2

Platform Security Concepts

by Simon Higginson

2.1 Background Security Principles

There's nothing new under the sun

Symbian has a long history of reuse – Symbian OS came from Psion, along with core staff. In addition, the object-oriented software in the operating system encourages reuse – so it should be no surprise that, when Symbian set about improving the security of its platform, the project team included people with a great deal of security experience and drew heavily on design principles firmly established in the field of computer science and already implemented in other systems. This background section outlines those tried-and-tested design principles, before we go on to describe in detail the main concepts in the Symbian OS platform security model. If you prefer to skip the background, you can turn to Section 2.3 where we begin describing how the platform security model is actually implemented.

2.1.1 Reference Monitor

The concept of 'a **reference monitor** which enforces the authorized access relationships between subjects and objects of a system' was first introduced by [Anderson 1972], who wrote that a reference monitor must have three features:

- The reference validation mechanism must be tamper-proof.
- The reference validation mechanism must always be invoked.
- The reference validation mechanism must be small enough to be subject to analysis and tests, the completeness of which can be assured.

Symbian OS has a micro-kernel that undertakes the role of the reference monitor. The combination of the computer hardware, the security kernel, and other highly-privileged OS components together make up the Trusted Computing Base (TCB) – the portion of the system responsible for enforcing security restrictions.

2.1.2 Protection Mechanism Design Principles

According to [Saltzer and Schroeder 1975], there are eight design principles that apply to protection mechanisms:

1. **Economy of mechanism:** keep the design small and simple, enabling easier verification, such as line-by-line examination, of software that implements the protection mechanism. Symbian has adopted this by only fully trusting a subset of Symbian OS (the TCB) which is small enough to be effectively code-reviewed. The TCB is described in detail in Section 2.3.2.
2. **Fail-safe defaults:** decisions are based on permission and, by default, software has no permission. The permissions to carry out particular services can be called privileges – software built for Symbian OS by default has no privileges; procedures are in place to ensure that privileges can be granted to suitably trustworthy software. These procedures, including 'Symbian Signed', are described in Chapter 9.
3. **Complete mediation:** access permission to a protected object must always be checked, and any temptation to cache decisions for performance reasons must be examined skeptically. Symbian OS provides this architecture in the client–server framework, described in Chapter 5.
4. **Open design:** security should not depend on keeping the mechanisms secret – such secrets are difficult to keep, and forego the benefits of others reviewing the design. This principle was first stated by Auguste Kerckhoffs [1883]. Symbian's design is open – you're reading about it!
5. **Separation of privilege:** a protection mechanism that requires two keys to unlock it is better than one requiring only one key. This is not universally applicable, but there are cases where requiring two competent authorities to approve something is beneficial. The ability to require software to have more than one signature in order to be granted privileges is described in Chapter 8.
6. **Least privilege:** a program should operate with the smallest set of privileges that it requires to do the job for which it has permission. This reduces the risk of damage through accident or error. This is similar to a military 'need to know' rule, which provides the best chance

of keeping a secret by disclosing it to as few parties as possible. Least privilege is achieved in Symbian OS by controlling privileges using several distinct 'capabilities' (introduced in Section 2.1.3). The different tiers of trust this enables are outlined in Section 2.3.1 and the rules for assigning capabilities are covered in Section 2.4.5.

7. **Least common mechanism:** the amount of common services, used by multiple programs, should be kept to a minimum because each provides the potential for a leak of information between programs. In Symbian OS, the best example of 'common services' may be shared libraries (DLLs, or Dynamically-Linked Libraries). The rules for the loading of DLLs, covered in Section 2.4.5, ensure that a privileged process cannot use a shared library which is less trustworthy than itself.

8. **Psychological acceptability:** the user interface must be easy to use and fit in with what the user might expect to be reasonable security concerns. Symbian OS provides for security-related prompts at the point when software is installed and tries to minimize security prompts when a service is actually being used, to avoid the user dropping into a routine that could lead to an ill-advised choice. The choices presented, determined by the privileges that may be granted, have been designed to be readily understood, e.g. whether the user wants to let the program make a phone call. Those privileges designed to be understood by users are covered in Section 2.4.4.

2.1.3 Capability-Based Security Model

[Dennis and Van Horn 1966], in a paper discussing the challenges of multi-tasking and multi-user computer systems, proposed a mechanism for 'protection of computing entities from unauthorized access'. They introduced the notion of each process in the system having a '**capability list**' determining which protected resources it can access, and what are its access rights. In their example (protected memory segments) each capability includes both a reference (or pointer), allowing the segment to be addressed, and the respective access rights granted to the process.

Following that influential paper, the concept of a capability-based security model has been somewhat broadened (the original model is sometimes referred to as the 'object capability model' for clarity). We consider the essential characteristics of a capability to be that it is something which is a persistent attribute of a process, that it is predetermined when that process is created and that it completely defines the access rights of that process to a protected resource (no other rules need to be consulted at the point at which the resource is accessed).

The role in Symbian OS of a capability permitting a process privileged access to a protected system operation is described in Section 2.4.

2.2 Architectural Goals

Symbian set a number of architectural goals for the platform security project. This section explains each of these goals.

2.2.1 Ensure Understandability

When exposed to the security model, the phone user should be able to understand it.

It is well understood by security professionals that the most vulnerable parts of a security system are its human users. A security system that baffles the user is likely to lead to unintentional security vulnerabilities – if a user is confused or just doesn't understand, then that user is going to do a risky thing at some point. To minimize this problem, Symbian's goal was to identify a small number of things that the user could clearly understand and make simple choices about, and to hide the rest.

This has been quite a challenge, as it is a strong temptation for a system designer to leave difficult questions to the user, on the basis that the users will be the ones facing the risk, and therefore they should make the decision. Although from a strictly logical viewpoint this is reasonable, it neglects the inevitable fact that most users won't understand, and don't want to learn about, security terminology.

Consider the sorts of prompts that users of PC web browsers see. 'This page contains both secure and non-secure items. Do you want to download the non-secure items?' Is there a right answer to that? When would a user want to say 'Yes' or 'No'? What does 'non-secure' mean? Is this prompt coming up simply because the web-browser designer couldn't decide what to do, or is it allowing the user to make a meaningful choice? The characteristics of mobile phones, such as limited screen size and limited input methods, only make such complicated dialogs an even worse user experience.

