BO LI, ELENA RESHETOVA, AND TUOMAS AURA

Symbian OS platform security model

Bo Li is a second-year student in the master's program in security and mobile computing at Aalto University, Finland. He got his bachelor's degree in communications engineering in 2008 from Fudan University, China.

Bo.Li@tkk.fi

Elena Reshetova is a senior security engineer at Nokia, as well as a postgraduate student at Aalto University. She is interested in various research areas related to platform security, security aspects of networking, and cryptography.

elena.reshetova@nokia.com

Tuomas Aura is a professor at Aalto University, Finland. His research interests are security and privacy in communications networks and online services.

tuomas.aura@tkk.fi

THE SYMBIAN OS BECAME FULLY OPEN sourced in February 2010, which opens even more possibilities for application developers to understand and analyze its security solution. We present a short introduction to the software features of Symbian platform security: three trust tiers, capability model, data caging, and the Symbian signed process. We also try to compare the security solution with the classical design principles in this area, as well as briefly discuss general design challenges and potential weaknesses.

Introduction

With the development of mobile devices and mobile computers, more and more people rely strongly on them. People use mobile devices and mobile computers to arrange their schedules, contact each other, process emails, and share rich media content. People believe it is safe to do so because it feels secure just knowing it is "right there with you" [8]. But, in fact, even their manufacturers agree that all mobile devices are facing various security threats and attacks [3]. Moreover, the more widespread the phone's operating system is, the higher the risk that it will be targeted by attackers, especially if it allows execution of third-party applications. According to Symbian site information, over 80 million devices running Symbian OS were sold in 2009 [7]. The public development tools for the Symbian platform have been easily accessible for a long time, and Symbian also supports execution of add-on applications. So the Symbian OS had and still has a strong need for a well-designed security solution.

Despite the fact that modern smartphones operate like minicomputers, several characteristics suggest that their security solutions cannot be the same as the general security solutions for PCs.

Compared with the users of computers, the users of mobile devices have some particular expectations. People believe they should be able to make an emergency call whenever necessary. This means that, no matter how many applications are running, the mobile phone must have very high reliability; people also do not wish to restart their mobile phones every day as they do with desktop computers. This requires the mobile OS to be very stable. And although it seems more and more necessary to install antivirus software on smartphones

now [13], most users still do not expect to install any antivirus application on a mobile device, at least not to have to install it themselves.

Alongside these particular expectations from users, the mobile device itself has some constraints which should be taken into account when designing a security architecture. This is mainly due to the limitations, especially when compared with computers, of the mobile devices' hardware: low processing power, limited memory and battery, small screen size, and inconvenient keyboard. All of these require a light but efficient security architecture.

Another important characteristic of mobile devices indicates even more serious vulnerabilities and challenges. "Phones are primarily used to communicate. They are built to make communication as easy as possible," notes Aaron Davidson, CEO of antivirus company SimWorks. "Phone users want to communicate, and viruses want to be communicated" [12]. Every mobile phone has a contact list; if one phone is infected by malware, the people on the contact list will be vulnerable to attack.

To sum up, mobile device security has some similarities with computer security, but faces additional challenges.

Basic Design Principles

Saltzer and Schroeder's classic article on computer security design [9] lays out essential principles to follow when designing a security solution. In this section we will explain some of these principles, as well as show how they were applied in the Symbian platform security design.

- Economy of mechanism: Economy of mechanism means the design should be small and simple. The simpler a system is, the fewer vulnerabilities it should have. When we apply this rule to the Trusted Computing Base (TCB) design, fewer trusted components will result in fewer internal trust relationships and a simpler system. In Symbian OS, the only fully trusted part is the Trusted Computing Base, which is small enough to be well tested and reviewed.
- Fail-safe defaults: In the security field this principle means that, by default, access to a system's resources should be denied. This is needed to prevent the situation where some sensitive resource is forgotten from the list of resources but must be protected. When applied to users of a computer system, the principle may also mean that a user should be given a secure default choice when presented with a security question. Symbian follows the "deny by default" rule in its design, but the graphical interface does not always advise a secure choice for a user. For example, when an application tries to connect to the network, the user is asked whether it is allowed to do so or not. The "yes" button (meaning "allow connection") is situated on the left and is the button the user usually presses. So, the user tends to choose the unsafe option presented in that graphical interface.
- Complete mediation: Every time any system resource is accessed, there should be a check whether this action is authorized for a caller. Symbian Platform Security includes access control checks on all platform resources, but it does not provide any mechanism for applications that need to provide flexible access control to the applications' own resources. We will discuss this problem in more detail in the discussion section.
- Open design: The security solution should not try to build security by obscurity, but, rather, rely on secure storage for cryptographic keys and other security material (e.g., certificates, initialization vectors, etc.). Nowadays, Symbian almost fully follows this principle by open sourcing the operating system itself, including the security solution. However, the information about hardware security support is still not fully public.

