Forensic Protocol Filtering of Phone Managers

Wayne Jansen and Aurélien Delaitre
Information Technology Laboratory, National Institute of Standards and Technology
Gaithersburg, MD, USA

Abstract – *Phone managers are non-forensic tools sometimes used by forensic investigators to recover data from a cell phone when no suitable forensic tool is available for the device. While precautions can be taken to preserve the integrity of data on a cell phone, inherent risks exist. Applying a forensic filter to phone manager exchanges with a device is suggested as a safer alternative that could be pursued as a solution to reduce risk.*

Keywords: Cell Phone, Computer Forensics, Protocol Filter

1. Introduction

Over 2.5 billion cell phones are estimated to be in use in the world today – with 3 billion expected before 2010. Digital evidence recovered from a cell phone can provide a wealth of information about the user, and technical advances in device capabilities generally offers opportunity for recovery of a broader range of information. Numerous forensic tools abound for automatic data recovery from cell phones. While the outlook should be positive, a number of factors conspire to impede progress in cell phone forensics. A key issue is the delay between the availability of a cell phone to the public and support for the phone by a forensic tool.

When a new phone appears, a forensic tool manufacturer must decide whether to adapt its tool for the phone, purchase exemplars for study, create and test an update containing support for the phone, and finally release the tool update to the user. The decision factors involved include the popularity of the phone model, the requirements of the customer base, and the overall support objectives of the company. The time required for needed tool updates to reach users, therefore, can be lengthy and for the least popular models may never occur. Validation of the updated tool for use in casework increases the delay, putting forensic specialists further behind the power curve of having a suitable means for automated data recovery. Figure 1 illustrates the situation.

Phone managers are sometimes turned to as a way to recover data when no suitable forensic tool is available. Phone managers are typically available from the manufacturer of the cell phone and kept up to date with support for newly released models. They allow various operations, including retrieval of core user data such as phonebook entries and photos. Tools not designed specifically for forensic purposes are questionable, however [4]. In particular, phone managers have the ability to both read and write data to a phone, which is problematic from a forensic perspective, if used without applying proper testing and procedural controls. Many anecdotes abound of a practitioner accidentally or unknowingly writing data to a phone when using such a tool.

To simplify the content recovery process, a forensically-sound access method would exist across all cell phones. More realistically, cell phones would support a common interface and protocol standard for handset communications that could be used for data recovery. A recently proposed standard from the Open Mobile Terminal Platform specifies the use of micro Universal Serial Bus (USB) as a universal, cross-manufacturer cable interface for power and communications. Its data synchronization capabilities might provide an opportunity, if adopted by manufacturers.

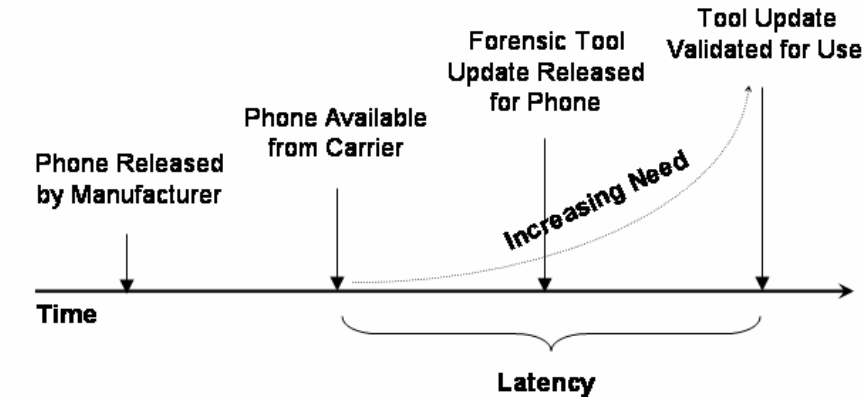

Figure 1: Forensic Tool Timeline

Until then, avenues to reduce latency need to be pursued. For example, tool manufacturers could improve their relationships with phone manufacturers or network carriers to gain a head start on development before phones are available to the general public. Another approach to reduce latency called phone manager protocol filtering is described in this paper. The idea is to build on the functionality of available phone managers by augmenting them with a protocol filter that limits their functionality to allow only safe exchanges to occur.