2.2.2 Support Open Phones

Enhanced platform security should not prevent third parties from producing exciting software for Symbian OS.

Symbian firmly believes that the mobile phone market will flourish when software for the phones flourishes, just as in the 1980s the PC market flourished because of the wide range of software that many developers wrote for it. PCs were open devices – third parties could write and sell software for them. Symbian wants the phone market to grow and believes

that the key to this is innovation through an open software development environment.

During the design and implementation of this platform security architecture, Symbian has constantly kept in mind the goal to minimize the impact on third-party developers. There are some inevitable effects, often resulting from the difficulty of distinguishing between legitimate, well-intentioned developers and authors of malware. We hope you will agree that Symbian has introduced mechanisms and infrastructure which allow legitimate developers to continue to produce compelling add-on software while helping to avoid 'bad apples' spoiling things for everyone.

2.2.3 Protect the Network

The network infrastructure should not be at risk from open phones.

Many network operators have taken an understandably cautious approach to promoting mobile phones on their networks that are open to third-party software developers, due to the risks of malware and so on. Symbian's goal is to provide mechanisms that ensure untrustworthy software is not able to affect adversely the network itself or other devices on that network, so that network operators can realize the benefits of open-OS mobile phones, such as easy deployment of new services to mobile phones in the field, without being exposed to undesirable risks as a consequence.

2.2.4 Provide a Lightweight Security Model

The model should be secure, but be so in a lightweight way.

This means that run-time performance should not be noticeably worsened and software should not be much more demanding of scarce silicon resources. Similarly, the new features should not overly affect the end-user's interaction with their mobile phone, or adversely impact on developers.

Naturally there is extra software, some APIs have had to be changed and various parts of the system architecture have been redesigned. There is an impact on developers and Symbian OS v9 is not binary- or source-compatible with earlier versions. However, changes that need to be made to add-on software are not extensive.

To minimize the impact on performance, Symbian OS only makes security checks where necessary to ensure the integrity of the system and protect sensitive services (this encompasses approximately 40% of all Symbian OS APIs) and the use of capabilities means that the access rights of processes are computed in advance, rather than every time a decision needs to be made.

2.2.5 Provide a Basis for Trust

Replace a trusting relationship with a trustworthy one.

In Chapter 1, we mentioned trust and confidence as separate but related concepts. Trust is about relationships – someone trusts someone else to do (or not do) something. The act of trusting someone is separate from having confidence that they will do as you expect, but such confidence would make you more likely to be willing to trust them. Confidence means they are trustworthy. The following example illustrates this.

If you give a stranger in the street some money to go and buy a loaf of bread for you, you trust them to do so and to bring you back the bread and the change. What's to stop them taking your money and never being seen again? This might make you think twice about giving them the money in the first place.

If, instead, the person is well known to you, is of good character, and has reliably carried out this service for you before, then you've no problem with asking them to do this. There is an established basis – you know the person is worthy of your trust.

The same concepts apply to computer systems in general and mobile phones in particular. The industry hopes that users will trust the phone and the network services to spend their money and to look after their private information. Symbian's goal is to provide mechanisms and infrastructure which increase mobile phone users' confidence in a Symbian OS phone's ability to do reliably just that.

2.3 Concept 1: The Process is the Unit of Trust

There are three concepts, which are the foundation of Symbian OS platform security architecture. This section looks at the first: what is the unit of trust? In other words, what is the smallest thing about which Symbian OS can make a decision regarding its trustworthiness?

2.3.1 Tiers of Trust

A mobile phone tends to be used by one person only – this is particularly true of 'smart' phones which hold personal information such as contact details and calendar entries. The design of Symbian OS assumes this – for example there's only one contacts book; there is no second contacts book for the owner's spouse for when they borrow the phone. If you've got a phone that's shared between members of a team at work, it's not likely to be used for much other than phoning the boss.

Because Symbian OS has been designed as a single-user operating system, there's no concept of logging on to the phone with a username and password. You can (and probably should) use a PIN to lock the

phone when it's not being used, but once that's entered you're in control of everything. Given that there is only one user, this simplifies the security model. There is no need for access control lists, which specify which of a number of users may access a particular file stored on the phone. This helps towards the architectural goal of a lightweight security model.

The security system does not need to concern itself with whether the phone's user is trustworthy – if they're able to use the phone, they're implicitly authorized to do so. It does, however, need to concern itself with the trustworthiness of the processes that are running on the phone on behalf of the user, which may include various programs downloaded to, or otherwise installed on, the phone after it has left the shop. Such programs might perform operations that spend money or affect the way the phone or network operates.

Symbian's platform security architecture has been designed to control what a process can do. A process is only able to carry out activities for which it has the appropriate privileges. How these privileges are assigned will be covered shortly. But the important point to note here is that, without a specific privilege, Symbian OS will not let a process carry out a requested service which requires that particular privilege (on the grounds that it is not considered trustworthy enough).

Let's consider what a process is. In Symbian OS, a process has at least one thread of execution and it has resources, particularly physical blocks of memory controlled by a Memory Management Unit (MMU) in the hardware. The process is the unit of memory protection; the hardware raises a processor fault if access is made to an address not in the virtual address space of the process. This hardware-assisted protection is what provides the basis of the software security model. Symbian OS can trust a process not to access directly any other virtual address space – the hardware won't let it. For the same reason, the process's own virtual address space is private to it – no other process can access it because the hardware won't let it. The part of the operating system that controls the MMU hardware directly, therefore, has ultimate control. Naturally, there also have to be mechanisms for somehow sharing data between processes and inter-process communications – these operations are managed by the operating system kernel, and we will cover how security is provided for them later in this book (see Chapter 7).

The trusted computing platform that is Symbian OS consists of the Trusted Computing Base (TCB), the Trusted Computing Environment (TCE), other signed software and the rest of the platform. In broad terms there are four corresponding tiers of trust that apply to processes running on a Symbian OS phone, ranging from completely trustworthy to completely untrustworthy. This is shown in Figure 2.1.

