PB173 - Tématický vývoj aplikací v C/C++ (podzim 2013)

Skupina: Aplikovaná kryptografie a bezpečné programování

https://is.muni.cz/auth/el/1433/podzim2013/PB173/index.qwarp?fakulta=1433;obdobi=5983;predmet=734514;prejit=2957738;

Petr Švenda svenda@fi.muni.cz

Konzultace: G.201, Úterý 13-13:50

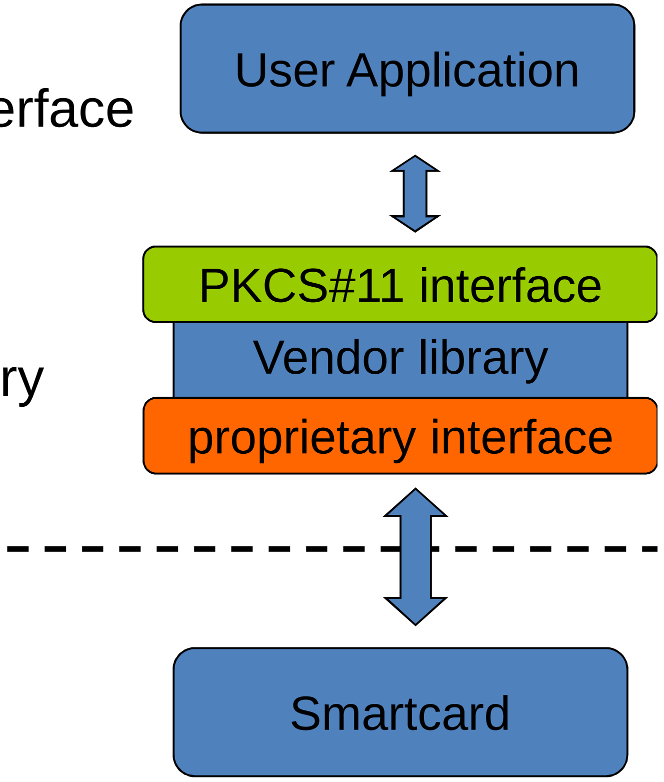
CROCS
Centre for Research on Cryptography and Security

Dynamic library usage

- Static linking
  - *library*.lib added to dependencies
  - (you already know that)
- Run-time dynamic linking
  - controllable run-time search for dynamic library
  - developer can control and respond on (un)available lib
  - LoadLibrary(*path*) & FreeLibrary(*hLib*)
- Run-time search for specific function
  - GetProcAddress(*hLib, “function_name”*)
  - cast to target function prototype (later)

PKCS#11

- Standardized interface of security-related functions
  - vendor-specific library in OS, often paid
  - communication library->card proprietary interface
- Functionality cover
  - slot and token management
  - session management
  - management of objects in smartcard memory
  - encryption/decryption functions
  - message digest
  - creation/verification of digital signature
  - random number generation
  - PIN management
  - lots of functions actually in software only ☹
- Secure channel not possible!
  - developer can control only App->PKCS#11 lib



PKCS#11 library

- API defined in PKCS#11 specification
  - http://www.rsa.com/rsalabs/node.asp?id=2133
  - functions with prefix ‘C_’ (e.g., C_EncryptFinal())
  - header files pkcs11.h and pkcs11_ft.h
- Usually in the form of dynamically linked library
  - cryptoki.dll, opensc-pkcs11.dll, dkck232.dll…
  - different filenames, same API functions (PKCS#11)
- We will use virtual token with storage in file
  - VirtPKCS11.dll, *disk_name*:\VirtToken.vtk
  - suitable for easy testing (no need for hardware reader)

