



Contents

- [1 Definition](#)
- [2 Using Import Directory](#)
- [3 Using Polymorphic DLLs](#)
- [4 Internal links](#)

Definition

Data partitioning involves separating code from data in the file system so that a simple trusted path is created on the platform. The central idea is that by "caging" non-TCB processes into their own part of the file system, system files become hidden to them.

Data caging means that the applications and the users have access only to certain areas of the file system. In practice the applications can access their own private directories and directories that are marked as open. It means, for example, that one application cannot access another application's private directory and data.

The file system has the following structure:

****/sys/***

The `/sys` folder is the restricted system area and is inaccessible to non-TCB programs (TCB capability is required to modify the `/sys` contents but AllFiles capability is enough to read `/sys` (file browser, diagnostics tools)).

The `/sys` folder contains a subdirectory `/sys/bin/` that holds all executables (EXEs, DLLs, etc.). Because it is a flat directory, all EXE files, DLLs, etc. must have a different name. The OS will refuse to run code placed in any other location.

****/private/***

Non-TCB programs have their own restricted view on the file system consisting of the directory sub-tree `/private/<SID>/`, where `SID` is a unique security identifier (SID), taken from the program's UID. Applications, for example, would use their sub-tree to hold `.ini`, `.mbm`, `.rsc`, and data files. Other processes need AllFiles capability to be able to access the directory.

This is a private playground for each application. Reading and writing is only allowed to the application's own directory. Backup software has read and write access to this directory.

****/resource/***

This directory is public, but is read-only for non-TCB processes. The purpose is to allow read-only files to be publicly shared without compromising their integrity. Applications without any capabilities can read these files.

Data_caging

Tcb capability is required to modify the `\resource` contents. Software installer adds and updates these files, but not anything else. Subdirectories can be utilized to ensure file name uniqueness.

These protected directories are located on either fixed or removable drives. All the other (previously used) directories are unprotected, so it is not safe to leave any sensitive data in them. It is normally used to store application icons, bitmaps, etc.

***/all the rest/**

Access to all the other folders is unrestricted. They can be used for anything, such as user's own photos, music, and documents.

Using Import Directory

Some servers can have their functionality extended by installing associated plug-ins; they discover the existence of any plug-ins by scanning a directory where the plug-ins' resource files are expected to be found.

Since these resource files are of no interest to any executables apart from the relevant server, it is natural to put them in their private directory. But how does Software Install decide whether it is safe for an apparently unrelated SIS package to install a file in the private directory of another process?

The solution is to be found in the concept of the import directory. Software Install will only allow the package to write files into a subdirectory of an unrelated process if it is named "import" (that is, `/private/<process_sid>/import/`) and if such a subdirectory already exists.

An example of this is the directory where third-party application registration files are stored:
`\private\10003a3f\import\apps.`

Using Polymorphic DLLs

It is no longer possible to look in the `\sys\bin` folder to find the DLLs without having the required access capabilities. Therefore searching for polymorphic DLLs should be done through `TFindLibrary`, which can be used to find DLLs whose full names match a specified pattern. Once the required polymorphic DLL has been found, the `Handle()` method can be used to obtain the find-handle number associated with the kernel object. The polymorphic DLL handle allows a program to load and close a particular polymorphic DLL. It also allows the caller to obtain pointers to functions exported by the DLL. The system can check that a polymorphic DLL is of the correct type by checking the type UID value.

Internal links

- [Symbian Platform Security Model](#)
- [Capabilities](#)