What does it mean for Symbian OS to decide the trustworthiness of a process? As we mentioned in Chapter 1, the user and the network service provider trust the mobile phone manufacturer to provide a phone, that

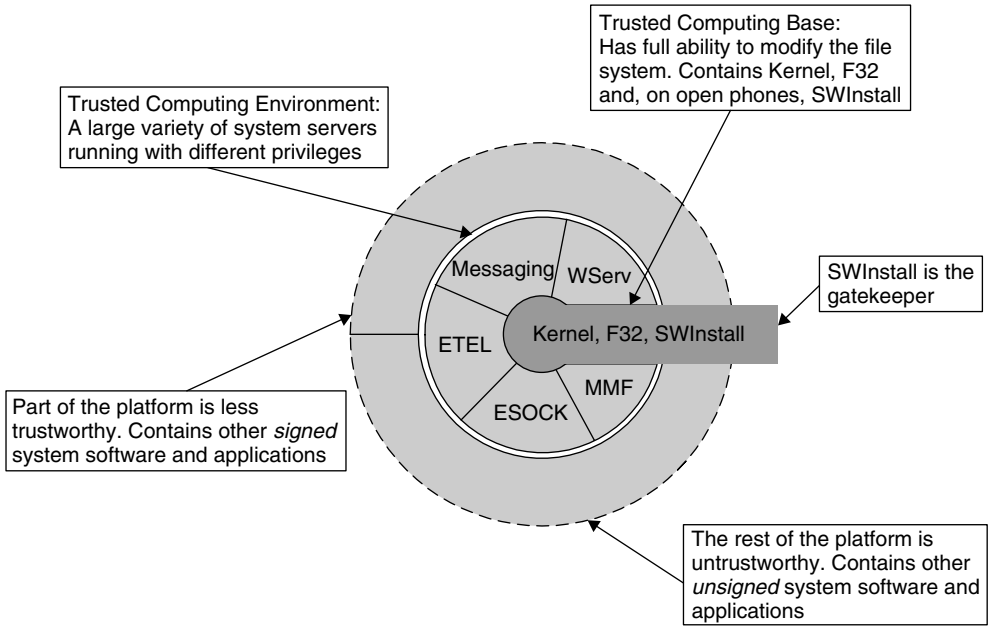

Figure 2.1 Tiers of Trust

preserves privacy, reliability and defensibility. The user will be judging the trustworthiness of the manufacturer (or perhaps the network operator, if it's an operator-branded model) at the time they buy the phone, probably based mainly on the brand reputation, or perhaps their previous experience with phones from the same manufacturer. The manufacturer, in turn, will judge the trustworthiness of software installed on the phone based primarily on where it comes from, but also potentially on testing or other evaluation of the software. For software that is supplied in the phone's read-only media (typically flash ROM), this trustworthiness is recorded directly, based on internal quality assurance processes. For add-on software, this trustworthiness is indicated by a digital signature. The mechanisms for checking this signature and assigning appropriate privileges are covered in Chapter 8. The phone manufacturer determines which authorities they will trust to determine the trustworthiness of add-on software for them, and configures the phone appropriately. Such authorities will typically include the manufacturer themselves, potentially the network operator, and probably also a common signing program for third parties, such as Symbian Signed, covered in Chapter 9.

In Symbian OS, there are now two new identifiers associated with each executable binary file (EXE) – the 'secure identifier' (SID) and the 'vendor identifier' (VID). SIDs are required to be present and unique for each EXE on the device (this is similar in intention to the existing UID3 identifier, which is in fact the default value for the SID if one is not explicitly

specified). There is a protected range of SIDs, which may be used to make security decisions; EXEs are only permitted to use SIDs in this range if they are signed by a trusted authority. VIDs are not required to be unique (the intention is that multiple EXEs from a single source would share the same value); they also may only be used if the EXE is signed by a trusted authority. More practical details on these special identifiers are given in Chapter 3.

The unit of protection remains the process; Symbian rejected allowing a program to link to a DLL whose code ran with different capabilities, or indeed allowing one thread to run with different capabilities from another in the same process. The principle of least privilege may have suggested permitting these; however a protection system not built on top of the fundamental hardware-assisted OS mechanism would have been very cumbersome and expensive. Malicious code could in such cases modify any data including function pointers, thereby perverting the path of execution and services accessed. Consequently many plug-in architectures have changed, including applications which, as described in Chapter 3, are now built as stand-alone programs.

2.3.2 The Trusted Computing Base (TCB)

The Trusted Computing Base, or TCB, is the most trusted part of Symbian OS, as it controls the lowest level of the security mechanisms and has the responsibility for maintaining the integrity of the system. The first protection mechanism design principle in Section 2.1.2 states the design should be small and simple, and therefore our TCB is as small as possible, in order to support this level of trust.

The TCB includes the operating system kernel, which looks after the details of each process, including the set of privileges assigned to it. The file server (F32) is also included in this tier because it is used to load program code to make a process. The process's privilege information is established in the kernel during this loading activity. Some Symbian OS phones are 'closed', that is they do not support installation of native add-on software; on such a closed phone, the kernel, including the kernel-side device drivers, and the file server are the only fully-trusted components. On an 'open' phone, the software installer (SWInstall) is also part of the most-trusted group. This is the program that runs when you install files from a Symbian OS Software Install Script (SIS) file package. It extracts the files from the package (for example, program binaries) and it has the important role of validating the privileges requested for the program binaries against a digital signature on the installation package. Note that most user libraries are not included in the TCB – only those few which need to be used by the file server or software installer are given the highest level of trust.

The kernel, the file server process and the software installer have been carefully checked to ensure they behave properly and are considered

completely trustworthy. They therefore run with the highest level of privilege of any processes on the phone. We should note here that strictly speaking the TCB also includes the phone hardware, including the MMU and other security-related hardware features; however, we will not be dwelling on that in this book as the hardware is not supplied by Symbian.

Figure 2.1 deliberately does not show the TCB as the center of a set of onion rings. First, although the kernel might normally be thought of as occupying this position, some of its services are available to all processes; secondly, the file server, rather like other servers, is both a client of the kernel and available to other processes; and thirdly, the software installer is shown stretching to the outer perimeter because it acts as the gatekeeper for the phone.

