(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
4 December 2003 (04.12.2003)

PCT

(10) International Publication Number
WO 03/100581 A2

(51) International Patent Classification[7]: G06F 1/00

(21) International Application Number: PCT/GB03/02311

(22) International Filing Date: 28 May 2003 (28.05.2003)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
0212314.9 28 May 2002 (28.05.2002) GB

(71) Applicant *(for all designated States except US)*: **SYMBIAN LIMITED** [GB/GB]; Sentinel House, 16 Harcourt Street, London W1H 1DS (GB).

(72) Inventors; and

(75) Inventors/Applicants *(for US only)*: **DIVE-RECLUS, Corinne** [FR/GB]; 51 Charmouth Road, St Albans, Herts AL1 4SE (GB). **HARRIS, Jonathan** [GB/GB]; 35 Caterham Road, London SE13 5AP (GB). **MAY, Dennis** [GB/GB]; 28 Woodquest Avenue, London SE24 0HD (GB).

(74) Agent: **LANGLEY, Peter, James**; Origin Limited, 52 Muswell Hill Road, London N10 3JR (GB).

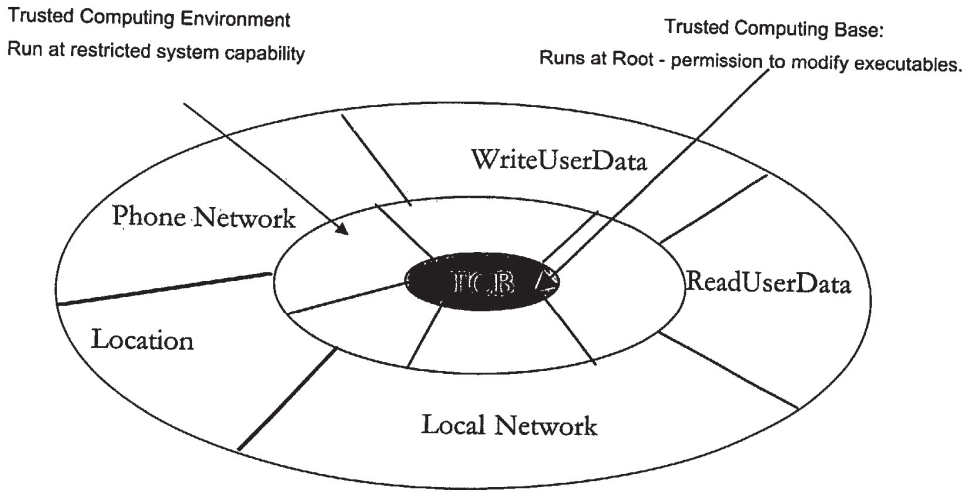
(81) Designated States *(national)*: AE, AG, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, BZ, CA, CH, CN, CO, CR, CU, CZ, DE, DK, DM, DZ, EC, EE, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MA, MD, MG, MK, MN, MW, MX, MZ, NO, NZ, OM, PH, PL, PT, RO, RU, SD, SE, SG, SK, SL, TJ, TM, TN, TR, TT, TZ, UA, UG, US, UZ, VN, YU, ZA, ZM, ZW.

(84) Designated States *(regional)*: ARIPO patent (GH, GM, KE, LS, MW, MZ, SD, SL, SZ, TZ, UG, ZM, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, BG, CH, CY, CZ, DE, DK, EE, ES, FI, FR, GB, GR, HU, IE, IT, LU, MC, NL, PT, RO, SE, SI, SK, TR), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, ML, MR, NE, SN, TD, TG).

Published:
— *without international search report and to be republished upon receipt of that report*

For two-letter codes and other abbreviations, refer to the "Guidance Notes on Codes and Abbreviations" appearing at the beginning of each regular issue of the PCT Gazette.

(54) Title: SECURE MOBILE WIRELESS DEVICE



(57) Abstract: A secure mobile wireless device in which executable code to be installed on the device is assigned a set of capabilities which define the protected resource(s) on the device which it can access. Hence, the present invention takes the idea of capabilities (known in the context of defining the capabilities or access privileges of different *users* in a multi-user system) and applies it to defining the capabilities or access privileges of different *native executable code* for secure, single-user mobile wireless devices.

