

Issues in Symbian S60 platform forensics

Antonio Savoldi, Paolo Gubian

(Department of Electronics for Automation, University of Brescia, Via Branze 38, I25121 Brescia, Italy)

Abstract: This paper aims at presenting the real and complete structure of the Symbian S60 v3 (third edition) platform file system, which is considered one of the most secure and armored systems issued by Symbian Ltd.^[1] in the last two years. By analyzing the unique features of security, which have been applied at the file system level, it will become clear how to develop applications for digital forensic purposes. Moreover, the limitations and main issues related to accessing the observable portion of the file system will be pointed out, and possible solutions will be presented.

Key words: Symbian OS; file system; data caging; digital forensics

1. Introduction

Nowadays embedded systems might be considered one of the most pervasive pieces of electronic equipment in our society, able as they are to keep plenty of sensitive and probatory data which are fundamental in a digital forensic scenario. As a matter of fact, a state-of-the-art smartphone, which can be considered as a de facto standard in the embedded device category, might contain phone basic information and SIM-card data, contacts list, caller groups, log records (missed, outgoing and incoming calls), SMS, MMS and E-mail messages (with service data records), calendar events schedule, To-Do items, text notes, photos, video, sound, Java and other files saved in the phone memory or on the flash card, and voice records. Indeed, it is not rhetorical to say that such a platform is getting closer and closer to the traditional desktop/laptop systems, in terms of

multimedia capabilities. For instance, a modern Nokia smartphone, which integrates functionalities of a cellular phone plus the PIM part of a PDA, might have up to 64 Mbytes of SDRAM, 160 Mbytes of internal flash memory, different wireless built-in capabilities, such as Wi-Fi (Wireless Fidelity), Bluetooth, IrDa (Infrared Device Application), GSM (Global System for Mobile Communications), UMTS (Universal Mobile Telecommunications System), HSDPA (High Speed Downlink Packet Access), a built-in high resolution camera, and even a GPS (Global Position System) receiver. Apart from the increasing rate of diffusion of such devices, we need to ponder about the misuse and abuse of these embedded systems, by increasing the awareness of how it is possible to extract all the digital content from the observable memory of such systems. Presently, one of the major operating systems used in the arena of portable devices is, without doubt, Symbian^[2,3,4,5]. One of the first needs for a forensics practitioner is to be able to collect almost all the memory content, RAM and ROM, from such devices, as a preliminary step in a forensic investigation.

The remaining part of the paper has been organized as follows. Firstly, a sound description of Symbian architecture will be provided, by pointing out the basic building blocks of this OS. In addition, the basic elements of the file system will be discussed for understanding how to interact with it. Secondly, the new Symbian security platform will be introduced and analyzed thoroughly, giving at the same time a proof of concept about the possibility to gather the full, observable memory content of a Symbian based device. Finally, general guidelines about how to collect

Antonio Savoldi, Ph.D. candidate; research fields: computer and embedded forensics, security of mission-critical computer systems.

Paolo Gubian, associate professor; research fields: computer and embedded forensics, reliability and robustness of electronic systems architectures.

probatory elements from the new Symbian platform will be introduced.

2. Symbian OS architecture

Today's Symbian operating system (OS) might be referred to as EKA2 as well, which means EPOC Kernel Architecture 2. EKA2 is the second iteration of Symbian 32-bit kernel architecture, and this in turn follows 8 and 16-bit kernels designed in the 1980s for Psion personal organizers and PDAs. As can be seen in Fig. 1, the architecture is modular, where operating system functionalities are provided in separate building blocks, and not in one monolithic unit. Symbian OS is single user. Indeed, there is no concept of multiple logins to a Symbian OS smartphone, unlike Windows, Unix or traditional mainframe OSes. In addition, it is

multi-tasking, being able to switch CPU time between multiple threads, giving the user of the mobile phone the impression that multiple applications are running at the same time. Moreover, it is a preemptively multi-tasking OS because it does not rely on one thread to relinquish CPU time to another, but reschedules threads perforce, from a timer tick. It is worth mentioning that Symbian OS is a priority-based multi-tasking OS with priority inheritance. The OS allocates CPU time based on a threads priority and minimizes the delays to a high-priority thread when a low-priority thread holds a mutex it needs. Furthermore, Symbian OS is real-time based and its services are bounded, that is it completes them in a known amount of time.