2.3.3 The Trusted Computing Environment (TCE)

The Trusted Computing Environment, or TCE, consists of further trusted software provided in the mobile phone by Symbian and others such as the UI platform provider and the phone manufacturer. This code is still judged to be *trustworthy*, but need not run with the highest level of privilege in order to get its job done, so it can be less *trusted* than TCB code. TCE code usually implements a system server process – failure of one server should not threaten the integrity of the operating system itself: the kernel can restart the server and maintain that integrity. Each server has limited privileges to carry out a defined set of services. By not granting all privileges to all servers, Symbian OS limits the threat exposed by any flaw in, or corruption of, a server. By requiring servers to have certain privileges, it is possible to limit access to sensitive low-level operations to selected servers and, thereby, prevent misuse of these operations by other processes.

For example, the window server has privileged access to the screen hardware but has no need to access the phone network; the telephony server (ETEL) has privileged access to the communications device driver but does not need access to the screen. The TCB controls access to the low-level operations (such as access to the screen hardware or to the communications device driver) and ensures that only the appropriately-privileged TCE components are able to perform them. The TCE components (such as the window server or the telephony server) then provide services to software outside the TCE, which is unable to perform directly the low-level operations.

2.3.4 Signed Software

It is possible to install software that adds to or modifies components in the TCB or TCE, but only if that software is signed by a trusted authority and

that authority is permitted to grant the necessary privileges. Most add-on software will, however, be outside the TCE.

Even though such software is not part of the TCE, it may still need certain privileges in order to use services provided by the TCE. One example of this is access to network services, such as opening a network socket. The socket server (ESOCK) is part of the TCE, and handles the low-level operations on the network interface. A program that wishes to open a network socket requests the socket server to do so on its behalf (the program is not able to control the network interface directly). The socket server will only honor that request if the program has been granted the appropriate privilege – we don't want to provide free access to network sockets to software that is completely untrustworthy; it might, for example, be malware trying to attack the network or other devices.

Signed software outside the TCE can be less trustworthy than software within it. When an authority is deciding whether to sign a program to allow it, for example, to open network sockets, the assurance (testing the program, checking the *bona fides* of the developer, and so on) does not need to be as strict as it would be for software that is inside the TCB or TCE. This is because such software is less trusted – it is not allowed to affect the integrity of the system or to access sensitive low-level operations.

2.3.5 Unsigned Software

With unsigned software, or indeed signed software if it has not been signed by one of the trusted authorities (as configured in the mobile phone), the system has no basis for determining its trustworthiness and so it is therefore *untrusted*. This does not necessarily mean that the software is evil or worthless – there are many useful operations that can be performed on a mobile phone that do not require privileges, because they do not have any security consequences. A solitaire game, for example, would not need to perform any actions that access user-sensitive data or system-critical data and services. Such software can be installed and run on the phone without needing a signature (at least not for security purposes – a signature may still be useful to give users confidence in the quality of the signed software). In effect, unsigned software is 'sandboxed' – it can run, but it is not trusted and thus can't perform any security-relevant operations.

2.4 Concept 2: Capabilities Determine Privilege

The second concept underpinning the Symbian OS platform security architecture is the privilege model – each process carries along with it capabilities which determine what sensitive operations it can perform.

2.4.1 Capability Definition

> *A token [is] usually an unforgeable data value (sometimes called a 'ticket') that gives the bearer or holder the right to access a system resource. Possession of the token is accepted by a system as proof that the holder has been authorized to access the resource named or indicated by the token.* [Shirey 2000]

A capability is a token that needs to be presented in order to gain access to a system resource. In Symbian OS, these system resources take the form of services provided via an API – different APIs may require different capabilities to gain access to restricted services, for example, functions provided by a server or device driver, or data such as system settings. Possessing a capability indicates that the process is trusted not to abuse resources protected by that capability.

In the earlier parts of this chapter we have used the term 'privileged' to describe software that has authority to carry out a restricted operation that provides access to sensitive system resources. Symbian OS platform security is built around using capabilities to represent these access privileges. Executable code can have no capabilities or it can have a collection of capabilities. The kernel holds a list of capabilities for every running process. A process can ask the kernel to check the capabilities of another process before deciding whether to carry out a service on its behalf.

Symbian OS defines 20 capabilities, aligned with specific privileges. This number is a balance between reducing the number of capabilities and hence reducing complexity ('economy of mechanism', 'psychological acceptability') and increasing the number of capabilities giving a fine degree of control ('least privilege'). One extreme would be having just one capability to authorize everything (similar in scope to the UNIX 'superuser' model) whereas the other extreme would be a different capability for every protected API (over 1000!) As we have identified four tiers of trust (see Section 2.3.1), we need at least three different capabilities in order to distinguish between them. In fact Symbian has chosen to define a few more than that, to provide some separation between the privileges of different TCE components and to subdivide privileges granted either by the user or by a signing authority.

Symbian OS supports three broad categories of capabilities: one capability only possessed by the TCB itself, other system capabilities and, finally, user capabilities. This is illustrated in Figure 2.2. For add-on software, the software installer, acting as gatekeeper, validates that the program is authorized to use the capabilities with which the executable code was built, and will refuse to install software that does not have the correct authorization (digital signature).

Capabilities are *discrete* and *orthogonal*. This means they are not a hierarchical set of access tokens, with each one adding more and more privileges until reaching the level of the TCB. Instead, any protected