- Principle of Least Privilege: Least privilege means that a process should be allocated sufficient privileges to accomplish its intended task but no more. However, this principle does not speak about any granularity of privileges, so often it is interpreted quite broadly. Symbian Platform Security addresses this principle by its capability model.
- Psychological acceptability: Psychological acceptability requires that the user interface should be intuitive and clear. An elegant interface, combined with a precise definition of its behavior, promotes the correct and secure use of the system. For example, Symbian should convert the various low-level capabilities into several user-friendly descriptions, such as: "The application wants to open the Bluetooth connection: allow or not?"

One principle that needs to be added to this classic list is the notion of "trusted root," meaning that there should be a chain of trust going from the hardware to upper parts of the operating system in order to secure the end of the chain from offline attacks.

Concepts of the Symbian OS Security Model

In this section we will introduce three important concepts in Symbian OS platform security: three trust tiers, capabilities, and data caging.

TRUSTED COMPUTING PLATFORM: THREE TRUST TIERS

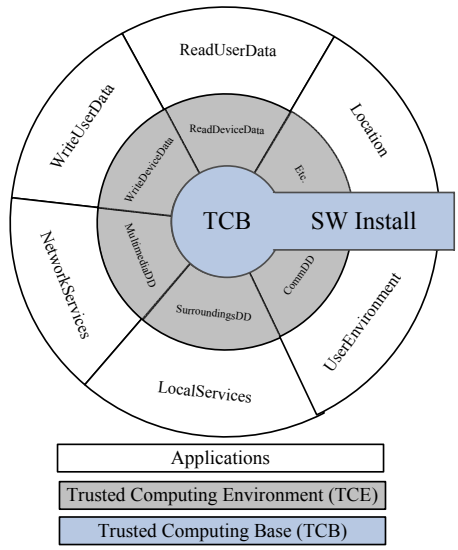

FIGURE 1: THREE TRUST TIERS [8]

The first essential concept in Symbian OS platform security is the "three trust tiers" model, shown in Figure 1. Figure 1 depicts a trusted computing platform comprising a Trusted Computing Base (TCB), a Trusted Computing Environment (TCE), and Applications.

- Trusted Computing Base (TCB): The heart of a trusted computer system is the Trusted Computing Base (TCB), which contains all of the elements of the system responsible for supporting the security policy and supporting the isolation of objects (code and data) on which the protection is based [10]. TCB is the most trusted part of Symbian OS. It has three components: the operating system kernel, the file server (F32), and the software installer. All of them have been carefully checked to ensure that they behave properly and can be completely trusted. The kernel has the responsibility for managing all the processes and assigning appropriate privileges to them.

The file server is used to load the code for running a process. The software installer is used to install applications from files packages. It also checks the digital signatures of the packages to validate the privileges requested for the program binaries, so it is the "gatekeeper" for the mobile device.

- Trusted Computing Environment (TCE): The next tier is the TCE, which is a set of different system servers running with different privileges. TCE consists of trusted software provided in the mobile phone by Symbian and others, such as the UI platform provider and the mobile device manufacturer [8]. The TCE usually implements a system server process and is less trusted, and thus it has only limited privileges to perform a defined set of functions. With this design, Symbian can ensure that failure of one server will not threaten the whole system, and it is also impossible for a misbehaving system server to compromise the security of another server, since it does not have access to the same APIs [16].
- Applications: Third-party applications are the last tier in this trusted computing model. As they are not highly trusted, they only have privileges to access the services that are unlikely to pose a security risk, and they are not allowed to access critical low-level operations. There are actually two kinds of applications: signed applications and unsigned applications. If a signed application wants to use some service provided by TCE, it must request that TCE run the service on its behalf. TCE will only accept the request if the application has been granted enough privileges. The unsigned applications, however, can only perform some operations that do not require privileges, or operations that may be allowed by the user. It is necessary to point out that there are a lot of useful operations which can be performed by unsigned applications and which do not require a signature because they are not security-relevant. A user may allow some operations to be performed even if an application is not signed, via a user-prompt dialog. An unsigned application, then, is not necessarily worthless software or malware: an application should have a signature only when it is necessary.

