERR rc;
        WLD_ERR wldErr;
        BOOL done = false;
        CK_SLOT_ID slotID = NO_SLOT_ID, embeddedSlotID;
        CK_OBJECT_HANDLE hObject;
        CK_SESSION_HANDLE hSession;

        while (!done)
        {
                if (GetWLDSlotID(&slotID, &embeddedSlotID) != WLDR_OK)
                {
                        // There are no slots available - this error is
                        // catastrophic - set the APP_ERR error appropriately
                        // and return
                        rc = APR_WLD_FAILURE;
                        break;
                }

                // Call a function to open a session and login
                OpenPKCS11SessionAndLogin(slotID, ..., &hSession);

                // Wrapper function for the PKCS#11 C_FindObjectInit/C_FindObject functions
                // from the key label
                rc = FindObject(hSession, "MyKey", &hObject);
                if (rc == APR_NO_KEY)
                        break;

                // Build the FM command block – include the embedded slot ID as this may
                // be required by the FM if accessing the PKCS11 partition
                rc = BuildFMCommandBlock(..., embeddedSlotID, hObject...);
                if (rc != APR_OK)
                        break;

                wldErr = SendWLDMessageToFM(slotID,  // NOT embeddedSlotID
                        fmNumber,
                        pReq,
                        0,                      // timeout - 0 is no limit
                        pResp,
                        pRecvLen,
                        &FMErrCode);

                if (wldErr == WLDR_MD_ERROR)   // Not done - loop and try again on another slotID
                        continue;

                (void)C_CloseSession(hSession);
                done = true;
        }

        (void) C_CloseSession(hSession);


        return rc;
}
```

MODE 2 – In this scenario, all functionality is done in the FM and no interaction with the HSM firmware is required. Note that in this mode, the next best available slot will be selected in the SendWLDMessageToFM() function call and used once. If multiple FM calls need to made to the same adapter as part of a single transaction however, then the application should call the GetWLDSlotID() function (as in the previous mode) to obtain a dedicated slot and pass this ID to the SendWLDMessageToFM() function. Note also that because any adapter would be suitable to perform this transaction, the WLD logic will automatically retry the command on another adapter in the event of an MD failure on the MD_SendReceive command.

```
APP_ERR PerformFMFunction(....)
{
        APP_ERR rc;

        rc = BuildFMCommandBlock(..., embeddedSlotID, hObject...);
        if (rc != APR_OK)
                break;

        wldErr = SendWLDMessageToFM(WLD_NO_SLOT_ID,  // No PKCS#11 slot associated with this call
                fmNumber,
                pReq,
                0,                      // timeout - 0 is no limit
                pResp,
                pRecvLen,
                &FMErrCode);

        // The logic will automatically retry on another adapter if the first
        // selected results in an MD error.  The following error will be returned
        // if there are no adapters available.
        if (wldErr == WLDR_NO_ADAPTER)
        {
                rc = APR_NO_ADAPATER;
        }
        else
        {
                rc = ProcessFMReplyBlock(pResp, FMErrCode, ...);
        }

        return rc;
}
```

Sample Source

Overview

The FM_WLDSample package contains source that implements an application (main.c) and wld functions (wld.c/.h) as described previously in this document. As well, a sample FM has been included that works with the sample client application. The package also contains makefiles for both the FM and application for the linux OS.

NOTE that this code is SAMPLE ONLY and Thales, Inc. assumes no responsibility for its incorporation and use in the user's application. Refer to the included README file for specific details concerning the version(s) of the Luna client and FM SDK that this sample has been built and tested on.

Package Components

FM_WLDSample/LunaFM_HA_ApplicationGuide.pdf – this guide

FM_WLDSample/README.txt - current release details

FM_WLDSample/fm/(startup.c, hdr.c, makefile) - folder contains source to build sample fm "wldsample.bin"

FM_WLDSample/wld/main.c - sample application source file

FM_WLDSample/wld/wld.c - sample source for wld functions (as described above)

FM_WLDSample/wld/makefile - linux makefile to build sample application

FM_WLDSample/include/wld.h - header application containing common WLD definitions and function declarations.

Installation

The FM_WLD_Sample.tar may be installed in any working directory on a linux system. The makefiles expect that the luna client and FM SDK are installed at /usr/safenet/lunaclient and /usr/safenet/lunafmsdk respectively. In order to run the wldapp application, first sign and install the wldsample.bin FM file as per the instructions in the Luna online documentation. Next, create at least one partition on each HSM adapter that should be included in the WLD group. Export the WLD_SLOT_LIST environment variable that lists those partitions to be included.