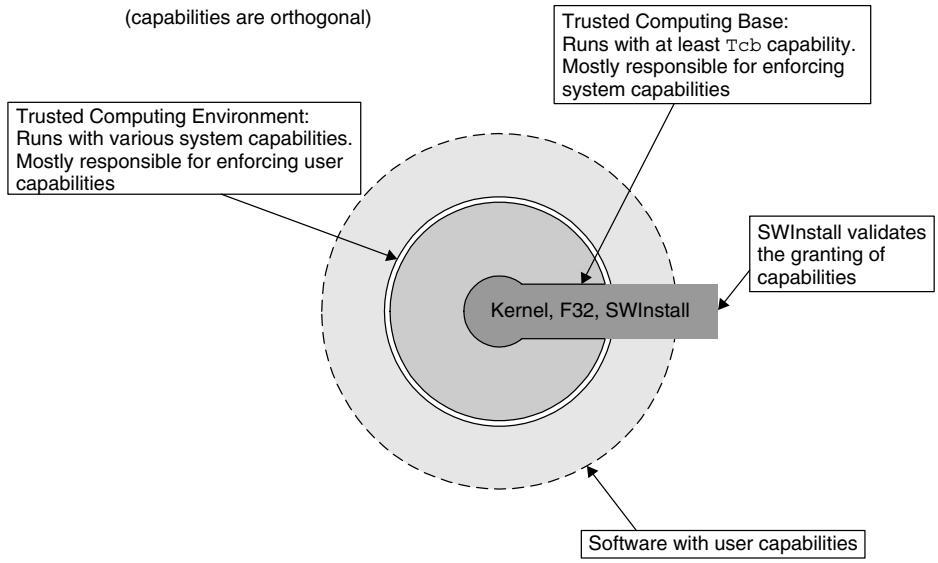

Figure 2.2 Categories of Capability

resource will be controlled by just one specific capability, and any process accessing that resource, even including TCB processes, must possess that specific capability in order for the access to succeed; in other words, capabilities do not overlap. It is also worth noting here that different operations may require different capabilities even if those operations are implemented using APIs within the same Symbian OS component – for example, different capabilities may be required depending on whether the data being accessed is considered to be user data or system data, even if it is stored using the same mechanism.

2.4.2 TCB Capability

The TCB runs with maximum privilege, in that it is granted all capabilities. As there are some things that TCB code must be able to do that nothing else can (such as creating new executables), there is one capability which is only given to TCB code: this capability is called '`Tcb`', logically enough. Processes with this capability can create new executables and set the capabilities which will be assigned to them. As this could be used by malware to create and then run another executable with any capability it wanted, `Tcb` capability is very much the 'keys to the castle', and therefore it should only be granted to add-on software in strictly controlled circumstances.

2.4.3 System Capabilities

The largest group of capabilities is the system capabilities. The granting of a system capability allows a process to access sensitive operations,

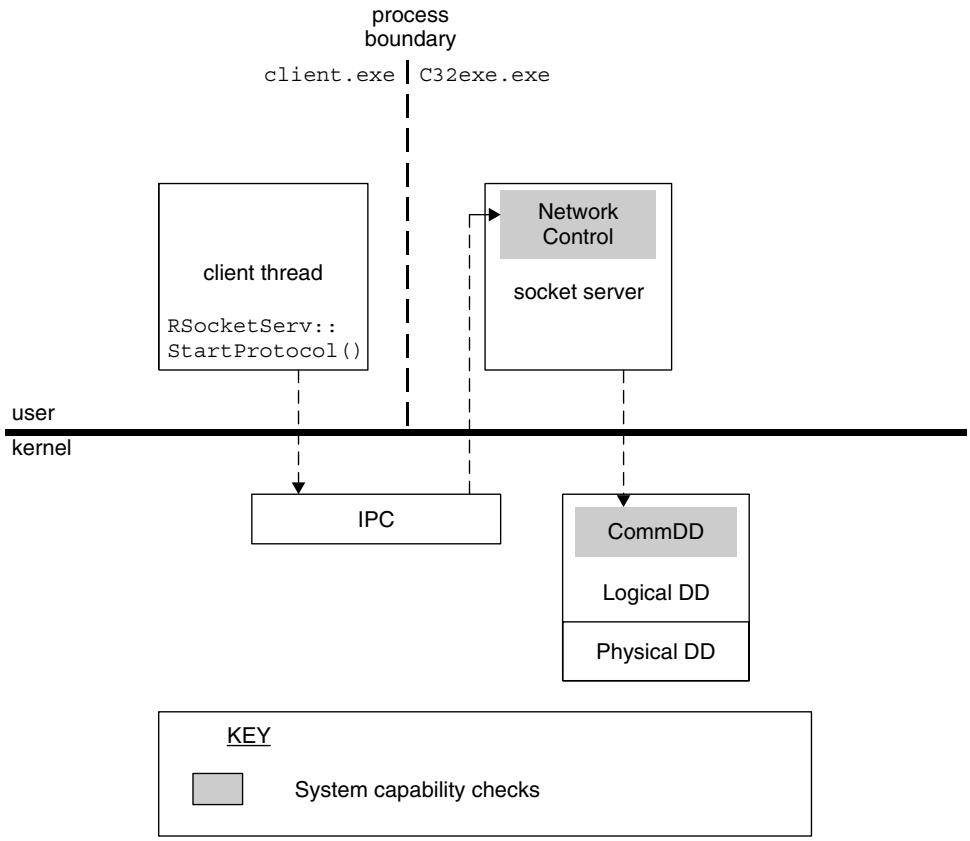

Figure 2.3 System Server Enforcing System Capabilities

misuse of which could threaten the integrity of the mobile phone. System capabilities are not particularly meaningful to the mobile phone user, and are not intended to be referenced in user dialogs. Software which needs system capabilities should be granted them either by building them into the phone ROM, or by being signed by a trusted authority.

The TCB is responsible for enforcing most of the system capabilities (for example `AllFiles`, which is enforced by the file server). Some device drivers, also part of the TCB, require a system capability to access them (for example `CommDD`). Other system capabilities are enforced by the TCE (see Figure 2.3), where the capability controls access to higher-level services (for example `NetworkControl`, which is enforced by the socket server, among others, which itself has `CommDD` so that it can access the necessary lower-level services).

Table 2.1 summarizes the available system capabilities. More detail on the use of each capability is provided in Appendix A, Section A.1.

Table 2.1 System Capabilities

Capability	Privilege Granted
`AllFiles`	Read access to the entire file system and write access to other processes' private directories.
`CommDD`	Direct access to all communications equipment device drivers.
`DiskAdmin`	Access to file system administration operations that affect more than one file or directory (or overall file-system integrity/behavior, etc.).
`Drm`	Access to DRM-protected content.
`MultimediaDD`	Access to critical multimedia functions, such as direct access to associated device drivers and priority access to multimedia APIs.
`NetworkControl`	The ability to modify or access network protocol controls.
`PowerMgmt`	The ability to kill any process, to power-off unused peripherals and to cause the mobile phone to go into stand-by, to wake up, or to power down completely.
`ProtServ`	Allows a server process to register with a protected name.
`ReadDeviceData`	Read access to confidential network operator, mobile phone manufacturer and device settings.
`SurroundingsDD`	Access to logical device drivers that provide input information about the surroundings of the mobile phone.
`SwEvent`	The ability to simulate key presses and pen input and to capture such events from any program.
`TrustedUI`	The ability to create a trusted UI session, and therefore to display dialogs in a secure UI environment.
`WriteDeviceData`	Write access to settings that control the behavior of the device.

