28-3-2005

# Personal firewall for Pocket PC 2003: design & implementation

Willy Susilo
*University of Wollongong*, wsusilo@uow.edu.au

R. J. Ang
*University of Wollongong*, uow@ang.edu.au

C. A. McDonald
*University of Wollongong*

J. Huang
*University of Wollongong*

# Personal firewall for Pocket PC 2003: design & implementation


## Abstract

Personal digital assistants (PDAs) are widely used and becoming indispensable tools of everyday life. Wired or wireless connection enables PDA users to connect to the Internet from any place, making security an extremely important issue in a pervasive computing environment. This paper investigates how to build a personal firewall for PDAs running Pocket PC 2003. This personal firewall allows the PDA user to perform access control based on a user-defined policy, and hence provides a security perimeter between the public network and the PDA. We provide a complete technical detail on how the firewall can be built in a Pocket PC 2003 device. To the best of our knowledge, this is the first design and implementation of personal firewalls for Pocket PC 2003 device which is explicit, open source and successfully implemented in any Pocket PC 2003 compatible devices, including HP iPaq H5550 and XDA2 02 smart phone.



## Disciplines

Physical Sciences and Mathematics

## Publication Details

This article was published as: Susilo, W, Ang, RJ, McDonald, CAG & Huang, J, Personal firewall for Pocket PC 2003: design & implementation, Proceedings 19th International Conference on Advanced Information Networking and Applications, 28-30 March 2005, vol 2, 661-666. Copyright IEEE 2003.

# Personal Firewall for Pocket PC 2003: Design & Implementation

Willy Susilo, Russell James Ang, Cameron Allen George McDonald, and Jianyong Huang
University of Wollongong, Australia
Email: {wsusilo, rja02, cam19, jyh33}@uow.edu.au

## Abstract

*Personal Digital Assistants (PDAs) are widely used and becoming indispensable tools of everyday life. Wired or wireless connection enables PDA users to connect to the Internet from any place, making security an extremely important issue in a pervasive computing environment. This paper investigates how to build a personal firewall for PDAs running Pocket PC 2003. This personal firewall allows the PDA user to perform access control based on a user-defined policy, and hence provides a security perimeter between the public network and the PDA. We provide a complete technical detail on how the firewall can be built in a Pocket PC 2003 device. To the best of our knowledge, this is the first design and implementation of personal firewalls for Pocket PC 2003 device which is explicit, open source and successfully implemented in any Pocket PC 2003 compatible devices, including HP iPaq H5550 and XDA2 0 2 smartphone.*

## 1. Introduction

The increasing integration of wireless networking technologies such as 802.11b into PDAs has meant that users can connect to public and private networks without the need for wires. Additionally, as wireless hotspots become more common, users will be able to connect to the Internet from public places. Unfortunately, this means that malicious users also have the capability to connect to such devices without the need for wires. Public hotspots can serve as attractors for both legitimate users and malicious users.

In such a pervasive connected environment, network security is critical to protect devices and the information they contain. Central to network security is the firewall, which acts as a barrier between an untrusted network and the trusted network (or device). Due to high-profile hacker attacks and proliferation of viruses, firewalls have become a fundamental security tool. At this stage, no security system can ensure with absolute certainty that all of its information will be protected all of the time. Firewalls are one of the most effective security tools that a network administrator can deploy to limit vulnerability.

A firewall is hardware or software that monitors the transmission of packets of digital information that attempt to pass through the perimeter of a network. A firewall can be described as a system for enforcing *access control policy* between two networks. A firewall can deny unauthenticated requests or request, with potential threats, while permitting authenticated requests, thus protecting the internal network. In most cases, firewalls are used to prevent outsiders from accessing an internal network. However, firewalls can also be used to guard one highly sensitive part of a private network against its other parts. Such sensitive parts are for payroll, payment processing, R&D systems, etc.

There are essentially three types of firewalls: 1) packet-filtering firewalls, 2) stateful firewalls and 3) application-level firewalls [9]. A packet-filtering firewall drops packets based on their source or destination address or ports. For example, it blocks all packets from a site that we do not trust, or block all packets to an internal machine that should be inaccessible from external network. Linux's *ipfwadm* and *ipchains* are examples of this type of firewall. A stateful firewall works at the session layer of the OSI model, or the TCP layer of TCP/IP. It monitors TCP handshaking between packets to determine whether a requested session is allowed. It creates a circuit then relays data between external and internal network. Stateful firewalls keep track of sessions and connections in internal state tables. It can protect against certain types of denial of service attacks. An example of this type of firewalls is Checkpoint FW-1. In addition to lower levels, application-level firewalls inspect packets at the application level. A packet reaches the firewall and is passed to an application-specific proxy, which inspects the validity of the packet.