WO 03/100581 A2

SAMSUNG 1026

SECURE MOBILE WIRELESS DEVICE

FIELD OF THE INVENTION

This invention relates to a secure mobile wireless device; in particular it relates to a new approach to platform security for mobile wireless devices which protects key operating system resources from harmful access.

DESCRIPTION OF THE PRIOR ART

Platform security covers the philosophy, architecture and implementation of platform defence mechanisms against malicious or badly written code. These defence mechanisms prevent such code from causing harm. Malicious code generally has two components: a payload mechanism that does the damage and a propagation mechanism to help it spread. They are usually classified as follows:

Trojan horse: poses as a legitimate application that appears benign and attractive to the user.

Worm: can replicate and spread without further manual action by their perpetrators or users.

Virus: Infiltrates legitimate programs and alters or destroys data.

Security threats encompass (a) a potential breach of confidentiality, integrity or availability of services or data in the value chain and integrity of services and (b) compromise of service function. Threats are classified in the following categories:

1. Threats to confidentiality and integrity of data. Examples: Get user's password; corrupt files.
2. Threats to confidentiality and integrity of services. Examples: Use bandwidth from phone network subscriber without paying for it; repudiate transaction with network service provider.

3. Threats to availability of service (also called denial of service). Examples: Prevent the user from sending a text message; prevent the user from accepting a telephone call.

Hence, mobile wireless devices offer very considerable challenges to the designer of a security architecture. A conventional approach typically operates by first of all determining whether the code (e.g. an application) is sufficiently trustworthy to be installed and then subsequently protecting the platform from errant behaviour once it is installed. In terms of determining trust, third party applications can be categorised according to the degree of trust associated with their authors. Hence, it is known to allow applications to be automatically installed on a device only if they can present a valid digital certificate issued by an acceptable Certificate Authority (CA).

Digital certificates are however only just becoming widespread; prior to their adoption, a primary platform security mechanism was to grant privileges to a user rather than to an application. At run time, the code being executed by the user is granted all his privileges. At install time, only designated users could install sensitive software. This meant that potentially harmful code could be vetted by an expert administrator prior to installation. Many well known operating systems adopt this approach to platform security – e.g. Unix-based operating systems.

Platform security based on giving different users different access privileges is not relevant however for mobile wireless devices for the simple reason that this category of product is *single user*: It is a personal device, typically a 'smartphone', enhanced mobile telephone, PDA or other personal, portable computing device. Furthermore, the single user is very likely not to be expert in computing and hence cannot reliably assess the risk of installing and running code. With mobile wireless devices, the key threats that a platform security model tries to address are unauthorised access to user data and to system services, in particular the phone stack. The phone stack is especially important because it controls a permanent data connection to the phone network; mobile commerce, mobile banking etc. will be performed using the phone stack and (as noted above) losing control over it to malicious or badly written code exposes the device owner to potentially very significant financial damage.

To date, there have been no effective proposals for platform security for mobile wireless devices.

SUMMARY OF THE PRESENT INVENTION

In a first aspect of the present invention, there is a secure mobile wireless device in which native executable code to be installed on the device is assigned a set of capabilities which define the protected resource(s) on the device which it can access. Native code is composed of instructions directly executed by a CPU on a device.

In a second aspect, there is a software application adapted to run on a secure mobile wireless device and written in native code in which the application is assigned a set of capabilities which define the protected resource(s) on the device which it can access.

In a third aspect, there is an operating system adapted to run on a secure mobile wireless device in which the operating system requires executable code requiring access to sensitive resources to be assigned a set of capabilities which define the protected resource(s) which it can access.

Hence, the present invention takes the idea of capabilities (known in the context of defining the capabilities or access privileges of different *users* in a multi-user system) and applies it to defining the capabilities or access privileges of different *native executable code* for secure, single-user mobile wireless devices.

An implementation of the present invention can be thought of as firewall protection of key operating system servers through the use of capability-based access control as applied to native executable code. Each capability may grant access to a specific API (or range of APIs), a file or any other applicable resource(s). Each executable (e.g. program (EXEs), or shared, static or dynamic libraries (DLLs)) contains the capabilities it has been granted at installation time. Each executable is permanently stored in a location only accessible to the Trusted Computing Base (kernel, loader, file server and software installer; see Detailed Description section 1.1) to preserve the integrity of the capabilities assigned to the executable. Once an executable is invoked, the loader will load the executable either to create

a new process or to load the library code into an existing process. When creating a new process, the capabilities of this process are equal to the capabilities granted to the program. Based on this set of capabilities, the loader controls which libraries can be loaded into n process. For each client-server communication, the kernel will pass the client capabilities to the server. Therefore, the server can fully trust the client capabilities, as they are not passed/approved by it but by a process part of the Trusted Computing Base, e.g. the kernel. The server can decide either to process the call or to reject it.