2.4.4 User Capabilities

User capabilities are a smaller group of capabilities deliberately designed to be meaningful to mobile phone users, relating to security concepts that should be easy for users to understand and about which to make choices. As a general principle, assigning a user capability to a process should not allow that process to be able to threaten the integrity of the mobile phone – user choices when installing add-on software should never prevent the phone from working. It may however be appropriate for mobile phone users to make decisions about whether add-on software can spend their money on phone calls (`NetworkServices` capability) or have access to their personal data (`ReadUserData` capability, for example).

User capabilities will typically be granted to add-on software that is making use of services provided by the TCE. The TCE is responsible for checking and enforcing user capabilities, and then performing the requested services on behalf of the add-on software. Those services will typically be performed using system capabilities granted to the TCE, as shown in Figure 2.4:

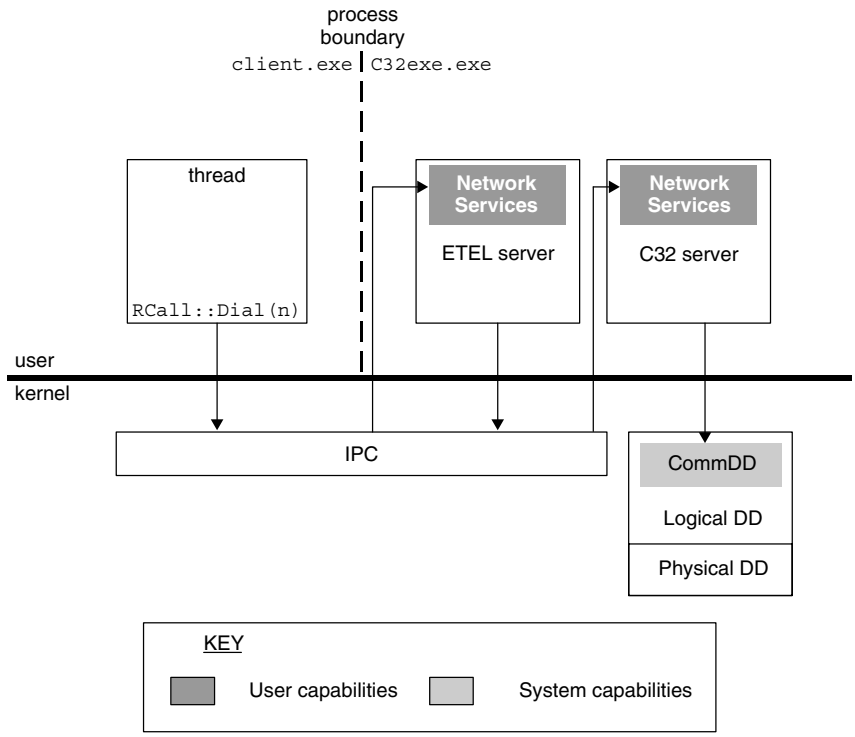

Figure 2.4 System Server Enforcing User Capabilities

Table 2.2 User Capabilities

Capability	Privilege Granted
`LocalServices`	Access to services over 'short-link' connections (such as Bluetooth or Infra-red). Such services will not normally incur cost for the user.
`Location`	Access to data giving the location of the mobile phone.
`NetworkServices`	Access to remote services (such as over-the-air data services or Wi-Fi network access). Such services may incur cost for the user.
`ReadUserData`	Read access to confidential user data.
`UserEnvironment`	Access to live data about the user and their immediate environment.
`WriteUserData`	Write access to confidential user data.

The available user capabilities are summarized in Table 2.2. More detail on the use of each capability is provided in Appendix A, Section A.2.

Although user capabilities are designed to be understandable by the mobile phone user, it may, nevertheless, not be appropriate to offer some of these choices to the user, depending on the environment in which the phone is being used. The platform security model is deliberately flexible and it is not realistic to expect a 'one size fits all' security policy. Mobile phone manufacturers therefore have some discretion in exactly how they configure platform security on their phones. These configuration options are discussed in detail in Chapter 3. Where the security policy permits, it is possible for a mobile phone user to choose to authorize the use of certain user capabilities by unsigned software at the time that software is installed.

We also note here that some actions, which are protected by user capabilities, can still be performed by unsigned and untrusted software. One example of this is sending an SMS message; there are three ways in which software can be permitted to do this:

- The software is signed by a trusted authority and granted `NetworkServices` capability.
- The software is granted `NetworkServices` capability at install time by the user (if the phone's security policy permits).

- The software uses the 'SendAs' API, which invokes a system server, which, if the software does not have `NetworkServices` capability, asks the user for a 'one-shot' permission to send the message.

2.4.5 Capability Rules

When a developer builds a binary (an EXE or a DLL) for Symbian OS v9, that binary has a set of capabilities (possibly just an empty set) declared within it (see Chapter 3). A binary will then be put onto a mobile phone, either by a phone manufacturer building it into ROM or by it being installed as add-on software. In either case, a decision will be whether that binary is sufficiently trustworthy to be assigned the capabilities that have been declared. In the first (ROM) case, this will be a manual process of checking the declared capabilities. In the second (add-on) case, it is an automatic process handled by the software installer (see Chapter 8).

Thus, after a binary is built into ROM or an add-on binary is installed, Symbian OS can assume that the binary is sufficiently trustworthy to be granted the capabilities declared within it. For an EXE, this means that a process created from that EXE will run with that declared authority to carry out certain privileged operations. In contrast, the capabilities declared within a DLL indicate the degree to which it is trusted, but that DLL may be loaded into a process that is running with less privilege. The code within a DLL cannot assume that it will necessarily be running with the capabilities declared within it, whereas the code within an EXE can (it is still prudent to check for run-time errors due to insufficient privilege in case the EXE was installed with incorrect capabilities declared).

Rule 1: Every process has a set of capabilities (as defined by the EXE) and its capabilities never change during its lifetime.

At run-time the loader, which is part of the TCB (it runs as a thread in the file server process), creates a new process, reading the executable code from the filing system and determining the set of capabilities for the process. The kernel maintains a convenient list of capabilities for all processes to save frequent re-reading of the relevant part of an EXE whenever one process wants to check the capabilities of another.

