

# Platform Security - a Technical Overview

Mark Shackman

Version 1.2

- 1. INTRODUCTION ..... 1**
- 2. WHY THE NEED FOR PLATFORM SECURITY?..... 2**
  - 2.1 DETECTION ..... 2
  - 2.2 PREVENTION ..... 2
- 3. PLATFORM SECURITY CONCEPTS AND TERMINOLOGY..... 2**
  - 3.1 CAPABILITIES..... 2
  - 3.2 USER-GRANTABLE PERMISSIONING ..... 4
  - 3.3 IDENTIFIERS – SIDS, VIDS AND UIDS ..... 4
- 4. PLATFORM SECURITY ARCHITECTURE..... 5**
  - 4.1 HOW CAPABILITIES AFFECT API CALLS ..... 5
  - 4.2 HOW CAPABILITIES AFFECT DLLS ..... 6
  - 4.3 SOFTWARE INSTALLER ..... 6
  - 4.4 API CHANGES ..... 7
  - 4.5 DATA CAGING..... 7
  - 4.6 REMOVABLE MEDIA..... 7
  - 4.7 BACKUP AND RESTORE ..... 8
- 5. CERTIFICATION ..... 8**
  - 5.1 CERTIFICATES ..... 9
  - 5.2 DEVELOPER CERTIFICATES..... 9
  - 5.3 OBTAINING CERTIFICATES ..... 9
- 6. FURTHER INFORMATION ..... 10**
  - 6.1 GLOSSARY ..... 10
- APPENDIX A – LIST OF CAPABILITIES..... 11**

## 1. Introduction

This document takes a look at the platform security enhancements for Symbian OS and is intended to give a brief technical summary of the changes that platform security has brought to the architecture of Symbian OS.

It should be noted that what is presented here is just an overview of platform security; it is necessarily a simplification and will not cover all services, situations etc. that have in fact been considered and accounted for within the architecture.

## 2. Why the Need for Platform Security?

Version 9.x is a major evolution of Symbian OS. It is specifically geared to target mid-range phones in tens and hundreds of millions of units – offering even larger opportunities for ISVs to market their products – and as such includes a significant number of new features and improvements to enable key emerging technologies required by advanced, open mobile phones for years to come. In order to provide comprehensive support for new functionality such as Digital Rights Management, Device Management and Enterprise-grade data handling, large changes have been made to the very core of Symbian OS to support vital concepts such as data protection or “caging” and restricting some API usage.

One aspect of these changes is the platform security enhancements. These represent an evolution of the existing perimeter security model of Symbian OS and help ensure the stability of the platform, providing even greater protection against malicious or badly-implemented programs. Platform security does this by operating at the software level in two ways:

### 2.1 Detection

The ability to detect unauthorised access attempts to hardware, software or data is an obvious requirement of any security system.

Platform security ensures that an application which tries to use an API that is considered sensitive has the right to call into that API and use the functionality it provides.

### 2.2 Prevention

The ability of a system to prevent programs from acting in unacceptable ways is also a necessary part of an effective security system, irrespective of whether these actions are intentional (by malware) or unintentional. Such actions might include unauthorised hardware access, attempts to read or write restricted data etc.

Under platform security, prevention is essentially achieved by giving all executable code a level of trust that determines what they can do within the system, and then enforcing that, so that programs are unable to act in an undesirable manner.

## 3. Platform Security Concepts and Terminology

The following concepts and terms are core to an understanding of platform security issues.

### 3.1 Capabilities

A *capability* is an entity of protection. Functionality (i.e. an API) within Symbian OS has a capability associated with it if it needs to be protected, and code wanting to use that functionality must be guaranteed to use the API in a safe way.

Code that needs to access capability-protected functionality must go through the process of *authorisation*, in order to gain the privilege of using a capability. Once this has been successfully completed, the code is considered *authorised* for that capability. Authorising a capability effectively confirms that the code is trustworthy enough to use the functionality protected by that capability.

On a phone, much of the functionality is provided by applications without a user interface, called servers. In practice, capabilities are required by servers in order to access sensitive APIs.

Usually it is the responsibility of any server that requires a capability to access its APIs to ensure that the calling process has the required capability. Only about 40% of Symbian OS APIs fall within this category; all other APIs do not require the calling process to possess any capabilities.

Capabilities that an application requests are listed in its project definition (MMP) file. This data is used by Symbian OS itself, the Software Install component of Symbian OS and also Symbian Signed to check what functionality an application should have access to.