2. Background[♦]

More than a billion cell phones were sold worldwide in 2007 and projections beyond continue to rise. Over the last decade the capabilities and features of cell phones, such as increases in performance and storage capacity, and additions of document and multimedia handling functionality, have also continued to improve rapidly, turning cell phones into data reservoirs with the capability to hold a broad range of personal and organizational information.

Forensic software tools are the preferred means for recovering digital evidence from supported cell phones. Data recovery is usually carried out through logical instead of physical acquisition, using one or more procedure supported by the device. The protocols include standardized and proprietary device synchronization protocols, command interface protocols, and diagnostic protocols.

The number and variety of phone models unveiled on the world market each year is considerable, creating a burden for forensic tool manufacturers to keep their product coverage up to date. Models introduced into one national market can be used elsewhere by replacing the identity module of a phone with one from another carrier, or through roaming features. Models of older functioning phones, though out of date, can also remain in use for years after their initial release.

Unlike the situation with personal computers, mobile phone manufacturers often employ different proprietary operating systems and storage structures. New phone models often have functional differences from previous models that must be taken into account to recover and report data properly. Complicating matters further are variations in data storage location assignments, which can occur in a specific model of phone subsidized and supplied by different network carriers, due to adaptations made for the carriers by the manufacturer. Firmware updates sent out by a network carrier can also affect data locations, creating additional hurdles for developing and maintaining a tool [3].

[♦] Certain commercial products and trade names are identified in this paper to illustrate technical concepts. However, it does not imply a recommendation or an endorsement by NIST.

Six manufacturers control about 80 percent of the cell phone market at any one time, while approximately forty others compete for the remaining 20 percent share. Nokia and Motorola led the group in 2006 with more than 50 percent; in 2007 Nokia and Samsung were in front with more than 50 percent [1, 2, 10]. New manufacturers occasionally enter the marketplace and others leave. For example, the iPhone from Apple was a new entrant in 2007.

Cell phone manufacturers such as Nokia, Motorola, and Samsung normally keep their phone manager software up to date for new and current phone models in the product line. Forensic specialists have long recognized the potential for phone managers as a tool for automated recovery of common types of core user data. Because phone managers are not forensically sound, additional steps must be followed, if used to recover data. They include validating the operation of the phone manager, testing and verifying the procedures to be followed for acquisition to safeguard against altering data on the phone, and producing a cryptographic hash of the acquired data.

Regrettably, even an experienced forensic specialist taking all available precautions could accidentally write data to a phone using a phone manager. Phone manager protocol filtering helps to safeguard against accidental modifications to data on the phone and provides a stopgap measure until a forensic tool update that supports the phone in question becomes available.

3. Filtering Considerations

Forensic cell phone tools often recover data employing the same protocols used by phone managers. To avoid the problem of altering data on a phone, forensic tools restrict the protocol used to communicate with the device to only functions that are either known to be safe or involve very minor forensic issues. A potential way to gain the same advantage for phone managers is to apply a filter between the phone manager application and the device being managed, which blocks harmful protocol commands from propagating. Filtering is an often used technique in computer forensics, commonly implemented in hardware or software write blockers for disk and USB device interfaces.

Most phone managers run under the Windows operating system and are distributed in binary form for installation. Communications with cell phones occur over a serial COM or USB port. Most serial port data transmission for Windows systems is done the same way as writing to a file. For example, the WriteFile function can be used to send data via a serial COM port. The same function also works with virtual serial ports established over USB, infrared, or Bluetooth communications. The technique used for the filter prototype involves intercepting the call from the phone manager to the Application Programming Interface (API)

for this function to capture the data, interpret the content, and return an appropriate response to the phone manager. Similarly, calls to other related functions, such as CreateFile and ReadFile, would need to be intercepted for the filter to work overall.