To summarize, the TCB is the most trusted part and controls access to all sensitive low-level operations such as the communication device drivers. The TCB ensures that only the privileged TCE components are able to perform these operations. The TCE then provides system services, such as the telephony server, to applications. This is the only way for the applications to perform the low-level operations [8].

CAPABILITIES DETERMINE PRIVILEGE

The second basic concept in the Symbian OS platform security model is capabilities. A capability is an unforgeable ticket, which when presented can be taken as incontestable proof that the presenter is authorized to have access to the object named in the ticket [9].

Capabilities Basics in Symbian OS

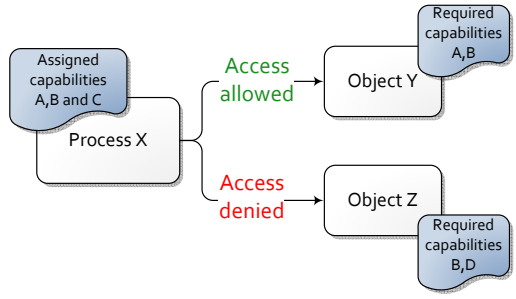

FIGURE 2: ACCESS CONTROL BASED ON CAPABILITIES

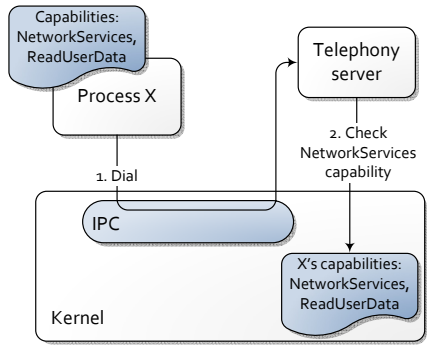

FIGURE 3: ACCESS TO SERVICES VIA THE TCE

In Symbian OS, each process runs with a list of capabilities, and these capabilities determine whether a given resource can be accessed by the process. As shown in Figure 2, process X (with capabilities A, B, and C) can access the resource Y, which requires capabilities A and B, but cannot access the resource Z, which also requires capability D. The concrete form used to access the resources is through APIs, and different APIs will require different capabilities for the services they provide.

The Symbian OS uses capabilities to represent access privileges. Each live process and its corresponding capabilities are listed and monitored by the system kernel. A process will ask the kernel to check the capabilities of another process before deciding whether to carry out a service on its behalf [8]. Figure 3 shows how a well-known example of a client application X accesses the Dial service via the telephony server (one component of the TCE). First, process X sends the Dial request to Inter-Process Communication (IPC, part of the kernel), which then delivers this request to the telephony server process. The Dial service requires NetworkServices capability, and the system kernel holds the capability list of X. The telephony server then checks X's capability list in the kernel and finds the capability NetworkServices there, enabling access from X to the Dial service. The capabilities architecture is discrete but not hierarchical; each capability is only associated with its corresponding resource, and capabilities do not overlap.

When the capabilities are designed and divided, the outcome is actually a trade-off between the principles of "economy of mechanism" and "least privilege" introduced in the section on Basic Design Principles, above. If only " economy of mechanism" is considered, the system will have only one capability with full access to all resources. This system is simplest and is easy to review, but it obviously doesn't work. If "least privilege" is the only principle followed, every API would have a dedicated capability, amounting to over 1000 capabilities in the Symbian OS. This is also impossible to manage and realize. The number of capabilities should be small enough to manage but also reasonably big enough to show the differences between different privileges. To meet this requirement, Symbian OS defines 20 capabilities with their special privileges.

Categories of Capabilities in Symbian OS

The capabilities are divided into several categories according to the three trust tiers model; basically, the three kinds of capabilities are TCB capability, system capabilities, and user capabilities. The categories and brief descriptions of the capabilities are shown in Table 1.

TABLE 1: CATEGORIES OF CAPABILITIES IN SYMBIAN OS [3, 6]

Category	Capability	Brief Description
User Capabilities	LocalServices	Grants access to sending or receiving information through USB, IR, and point-to-point Bluetooth profiles
	ReadUserData	Grants read access to confidential user data
	WriteUserData	Grants write access to confidential user data
	NetworkServices	Grants access to remote services such as dialing a number or sending a text message
	UserEnvironment	Grants access to recording the user's voice and using the camera
	Location	Grants access to data about the location of the device
System Capabilities (extended capabilities)	SwEvent	Grants the right to simulate key presses, pen input, and capture such events from any program
	ProtServ	Grants the right to a server to register with a protected name
	TrustedUI	Grants the right to create a trusted UI session, and, therefore, to display dialogs in a secure UI environment
	PowerMgmt	Grants the right to kill any process in the system, to power-off unused peripherals, and to cause the mobile phone to switch its machine state
	SurroundingsDD	Grants access to logical device drivers that provide input information about the surroundings of the device
	ReadDeviceData	Grants read access to sensitive system data
	WriteDeviceData	Grants write access to sensitive system data
System Capabilities (manufacturer-approved capabilities)	CommDD	Grants access to communication device drivers
	DiskAdmin	Grants the right to disk administration functions that affect more than one file or directory such as formatting a drive
	MultimediaDD	Controls access to all multimedia device drivers (sound, camera, etc.)
	NetworkControl	Grants the right to modify or access network protocol controls
	AllFiles	Grants visibility to all files in the system and extra write access to files under /private
	DRM	Grants access to alter DRM-protected content
TCB Capability	TCB	Grants access to the /sys and /resource directories in the device