Capabilities are classed as “basic” or “system”. For the purposes of authorisation via Symbian Signed, “basic” and “user” capabilities are identical and the “system” capabilities have been split in two, resulting in three sets of capabilities: User, Extended and Phone-Manufacturer approved.

### 3.1.1 User Capabilities

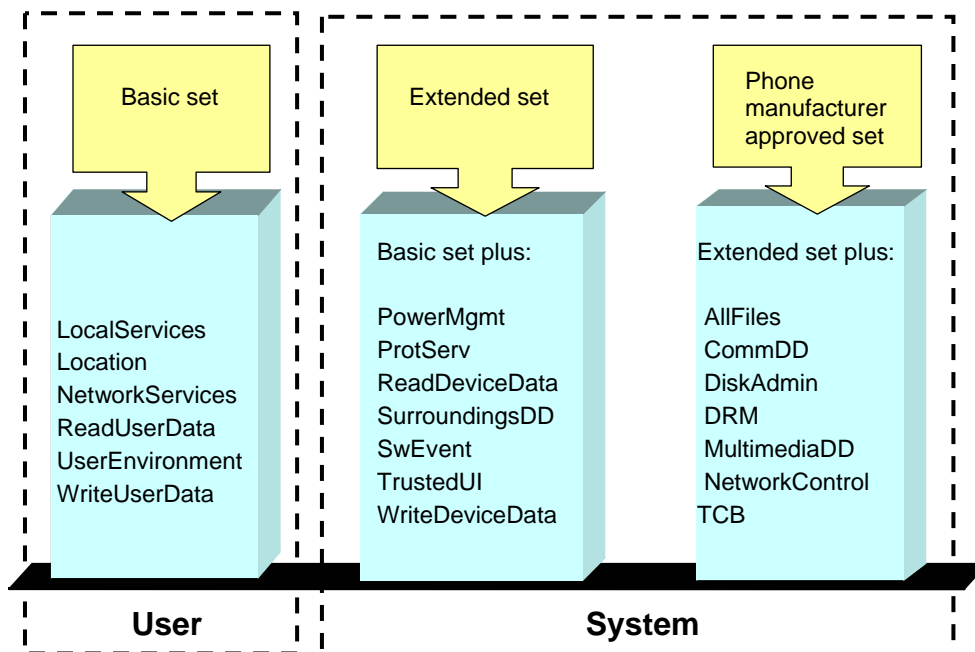
The “User” set of capabilities is granted to certified Symbian Signed applications. This grants automatic permission to all the capabilities in this set, so the user is never asked for authorisation when using these applications.

See Appendix A for a list of capabilities within this set.

### 3.1.2 Extended Capabilities

The “Extended” set of capabilities is granted to Symbian Signed applications that use non-user protected functionality, for which they undergo additional testing. This set includes capabilities from the User set, and also other capabilities providing lower-level access to system functions, such as SwEvent or ReadDeviceData.

See Appendix A for a list of capabilities within this set.



### 3.1.3 Phone-manufacturer Approved Capabilities

The right to grant the most sensitive capabilities, such as DRM, TCB and AllFiles, is reserved by the phone manufacturers. As part of the signing process, an application requiring any of these capabilities must gain the approval of a phone manufacturer.

See Appendix A for a list of capabilities within this set.

## **3.2 User-Grantable Permissioning**

A phone manufacturer may choose to allow the user to authorize selected capabilities. These capabilities are called the “User-grantable” set and, if a capability has been included in the “user-grantable” set, it is possible for the software install tool to prompt the user to request that an application be granted “blanket permission” (see 3.2.1 below). Applications requiring only “User-grantable” capabilities therefore do not need to be submitted for signature if user-granted authorisation is acceptable. However, it should be noted that these capabilities included in this set may vary according to manufacturer and/or device, so should not be relied upon.

### **3.2.1 Blanket and single shot permissioning**

When a program attempts to use a capability-restricted server, two types of authorisation can be given to allow the program to use the server: blanket permission or single shot permission.

If the granting authority or an end-user gives blanket permission to the program for particular functionality, this capability is granted to the program whilst it is installed. Note that, once blanket permission is given, the end-user is unable to revoke the permission without uninstalling the application and re-installing it.