API hooking is a term used to describe intercepting calls to a function for some purpose, usually to customize and extend its functionality and also to monitor aspects of an application. The target function may be in an executable application, a library, or a system Dynamic Link Library (DLL). In the case of Windows operating systems, the functions of interest are part of the so-called Win32 API. Hooking Win32 APIs is not new; security add-ons, such as personal firewalls and anti-virus applications, as well as malicious code such as rootkits, have used these techniques to insert themselves seamlessly into an operating system. The interception process is performed at run time against a running process rather than modifying static binary images at rest.

Several different techniques have been used to hook Windows APIs. A common way is to alter the import address table (IAT) of a given module and replace the target function with the substitute function. The IAT contains the address of each imported function and is used by the loader to map function calls to entry points of loaded routines. Alternatively, an unconditional jump can be inserted in the first few bytes of a target function to change the flow of execution to the substitute function. When the substitute function completes its task, control is returned to the modified function or, optionally, back to the calling program.

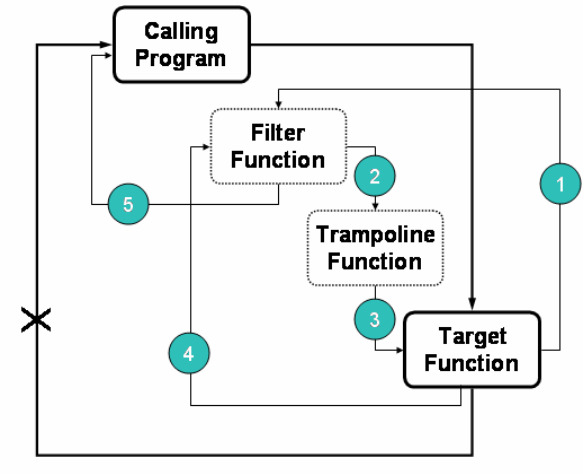

Figure 2: API Interception

The approach used for the phone manager filter is to have the substitute function serve as a wrapper for the target function, as illustrated in Figure 2. The first few instructions of the target function are replaced with a jump to the filter function, and the replaced instructions from the target function are preserved in a so-called trampoline function [6]. The trampoline function acts like a relay, ending with a jump back to the target function to complete processing after the preserved instructions are executed. The filter function can either call the trampoline function to invoke the target function, or return directly to the calling program and bypass the target function altogether. The target function is also adjusted to return control to the filter function upon completion to allow the filter to perform any needed post-function operations.

The use of this technique makes the filter somewhat system-dependent. Certain functions of the Win32 API are partially overwritten and the binary code of the Win32 API can vary with the version of the operating system. The operation also must observe the right alignment with the next, not overwritten instruction. It is typically a simple task to adapt the filter to a particular release of Windows, including the version of its service pack.

4. Phone Manager Protocol Considerations

The Nokia PC Suite provides a good example of a candidate phone manager for protocol filtering. The current version for the U.S. market supports approximately 75 models, including the very latest. The versions for other countries support about the same number of models, some of which are different from the models in the U.S. version. PC Suite can be used for a number of things, including copying personal data (e.g., phonebook entries) to a computer for safekeeping; transferring images, video clips, and other files from the phone to a computer; and viewing contacts and messages on a device. Certain features work only when used with those models of Nokia phone that employ compatible functionality. Various types of communications with the phone are supported, including serial COM and USB cables. Wireless options also exist.