Newer types of firewalls include distributed firewalls and personal firewalls, In distributed firewalls, security policy is centrally defined but is enforced at individual network end points, such as routers, gateways, servers or user PCs. Policy can be distributed to end points in various forms. For example, policy can be pushed to end points where such policy should be enforced. Policy may also be provided to

Proceedings of the 19th International Conference on Advanced Information Networking and Applications (AINA'05)
1550-445X/05 $20.00 © 2005 IEEE

the users as credentials that will be presented when needed.

Unlike traditional firewall sitting between the external and internal networks, a personal firewall runs on a user's computers. Personal firewalls do not have problems such as end-to-end network encryption, malicious insider attacks, single point of failure. etc. In this paper, we are interested in building a personal firewall for Pocket PC 2003 handheld devices. We will discuss our design and implementation in detail.

### 1.1. Review on Handheld Devices

Handheld computers/devices have come a long way in just a few years, but currently they have become indispensable part of everyday's life. Currently, there are two major operating systems that are applied on handheld devices: Palm OS and Pocket PC. Pocket PC is developed by Microsoft with the aim to extend today's personal computer platform to mobile computers. It provides essential PIM (Packet Information Manager) style applications on the handheld, as well as expansion, synchronization and handwriting recognition. However, the implementation is dramatically different from manufacturer to manufacturer. The OS is condensed to work on a PDA (Personal Data Assistant) with a small display and no keyboard. The Pocket PC operating system is immediately familiar to Windows users. The most current version of the OS is Packet PC 2003, also known as Windows Mobile 2003. Pocket PC 2003 directly supports mobile-phone hardware to provide a standard user interface for phone functions across all devices.

These days, important information is often stored in a PDA. Therefore, the security of the PDA is essential, especially when we consider that the user will connect the PDA to the Internet. This security issue has been considered in the design of the hardware, for example by incorporating a fingerprint detection mechanism before allowing people to access the contents of the PDA. It was announced recently that the security of the PDA will be embedded into the operating system itself for the future OS, known as the Windows CE 5 [8]. A new report from the Burton Group says that data encryption capabilities should be the centerpiece of mobile device configuration [4].

There have been several studies in the security of Palm OS devices, for example [2, 3]. Recently, the first virus, called "Duts", to attack handheld computers running Microsoft's Windows Pocket PC software has been found [1]. This virus is a parasitic file infector. Upon infecting files, it appends itself to the host file, modifying the entry point to point to the virus body. Moreover, the first backdoor Trojan horse against Pocket PC mobile device, known as "Brador", has been detected [7]. This backdoor program may give an attacker complete control over the handheld by sending the IP address of the infected device to the attacker and opening TCP port 2989. Building firewalls for wireless network has been studied in [6]. In this paper, we focus on building a personal firewall on Pocket PC 2003 platform.

### 1.2. Our Contributions

We have designed and implemented a personal firewall for Pocket PC 2003, which filters TCP/IP traffic passing to and from the device via an Ethernet card. The concept can be extended to allow other protocols such as 802.11b to be covered. The personal firewall is a packet-filtering type firewall, which analyzes the contents of network packets. The firewall examines, if applicable:

1. source and destination IP address
2. source and destination port
3. protocol
4. HTTP domain and URL

It establishes the validity of network packets, based upon a strict set of rules. Depending on the packet validity, the firewall either accepts or rejects the network packet. If the firewall encounters a packet that is suspicious or unknown, the user will be notified, depending on the security level, and asked to respond with the appropriate action to be taken.

The rest of this paper is organized as follows. In the next section, we will describe our design of personal firewall for Pocket PC 2003 devices. In Section 3, we discuss our implementation detail of personal firewalls in Pocket PC 2003 devices, We provide some screenshots of our implementation at the end of this section. Section 4 concludes the paper.

## 2. Design of Personal Firewall for Pocket PC 2003

The design of our firewall consists of three separate modules:

1. `IMDrv.dll` - network device driver
2. `ChkPkt.dll` - dynamic link library
3. `FWApp.exe` - application

The relation between these three modules are illustrated in Figure 1. We will elaborate these modules in the following subsections.