A small number of (usually high-level) APIs can be authorised by single-shot grant, if their capabilities haven’t been authorised with a blanket-grant (i.e. the capability is not listed in the MMP file). Single shot permission is granted on a one-off basis, permission being requested directly from the end-user each time that action is performed. Single shot grant can be offered to both signed and unsigned applications, and is dependant on the phone manufacturer’s chosen configuration.

## **3.3 Identifiers – SIDs, VIDs and UIDs**

In a secure environment, a server needs to know which programs are permitted to access its API. To achieve this, a server can either maintain a list of these programs, or it can use the capabilities paradigm as in the platform security model. This model avoids the need for specific identification of programs, so a server can control access to its APIs without generally having to know who is making the calls. However, there is sometimes a need to uniquely identify a program, for example when data needs to be tied to a specific program. Occasionally it may also be necessary to be able to identify the vendor of the program. To achieve this, two identifiers have been defined.

### **3.3.1 SID (Secure Identifier)**

From Symbian OS version 9.x, all executables must contain a Secure Identifier, which is guaranteed to be locally unique. This is an implementation detail which most developers will not need to consider – whilst it is possible to explicitly set a SID for the application, by default it is set to match the UID3 value which has always been required.

The inclusion of the SID concept allows platform security to:

- protect APIs
- limit access to APIs to specific applications
- protect access to file system areas on the phone that are used when upgrading content

If required, SID values are specified in the program’s MMP file, using the keyword SECUREID. To ensure that the claimed owner of a SID can be authenticated as part of a signing program,

the standard Symbian OS UID range has been split in two – there now exists both a protected and unprotected range. All UIDs will be allocated from one of these ranges, depending on whether the application will be Symbian Signed or not.

### **3.3.1.1 Protected range**

If the application is to be Symbian Signed, a UID will have to be obtained from the protected range. Allocation of these UIDs is tracked and, when signing the application, the Test House will validate that the UID is globally unique for that application and belongs to the developer.

The software installer ensures that unsigned applications cannot have a UID from the protected range and that no UID is associated with more than one executable (i.e. all UIDs are locally unique).

The protected range includes all UID values below 0x7FFFFFFF and thus also covers all legacy UID allocations. However, if a developer of a version 9.x application with a legacy UID value wishes it to remain unsigned, a new UID from the unprotected range must be obtained.

### **3.3.1.2 Unprotected range**

The unprotected range of UIDs provides “co-operative” allocations for unsigned applications – they gain some of the runtime benefits of protected range UIDs (e.g. local uniqueness) but without the guarantee of global uniqueness. At the same time, Symbian OS code, “firmware” applications which exist on the phone already and Symbian Signed applications are protected by unsigned applications being excluded from the protected range and thus preventing them from using a duplicate UID.

### **3.3.2 VID (Vendor Identifier)**

Executables may also contain a vendor identifier that distinguishes the origin of the executable. If an application needs a VID, it must be signed – unsigned applications must use a VID of 0 (KNullUid), which is the value applied by default.

VIDs allow a specialist server to only accept calls to its API from applications created by the same vendor or a specified set of vendors; for example, Network Operators may release APIs but wish to limit their use.

Further details for obtaining both protected and unprotected UIDs and for obtaining VIDs are published on <http://www.symbiansigned.com/app/page/uidfaq>.

## **4. Platform Security Architecture**

This section details the changes made to Symbian OS architecture in incorporating platform security.

### **4.1 How Capabilities Affect API calls**

The platform security enhancements have restricted the access to many of the APIs in those core components that comprise the Trusted Computing Environment (TCE). An application that wishes to use these APIs must be created with the appropriate capabilities, and must be authorised as being trustworthy to access these APIs. If the application is signed, and the capabilities requested by the binaries match those granted in the .SIS file header, the installer will authorise all the capabilities in the .SIS file.

The capabilities will subsequently be checked by individual servers when calls are made to their protected APIs.

For example, a program with LocalServices capability would be able to request the messaging server to send data by infra-red, but (without the NetworkServices capability) would be prevented from sending an SMS.

## 4.2 How Capabilities Affect DLLs

Capabilities are granted to programs (i.e. processes) only. Where a program (an .EXE file) loads a library (a .DLL file), the library is loaded only if it is authorised for all the capabilities already possessed by the loading program; if not, the program will fail. Capabilities of both programs and libraries are assigned in the source code (the MMP file) and cannot be altered after compilation.