Preferably, the capability model is deliberately limited to a small number of capabilities. *System* capabilities are mandatory and control access to the operating system kernel, file system and system data. They guarantee the integrity of the 'Trusted Computing Environment' (or TCE – see Detailed Description at 1.2). The way they are controlled cannot be modified by the user of the device and they are never exposed to them. Other kinds of capability are discretionary, as they are meaningful enough to average users to let them decide whether or not to grant some of them to code they have installed. These capabilities are categorised are follow:

Root (mandatory) "Full access to all files - Can modify capabilities associated with executables"

Used by the Trusted Computing Base only.

System capabilities (mandatory)

As examples, we may define:

WriteSystemData allows a process to modify configuration system data.

CommDD grants access to all communication and ethernet card device drivers.

User-exposed capabilities (discretionary)

As examples, we may define:

PhoneNetwork. "Can access phone network services (and potentially spend user money)"

ReadUserData. "Can read the device user's private information"

BRIEF DESCRIPTION OF THE DRAWINGS

The present inventionwill be described with reference to the accompanying drawings, in which

Figure 1 shows the Run time processes, system capability rings and user visible capability groupings for an implementation; and

Figure 2 illustrates a possible process of installation.

DETAILED DESCRIPTION

The present invention will be described with reference to the security architecture of the Symbian OS object oriented operating system, designed for single user wireless devices. The Symbian operating system has been developed for mobile wireless devices by Symbian Ltd, of London, United Kingdom.

The basic outline of the Symbian OS security architecture is analogous to a medieval castle defences. In a similar fashion it employs simple and staggered layers of security above and beyond the installation perimeter. The key threats that the model is trying to address are those that are linked with unauthorised access to user data and to system services, in particular the phone stack. The phone stack is especially important in the context of a smart phone because it will be controlling a permanent data connection to the phone network. There are two key design drivers lying behind the model:

- **Firewall protection** of key system resources through the use of **capability-based access control.**
- **Data partitioning** which creates a protected part of the file system which standard software is not able to access.

The main concept of the capability model described below is to control what a *process* can do rather than what a user can do. This approach is quite different to well-known operating systems as Windows NT and Unix. The main reasons are:

- The very nature of Symbian OS is to be mono-user.
- Symbian OS provides services through independent server processes. They always run and are not attached to a user session. As long as power is supplied, Symbian OS is always on even if no user is logged on.
- Symbian OS is aimed to be used in devices used by a large public with no technology knowledge. When installing software, the user may not have the skills to decide what permissions to grant to an application. Furthermore, with always-

connected devices, the consequences of a wrong or malevolent decision may impact a domain much larger than the device itself.

A remarkable point of this invention is that the use of cryptography has been deliberately excluded from the kernel and the loader: executables are not signed, capabilities are not stored encrypted or signed. That has been possible because data partitioning provides a secure storage area for executables. It allows the system to treat executables from ROM (built in the device) the same way as code from RAM (installed by the user) without compromising either on security or performance. Therefore any misuse or exploit of device manufacturer code is restricted to the capabilities assigned to this code.

1 Trusted Computing Platform

1.1 Trusted Computing Base

A trusted computing base (TCB) is a basic architectural requirement for robust platform security. The trusted computing base consists of a number of architectural elements that cannot be subverted and that guarantee the integrity of the device. It is important to keep this base as small as possible and to apply the principle of least privilege to ensure system servers and applications do not have to be given privileges they do not need to function. On closed devices, the TCB consists of the kernel, loader and file server; on open devices the software installer is also required. All these processes are system-wide trusted and have therefore full access to the device. This trusted core would run with a "root" capability not available to other platform code (see section 2.1).

There is one other important element to maintain the integrity of the trusted computing base that is out of the scope of this invention, namely the hardware. In particular, with devices that hold trusted computing base functionality in flash ROM, it is necessary to provide a secure boot loader to ensure that it is not possible to subvert the trusted computing base with a malicious ROM image.