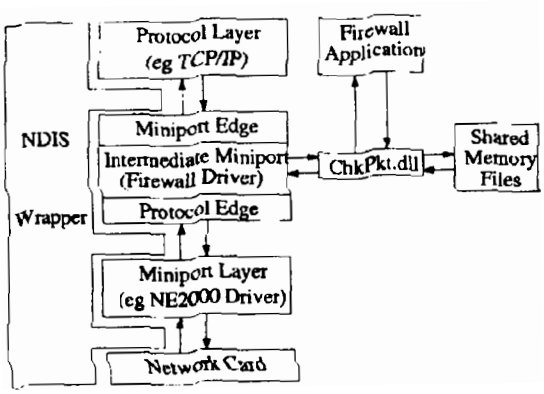


**Figure 1.** *Relationships between the three modules*

## 2.1. Network Device Driver

The network device driver, `IMDrv.dll`, is responsible for capturing all incoming and outgoing network packets. It is an Intermediate Miniport (IM) driver, which is a low-level interface that separates the miniport driver layer from the protocol driver layer. The IM driver is situated above the miniport layer, and below the protocol layer. These drivers communicate via NDIS (Network Driver Interface Specification). Microsoft networking protocols use the NDIS interface to communicate with network card drivers. The NDIS implementation on Windows CE is a subset of the NDIS 4.0 implementation used on Windows NT [5]. The IM driver consists of a protocol driver interface at its lower edge, and a miniport driver interface at its upper edge. The protocol interface of the IM driver receives data (network packets) from the miniport layer below, these are termed *incoming network packets*. The miniport interface of the IM driver receives data (network packets) from the protocol layer above, these are termed *outgoing network packets*. Once the IM driver receives a packet, it sends the packet data to the `ChkPkt.dll` for content checking. The IM driver receives notice from the `ChkPkt.dll` regarding the packets authenticity/validity. The IM driver is then responsible for taking appropriate action depending on the packets validity. For valid incoming packets, the IM driver relays the packet to the protocol layer above. For valid outgoing packets, the IM driver relays the packet to the miniport layer below. For all invalid packets, the IM driver drops the packet, by not relaying it to the appropriate layer.

## 2.2. Dynamic Link Library (DLL)

The dynamic link library, `ChkPkt.dll`, has two major functions. It is responsible for analyzing the contents of all network packets, and provides a link between the application and the IM driver.

*Packet Filtering*

`ChkPkt.dll` receives a network packet from `IMDrv.dll`, and disassembles it into the appropriate sections. First the IP header is extracted, exposing the source IP, destination IP, and protocol type. These components are checked against a list of allowed records. Secondly, if the protocol type is either TCP or UDP, the appropriate header is extracted, exposing the source and destination port. These components are also checked for validity. Finally, if the protocol is TCP, and contains HTTP data in the body, the URL is extracted from the HTTP data. The URL is then checked against a list of domains and keywords. After all the packet component checking has been done, `ChkPkt.dll` then reports back to `IMDrv.dll`, conveying the validity of the packet.

*Application - IM Driver Link*

`ChkPkt.dll` acts as a link between the user mode application (`FWApp.exe`), and the kernel mode IM Driver (`IMDrv.dll`). `ChkPkt.dll` receives data from the application, including the specific criteria for allowed network packets. It is responsible for keeping an up-to-date list of this criteria, which it uses to check against the packet components. In the case of examining a suspicious/unknown packet, `ChkPkt.dll` will notify `FWApp.exe` and prompt the user to reply with the appropriate action. This is dependent on the security level, which will be discussed later. The reply is received from `FWApp.exe`, and `ChkPkt.dll` then notifies `IMDrv.dll` of the intended action.

## 2.3. Application

The application, `FWApp.exe`, is the user interface which allows the user to specify the criteria for which network packets are examined against. It contains a setting section where the user can specify which IP addresses to block, which network protocols are allowed, which ports are allowed to listen/receive data. The user can also specify what domain names to block, and also what URL keywords to block. This criteria is then sent to `ChkPkt.dll`. `FWApp.exe` is also responsible in notifying the user of suspicious packets that are entering/leaving the device, and retrieving the user's response to send to `ChkPkt.dll`. The firewall operates on effectively four different security levels (illustrated in Figure 2).