If a process that has been assigned specific capabilities calls code inside a DLL, that DLL must have been granted the same (or more) capabilities as those given to the calling program. Note that, even though the DLL may make no use of APIs that are protected by a capability, the DLL may be called by a variety of processes with different capabilities. Therefore, the DLL must contain sufficient capabilities itself in order to ensure that the requirements of all callers are fulfilled.

For example:

The program P.EXE is linked to the library L1.DLL.

The library L1.DLL is linked to the library L0.DLL.

Case 1:

P.EXE holds Cap1 & Cap2

L1.DLL holds Cap1 & Cap3

L0.DLL holds Cap1 & Cap2.

The load fails because P.EXE cannot load L1.DLL (no Cap2).

Case 2:

P.EXE holds Cap1 & Cap2

L1.DLL holds Cap1 & Cap2 & Cap3

L0.DLL holds Cap1 & Cap2 & Cap3 & Cap4

The load succeeds and the new process is assigned Cap1 & Cap2.

## 4.3 Software Installer

The Software Installer (SWInstall) component has been significantly enhanced to provide run time security by policing files that are being installed on the phone. It carries out both capability checking and data caging.

### 4.3.1 Validation of Installable (.SIS) Files

SWInstall validates a .SIS file by checking that an application has the authority to access the capabilities it wishes to use.

It also ensures that the application has been signed and a certificate chain can be constructed back to a trusted root. SWInstall can check whether a certificate has been revoked, and can

also mark root certificates as mandatory, ensuring that all installable packages must be signed with it.

SWInstall compares the capabilities requested by an installable file with those that the root certificate can grant, and calculates the largest set of capabilities that can be granted as a union of the set of capabilities associated with all validated signatures. A configuration option allows the end-user to grant additional capabilities if the .SIS file requests additional capabilities than SWInstall has calculated it can be granted. If the user declines to grant the additional capabilities, SWInstall will not install the software.

### 4.3.2 Validation of file system data caging

SWInstall ensures that use of and access to the file system is restricted, to protect private data and separate data and code. See section 4.5 below for further details.

## 4.4 API Changes

A number of API changes have been made in order to ensure that the platform security enhancements are accessible and work effectively. The full details of these are outside the scope of this document; they will be published at a later date in the Symbian Developer Library.

## 4.5 Data Caging

Platform security not only implements protection for APIs, but also provides a facility for protecting private data. Coupled with the necessity to separate code and data (preventing code in the data space being executed), the file system has been reorganised to implement data partitioning. By “caging” processes into a specific part of the filing system, privacy of data is ensured.

The filing system now has the following structure:

/sys/	Restricted system area, accessible only to programs with TCB (see Appendix) capability. Executables are placed in, and can only be run from, /sys/bin/
/private/	Contains the private data for all programs, held in the directory /private/<SID>/. If the subdirectory import/ already exists, SWInstall may write data into this directory during installation of any program
/resource/	Contains public data, but is read-only for program without TCB capability

All other directories are public and may be read from and written to by any program. This facilitates the use of removable media.

This structure is enforced by the software installer when new programs are installed or existing programs updated. Since only the software installer may change executables in /sys/bin/ on the internal drive, executables cannot be tampered with by other software.

## 4.6 Removable Media

The end-user may choose to install a program on removable media rather than on the phone’s internal memory. In this case, the platform security architecture must be able to verify that the executable has not been tampered with since installation.

This is achieved by the software installer computing and storing (in the tamper-proof /sys/ directory) an initial hash of the program file on installation, which is recomputed and compared whenever the program is to be run. If the two hashes fail to match, or if the internal hash doesn't exist, the program will not run.

### 4.7 Backup and Restore

Under platform security, file backup and restore is implemented differently depending on the type of the file.

Executables are backed up along with elements of the associated meta-data (which is supplied with the initial SIS archive and contains the information needed to install the files contained in the archive). This preserves the capabilities of the files and ensures that, during the corresponding restore operation, the associated meta-data can again be used to verify both the integrity of the executable being restored and the validity of its digital signature against one of the root certificates always present on the device.