1.2 Trusted Computing Environment

Beyond the core, other system components would be granted restricted orthogonal system capabilities and would constitute the Trusted Computing Environment (TCE); they would include system servers such as phone and window servers... For instance the window server would not be granted the capability of phone stack access and the phone server would not be granted the capability of direct access to keyboard events. It is strongly recommended to give as few system capabilities as possible to a software component to limit potential damage by any misuse of these privileges.

The TCB ensures the integrity of the full system as each element of the TCE ensures the integrity of one service. The TCE cannot exist without a TCB but the TCB can exist by itself to guarantee a safe "sand box" for each process.

2 Process Capabilities

A capability can be thought of as an access token that corresponds to a permission to undertake a sensitive action. The purpose of the capability model is to control access to sensitive system resources. The most important resource that requires access control is the kernel executive itself and a *system* capability (see section 2.1) is required by a client to access certain functionality through the kernel API. All other resources reside in user-side servers accessed via IPC [Inter Process Communication]. A small set of basic capabilities would be defined to police specific client actions on the servers. For example, possession of a *make calls* capability would allow a client to use the phone server. It would be the responsibility of the corresponding server to police client access to the resources that the capability represents. Capabilities would also be associated with each library (DLL) and program (EXE) and combined by the loader at run time to produce net process capabilities that would be held by the kernel. For open devices, third party software would be assigned capabilities either during software installation based on the certificate used to sign their installation packages or post software installation by the user, as detailed in section 3. The

policing of capabilities would be managed between the loader, the kernel and affected servers but would be kernel-mediated through the IPC mechanism.

The key features of the process capability model are:

- It is primarily focused around system servers and client-server IPC interactions between these entities.
- Capabilities are associated with processes and not threads. Threads in the same process share the same address space and memory access permissions. This means that any data being used by one thread can be read and modified by all other threads in the same process.
- The policing of the capabilities is managed by the loader and kernel and through capability policing at the target servers. The kernel IPC mechanism is involved in the latter.
- When the code is not running, capabilities are stored inside of libraries and programs. Capabilities stored in libraries and programs are not modifiable, as they would be stored during installation in a location that is only accessible by the Trusted Computing Base.
- Not all servers would have to handle client capabilities. Servers would be responsible for interpreting capabilities as they wish.
- The only cryptography involved in this scheme might be at the software installation stage where certificates would be checked off against a suitable root certificate (refer to sub-section 3)

2.1 System capabilities: Protecting the integrity of the device

Root. "Full access to all files - Can modify capabilities associated with executables "

"Root" capability – Used by the Trusted Computing Base only, it gives full access to all files in the device.

System capabilities

Some system servers require some specific access to the Trusted Computing Base. Because of the object-oriented implementation of Symbian OS, the kind of resources required by a system server is most of the time exclusive to it. Therefore, one system

server would be granted some system capability that would be orthogonal to those required by another. For instance, the window server would be granted access to keyboard and pen events issued by the kernel but it would not have permission to access the phone stack. In the same way, the phone server would be granted access to the phone stack but would not have permission to collect events from the kernel.

As examples, we can name:

WriteSystemData	Allows modification of configuration system data
CommDD	Grants access to all communication and Ethernet card device drivers.
DiskAdmin	Can perform administration task on the disk (reformat, rename a drive,...).

2.2 User-exposed capabilities: Mapping real-world permissions

The process of generating capabilities can be difficult. One has first to identify those accesses that require policing and then to map those requirements into something that is meaningful for a user. In addition, more capabilities means greater complexity and complexity is widely recognised as being the chief enemy of security. A solution based on capabilities should therefore seek to minimise the overall number deployed. The following capabilities map fairly broadly onto the main threats which are unauthorised access to system services (eg. the phone stack) and preserving the confidentiality/integrity of user data.

PhoneNetwork. "Can access phone network services and potentially spend user money"

"Make telephone calls"

"Send short text messages".

WriteUserData. "Can read and modify users private information"

"Add a contact".

"Delete an appointment".

ReadUserData. "Can read users private information"

"Access contacts data".

"Access agenda data".

LocalNetwork. "Can access local network"

"Send Bluetooth messages".