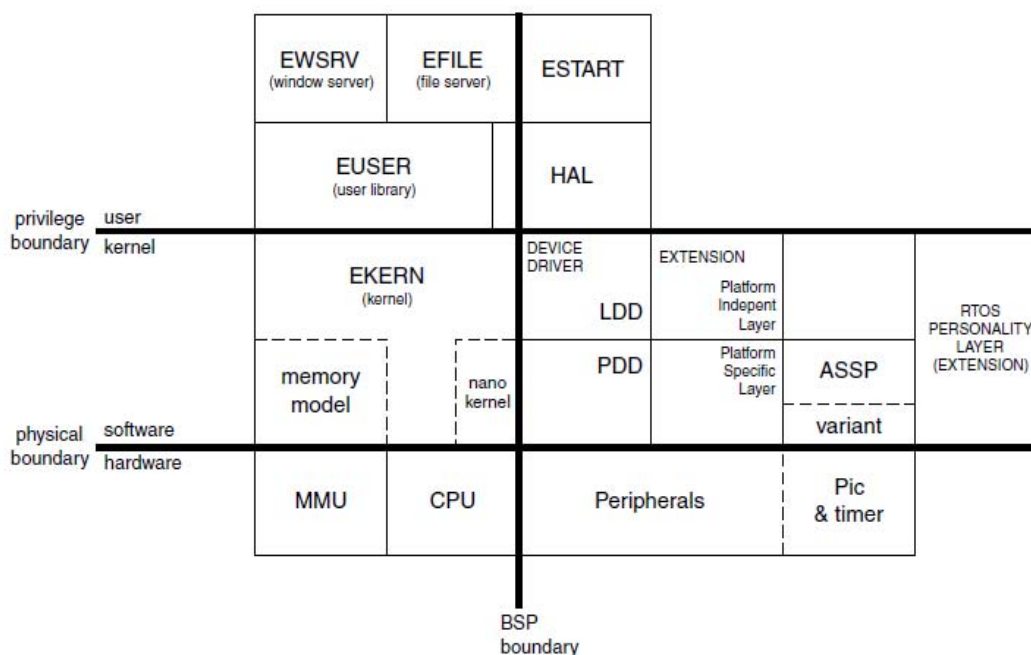


Fig. 1 Symbian OS architecture

The core OS is formed by a microkernel built as a personality layer on top of a real-time (RTOS) nanokernel. This block is responsible for primitives such as fast synchronization, timers, initial interrupts dispatching, and thread scheduling. The Symbian kernel might be referred to as the combination of the

nanokernel and the microkernel. Interestingly, the nanokernel has been designed to provide just enough functionality to run both the telephone (GSM/GPRS/UMTS) signaling stack, already written for RTOS platforms such as Nucleus and μ TRON, and the personal information management (PIM) part on a

single processor. This approach provides considerable cost savings over the usual two-processor solution.

Generally speaking, Symbian OS is intended to run on open, small, battery-powered portable computers, which are modern advanced state-of-the-art mobile phones. Besides this, it is designed for 32-bit CPUs, running at low speeds, usually less than 400 MHz, relative to those in desktop or workstations. Today's Symbian OS systems are based on XScale, ARM9 and ARM11-based CPUs, while older devices are powered by the ARM7 family of CPUs.

2.1 Symbian filesystem

On a Symbian smartphone, the file system can be accessed by means of the file server component, also referred to as F32, which manages every file device. It provides services to access the files, directories and drives on those file mapped devices. The file server uses the client/server framework, by receiving and processing file-related requests from multiple clients. Moreover, it is able to deal with different file system formats, such as the FAT format used for removable disks, by using components that are plugged into the file server itself. In addition, it supports a maximum of 26 drives, each identified in DOS-like convention by a different drive letter, in the range A: - Z:. The main ROM drive, where the firmware resides, is always designated as "Z:". This drive holds system executables and data, which are referred to as XIP (eXecutable In Place) because they are directly launched without being loaded into RAM. Besides this, the firmware, or ROM image, is usually programmed into Flash memory, known also as EEPROM, the nonvolatile memory that can be programmed and erased electronically. The C: drive is always designated as the main user data drive, which can be mapped onto the same Flash memory chip of the firmware, whereas any removable media device is generally designated as D: or E:.

It is worth mentioning that every access from a client to file server (F32) takes place via a file server session, by means of RFs server session class, which

implements all the basic services to interact with the file system. So far, we can obtain information about drives and volumes, act on directories, obtain notification about the state of files and directories, analyze file names, verify the integrity of the file system, and finally, manage drives and file systems.