1. *Low* - All network packets are accepted.
2. *Medium* - Network packets are accepted/rejected based on the criteria set by the user. If a suspicious/unknown packet is found, the user is asked to submit the appropriate action.
3. *High* - Network packets are accepted/rejected based on the criteria set by the user. If a suspicious/unknown

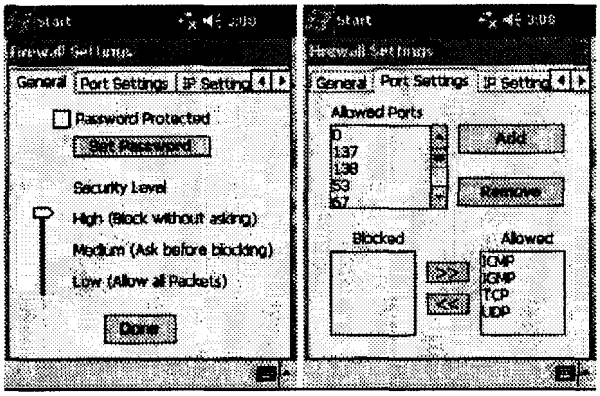


**Figure 2.** *Choosing Security Level and Setting*

packet is found, the packet is automatically rejected. The user may get information about the rejected packet if necessary.

4. *Block All* - This is an emergency like state, where all network packets are rejected.

The firewall security setting can be changed in the firewall settings section (Figure 2).

## 3. Personal Firewall Implementation

### 3.1. IMDrv.dll

The implementation of `IMDrv.dll` is based upon the *Passthru intermediate Driver for Windows CE 4.0,* which is readily avaiiable from Microsoft.

For an IM driver to be loaded it is required that there is registry key in HKLM→COMM specifying the driver's credentials. This key must have a group field with value "NDIS". On startup, the system loads all drivers with this "NDIS" group field. The system first calls the driver's **DriverEntry** function, which is the initialization phase for the driver. The DriverEntry function creates the binding between the IM driver and the NDIS library. There are three fundamental tasks performed by DriverEntry:

1. **NdisMInitializeWrapper** - Notifies NDIS that the driver is about to register
2. **NdisIMRegisterLayeredMiniport** - Registers as a Layered Miniport, telling NDIS the envy points for its Miniport edge
3. **NdisRegisterProtocol** - Registers as a Protocol, telling NDIS the entry points for its Protocol edge

Once initialization has taken place, `IMDrv.dll` is ready to function as an IM Driver. `IMDrv.dll` is responsible for handling both incoming and outgoing network packets (Figure 3).

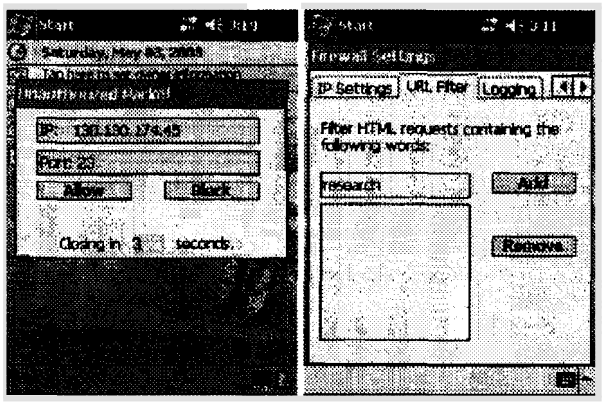


**Figure 3.** *Capturing Packets and Filtering Words*

*Outgoing Network Packets*

`IMDrv.dll` receives outgoing network packets at its miniport edge. The entry point for the packets is **MiniportSend** and **MiniportSendPackets**, specified in the call to **NdisIMRegisterLayeredMiniport**. The packet is received as a NDISPACKET, a network packet covered by a NDIS wrapper. To extract the contents of the NDISPACKET, a call to **NdisGetFirstBufferFromPacket** is made. This function returns the first buffer associated with the packet. The first buffer contains the ***Ethernet Header,*** which we are not interested in. We then make a call to **NdisGetNextBuffer**, which returns the second buffer in the packet. The second buffer contains the ***IP header and body.*** If the packet is a TCP/IP packet, there is often HTTP data in the packet as well. Another call to **NdisGetNextBuffer** returns the third buffer which contains the HTTP data. Once we have retrieved all the required buffers, we need to call **NdisQueryBuffer** on each buffer. This returns the data in the form of an array. We then send the data to ChkPkt.dll for checking (this will be examined in the next section). We then receive a reply from `ChkPkt.dll` in regards to the packet's validity. If the packet is accepted, then it is sent down to the miniport driver using **NdisSend**. If the packet is rejected, we do not send the packet, we return NDISSTATUSSUCCESS immediately. This response tells NDIS that we are finished with the packet.