Once the set of capabilities is determined for a process, it never changes – it remains the same until that process terminates. This is in contrast with some systems that allow privileges to be voluntarily dropped or given up by running processes. Symbian has chosen not to allow this, for simplicity and security – security vulnerabilities have resulted from the careless use of such features in other systems.

Rule 2: A process can only load a DLL if that DLL has been trusted with at least the capabilities that the process has.

When a process loads a DLL it does not enlarge or reduce the capability set of the process; it remains the same, as required by Rule 1. The DLL load will fail if the DLL does not have a superset of (that is, at least the same set as) the capabilities of the process loading it. This prevents untrusted (and thus potentially malicious) code being loaded into sensitive processes, for example a plug-in into a system server. The loader provides this security mechanism for all processes; relieving them of the burden of identifying which DLLs they can safely load. This is illustrated in Figure 2.5 – capabilities declared in the binaries are shown as 'C*n*':

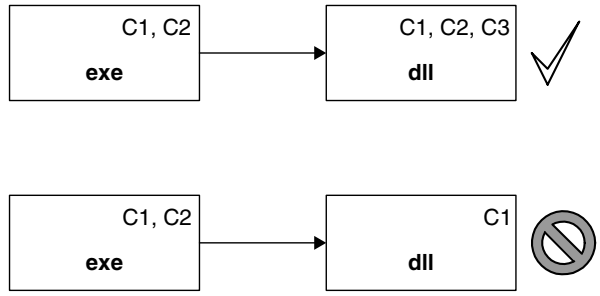

Figure 2.5 Direct Loading of DLLs

Rule 2 has some interesting consequences for *statically-linked* DLLs. Most programs using DLLs will be built using static linking – dynamic loading of DLLs is primarily used when a program has plug-ins that may or may not be present at run time. Static linking resolves references to symbols in the linked DLL at build time so that the run-time loading is more efficient. The most interesting case with regard to capabilities is when one DLL statically links to another – this means that when the first DLL is loaded by a process, the second DLL is also loaded into that process at the same time. Consider the case where the first DLL has a capability that the second DLL does not have, as in Figure 2.6.

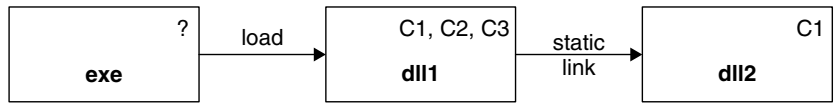

Figure 2.6 Static Linking of DLLs

In this case, because DLL1 is statically linked to DLL2, DLL1 can never be loaded into a process that has capability C2, because that process cannot load DLL2 (which is not trusted with C2). There is therefore no point in declaring capability C2 for DLL1, because it can never be used. In fact,

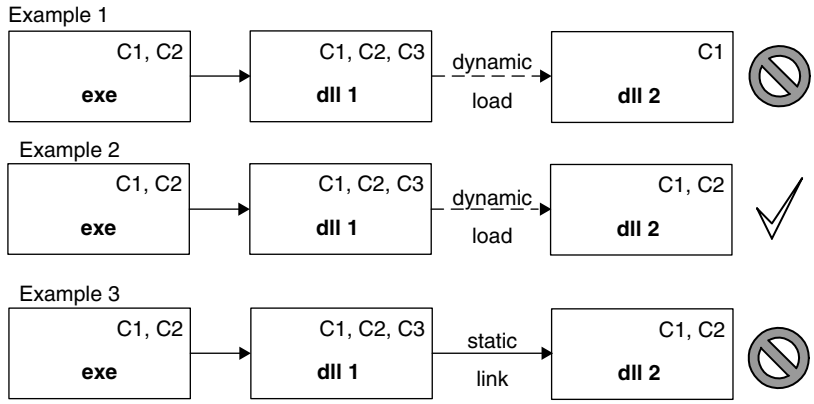

Figure 2.7 Indirect Loading of DLLs

because other processes often reuse loaded DLLs, Symbian has made an optimization to Rule 2 (see also Figure 2.7):

Rule 2b: The loader will only load a DLL that statically links to a second DLL if that second DLL is trusted with at least the same capabilities as the first DLL.

If a DLL that has already been loaded for another process is being reused, the loader only needs to check the capabilities of the first DLL, and not those of any of the other DLLs to which the first DLL may be statically-linked. To avoid being impacted by this rule, developers should not statically link any DLL to another DLL with a subset of capabilities.

2.4.6 Run-time Capabilities for DLLs

There is a consequence of these rules that developers should be aware of. There is a clear difference between capabilities that a DLL is trusted with (declared at build-time) and the subset of capabilities that it relies on having (determined at run-time) to perform its functions. Whilst the former is recorded in a third-party DLL binary's header, there is no similar information about the latter.

You may ask 'Which capabilities do I assign to my executable, because I don't know which capabilities are used by the DLLs to which it links?'. To answer this you need to look at the documentation for each DLL function you use. That documentation should list the capabilities required for the function, and hence the capabilities that you should assign to your executable (in addition to any other capabilities used by system functions that your executable calls directly).

To support this, all DLL authors are strongly encouraged to adopt the Symbian practice of documenting their DLL functions, indicating

which capabilities a function (or the ones it calls) *uses*. Symbian uses an in-source comment tag '@capability'. They also need to indicate whether these capability checks will always occur, or under which run-time conditions they apply (for example, when certain options are specified as a parameter). This is much like having to document any function's behavior, parameter input values, output values, return values, leave codes or panic codes.

2.5 Concept 3: Data Caging for File Access

The third and final concept of Symbian OS platform security architecture is the file access control model. This section explains how the integrity and confidentiality of stored files is preserved.

2.5.1 Data Caging Basics

Data caging is used to protect important files (the term 'data caging' applies to files regardless of whether the content is code or data, so perhaps Symbian should really have called it file caging). These can be either system files or user files. Many system files are critical to the functioning of the system – securing the mobile phone means protecting the integrity of these files. Many user files are personal to the user; securing the mobile phone also means protecting the confidentiality of this user data. The system needs to protect program code from corruption and to prevent undesirable access to system-critical or confidential data.

Not all files are protected – data caging is only applied where necessary. In Symbian OS data caging is achieved by providing special directories that 'lock away' files in private areas; these directories are in addition to the public parts of the filing system. There are three special top level paths: `\sys`, `\resource` and `\private`; access restrictions apply to these directories and, also, to all subdirectories within them. Other paths remain public, this provides compatibility for existing software, which does not need its data to be protected, and allows free access to existing files, which may be on removable media.