3. Platform security

Since the introduction of Symbian OS version 9.1, which can also be referred to as Symbian S60 series third version (S60 v3), Nokia engineers have introduced comprehensive support for new functionality such as Digital Rights Management (DRM), Device Management and Enterprise-grade data handling, which rely on the Trusted Computing Base (TCB) concept. At this level of security, the deepest and strongest, every component needs to be certified and formally verified to be secure for having unrestricted access to the device's resources. Only the kernel, the file server, the software installer and its registry are part of the TCB set. Beyond the core of the TCB there is the Trusted Computing Environment (TCE), which protects the device resources from misuse, when system components require access to some, but not all, sensitive system resources. The final level of trust is represented by ordinary applications, which need to be authorized to access privileged restricted resources by means of a capability set. This defines an authorization token that can grant access to sensitive APIs such as device drivers or system settings. Indeed, among all APIs there is a subset which can be referred to as capability-protected and represents 40% of the Symbian OS APIs. As a consequence, an application needs to be authorized, by using a certification process, for using the protected functionality. There are two basic sets of capabilities, basic and extended. The former set protects APIs that access confidential user data, provides data about user's environment and accesses network services. The latter set, the extended group, which requires a

phone-manufacture's approval, grants the access to the most reserved area of the phone. This set includes DRM, DiskAccess, NetworkControl and AllFiles capabilities, where the last is referred to as the complete access to the file system. The following rules will be applied when dealing with capabilities.

- Every process has a set of capabilities and its capabilities never change during its lifetime.
- A process cannot load a DLL that has a smaller set of capabilities than it has itself.
- A DLL cannot statically link to a DLL that has a smaller set of capabilities than it has itself.
- The access rules of a file are entirely determined by its directory path, regardless of the drive.

The latter rule explains how to control the file access by means of the various processes running on the phone. This solution is different from a traditional access control list, defined per process, being based on a fixed access control policy which is equally valid for all the processes. This concept might be referred to as data caging and there are four different sets of access rules, which are represented by four different directory hierarchies under the root of the file system, according to the following scheme.

- **\sys:** Only TCB granted processes can read and write files under this directory, which contains system and installed binaries, vital for the integrity of the platform.
- **\resource:** All processes can read files in this directory, but only TCB processes can write to them. In this directory we might find fonts and help files that are not expected to change once installed.
- **\private:** Every program has its own private sub-directory, with the name <application SID>, where SID is a secure identifier which uniquely identifies it. Only the granted processes and the backup server can read and write files in a process's private directory. Other processes may neither read nor write.
- **Other root files and directories:** The remaining portion of the file system has no restrictions in

reading or writing, being considered public and thus accessible space.

So far, if an application needs to have the complete access (e.g. read) to the phone file system, it has to be authorized by means of the Symbian signing procedure with AllFiles capabilities, which requires a special certificate released by TC Trust Center. Three steps are required to sign an application. Initially, the installation file generator, makesis.exe, creates the installation files (extension.sis) from information specified in the package file (extension.pkg). After that, if the application is for the international market, it will be signed with an ACS Publisher ID, by means of the Symbian Signed service. Conversely, it will be signed with a user-generated certificate, which might be created with makekeys.exe. Finally, the Installation File Signer (signsis.exe), digitally signs the installation files with the proper certificate, by generating, as a result, a .six file.

4. Data collection

After the exhaustive preamble of previous paragraphs it is worth pointing out what the main issues are when dealing with the implementation of an open source application, for forensic purposes, which aims at collecting the entire set of digital objects present on a Symbian mobile phone file system. It should be evident that with the new Symbian security platform the collection of the entire, observable, memory content is restricted by means of data caging, at least for C: user data disk, where all probatory data of interest are located. This implies having the phone-manufacturer approval to be able to gather the full and complete file system content. Interestingly, the file system restriction policy is fully contained in the file known as SWIPOLICY.INI^[1], located in the folder z:\system\data\. The original policy file related to a Nokia E65 Symbian based smartphone is illustrated as follows.

```

AllowUnsigned = false
MandatePolicies = false
MandateCodeSigningExtension = false
Oid = 1.2.3.4.5.6
Oid = 2.3.4.5.6.7
DRMEnabled = true
DRMIntent = 3
OcspMandatory = false
OcspEnabled = true
AllowGrantUserCapabilities = true
AllowOrphanedOverwrite = true
UserCapabilities = NetworkServices LocalServices
    ReadUserData WriteUserData UserEnvironment
AllowPackagePropagate = true
SISCompatibleIfNoTargetDevices = false
RunWaitTimeoutSeconds = 600
AllowRunOnInstallUninstall = false
DeletePreinstalledFilesOnUninstall = true
    
```