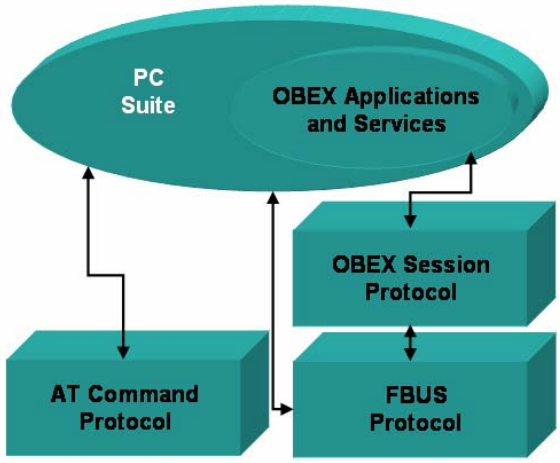

Figure 3: Phone Manager Protocol Stack

Byte	0	1	2	3	4 - 5	6 - n	n+1 - n+2
Contents	Frame ID	Destination	Source	Command	Length	Data	Checksum

Figure 4: FBUS Frame

The Nokia PC Suite uses a proprietary protocol called the FBUS protocol to perform its functions. An AT modem command is sent to the phone to switch into FBUS mode. The FBUS protocol is used to extract the model number of the phone, presumably to determine how to proceed. The FBUS protocol can also be used to recover other information, such as phonebook, call logs, SMS messages and calendar entries. Another protocol, OBEX, which rides over the FBUS frames, is also used to extract media files, ring tones and downloaded applications that are present. The physical interface is a bidirectional serial communication bus that runs at 115,200 bits per second [7]. Figure 3 illustrates the situation.

The FBUS frame is byte oriented. The first byte of the frame, byte 0, holds the hexadecimal value of the identifier for the FBUS protocol. The value 1E is the frame identifier for cable. Bytes 1 and 2 respectively contain the destination and source addresses [7, 8]. For data sent to the phone, the destination address is 00. The source address for the personal computer is 10 or 0C. Byte 3 contains the command identifier, which potentially supports up to 256 (i.e., 2^8) commands. Bytes 4 and 5 hold the length of the data that follows. The bytes following byte 5 convey the data segment of the frame. The last byte of the data segment contains a 3-bit sequence number and fragment flag, while the penultimate byte indicates the remaining frames to go to complete the payload. The last two bytes of the frame contain a checksum [7, 8]. Only frames of an even length are transmitted. A byte of all zeros is inserted before the checksum, if needed, to make the total length of the frame even. Figure 4 illustrates the frame composition.

The FBUS protocol is an acknowledged request-response protocol, with the phone manager issuing command requests and the phone answering [7, 8]. Responses use the same command identifier as the request being answered, but reverse the source and destination address. Every request or response, except for the first request, is prepended with an acknowledgment frame indicating receipt of the last protocol element sent by the other party. This convention means that for a blocked request, the filter may need to forge a receipt acknowledgment, in addition to an appropriate negative response, to prevent the phone manager from resending a disallowed frame.

Because the FBUS protocol is proprietary, the function of all command identifiers is not known. However, over the years many of the commands have been determined through experimentation by various parties. Furthermore, the communications of available cell phone forensic tools can be monitored to help to identify commands considered safe by tool manufacturers. To avoid propagating frames containing unsafe commands to a phone, the phone manager filter incorporates a white list of known commands deemed safe; all other command frames are blocked.

The Object Exchange Protocol (OBEX) performs a function similar to HTTP for devices that are resource constrained. OBEX consists of the following pieces:

- An object model that conveys information about the objects being sent, as well as the objects themselves
- A session protocol, which uses a binary packet-based client/server request-response model.

The OBEX File Transfer Protocol (OBEXFTP) service is used to access the file structure of the device. The OBEX Object Push (OBEXOBJECTPUSH) service is used to exchange objects such as vCard and vCalendar and, for some devices, to access the file structure. In addition, other proprietary methods can be defined by the manufacturer.

5. PC Suite Design and Operation

Nokia PC Suite (PCS) release 6.84.10.3 is made of several standalone programs. The Graphical User Interface (GUI), LaunchApplication.exe, allows the user to start other operational subprograms such as PCSync2.exe and ContactsEditor.exe, used respectively to synchronize data with a computer and to edit phonebook entries. These programs use a Remote Procedure Call (RPC) channel for communications with ServiceLayer.exe, a resident PCS service. The service is started automatically by the operating system and is responsible for communicating with the phone. It makes use of the different protocols supported by Nokia phones (i.e., AT, FBUS, and OBEX over FBUS).