"Establish an IR connection"

"Establish an USB connection"

Location. " Can access the current location of the device"

"Locate the device on a map"

"Display closest restaurants and cinema"

It is necessary to make a distinction between PhoneNetwork and LocalNetwork because it is possible to transmit information across a network without spending any money (eg. Bluetooth piconet). This kind of access may be a very useful third party software enabler but nonetheless represents a local way of leaking sensitive information via a trojan horse so it must be protected with a capability, albeit LocalNetwork. PhoneNetwork, if granted by the user, would allow trojans seeking to use the phone network as their exit route; that is potentially much more damaging and hence the blunt warning in its description.

Root and system capabilities are mandatory; if not granted to an executable, the user of the device cannot decide to do it. Their strict control ensures the integrity of the Trusted Computing Platform. However the way servers check user-exposed capabilities or interpret them may be fully flexible and even user-discretionary.

Figure 1 shows the Run time processes, system capability rings and user visible capability groupings.

2.3 Assigning capabilities to a process

The association of a run-time capability with a process involves the loader. In essence, it transforms the static capability settings associated with individual libraries and programs into

a run-time capability that the kernel holds and can be queried through a kernel user library API. The loader applies the following rules:

Rule 1. When creating a process from a program, the loader assigns the same set of capabilities as its program's.

Rule 2. When loading a library within an executable, the library capability set must be *greater than or equal to* the capability set of the loading executable. If not true, the library is not loaded into the executable.

Rule 3. An executable can load a library with higher capabilities, but does not gain capabilities by doing so.

Rule 4. The loader refuses to load any executable not in the data caged part of the file system reserved to the TCB.

It has to be noted that:

- Libraries' capabilities are checked at load time only. Beyond that, all code contained in libraries is run freely and is assigned the same capability set as the program it runs into when initiating some IPC calls.
- For ROM images with execution in place, the ROM build tool resolves all symbols doing the same task as the loader at runtime. Therefore the ROM build tool must enforce the same rules as the loader when building a ROM image.

These rules

- Prevent malware from being loaded in sensitive processes, for example, a plug-in in a system server
- Encourage encapsulation of sensitive code inside processes with no possible bypass

The examples below show how these rules are applied in the cases of statically and dynamically loaded libraries respectively.

2.3.1 Examples for linked DLLs

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 & Cap2 & Cap3

L0.DLL holds Cap1 & Cap2.

Process P cannot be created, the loader fails it because L1.DLL cannot load L0.DLL. Since L0.DLL does not have a capability set greater than or equal to L1.DLL, Rule 2 applies.

Case 2:

P.EXE holds Cap1 & Cap2

L1.DLL holds Cap1 & Cap2 & Cap3

L0.DLL holds Cap1 & Cap2 & Cap3 & Cap4

Process P is created, the loader succeeds it and the new process is assigned Cap1 & Cap2. The capability of the new process is determined by applying Rule 1; L1.DLL cannot acquire the Cap4 capability held by L0.DLL, and P1.EXE cannot acquire the Cap3 capability held by L1.DLL as defined by Rule 3.

2.3.2 Examples for dynamically loaded DLLs

The program P.EXE dynamically loads the library L1.DLL.

The library L1.DLL then dynamically loads the library L0.DLL.

Case 1:

P.EXE holds Cap1 & Cap2

L1.DLL holds Cap1 & Cap2 & Cap3

L0.DLL holds Cap1 & Cap2

Process P is successfully created and assigned Cap1 & Cap2.

When P requests the loader to load L1.DLL & L0.DLL, the loader succeeds it because P can load L1.DLL and L0.DLL. Rule 2 does apply here the loading executable being the process P not the library L1.DLL: the IPC load request that the loader processes is sent by the process P. The fact that the call is within L1.DLL is here irrelevant. Rule 1 & 3 apply as before and P does not acquire Cap3 by loading L1.DLL

Case 2:

P.EXE holds Cap1 & Cap2

L1.DLL holds Cap1&Cap2&Cap3

L0.DLL holds Cap1&Cap2&Cap4

Process P is successfully created and assigned Cap1 & Cap2. When P requests the loader to load L1.DLL & L0.DLL, the loader succeeds it because P can load L1.DLL and L0.DLL. Once again, Rule 2 does apply with P as the loading executable rather than L1.DLL, while Rule 3 ensures P acquires neither Cap3 nor Cap4.