It is interesting to observe that the capability set defined in the previous file is limited, and it restricts the interaction with the file system of the mobile platform. According to the standard documentation, the various parameters can appear in any order. Moreover, UserCapabilities set might be changed, by adding the required capabilities such as those illustrated in the following modified version of policy file.

```

AllowUnsigned = true
MandatePolicies = false
MandateCodeSigningExtension = false
Oid = 1.2.3.4.5.6
Oid = 2.3.4.5.6.7
OcspMandatory = false
OcspEnabled = true
AllowGrantUserCapabilities = true
UserCapabilities = AllFiles DiskAdmin NetworkServices
    LocalServices ReadUserData WriteUserData
UserEnvironment MultiMediaDD NetworkControl
    CommDD ReadDeviceData WriteDeviceData
SISCompatibleIfNoTargetDevices = false
AllowRunOnInstallUninstall = true
AllowPackagePropagate = true
DeletePreinstalledFilesOnUninstall = true
    
```

The illustrated policy file can be written directly in the original firmware of the phone and, subsequently, uploaded by means of reflashing. As a result, the complete C: disk content might be collected, with

standard self-signed APIs, and thus analyzed, to extract the full set of probatory data which might be usually found on a mobile platform. This is the usual approach for other mobile platforms such as Windows CE^[6] or for traditional mainframe systems, where the primary image, the one which contains the entire set of evidence, can be obtained without any restrictions. Unfortunately, such a scenario is far from the reality, and we need to evaluate others ways to access the digital data content of the smartphone.

General guidelines for PDA and smartphone forensics have been released by reference [7], where the authors pointed out that investigators cannot change the digital data content of the device being analyzed. Moreover, an audit trail of the investigation process, suitable for independent third-party review, must be created and preserved, accurately documenting each investigative step. Finally, the person in charge of the investigation has the overall responsibility for ensuring that the above-mentioned procedures are followed and are in compliance with the governing laws. Another important issue to be aware of is the general set of requirements that a forensically sound method should have when used on a mobile device. The method should minimize changes on the device, be able to retrieve the full set of data, and finally minimize user interaction with the device itself.

Ideally, the full memory content of a generic embedded device should be collected, to preserve the full inner state and obtain a forensically sound acquisition. As a matter of fact, with Symbian S60 platform this is not possible. Nevertheless, as detailed by^[8], there is a considerable number of other approaches to capture data from the mobile platform, which are as follows.

4.1 Manual examination

This approach requires a manual inspection of the device being investigated, by means of an external recording device (e.g. photcamera) which is able to take snapshots of visual evidence. The method is error-prone because of the direct interaction between

the investigator and the device, which can cause an easy alteration of the content. It should be used only when there are no other ways to extract digital data, and a sound and complete audit trail of the investigative procedure must be provided for third-party examination.

4.2 Connectivity services

This is a software approach based on command-response protocols, such as AT Command Set, SyncML, OBEX, Nokia FBUS proprietary protocol or applications based on Nokia PC Suite Connectivity^[9]. All mentioned protocols are unable to have the full and complete access to the memory areas where probatory data are located, the C: and external drive disks. Even so, such protocols might be used for collecting digital objects at the logical level, by ensuring that no command will be issued to modify the content of the phone.

4.3 Connection agent

This method, again software based, uses a specific program, the so-called connection agent, which is installed on the target device. It is capable of establishing a connection and exchanging data with an externally connected host computer. The main issue with this approach is that it requires a piece of software to be loaded on the device, thus modifying the target device. This kind of approach has been successfully used by the authors with Windows CE platform^[6].

4.4 Hardware approach

This method permits to obtain a forensically sound copy of the entire flash memory content, making it possible to access deleted or partially overwritten entries of the file system. This might be obtained by means of the JTAG test access port of an embedded device. A JTAG port is used for debugging purposes, but can also be used to access the flash memory^[10]. Despite the difficulties related to this process, which depends on the possibility to easily find the hardware interface, it can be guaranteed that no data is written during the collection phase. The other possible approach to collect data from embedded devices'

memory is by de-soldering the flash chips from the board and reading the content with a memory chip programmer or reader. Although this method is the most invasive for the equipment, it guarantees that data can be recovered even from broken or damaged embedded systems.