PCS can be envisaged as two distinct parts: the application, which bundles the GUI and the operational subprograms, and the service layer, which is a sublayer of the application. Figure 5 illustrates the design. The upper-level applications run with the privileges of the user executing them. In contrast, the service layer runs with System privileges, which gives it total access to the operating system and the resources of the computer.

The PCS service uses the Win32 API provided by the operating system. In this case, to communicate with a Nokia 6101 device, it uses a variant of the CreateFile function, CreateFileA, to get a handle on the serial port to which the phone is connected. In the main thread, the service runs a loop that scans for available devices on a regular basis. Once a device is detected, it calls CreateFileA to open a communication channel to the device. The

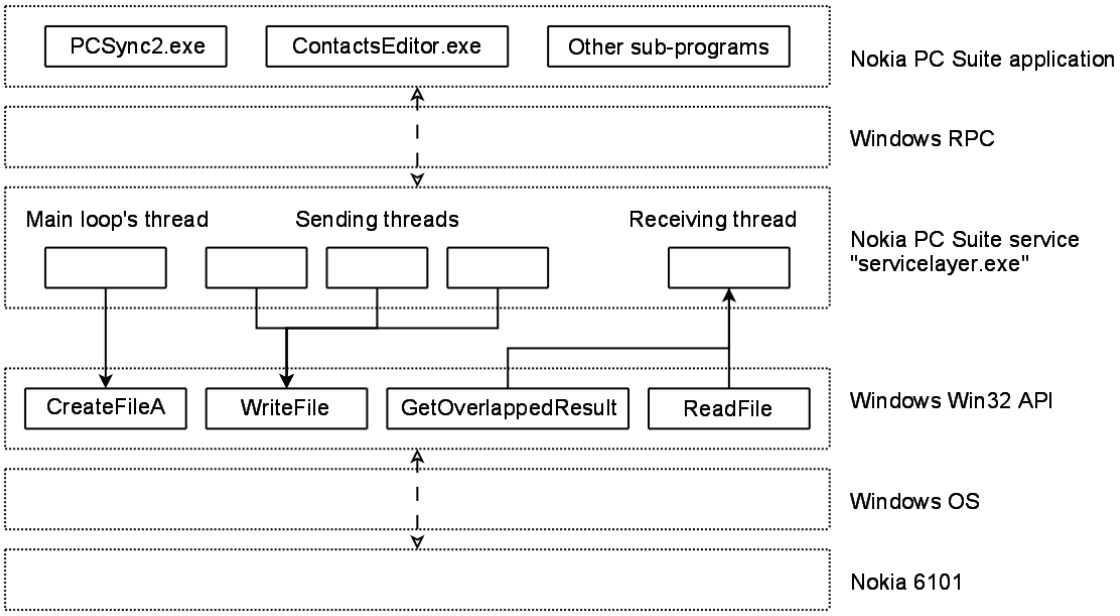

Figure 5: PC Suite Design

functions WriteFile and ReadFile are used respectively to send requests to the phone and to receive the responses. Depending on the upper-level application being used, several threads are created to send requests over the newly created channel. A different thread is used to read the responses from the device using the functions ReadFile and GetOverlappedResult. GetOverlappedResult is used to read the data after a call to ReadFile to accommodate the asynchronous communication channel to the phone.

In the beginning of a data exchange, the phone is in the default AT mode. PCS sends the standard AT command "at&f" to initialize the phone's modem, followed by a second non-standard AT command, "at*nokiafbus", to have the phone switch to the FBUS mode. Using FBUS, PCS requests the phone's model. For example, for a Nokia 6101, the application asks for the phone capabilities using an OBEX over FBUS session. The phone replies with an XML file containing the requested information. The rest of the operations are performed, ending with an FBUS command that switches the phone back to the default AT mode.