2.4 Protecting system servers names

To prevent impersonation of critical servers, a specific system capability ProtServ has been created. When a process is granted this capability, it can register as a 'Protected Server' to the kernel under a specific name format, for instance starting by '!'. The kernel will reject any name registration starting by '!' if the calling process does not have ProtServ. However all servers from the protected server community must be trusted not to impersonate another server of this community. It is believed this simple mechanism will achieve a good enough level of security providing that ProtServ is granted only to a small set of services. If two server domains were proven not to be enough, nothing would prevent in the future adding a new domain linked to a new capability and name format.

2.5 Firewall-ing Symbian OS system servers

2.5.1 Capability Permission lifetime

With a kernel-mediated process capability model in place, specific policy decisions are delegated to the servers themselves. Each server would have to police itself and would do so differently - some would be more paranoid than others and may decide to check client capability settings per access or may refuse to allow access at all without the requisite capability bit set. Devolving security policy to a local decision at servers represents good abstraction as it means that security policy can be interpreted locally as desired. In addition,

interpretation of the presence of the capability bit could be done in one of four ways, namely blanket permission, process permission, session permission and selective access permission.

From a user point of view, it has become more and more difficult to identify the lifetime of a process or a session. For instance, in most smart phones, when the user accesses her Agenda, she does not know if a new Agenda process has been created or if an existing one is being re-used. In the same way, she does not have any idea, when making a telephone call, if the telephone application has created a new session to the phone server or re-use the previous one. It is our belief that users can understand only two ways of granting capabilities: the blanket and the selective access permission.

In line with this, there would be two possible alternatives for interpretation of the presence of a user-exposed capability bit at capability-aware servers:

Blanket permission: Capability is granted as a one off act. If the capability is not present when the process is created, the request fails. All access requiring system capability will be policed this way.

Single action permission: Capability is checked upon each sensitive API call to the server. This means that on each access to the sensitive API call, a dialog box created by the server asks the user if they wish to grant that capability. However the capability only lasts for the duration of the corresponding sensitive API call.

The user perception of this kind of permission must not be confused with when a server will check a required capability. For instance, an application may be granted PhoneNetwork as blanket permission but the phone server will check PhoneNetwork capability at each relevant call made by this application as the kernel will send the client capability set at each IPC call anyway. But as the user will not be asked to make a decision (to grant or not to grant PhoneNetwork for this call), this security check is transparent to the user.

The advantages of this design are:

- To take immediately into account any modification of the process blanket set of permission
- To maintain the time of check as close to the time of use as possible (to avoid any TOCTOU(time of check - time of use flaws)
- To avoid the server storing capability-related information for a process.

3 Software Installer – Enhancing perimeter security

The software installer has to support the process capability model to determine what capabilities can genuinely be granted to a piece of code to be installed after manufacture time. There are many ways to do it and this invention does not mandate any. In the following sections, we explore different scenarios based on the existence of a Public Key Infrastructure to illustrate how someone may choose to do it and which issues should be considered.

The software installer may support three main scenarios:

- The [un]installation/revocation of trusted (signed) applications
- The [un]installation/revocation of untrusted (signed) applications
- The [un]installation/revocation of unsigned applications

The trust distinction here is a function of the presence of a root certificate in the signing chain that corresponds to one of the root certificates stored on the device. In the next subsections, we examine these alternatives from a user perspective.

3.1 Installation of trusted (signed) software

In this case, approved client applications would be checked by a recognised authority. Software would then be signed, along with a list of the capabilities it required, into installation packages. Capabilities that could be signed for would be system and user-exposed capabilities. The chain of certificates included with the software is terminated by a root CA known by the software installer, in which case the user can install the software safe

in the knowledge that the root CA vouches for it. At install time, the software would be installed on the device without any user feedback, though on some devices it would be possible to examine the details of the signing certificate. Note that any third party software that sought system capability would be forced to go via this route.

3.2 Installation of untrusted (signed) software

In this case, the third party software would not check out against any of the root certificates. The software is signed by an unknown root CA in which case the user would be presented with a warning but could just go ahead and install the software anyway after viewing the contents of the certificate. The software installer would throw up a user dialog at install time, which would offer the user the option of installing the application plus a list of the capabilities the application wished to request. Capabilities that could be granted via this route would be in the range of user-exposed capabilities. In other words, it would not be possible for users to grant system capability to signed software that did not check out against one of the root certificates on the device.