- TCB is fully trusted and has the highest privilege, so there should be a capability owned only by TCB. This capability is just called *TCB*, which allows one process to create new processes and assign capabilities to them. Generally, the TCB capability is not granted to other parts besides TCB, as it is really critical and closely related to the integrity and security of the whole OS.
- System capabilities are assigned to access sensitive operations. Basically, system capabilities are divided into two parts: manufacturer-approved capabilities and extended capabilities. Manufacturer-approved capabilities are the most sensitive capabilities. They are reserved by the mobile device manufacturers and are enforced by TCB. Extended capabilities control access to higher-level services and are enforced by TCE.
- User capabilities are also called "basic capabilities." Following the psychological acceptability principle, user capabilities should be simple and easy to understand, as they are defined for user interaction. User capabilities are

usually granted to third-party applications to access the service provided by the TCE, and the TCE is responsible for enforcement.

Capability Rules

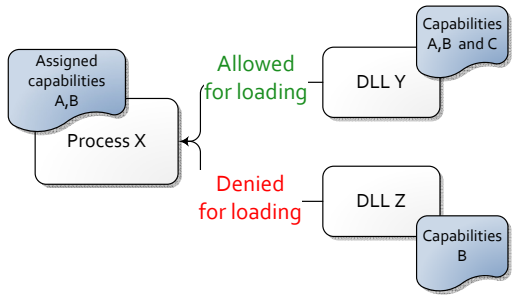

FIGURE 4: LOADING OF DLLS

There are two basic capability rules in the design of Symbian. The first rule is that every process has a set of capabilities and its capability never changes during its lifetime [8]. This rule is included in Symbian OS for simplicity and security. The second rule states that a binary cannot load any DLL that has fewer capabilities than itself (see Figure 4) [6]. This prevents untrusted code being loaded into sensitive processes. The capabilities of a DLL do not affect the capabilities of the process that loads it; process capabilities are entirely defined by the capabilities of the EXE.

FILE ACCESS CONTROL SYSTEM

The last essential principle in Symbian OS platform security architecture is the file access control system. It is designed to protect the integrity and confidentiality of critical files. The concrete solution is called "data caging" or "file caging."

Basic Data Caging Principles

The original idea of data caging is simple: put the most important treasures into a coffer and enforce strict access control to it, so that no one else can easily access the treasures inside it. In Symbian OS, data caging is achieved by providing special directories that "lock away" files in private areas [8]. Data caging is a lightweight mechanism, as the access control of the files only relies on their path; thus, it works without another access control list, which would consume additional system resources.

Data caging also follows the Principle of Least Privilege, by which a process cannot access these special directories unless it is authorized. If we take a close look at this aspect, there are two special capabilities closely related with data caging: TCB and AllFiles. These are enforced by the TCB. If these capabilities are granted, they will apply to all the files in the system. So they are really critical and should not be assigned to one specific application. In Symbian, the solution for this problem is to provide several different system services with more limited privileges such as ReadUserData, WriteUserData, and so on, so the access permission is narrowed and divided.

Caged Paths in Symbian OS

In Symbian OS, there are three caged top paths: \sys, \resources, and \private. All their subdirectories are also access restricted. Only the processes with TCB and AllFiles capability may access these directories.