6. Filter Design and Operation

The filter is injected in the memory of the service layer, ServiceLayer.exe. It serves as a wrapper for the Win32 API, intercepting calls to the functions used to communicate with the phone, as illustrated in Figure 6. Instead of calling the genuine function of the Win32 API, the service calls the matching detour functions of the filter. The filter then decides how to handle calls to the Win32 API.

During a data exchange, PCS controls the whole operation, sending requests for the phone to answer. Hence, the filtering is done primarily by analyzing the data sent to the phone by the computer (i.e., through the intercepted WriteFile function). Only requests that are considered safe are forwarded onward. Unsafe requests are blocked and an error status is returned, but they also could be used to trigger a negative response (e.g., object unavailable) from the filter.

Responses sent back by the phone are not blocked by the filter. The filter analyzes and logs them using the intercepted ReadFile and GetOverlappedResult functions, before forwarding them onto the service layer. Since blocked requests are not received by the phone, no responses are sent back. In general, there should be no need to filter the data sent to the computer by the phone.

6.1 Injection of the filter

The filter consists of a DLL that is loaded into the service layer's address space by a loader. The goal of the loader is to find the right process to inject, namely ServiceLayer.exe, and to load the DLL into its memory. Since the service layer runs with System privileges, the loader also needs System privileges to carry out its work. System privileges are not granted to regular users, or even administrators, on Windows computers.

One way to obtain System privileges is to use the administrator's ability to create new services [9]. A member of the Administrator group can create a service that runs a command prompt, cmd.exe, with System privileges. The Service Create tool, Sc.exe, with the syntax: *sc*

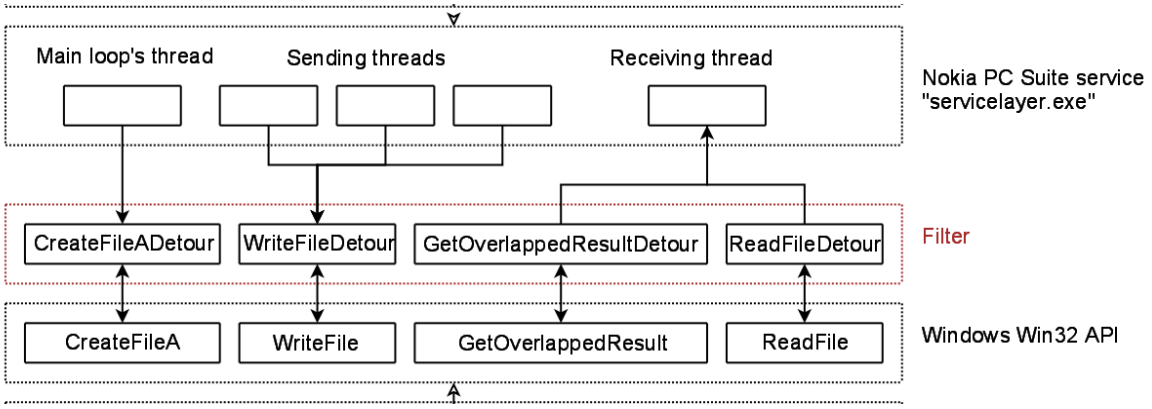

Figure 6: Filter Loaded into PCS

Command Servicename [Optionname= Optionvalue...] is used as follows: *sc create systemprompt binpath= "cmd.exe /K start" type= own type= interact* [11]. Once this new service is created, it can be started at any time to launch a System level command shell (i.e., *sc start systemprompt*).

From the command shell, the user navigates to the directory containing the filter's DLL and the loader. Before running the loader, it is necessary to ensure no phone is connected to the computer. It is safer to stop PCS's service first and then start it again, before injecting the filter. Once the loader is executed and the filter is injected into the service layer, the phone can be plugged in and used with PCS. It is not possible to unload the filter once it is injected. The only safe way to resume the regular work of PCS is to stop its service and start it again.