3.3 Installation of unsigned software

The scenario here would be similar to 3.2. Third parties would have to request capabilities with their installation package. Again, only user-exposed capabilities would be available via this route - it would not be possible for users to grant system capability to unsigned software.

Figure 2 illustrates a possible process of installation when it is wished to let the user make discretionary decisions about the policy to apply to either an unsigned application or a signed application the root-signing certificate of which didn't match against any of the device root certificates.

3.4 Control Panel for revoking capabilities

In case it is wished to let the user make discretionary security policy decisions after install time, a software installer control panel dialog would be required to provide access to the capability settings currently granted to various installed third party applications. The user would be able to view and optionally grant or revoke user-exposed capabilities for installed third party code. In addition, it would be possible to view a corresponding code signing certificate (if necessary). The following table illustrates what the control panel dialog might look like:

Installed	PhoneNetwork	ReadUserData	WriteUserData	LocalNetwork	Certificate
Good.exe	Not granted	Granted	Not granted	Granted	Checked OK – click to view root signing cert
Bad.exe	Not granted	Not granted	Not granted	Not granted	Unsigned

CLAIMS

1. A secure mobile wireless device for a single user in which native executable code to be installed on the device is assigned a set of capabilities which define the protected resource(s) on the device which it can access.

2. The device of Claim 1 in which a capability grants access to a specific API (or range of APIs), file or other operating system protected resource(s).

3. The device of Claim 1 in which the executable code contains the capabilities it has been granted at install time.

4. The device of Claim 3 in which the executable code is permanently stored in a location in memory of the device which is only accessible to the trusted computing base to preserve the integrity of the capabilities assigned to the executable code.

5. The device of Claim 4 in which the loader refuses to load executables not permanently stored in a location in memory of the device which is only accessible to the trusted computing base, to preserve the integrity of the system.

6. The device of Claim 1 in which there are system capabilities which are mandatory and control access to the operating system kernel, file system and system data.

7. The device of Claim 1 in which for each client-server communication within the device, the device kernel will pass the client capabilities to the server.

8. The device of Claim 1 in which, for each library to be loaded in an executable, the loader verifies that this library is assigned a superset of the capabilities granted to the requesting executable.

9. The device of Claim 1 in which the capability is selected from a list of capabilities which comprise root capabilities, system capabilities and user –exposed capabilities.

10. The device of Claim 9 in which the user – exposed capabilities allow the following functions:

(a) Can spend subscribers money by using the phone network

(b) Can read users private information

(c) Can modify users private information

(d) Can send information to a local network without spending subscribers money.

(e) Can know the device location.

11. A software application adapted to run on a secure mobile wireless device in which the application is assigned some capabilities which define the protected resource(s) on the device which it can access.

12. An operating system adapted to run on a secure mobile wireless device in which the operating system requires native executable code requiring access to sensitive resources to be assigned some capabilities which define the protected resource(s) which it can access.