- \sys
  There are two important subdirectories under \sys: \sys\bin and \sys\hash. \sys\bin is the default path where all the binary files are stored. The pre-installed binaries are located at z:\sys\bin, which is a path in the ROM; and add-on applications are under \sys\bin on some writable devices. In Symbian OS, there are two rules defined for this path. First, only TCB can write new executables into this path (via SWInstall) or execute the binaries under this path (via F32). Second, only the binaries under this path are runnable; all the other paths will be ignored by the loader. This ensures the security and integrity of the system, since only TCB, not malware, can create a new process.
  \sys\hash is used to check whether a binary on removable media can be launched. It is managed by the software installer. The installer will check the presence and correctness of the hash, and the hash entry for a binary will not be generated unless the installation has been validated. This mechanism ensures the security of running a binary on removable media.
- \resources
  Similar to \sys, \resources exists in both ROM and writable storage. The files under \resources are read-only for most of the applications, because this path is used to store the resource files, which will not be changed after installation. Only an application with TCB capability (installer application) can modify the files under this path, ensuring that resources installed for one application will not be destroyed by other applications.
- \private
  In Symbian, every EXE is assigned its own caged subdirectory under \private. The subdirectory is open to its own process but is inaccessible to other normal processes. Particularly, if two processes are loaded from the same EXE, they share the same subdirectory [8]. The subdirectory under \private is the default path to store the data of the process, but the developer can also choose to store the data of their application in a public directory.

Software Installer in Symbian OS

The software installer plays an important role in Symbian OS platform security, as it is one of the gatekeepers of the system. It is responsible for ensuring that add-on native software is installed on the mobile phone with the correct set of security attributes [8]. The software installer discussed here only handles the installation of the software directly running on Symbian OS, but not the software running on a Java Virtual Machine.

There are three main tasks for the installer of Symbian OS: first, to validate and install the native Software Install Scripts (SIS files) on the mobile device; second, to validate pre-installed software on removable media; lastly, to manage upgrades and removals and provide the package management service to the rest of the system.

IDENTIFIERS

Identifiers are used by the servers to identify processes. There are several different kinds of identifiers in Symbian OS which are important for the software installers. They are the Secure Identifier (SID), Vendor Identifier (VID), and package UID (pUID).

The SID is used to identify the binaries, and it is locally unique. The VID is used to distinguish the origin of the executable. If an application needs a VID, it must be signed [17]. The pUID is the identifier of a package, or set of

files, that forms an installable unit [8]. For example, if we download one installable package of a game from Electronic Arts (EA)'s Web site and install it into our Symbian mobile phone, the pUID is the ID of this package, the VID is the ID of the EA company, and the SID is the ID of the executable binary of the game after installation.

Identifiers are split into protected and unprotected ranges. Every identifier will be allocated in these ranges, according to or not whether the application will be signed.

SIS FILES AND REMOVABLE MEDIA

SIS (Software Installation Script) files in Symbian are used to deliver software packages to mobile phones for installation and can be installed from a PC, downloaded via a browser, or sent to a mobile device by MMS [8]. When the software installer validates the SIS files, it checks a lot of parameters. In general, it checks the capabilities the software wishes to use for authorization. First, it checks to see whether the software is signed; second, it checks that the certificate chain can be followed back to the root; then it checks whether the certificate has been revoked. It can also mark root certificates as mandatory, ensuring that all installable packages must be signed with it.

After the validation of authority, the installer compares the capabilities requested by the installable package with those that the root certificate can grant, and calculates the largest set of capabilities that can be granted. A configuration option allows the end user to grant additional capabilities to the SIS file if these capabilities are not granted by the installer directly. If the user refuses to grant the additional capabilities, the software will not be installed [17]. Another possibility is to ask for the user grantable permissions during runtime, but this is annoying to some users. The overall process of installing a signed application is shown in Figure 5.

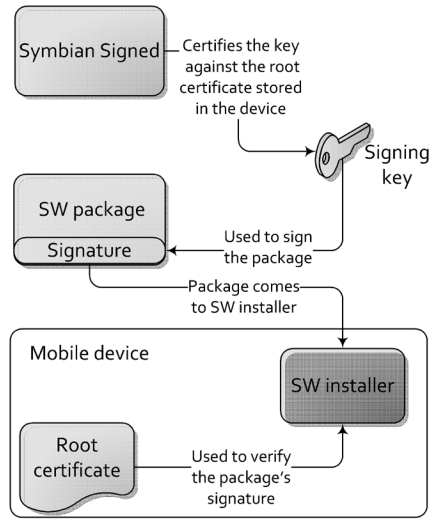

FIGURE 5: INSTALLATION PROCESS FOR SIGNED PACKAGES

Besides the installation from the SIS files, the software can also be installed onto removable media. The most important security issue is to prevent tampering with the binaries installed on removable media. This is achieved by the software installer computing and storing (in the tamper-proof \sys\ directory) a reference hash of the program file on installation, which is recom-

puted and compared whenever the program is to be run [17]. If the reference hash does not exist, or the two hashes do not match each other, the program will not be executed.