Function prototypes

- GetProcAddress() returns untyped function pointer
- We need to cast this function pointer to known function type
- Function types for PKCS#11 are in pkcs11_ft.h

```
typedef CK_RV CK_ENTRY (*FT_C_Encrypt)(
  CK_SESSION_HANDLE hSession,
  CK_BYTE_PTR       pData,
  CK_ULONG          ulDataLen,
  CK_BYTE_PTR       pEncryptedData,
  CK_ULONG_PTR      pulEncryptedDataLen
);
```

PKCS#11 role model

- Functions for token initialization
  - outside scope of the specification
  - usually implemented (proprietary function call), but erase all data on token
- Public part of token
  - data accessible without login by PIN
- Private part of token
  - data visible/accessible only when PIN is entered

Load and init PKCS#11 library

```
int LoadAndInitLibrary(const char* path, HINSTANCE* phLib) {
    CK_RV   status = CKR_OK;
    FT_C_Initialize fInitialize = NULL;

       if (phLib) {
        if ((*phLib = LoadLibrary(path)) != NULL) {
            // INITIALIZE LIBRARY
            fInitialize = NULL;
            if ((fInitialize = (FT_C_Initialize) GetProcAddress(*phLib, "C_Initialize")) != NULL) {
                (fInitialize)(NULL);
            }
            else status = GetLastError();
        }
        else status = GetLastError();
    }
    else status = -1;

    return status;
}
```

CROCS

Finalize and unload PKCS#11 library

```
int FinalizeAndCloseLibrary(HINSTANCE hLib) {
    CK_RV    status = CKR_OK;
    FT_C_Finalize   fFinalize;
       if (hLib != NULL) {
        // UNINITIALIZE LIBRARY
        fFinalize = NULL;
        if ((fFinalize = (FT_C_Finalize) GetProcAddress(hLib, "C_Finalize")) != NULL) {
            (fFinalize)(NULL);
        }

        FreeLibrary(hLib);
    }
    else status = -1;

    return status;
}
```

List tokens in system

- Slots in system are equivalent to readers
  - C_GetSlotList
  - C_GetSlotInfo
- Slot can be empty or with inserted token
  - C_GetTokenInfo

Connect to token

- When slot with token is found
  - C_OpenSession
  - public session is opened
- Switch to private session by inserting PIN
  - C_Login
  - C_Logout
- C_CloseAllSessions

PKCS#11 arguments lists

- Most of the PKCS#11 functions accept parameters as CK_ATTRIBUTE[] array
- Every value is encoded in single CK_ATTRIBUTE
  - CK_ATTRIBUTE_TYPE type
  - CK_VOID_PTR pValue
  - CK_ULONG ulValueLen

```
CK_CHAR label_public[] = {"Test1_public"};      //label of data object
CK_CHAR data_public[] = {"CxxTest Public"};
CK_ATTRIBUTE dataTemplate_public[] = {
       {CKA_CLASS, &dataClass, sizeof(dataClass)},
       {CKA_TOKEN, &ptrue, sizeof(ptrue)},
       {CKA_LABEL, label_public, sizeof(label_public)},
       {CKA_VALUE, (CK_VOID_PTR) data_public, sizeof(data_public)},
       {CKA_PRIVATE, &pfalse, sizeof(pfalse)}  // private object
};
BYTE    numAttributes_public = 5;
C_CreateObject(hSession, dataTemplate_public, numAttributes_public, &hObject);
```

Store/search/get data (public, private)

- Data created in public/private part of the token
  - CKA_PRIVATE attribute
  - C_CreateObject()
- User must be logged when creating/read private objects
- You must find target object
  - attribute template, must be logged when searching private objects
  - C_FindObjectsInit()
  - C_FindObjects()
  - C_FindObjectsFinal()
- Read data from object
  - C_GetAttributeValue()

Practical assignment

- Write your own code that will utilize PKCS#11 lib
  - run-time dynamic linking
  - use to store secrets for DRM controller
  - assume that user has no access to private part of PKCS#11 token
- Write unit tests
  - Get value from token
  - Write value to token
  - Write/read value protected by PIN