Figure 1

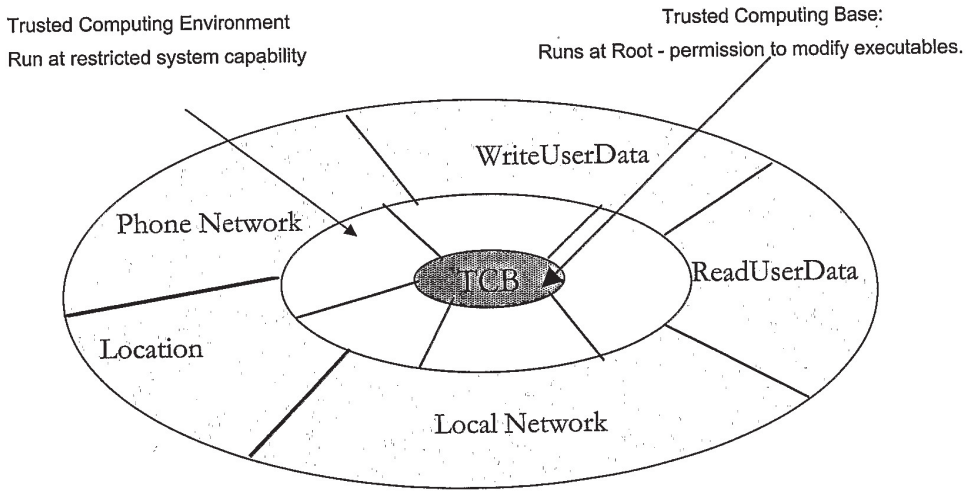
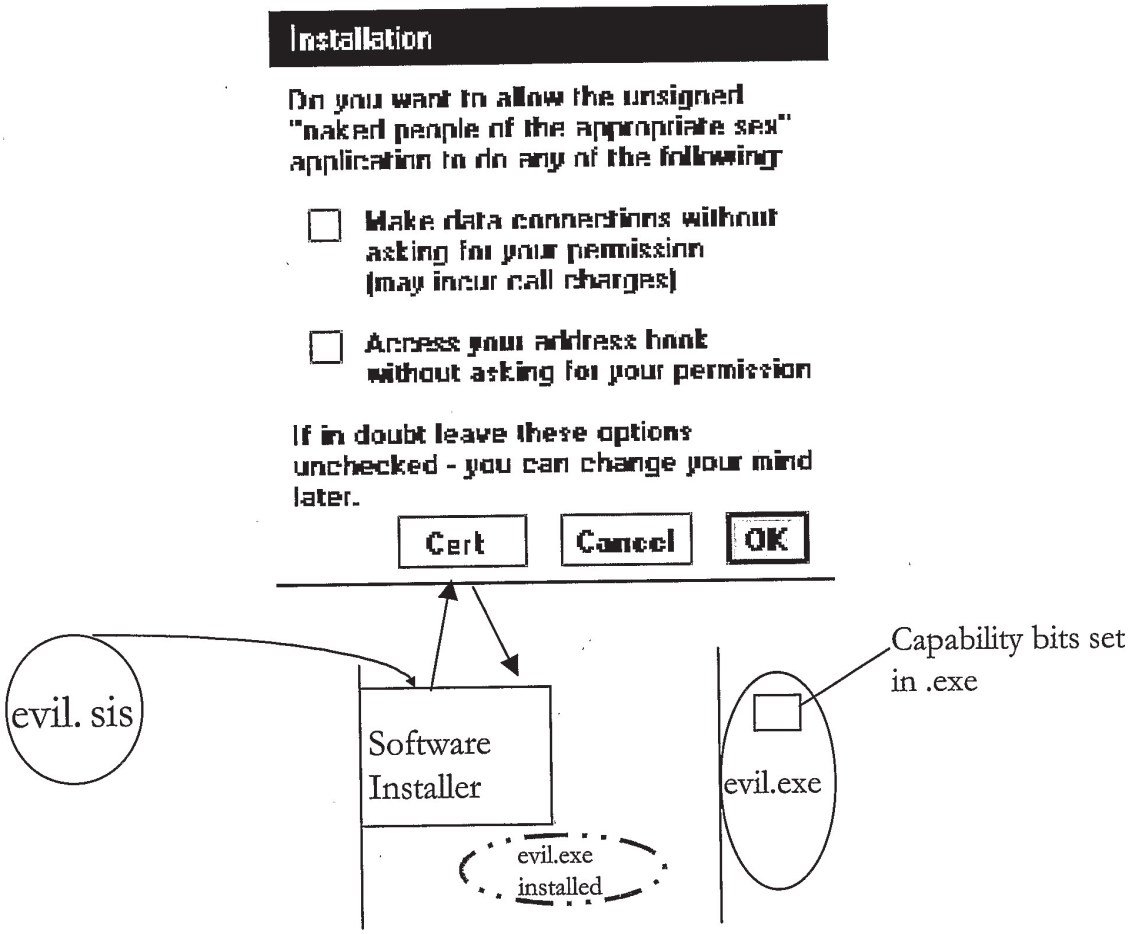

Trusted Computing Environment
Run at restricted system capability
Trusted Computing Base:
Runs at Root - permission to modify executables.
WriteUserData
Phone Network
TCB
ReadUserData
Location
Local Network

Figure 2


Installation
Do you want to allow the unsigned "naked people of the appropriate sex" application to do any of the following
Make data connections without asking for your permission (may incur call charges)
Access your address book without asking for your permission
If in doubt leave these options unchecked - you can change your mind later.
Cert
Cancel
OK
evil. sis
Software Installer
evil.exe installed
evil.exe
Capability bits set in .exe