Code Example

There are many Symbian C++ code examples of differing complexity on the Internet, as well as many developers' forums where people ask security-related questions. The most common question regarding access control concerns capabilities, which are needed in order to use a particular class or function. Let's take as an example a class CTelephony. The official API reference documentation [1] explains that this is the class that can be used to make a call (and for some other purposes, too). The actual function which initiates a call is CTelephony::DialNewCall, and the documentation says that it requires the user capability "Network Services" to use this function. The full example of the class, which can be used in order to handle a call, can be found in the Forum Nokia pages [2]. Listing 1 shows a small fragment of the code, which was modified in order to check the return status of the function.

```
1 iTelephony = CTelephony::NewL();
2 CTelephony::TTelNumber telNumber(aNumber);
3 iCallParams.iIdRestrict = CTelephony::ESendMyId;
4 iTelephony->DialNewCall(iStatus, iCallParamsPckg,
5                                   telNumber, iCallId);
6 User::WaitForRequest(iStatus);
7 if ( iStatus == KErrPermissionDenied )
8   {
9     // if we are here, then the call
10    // was stopped by the access control
11 }
```

LISTING 1: MAKING A CALL CODE EXAMPLE

The code (Listing 1) simply creates a class object CTelephony (line 1), prepares the function parameters, such as phone number (line 2) and CallID restriction setting (line 3), makes a new call (lines 4–5), and checks for the "KErrPermissionDenied" error code in order to check whether access was denied (lines 7–11).

Symbian Signed Model

The purpose of Symbian Signed is to enable third-party access to protected APIs and give users a basis for trusting third-party applications. If platform security is like a firewall locking down access to the system, then Symbian Signed is the mechanism that allows developers to negotiate entry through the firewall [14].

There are three different options for Symbian Signed: Open Signed, Express Signed (online and offline), and Certified Signed [15]. Open Signed can grant user and system capabilities to an application, and Open Signed offline can also grant the restricted capability set. However, the signature is limited to a device IMEI number and therefore is supposed to be used only for development purposes. The Open Signed online option also doesn't require a Publisher ID, which costs money for developers and can be granted only to a member of a company or organization. The Express Signed enables developers with a Publisher ID to sign and distribute their applications without IMEI restrictions. The cost per application's signature is small (10

euros), and the submitted applications are randomly tested for compliance with the Symbian Signed Test Criteria. This signature can be used for commercial purposes, but the restricted capability set can be granted to such an application. Certified Signed is the most common option for commercial software developers and can grant, in some cases, even access to manufacturer capabilities.

THE SIGNING PROGRESS

According to [8], the signing process consists of four key steps: (1) development, (2) developer authentication, (3) testing against industry-defined criteria, and (4) signing against the mobile device's root certificate.

The basic mechanism of the signing model works as follows: sensitive APIs are protected by capabilities; if the programs wish to access protected APIs, they must be granted the corresponding capabilities. The signing process is responsible for granting these capabilities, and it will encode the capabilities into digital certificates accompanying the application. The application certificates are validated at install time and the accesses to the critical APIs are also policed at runtime. Finally, when the process is running, the resource and privacy of the application will be protected by the additional security mechanisms of the system.

Analysis and Discussion

Earlier, the Symbian OS platform security was closed, no one was talking about the real implementation, and there were no publications about it. Nowadays, Symbian is fully open sourced, which is a very good step towards understanding its security solution. As there is no absolutely right solution, in this section we will discuss some problems of Symbian OS platform security and try to suggest some possible solutions. But before we start to go into detail, it is important to remember that while analyzing the solution to any problem, the constraints and given environment should be taken into account. One big constraint on the devices running the Symbian OS, at the time when the platform security was designed, was the processing power, which had a huge impact on the design of the security solution. However, today the processing power is no longer so constrained, and some decisions can be revised.

LIMITATIONS OF THE CAPABILITY MODEL

First, to ensure the security of the OS, Symbian OS restricts the capability set so that no one else can define a new capability. Of course, a fixed set of capabilities is easier to manage both inside the platform and during the Symbian Signed verification process, but sometimes having a flexible capability set is a very useful feature. For example, Symbian OS currently has no way for an application which would like to protect its own data with its own capability to enforce its own rules on how this capability can be granted to other applications.

Another problem is that the capability set is not fine-grained enough. As previously discussed, in order to keep the capability system simple, Symbian only defines 20 capabilities, not enough nowadays to achieve user-desired security. A lot of APIs are associated with the same capability, such as ReadUserData. Since user data is not fine-grained, one process can access either all user data or nothing. The result is that although a process—say,

a music player—should only be allowed to access part of the user data (the music files), through this capability it can also access data that should be off limits, such as the user's photos, documents, and even private email if they are stored in the user's folder.