The access controls on a file are entirely determined by its directory path, regardless of the drive. This is a simple, lightweight mechanism – the directory in which a file resides determines the level of access protection. It is not necessary to have explicit access control lists for each file to determine which processes may access it. Therefore no precious storage space is taken up with recording information for each file, and the impact on performance and battery life is minimized as there is no list to walk through and process each time.

The consequence of the access rules being implied by the full path to a file is that, should the developer want to limit access, the file only

Table 2.3 Data Caging Access Rules

Directory Path	Capability Required To:	
	Read	**Write**
\sys	AllFiles	Tcb
\resource	none	Tcb
\private\<ownSID>	none	none
\private\<otherSID>	AllFiles	AllFiles
\<other>	none	none

needs be moved to another directory. The access rules are summarized in Table 2.3.

2.5.2 Caged File Paths

\sys

Only TCB code can access the directory \sys and its subdirectories. There are two subdirectories of particular interest: \sys\bin and \sys\hash.

The \sys\bin directory is where all program binaries (executables) reside. Executables built into the mobile phone ROM run from z:\sys\bin; add-on software is written into c:\sys\bin (or \sys\bin on some other writeable drive). This ensures that only TCB software, the most trustworthy, can create new executables (via SWInstall) or load executables into memory (via F32). Executables stored elsewhere will not be runnable, which protects against malware creating new, or modifying existing, binaries in order to get the system to execute malicious code. If your program code attempts to load executable or library code from a different directory, the loader ignores the path and only looks in \sys\bin.

The\sys\hash directory is used to check for tampering with executables which are stored on removable media. It could be possible to make changes to files on removable media by putting that media in another, non-Symbian OS device. This directory is managed by the software installer, which we will cover in more detail in Chapter 8.

\resource

This is a directory tree that is intended for resource files that are strictly read-only. Examples are bitmaps, fonts and help files as well as other

resource files that are not expected to change after installation. Only the TCB can write into this directory; this provides software with assurance that its resource data cannot be corrupted. As with `\sys`, files here are either built into the mobile phone ROM or are installed on writeable media as add-ons by the software installer.

`\private`

Each EXE has its own caged file-system area as a subdirectory under `\private`. This subdirectory is identified by the SID of the EXE, thus it is private to that process. If there are two processes loaded from the same EXE they share the same private subdirectory.

It is up to the developer to choose in which directory a program should store its data, for example, `\resource`, its private subdirectory or a public directory. By default it will be its private subdirectory, though the path will still need to be created the first time the executable runs. There is an API to discover the name of the private path, and also to restore the current path to this setting. Note that data files on removable media, even those under the private path, can still be subject to tampering; protecting data from tampering is covered in detail in Chapter 7.

2.5.3 Data Caging and Capabilities

The concepts of capabilities and data caging work together in providing flexible options for controlling access to a program's data. The various options are covered in detail in Chapter 7, but we will discuss here the two capabilities which allow processes to bypass the normal data caging access controls, as shown in Table 2.3:

- `Tcb` – allows write access to executables and shared read-only resources.
- `AllFiles` – allows read access to the entire file system and write access to other processes' private directories.

The enforcement of these capabilities is the responsibility of the TCB (specifically the file server) and, once granted, they apply to all files on the mobile phone. In general, software should not be given these capabilities if it just needs to read or write one specific sort of data – it would be like giving out a master key to someone who just needs to open one particular safety deposit box in a bank. System services and applications should offer APIs that allow data to be shared with more limited privileges, for example by checking the `ReadUserData`, `WriteUserData`, `ReadDeviceData` and `WriteDeviceData` capabilities, or by checking the requesting process's SID or VID.

2.6 Summary

This chapter sets out the main concepts in Symbian's implementation of its secure platform. Implementation of these concepts, while new to Symbian OS, is not new to the industry.

In the first section we pointed out that in 1966 Dennis and Van Horn came up with the idea of a capability-based security model, then Anderson described a TCB/Reference model in 1972 and in 1973 Saltzer established a set of protection-mechanism design principles.

Symbian's platform security project had a number of architectural goals that strongly influenced the implementation. The mobile phone user should understand the exposed parts of the security model, the model should be a lightweight one, it should replace a trusting relationship with a trustworthy one, and, finally, network operators should confidently provide open mobile phones supporting after-market software.

The next three sections outlined the main concepts around which Symbian based the development, drawing on those set by Saltzer *et al*.

The first was that the 'sphere of protection', the unit of trust for privileged operations, is a Symbian OS process, which the underlying memory-protection hardware supports. The unit of trust gives rise to four tiers of software in Symbian OS. The TCB is the smallest part of the system that processes can fully trust. Many system servers are part of the TCE and therefore can be highly trusted by other processes to carry out particular, privileged operations. Network operators, mobile phone manufacturers and users are able to put appropriate trust in other signed software because of the systems put in place and agreed with industry stake-holders by Symbian Signed prior to signing that software. Lastly, unsigned software is completely untrustworthy (i.e. there is no basis for trusting it beyond safe operations) but large parts of the system API are available to it to carry out unprivileged operations.

The second concept is that of capabilities as a means of restricting access to privileged operations (and data) in the secured system. There are system capabilities that are there to protect the integrity of the phone operating system, and user capabilities that are there to protect access to user sensitive operations and data. Capabilities apply to a process. They are assigned to executables as a statement of intent as to which capabilities a process should have, subject to the loader rules. The rules on DLL loading mean that the capabilities of a process do not change after a DLL is loaded; thus they remain the same as the ones assigned to the EXE at build time. A process is prevented from loading DLLs that should not be trusted with the capabilities it possesses.

The third concept is that of data caging as a means of protecting code and data from processes that should not have access to it. This is an example of the least privilege principle cited by Saltzer, whereby processes do not have access to things unless they are authorized to do

so. Data caging is restricted to three special directory trees under `\sys`, `\resource` and `\private`. The TCB processes have full read/write access to these directories, whereas other processes have limited access. The mechanism for policing access is a combination of the directory path itself and the privileges of the process that wants to make the access.

In the rest of this book we consider and introduce some of the implications of the above concepts on how to organize and secure the design of software.