*Incoming Network Packers*

`IMDrv.dll` receives incoming packets at its protocol edge. The entry point for the packets is **ProtocolReceive**, specified in the call to **NdisRegisterProtocol**. The packet is received in the form of two arrays, *HeaderBuffer* and *LookAheadBuffer.* The HeaderBuffer contains the Ethernet header, while the LookAheadBuffer contains the IP, TCP/UDP data. It is the LookAheadBuffer that we are

interested in, which we send to ChkPkt.dll for checking. We receive a reply from ChkPkt.dll in regards to the packet's validity. If the packet is accepted, then it is indicated up to the above protocols using **NdisMEthIndicateReceive, NdisMTrIndicateReceive, or NdisMFddiIndicateReceive,** depending on the hardware (Ethernet, wireless, etc.). If the packet is rejected, we do not indicate the packet ahead, we return NDIS_STATUS_SUCCESS, telling NDIS that we are finished with the packet.

## 33. ChkPkt.dll

ChkPkt.dll has two main tasks, content checking of incoming/outgoing packets and maintaining an up-to-date list of user supplied criteria (Figure 3). There are 6 separate lists: IP address, protocols, source ports, destination ports, domains, and URL keywords. Each of which is controlled by ChkPkt.dll.

*Outgoing Network Packets*

ChkPkt.dll receives the outgoing packet data from IMDrv.dll. A simple **memcpy** call copies the first appropriate number of bytes into an IP Header struct. From this we can easily access the source and destination IP, as well as the protocol type. We check the destination IP against the list of blocked IP addresses, and also, the protocol type against the list of allowed protocols. If either of these return a match, we respond to IMDrv.dll with a packet rejection flag. Otherwise. we continue checking the data. If the protocol is TCP or UDP, we copy the next appropriate amount of bytes into its respective header struct. From this we can easily access the source and destination port. We then check these components against the appropriate list. If the protocol is TCP and contains HTTP data. we extract the URL from the **"HTTP GET"** section of the data. We then extract the domain name from the URL, and check it against the list of blocked domains. We also scan the entire URL, checking against the list of blocked keywords. If any match is found, we respond to IMDrv.dll with a packet rejection flag.

*Incoming Network Packets*

ChkPkt.dll performs similar tasks on incoming packets, as it does with outgoing packets. There are a few minor differences, we check the source IP address instead of the destination, and we do not check HTTP data. The port checking section is discussed below.

*Stateful Implementation*

Some programs (eg. Internet Explorer) when accessing the Internet, can open up several ports. Sometimes these ports seem to be random in a sense. When we *are* checking outgoing packets, if the source port is unknown, the packet may or may not be valid. We then check the destination port to see if it is destined for a known port (eg. 80). If the destination port is valid, we then add the source port to a "stateful" list which is checked during incoming packet verification. When we are checking incoming packets, if the destination port is unknown, we can then check it against the "stateful" list to determine if the packet is valid.

*Suspicious Packet*

This section is relevant to Medium and High security levels only. If we receive an incoming packet from an unfamiliar IP address, or that is connecting to an unfamiliar port, we are unsure if the packet has good intentions. The following action is then taken. The details of the packet are recorded and a flag is set which notifies FWApp.exe. If the security level is set to ***High,*** the packet is rejected automatically, If the security level is set to ***Medium,*** ChkPkt.dll waits on the reply from FWApp.exe. If there is not a reply from FWApp.exe (ie. user) within 5 seconds, the packet is rejected. Otherwise, it depends on the reply.

## 33. FWApp.exe

*Application*

In order to dynamically update which IP addresses and port numbers are blocked or allowed by the firewall, we implemented a simple application. Communication between the application and the firewall (for setting IP addresses as being allowed or not allowed, etc.) occurs through the use of shared memory files. A thread was also created, with the task of notifying the user if any unallowed packets are found.

*Application Details*

Once it is launched, the application is minimized and an icon is placed in the Taskbar (the bottom right hand side of the screen). This is achieved through the **Shell_NotifyIcon** function. Once a user 'clicks' on the icon, a pop-up window is displayed using the **TrackPopupMenu** function, giving the user the option to change the settings, view the details of a blocked packet, enter 'block all' mode, or exit the application (close it permanently).If the user chooses to modify the firewall settings, a **PropertyPage** is created containing a number of **PropertySheets**. PropertySheets are similar to Tab controls commonly found in other PC applications. Each PropertySheet contains a set of firewall settings the user can change, grouped according to common attributes. For example, all changes to port numbers, such as allowing destination and source ports. are all done in one PropertySheet. Changes to these firewall settings are saved in the firewall applications' settings file. These changes are also reflected in the shared memory file, which is used by the firewall module. Accessing the settings section of the application also allows the user to view a log of any packets that were blocked, including the port numbers and the type

of packet that was blocked. (For example TCP. or UDP). This information is stored in a plain text file, and can be reset by the user through the settings section. Once these changes are made, the user closes the PropertyPage, and the firewall application returns to the background. Note that most applications exhibit this behaviour on the Pocket PC platform when exiting.