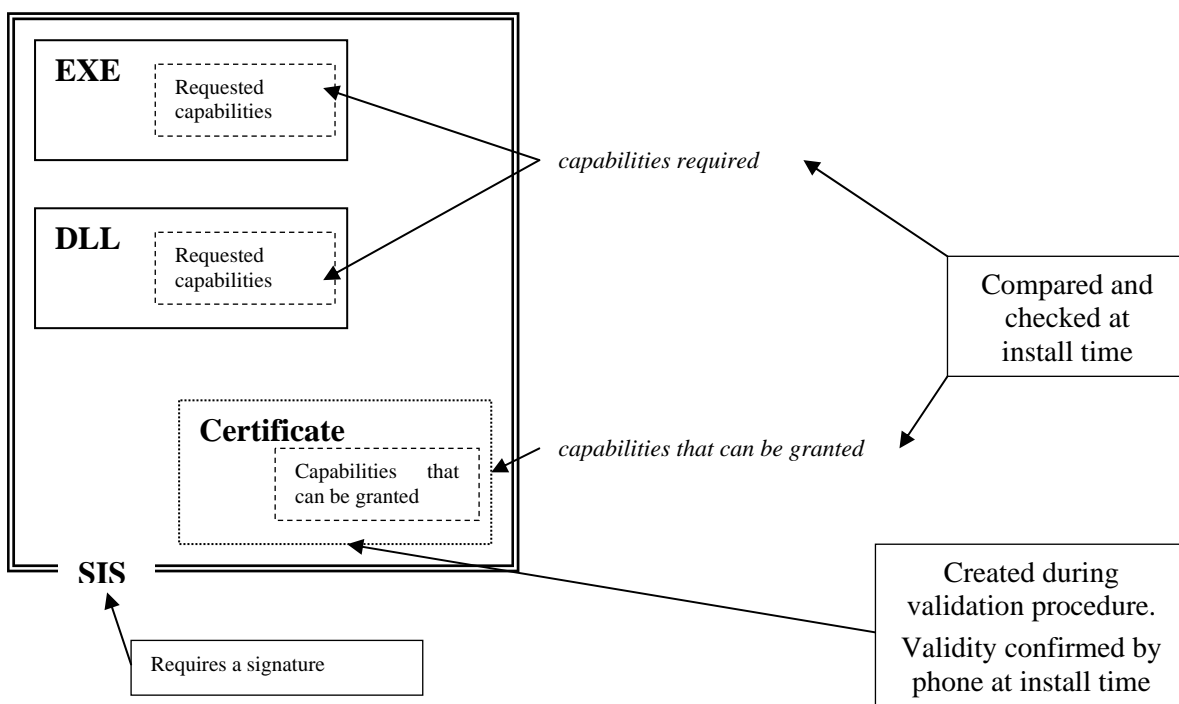
Private data files may have been changed since installation, so would no longer match the hash stored within the associated meta-data. They are therefore backed up and restored with the co-operation of the owning processes.

Most public data files are backed up and restored as previously. Some, such as the /resource/ directory which is only written during installation, are excluded.

Note that backup and restore will be introduced fully into Symbian OS in future releases.

## 5. Certification

Platform security uses certification to grant access to capabilities. Certified applications must pass certain tests in order to be signed against the certificate. The mechanism whereby the certificate provides access to these capabilities is summarised by the diagram below.



## 5.1 Certificates

The process of signing an application has two stages. Initially the application is signed with a Publisher certificate which identifies the developer. In addition to verifying the developer, the signature provides a hash for the submitted application, which prevents tampering.

Once independently verified, the .SIS file is then signed with the Content certificate, which has a chain of trust to the Symbian root certificate on the phone. This certificate enables applications to be installed without the phone displaying warnings and provides the grant authority for restricted capability access. More specific details of the process can be found at <http://www.symbiansigned.com/>.

If a rogue signed application is identified, the software installer can be configured to check for the application's Content certificate via Online Certificate Status Protocol (OCSP) and prevent installation of packages that were signed with this certificate.

## 5.2 Developer Certificates

Developers may wish to test an application on reference hardware prior to the application being submitted to Symbian Signed. However, without possessing a signed certificate, the application cannot be loaded on the phone for testing. Rather than relying just on the emulator or a special "all capability" enabled phone for the testing, Symbian will be providing "Development" certificates, which will be locked to specific phones (via IMEI/ESN number) and allow applications to be tested on standard phones.

Further details are available from <http://www.symbiansigned.com/app/page/devcertgeneral>.