4.5 Service provider

In the GSM/UMTS network environment, a great deal of information might be recovered from the service provider^[8]. The set of information which can be successfully collected with this method is related to the SIM/USIM data set, such as SMS, MMS, list of last called numbers, and the location of the subscriber^[11]. Clearly, information such as photos, videos, phone book, web browser logs, audio recordings, or user's notes cannot be gathered in such a way.

5. Experimental results

So far, Nokia S60 v3 (third edition) mobile phone platform does not permit to acquire the complete flash memory content at the logical level, by means of software techniques based on not signed and certified tools, for the limitation which has been introduced with the new security model, which relies on data caging. With the previous Nokia phone generations, for instance the S40 series, the logical acquisition of the device content was possible, and almost trivial, by means of Symbian OS APIs, which were able to copy the entire file system content on an external memory device (e.g. SD card)^[12]. This approach might be used with the newer Symbian phones (e.g. Nokia S60 devices) only when the OS has been provided with the "AllFile" capabilities, as it has already been shown with regard to the E65 Nokia phone example. Is such a case, as it has been demonstrated, the whole logical content might be acquired. For instance, by interpreting the raw data objects of the collected file system, we can carve out all the information related to SMS and MMS. Every object of this type is saved in a file with a name that follows the pattern 00100XXX, with XXX a

hexadecimal number. All the files are saved according to the following path.

```
C:\Private\1000484b\Mail2\00001001_S\[0..F]
..\Mail2\00001001_S\0001000c0
..\Mail2\00001001_S\0001001a0
..\Mail2\00001001_S\0001001c0
..\Mail2\00001001_S\000100020
..\Mail2\00001001_S\000100040
..\Mail2\00001001_S\000100080
..\Mail2\00001001_S\000100280
```

6. Conclusions

To summarize, digital forensic methodologies for the new Symbian S60 v3 OS based phones are still in their infancy stage, the security paradigm being new and very restrictive towards file system access. Software approaches for imaging tool might only be used for acquiring specific items. For instance, it is certainly possible to extract the entire set of probatory data, such as SMS, MMS, pictures, video clip and phone book, by using Symbian application APIs. From an integrity perspective, the hardware approach seems the only one which should be able to give a bit-by-bit image of the flash memory content thus preserving the content of the investigated phone.

References:

[1] Symbian-Ltd. Symbian OS library for application developers. Available at: <http://www.symbian.com>.

- [2] Morris B. *The Symbian OS Architecture Sourcebook: Design and Evolution of a Mobile Phone OS*. John Wiley & Sons, Ltd, 2007.
- [3] Sales J. *Symbian OS Internals: Real-time Kernel Programming*. John Wiley & Sons, Ltd, 2005.
- [4] Jipping M. *Smart Phone Operating System Concepts with Symbian OS: A Tutorial Guide*. John Wiley & Sons, Ltd, 2007.
- [5] Harrison R. and Shackman M. *Symbian OS C++ for Mobile Phones(Vol.3)*. John Wiley & Sons, Ltd, 2007.
- [6] Savoldi A. and Gubian P. Data hiding and recovery on wince based handheld devices. *In Fourth Annual IFIPWG 11.9 International Conference on Digital Forensics (IFIPWG 11.9 2008)*, 2008.
- [7] Jansen W. and Ayers R. An overview and analysis of PDA forensic tools. *In Digital Investigation*, 2005.
- [8] Mokhonoana P. and Olivier M. Acquisition of a symbian smart phone's content with an on-phone forensic tool. *In Southern African Telecommunication Networks and Applications Conference 2007 (SATNAC 2007) Proceedings*, 2007.
- [9] Campbell I. *Symbian OS Communication Programming (2nd ed.)*. John Wiley & Sons, Ltd, 2007.
- [10] Breeuwsma M. Forensics imaging of embedded systems using JTAG (boundary-scan). *In Digital Investigation*, 2006, 3: 32-34.
- [11] Savoldi A. and Gubian P. Sim and Usim filesystem: A forensic perspective. *In Proceedings of Symposium on Applied Computing (ACM SAC 2007)*, 2007.
- [12] Breeuwsma M., Jongh M. D., Klaver C., et al. Forensics data recovery from flash memory. *In Small Scale Device Forensics Journal*, 2007, 1.

(Edited by Alex, Yunflyer)