*Shared Memory* Files

In order to communicate between the application and the firewall, we use a set of shared memory files. These are hidden files existing for the duration of the firewall's existence, and can be written and read by both the firewall and the application. Because these files on the Pocket PC device exists only on RAM, as opposed to permanent storage such as a hard disk, the performance impact of this design decision is minimal. The shared memory files are created using the **CreateFileMapping** and **MapViewOfFile** functions. These set aside memory for the files, and maps a variable onto the files' position. Hence, reading and writing to and from the file is a simple task of reading and writing to and from the variable. Once we are finished with the shared memory files, we un-allocate the memory via the **UnmapViewOfFile** function.Upon receiving a network packet, the firewall checks the shared memory file to check whether it is from a blocked IP address or port. If it is, it writes to another shared memory file that a packet has been blocked. and the packet is not passed on to its destination. Otherwise, the packet is passed on as usual. In this way, the firewall can communicate back to the application any details of blocked packets.

Threads

When the application is run for the first time, a thread is created. This is achieved through the use of the **CreateThread** function. The only function of this thread is to continually poll a shared memory file, looking for indications that a packet has been blocked by the firewall. Once the shared memory file indicates this has occurred, the contents of the file are read, (containing the details of the packet such as IP address and source and destination port numbers), and the file is reset to its original state. The thread then indicates to the user that a packet has been blacked. This can be done explicitly by bringing up a message box, or discretely as a simple change in the icon in the Taskbar, depending on the settings chosen by the user in the settings screen.

### 3.4. Limitations

The firewall currently only has the capability of storing details of one blacked packet in memory. However, it is possible with a slight performance cost, to log the details of all blocked packets to file. The firewall also only handles a small number of protocols. However, it demonstrates one possible method of capturing. examining, and destroying any type of network data passing to and from the Pocket PC device. Hence an extension of the firewall could be performed to allow support for other network protocols and other features. The application runs with a minimal footprint on the system (the application is less than 40k in size). Because of this, the performance impact of the application running in the background is not noticeable. The threaded portion of the application also has no observable impact on performance of the system. Heavy network traffic passing to and from the Pocket PC device, however, has a noticeable impact on the performance of the system, especially when a verbose logging of the packet information is implemented. When logging only the details of blocked packets, this performance decrease is very minimal. Additionally, the ping times on a local network were found to still be under 10 milliseconds while the firewall was in operation.

## 4. Conclusions

PDAs play an important role in the pervasive computing environment. Along with the growth of PDA usage, there will also be an increase in the number of threats. We designed and implemented a personal firewall for Pocket PC 2003, and this firewall provides a safety barrier between the Internet and the PDA.

## References


[1] BitDefender. *Available online at http://www.bitdefender.com/bd/site/presscenter.php?menu_id=24&n_i d=102*, 1 August 2004.

[2] Kingpin and Mudge. Security analysis of the palm operating system and its weaknesses against malicious code threats. *Proceedings of the 10th USENIX Security Symposium,* pages 135–151, 13-17 August 2001.

[3] N. Leavitt. Malicious code moves to mobile devices. *Computer, December 2000,* pages 16–19, 2000.

[4] I. Lynch. Encryption key to mobile data security. *Intellisync report, Online.*

[5] Microsoft. *Microsoft Windows CE Developer's Kit (Microsoft Professional Editions).* Microsoft Press, 1 April 1999.

[6] U. Murthy, O. Bukhres, W. Winn, and E. Vanderdez. Firewalls for security in wireless network. *Proceedings of the 31th Hawaii International Conference on System Sciences, 5-8* January 1998.

[7] PC World. *Available online at http://www.pcworld.idg.com.au/index.php/id;2020686279;fp;16;fpdid;0,* 10 July 2004.

[8] D. Robinson. Windows CE gains new capabilities. *Intellisync report, Online.*

[9] E. Zwicky, S. Cooper, and D. Chapman. *Building Internet Firewalls.* O'Reilly & Associates, 2nd edition, 15 January 2000.