6.2 Operation of the Filter

The operation of the filter is illustrated in Figure 7. The filter is first activated when the main loop of the PCS service layer tries to open a device, while scanning for a connected phone. PCS calls the Win32 function CreateFileA, which jumps straight to the filter's CreateFileADetour. The filter, then, calls the genuine Win32 CreateFileA function to open the file as expected by the caller. If the operation is successful, the name of the open file or device is tested and, if it happens to be a serial device, the filter stores the resulting handle for later use. In the last step, the handle is returned to the caller.

Later, when an upper-level application asks for an operation to be performed on the phone, a new thread is created by the

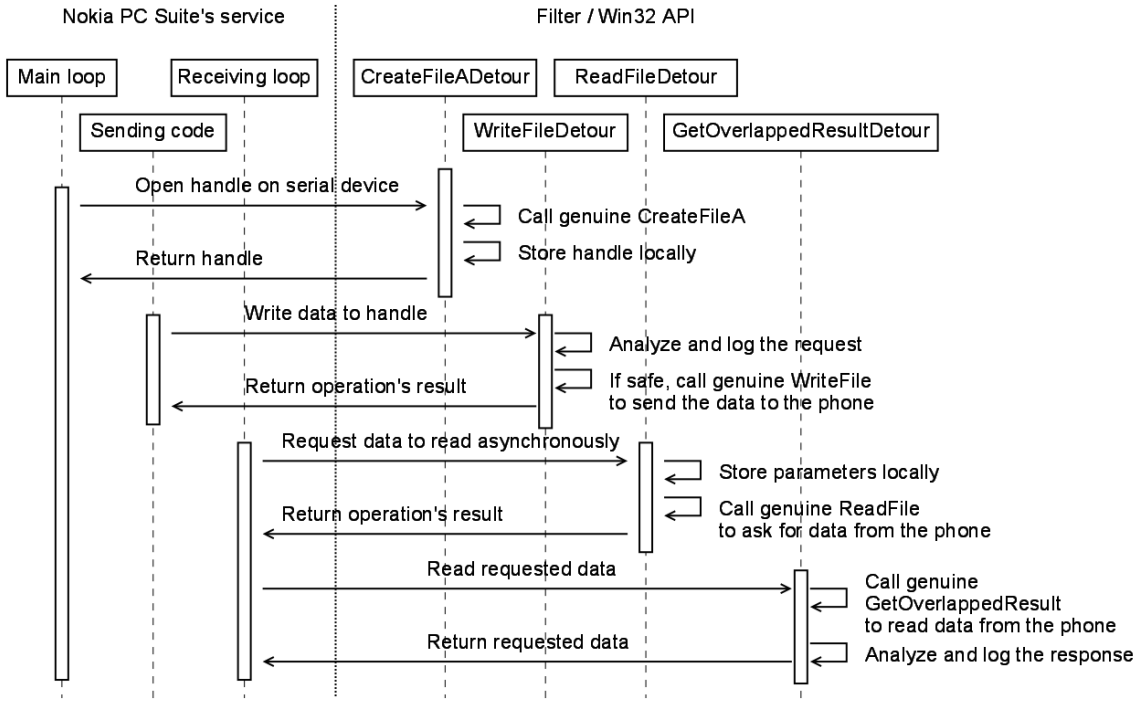

Figure 7: Filter Sequence of Operations

service to send a request through the previously opened device. This thread calls WriteFile to send these data and, since this function is intercepted, jumps to the filter's WriteFileDetour. If the handle to be written to is the same as the handle saved by CreateFileADetour, the caller is trying to send data to the phone. The request is analyzed by the filter to determine whether it is safe or not. If it is safe, the data is sent through the handle, by calling the genuine Win32 WriteFile, and the status of the operation is returned to the caller. If it is unsafe, then the data is not forwarded to the phone and an error status is returned to the caller.