Many people would argue that users should not be bothered with fine-grained security, which is of course true, but where is the border between needed flexibility and fine-grained solution? Who should define the granularity level? One possibility is to leave this choice to application developers and create a flexible mode which allows them to specify the granularity they need. Some developers prefer not to know much about platform security features, however, but simply to develop their applications "as before." Having a flexible but complicated system may frighten some beginners starting to develop applications for a particular platform.

Another possibility is to let the application certification authorities define the granularity, since they have to maintain the tools to check what permissions can be granted to a particular application. The last choice is to define the granularity of the level that an end user understands. Taking the end user into consideration is especially important if the user has to make any access control decisions on a platform, simply because users have to understand the ramifications of their actions in order to make a correct choice.

The question of granularity, then, should balance the needs of application developers, certification authorities, and end users. In the case of access control, end users become more and more security educated over time, especially users of high-end devices such as smartphones and minicomputers, as the generations change. Today, mobile device users understand the difference between their media files and email contents. However, Symbian OS platform security was designed when the notion of mobile device security was not very commonly understood, and ordinary users were not educated enough to think about their security and privacy to the deeper degree that Symbian proposed. At that time, it was reasonable to make the capability set small and simple, but for future solutions, the granularity probably should be increased.

PROBLEMS OF THE USER PROMPTING

As we mentioned before, when applications are installed or executed, the system will sometimes ask whether the user would like to grant some additional capabilities to the application; this can be dangerous [18]. Most applications downloaded from the Web are not signed, and, as normal users usually do not have enough security knowledge, this mechanism leaves open the possibility for malware to ask for some "user capabilities" from the users directly and then misuse the privilege. Many usability studies have shown that users tend to click "yes, yes, ok" in response to questions, after having become familiar with the process, which happens quickly. Moreover, since Symbian platform security does not fully provide a Trusted UI, there is no guarantee that a user prompt is fully secure and that it's actually an end user who presses the "allow" button on the screen.

Of course, many mobile operating systems use user prompts, mostly to guard themselves against liability, and put the consequences of a bad security decision on the user's shoulders. So how can we help the users to make better security decisions? What can be an alternative to a user prompt? One possible solution might be to allow a user access to a good software reputation system during the installation process. Such a system, if trusted and easy to use, may help users to make the right security decisions. Reputation

systems are hard to build, but if built correctly, they could provide a great service for users.

ISSUES WITH THE SYMBIAN SIGNED MODEL

Before starting to speak about issues with the Symbian Signed Model, let us recall its purpose. Symbian Signed allows third-party application developers to certify their software and therefore to get access to needed protected resources. This process is very important for any operating system that wants to support third-party applications and still check their quality. However, the process should be organized very carefully in order not to become a nightmare for applications developers and process maintainers.

Most software submitted to the Symbian signing process is checked by an antivirus engine; only some samples are checked manually. As the antivirus engine cannot guarantee detection of all potential malware, it is possible for evil codes to pass the signing process and acquire a digital signature. Recently, a Trojan called Sexy Space passed the security check of Symbian and obtained the digital signature [11]. Thus, there is always a possibility that malware can bypass the binary checks of Symbian Signed.

Also, the Symbian Signed process used to be quite painful for developers: it is slow, costly, and difficult. If your application needed to be certified for the sales version, for example, it used to take at least seven steps and one week to wait for the result. You also had to pay for the Publisher ID (200 USD/year) and the testing costs (at least 250 EUR) to get the signature [15]. The situation is slowly changing now, since the platform became open sourced and processes are evolving, but originally the service was not well executed.

This issue led to the current situation in China. The biggest Symbian fans community [5] in China made a survey on the N95 discussion board [4] asking if users wanted to "crack" their mobile phones. "Crack" here basically means disabling the installer's capability check mechanism to bypass the Symbian Signed problem. The results, which we translated from Chinese, showed that over 70% of the 1127 active users participated in the survey would like to do so (or already had done so) for a number of reasons. One reason is that they wanted to install some cool application which was not certified by Symbian Signed but which requires some system capabilities. Some other users bypassed the system by submitting the IMEI code of their devices in order to get their own developer certificates, so they could self-sign the application and therefore allow it to access needed resources on the devices. This simple example shows that if the process of application submission and certification is problematic, users and developers can always find a way to bypass it.

The last issue is that it is actually difficult for developers to know the exact capability set for their applications. As there is no automatic tool for the developers to detect the capabilities needed by their software, sometimes they just try to include as many capabilities as possible, whether or not the capability is needed.