## 5.3 Obtaining Certificates

Full details of the process of getting a .SIS file signed with a trusted certificate for Symbian Signed can be found at <http://www.symbiansigned.com/>. The essential steps are:

- obtain an ACS Publisher ID from VeriSign, to have your corporate identity confirmed
- create a .SIS file for your application
- sign the .SIS file with the ACS publisher ID key and submit it (zipped, along with the .PKG file and the user documentation) to selected Test House
- Test House ensures signature valid, installs .SIS file and tests
- if the .SIS file passes validation, the ACS Publisher ID is removed and the .SIS is re-signed with a Content certificate linked to the Symbian root and which has a unique identifier. (It is this identifier that is used in the revocation process.)

## 6. Further Information

### 6.1 Glossary

The following technical terms and abbreviations are used within this document.

Term	Definition
Capability	Access to restricted APIs
DLL	Dynamic Link Library
ESN	Electronic Serial Number – identifies each phone uniquely
EXE	Executable file
IMEI	International Mobile Equipment Identity – identifies each phone uniquely
Malware	Malicious Software, designed to disrupt or damage a system
SID	Secure Identifier
SIS	Symbian Installation file
SMS	Short Message Service
SWInstall	Software Installer component from Symbian OS
Trusted Computing Base (TCB)	Components at the core of Symbian OS, with unrestricted hardware & software access
Trusted Computing Environment	Symbian OS system components, which access the TCB
VID	Vendor Identifier

## Appendix A – List of Capabilities

The Basic capabilities are:

LocalServices	Grants access to local network services that usually do not incur a cost (e.g. Bluetooth, infra-red)
Location	Grants access to data giving the location of the phone
NetworkServices	Grants access to remote network services that may incur a cost (e.g. voice calls, SMS)
ReadUserData	Grants read access to data that is confidential to the phone user.
UserEnvironment	Grants access to live confidential information about the user and their immediate environment (e.g. audio, video or biometric data)
WriteUserData	Grants write access to data that is confidential to the phone user.

ReadUserData and WriteUserData protect the user's privacy by allowing data to be protected where necessary. In the examples below, which are for illustration only, the user probably would not be concerned if anyone sees the public data, but certainly does not want any malware to access the private data or another person to see it accidentally.

Type of Data	Public user data	Private user data
<b>Images</b>	Images or videos from "neutral" subjects	Personal and sensitive images
<b>Text</b>	Ordinary scribble	"Secret" notepad memos, PIN numbers and passwords
<b>Mail</b>	Junk mail	Messages containing personal and sensitive information
<b>Calendar</b>	Public holidays, sporting events	Medical appointments
<b>Contacts</b>	Local library, pharmacist, dry cleaner	Rival employers

The capabilities above essentially control access to services and data. They are defined at a more general level, such that a server can present them to the end-user of a phone when installing applications, allowing the end-user some control over what an application may do.

The Extended capabilities include:

PowerMgmt	Grants the right to power off unused peripherals, switch the phone into/out of standby state and power phone down
-----------	---

ProtServ	Grants the right to a server to register with a protected name. Protected names start by a "!". The kernel will prevent servers without ProtServ capability from using such a name, and therefore will prevent protected servers from being impersonated.
ReadDeviceData	Grants read access to phone confidential settings or data
SurroundingsDD	Grants access to the surroundings device driver
SwEvent	Grants the right to generate software key and pen events
TrustedUI	Grants the right to create a trusted UI session, and therefore to display dialogs in a secure UI environment
WriteDeviceData	Grants write access to phone confidential settings that control the phone's behaviour

The Phone-manufacturer approved capabilities include:

AllFiles	Grants read access to entire file system; grants write access to other process' private directories
CommDD	Grants access to communications device drivers
DiskAdmin	Grants access to specific disk administration operations
DRM	Grants access to protected content
MultimediaDD	Grants access to multimedia device drivers
NetworkControl	Grants access or modification rights to network protocol controls
TCB	Grants write access to executables and shared read-only resources

Symbian OS contains a Trusted Computing Base (TCB), which comprises a set of components at the core of Symbian OS which have unrestricted access to all the hardware & software on the platform. The TCB capability, which allows this access, is granted to the kernel, file server and (on open phones) the software installer only.

Surrounding the TCB are other system components that comprise the Trusted Computing Environment (TCE). Access to the APIs available within the TCE (and the TCB) is determined by the possession of extended capabilities. Components within the TCE are therefore granted a subset of the extended capabilities depending on their requirements.

[Back to Developer Library](#)

Want to be kept informed of new articles being made available on the Symbian Developer Network?

[Subscribe to the Symbian Community Newsletter.](#)

The Symbian Community Newsletter brings you, every month, the latest news and resources for Symbian OS.