The service layer has a thread dedicated to reading asynchronously received data coming from the phone. From within a loop, it asks for data to read by calling ReadFile, and then tries to read it with a call to GetOverlappedResult. During the first call, the filter's function ReadFileDetour is executed in place of the expected Win32 function. The filter stores the parameters of this reading request for later use by GetOverlappedResultDetour. It then calls the genuine ReadFile function and returns the status of the operation. If it is successful, the service calls GetOverlappedResult and executes the filter's GetOverlappedResultDetour function, which jumps to the genuine Win32 function. When returning, the read data is analyzed and logged by the filter, and then forwarded to the calling service.

The entire data exchange between the phone and the computer is analyzed. Every event of interest is logged in the file C:\NPSfilter.log. For example, if a frame is not understood, it is blocked and the action logged along with a dump of the frame. When a frame is allowed through, it is appended to the log file, with much of the data translated to a human-readable form.

The log file is not accessible during the operation of PCS due to access restrictions placed on non-System users by the operating system. The filter must be unloaded for the file to be opened, requiring the PCS service layer to be stopped.

7. Conclusions

Cell phone forensics is an emerging discipline. Various impediments exist that create problems for forensic specialists working in this area, and need to be overcome for the discipline to flourish. The technique presented in this paper attempts to resolve the problem with the latency in forensic tool coverage of newly available phone models by phone manager protocol filtering. It is intended as a stopgap measure until forensic tool support becomes available.

Initial testing of the prototype implementation indicates that the approach could provide a practical and effective solution for addressing the latency in forensic tool coverage of available phones. The basic technique described is extendable beyond the specific phone manager example given. Intercepting low-level Windows APIs in the application, as opposed to higher-level internal APIs, allows components of the solution to be applied to phone managers from other cell phone manufacturers. Reprogramming the filter for the different protocols involved would, of course, be required. As with any forensic tool, the resulting filtered phone manager program requires validation before its use.

8. References

[1] Nokia and Motorola Gain Market Share as Arena Grows, International Herald Tribune, Tech/Media November 22, 2006, <http://www.iht.com/articles/2006/11/22/yourmoney/mobile.php>.

[2] Nokia and Motorola Account for Nearly 50% of Worldwide Sales, Mobiledia, August 25, 2005, <http://www.mobiledia.com/news/35125.html>.

[3] Robert Vamosi, Cell Phone 'CSI,' CNET Reviews, May 25, 2007, < http://reviews.cnet.com/4520-3513_7-6737586-1.htm>.

[4] Annalee Newitz, Courts Cast Wary Eye on Evidence Gleaned from Cell Phones, WIRED, May 10, 2007, <http://www.wired.com/politics/law/news/2007/05/cellphone_forensics>.

[5] Tyler Moore, The Economics of Digital Forensics, Fifth Annual Workshop on the Economics of Information Security, June 2006, <http://www.cl.cam.ac.uk/~twm29/weis06-moore.pdf>.

[6] Galen Hunt, Doug Brubacher, Detours: Binary Interception of Win32 Functions, 3rd USENIX Windows NT Symposium, Seattle, WA, July 1999, < http://research.microsoft.com/~galenh/Publications/HuntUsenixNt99.pdf>.

[7] Wayne Peacock, An Introduction to Nokia F-Bus, Embedtronics, April 2005, <http://www.embedtronics.com/nokia/fbus.html>.

[8] Paul McCarthy, Forensic Analysis of Mobile Phones, BS CIS Thesis, University of South Australia, School of Computer and Information Science, October 2005, <http://esm.cis.unisa.edu.au/new_esml/resources/publications/forensic%20analysis%20of%20mobile%20phones.pdf>.

[9] SincereHacker, DOS Help (Includes obtaining system privileges on windows), July 2007, <http://www.elitehackers.info/forums/showthread.php?p=50838>.

[10] Global cellphone sales slowing, IDC says, CBC News, January 25, 2008, <http://www.cbc.ca/technology/story/2008/01/25/tech-cellphones.html?ref=rss>.

[11] How to create a Windows service by using Sc.exe, Microsoft, Article ID 251192, Revision 3.6, December 5, 2007, <http://support.microsoft.com/kb/251192>.