Conclusion and Recent Developments

Symbian, as a leading mobile operating system, has a complete set of security solutions. The design of Symbian Security resolves the most important issues of mobile device security: the reliability of the OS, and the integrity and privacy of the critical data on the mobile device. It follows several classic design principles and makes careful decisions between trade-offs, given

the constraints imposed on the platform security solution at the time of its creation. The three essential concepts ensure the security of the system, while the software installer and signing model also keep the platform open for third-party developers. However, Symbian also has some weaknesses and drawbacks, which we hope will be taken into account when new platform security solutions for mobile devices are designed.

Recently a number of new OS security designs for mobile devices have been introduced, such as the Android and Maemo security frameworks. Their main difference from the Symbian solution comes from trying to utilize basic existing UNIX security in order to implement the desired functionality. The Maemo security solution tries to overcome a number of the earlier mentioned issues by providing an extensible set of capabilities (called "resource tokens" in Maemo), better support for development tools and processes, and a special mode of device operation where advanced developers can get almost full control over their devices. Time will tell, based on actual usage by users and developers, whether any of these frameworks will succeed, which is possible only if the lessons are learned from such examples as Symbian OS security.

REFERENCES

[1] CTelephony class API reference: http://carbidehelp.nokia.com/help/index.jsp?topic=/S60_5th_Edition_Cpp_Developers_Library/GUID-35228542-8C95-4849-A73F-2B4F082F0C44/sdk/doc_source/reference/reference-cpp/ETel_3rd_Party_API/CTelephonyClass.html.

[2] Example of CTelephony class usage: http://wiki.forum.nokia.com/index.php/Make_call_with_CTelephony.

[3] Nokia Online Documents: Symbian security model: http://www.forum.nokia.com/document/Cpp_Developers_Library/?content=GUID-232258EC-D3B4-4D72-B12B-FFC34F070B4B_GUID-644A092A-6999-46F8-A81F-363591CC0C03.html.

[4] Symbian China community discussion: http://bbs.dospy.com/viewthread.php?tid=2474219&extra=page%3D1%26amp%3Bfilter%3Dpoll&bbsid=147.

[5] Symbian China fans community: http://bbs.dospy.com/.

[6] Symbian SDK: http://www.forum.nokia.com/info/sw.nokia.com/id/05c63dfd-d6e9-4c0e-b185-d365e7001aeb/S60-SDK-0548-3.0-f.3.215f.zip.html.

[7] Symbian, "The Future of Your App": http://www.symbian.org/yourapp.

[8] Craig Heath, *Symbian OS Platform Security: Software Development Using the Symbian OS Security Architecture* (Wiley, 2006).

[9] Jerome H. Saltzer and Michael D. Schroeder, "The Protection of Information in Computer Systems," *Proceedings of the IEEE*, vol. 63, September 1975, pp. 1278–1308.

[10] Donald C. Latham, Department of Defense Trusted Computer System Evaluation Criteria, DoD 5200.28-STD, December 1985: http://csrc.nist.gov/publications/history/dod85.pdf.

[11] George Lawton, "Sexy Space Threat Comes to Mobile Phones," *Computing Now*, August 2009.

[12] Neal Leavitt, "Mobile Phones: The Next Frontier for Hackers," *Computer*, April 2005, pp. 20–23.

[13] Robert Lemos, "A Moving Target," *PC Magazine*, June 2006, p. 124.

[14] Ben Morris. *Platform Security and Symbian Signed: Foundation for a Secure Platform* (Symbian Developer Network, 2008): http://developer.symbian.com/main/downloads/papers/PlatSec_and_Symbian_Signed.pdf.

[15] Ben Morris and Ashlee Godwin, *A Guide to Symbian Signed*, 3rd edition (Symbian Software Ltd, 2008): http://developer.symbian.org/wiki/index.php/Complete_Guide_To_Symbian_Signed.

[16] Joe Odukoya, Elise Korolev, and Ashlee Godwin, *Platform Security for All* (Symbian Software Ltd, 2008): http://developer.symbian.com/main/documentation/books/books_files/pdf/Plat+Sec+FINAL+-+WEB.pdf.

[17] Mark Shackman, *Platform Security: A Technical Overview* (Symbian Developer Network, 2008): http://developer.symbian.com/main/downloads/papers/plat_sec_tech_overview/platform_security_a_technical_overview.pdf.

[18] Niu Xuelian and Ling Li, "Research and Improvement of the Symbian OS Kernel Platform Security Design," *Computer Engineering*, June 2006, pp. 194–196.