available at www.sciencedirect.com

ScienceDirect

www.compseconline.com/publications/prodinf.htm

Information Security Technical Report

Windows device interface security

Stephen D. Wolthusen

ABSTRACT

This paper discusses both risks and mitigation strategies for risks and threats associated with physical device interfaces. To this end, a brief discussion of the I/O architecture found in the Microsoft Windows operating system is followed by a review of several classes of attacks possible using only external devices attached to standard device interfaces of host computers. Based on this analysis, a selection of possible countermeasures including the modification of the host operating system by wrapping the I/O mechanisms into a hardened protective layer is discussed.

1. Introduction

While the number of potential attackers over a network connection is much larger than adversaries within physical proximity of a target system, it is nevertheless worthwhile to consider the threats and security countermeasures available for securing device interfaces. This is particularly relevant for mobile devices that are potentially exposed to a large number of individuals, but is not necessarily confined to this class of devices as even a momentary lapse of physical supervision (e.g. in a meeting) may be sufficient to initiate an attack. In addition to external attackers, these interfaces also represent a threat vector for internal users intending to subvert systems or to gain unauthorized privileges without leaving readily identifiable traces that would occur in other cases. Depending on an overall risk assessment, the threats described here may even represent a major concern in environments that have stringent network security mechanisms in place or are even protected by an air gap from external networks since the threats described in this paper can provide an efficient pathway for subverting systems to gain privileged access or to extract information from a target system surreptitiously while retaining plausible deniability since the devices used for conducting these attacks are inherently dual-use in their nature (e.g. smart phones or media players) (Arce, 2005); at the same time locking out all interfaces is increasingly infeasible since dedicated ports, particularly for human interface devices, are being phased out.

This article focuses on the risks posed by the use of devices running the Microsoft Windows (2000/XP/Vista) operating system along with several typical application programs found in commercial environment with a focus on the host to which the devices are attached. To this end, Section 2 describes the pertinent aspects of underlying operating system structure while Section 3 identifies a number of threats, both generic in type and also exemplified with specific scenarios. Several possible countermeasures and tactics are then described in Section 4.

2. Microsoft Windows device interface architecture

The Microsoft Windows NT architecture (encompassing later revisions including 2000, XP, 2003, and Vista) is based on a layered design in which the central abstractions are file objects (similar to Unix derivatives) which are also used to represent devices and device drivers. However, while the external interface presented to application programs is procedural (represented by environmental subsystems such as the native Win32 or the rarely used POSIX subsystem), the underlying operating system itself is asynchronous and packet-based. Fig. 1 omits the procedural interface layer and several other kernel components not immediately relevant while identifying the component interactions involved in device I/O relevant to this discussion (Russinovich and Solomon, 2004; Solomon and Russinovich, 2000; Oney, 2002).

E-mail address: stephen.wolthusen@rhul.ac.uk

1363-4127/$ – see front matter © 2006 Elsevier Ltd. All rights reserved.
doi:10.1016/j.istr.2006.10.001

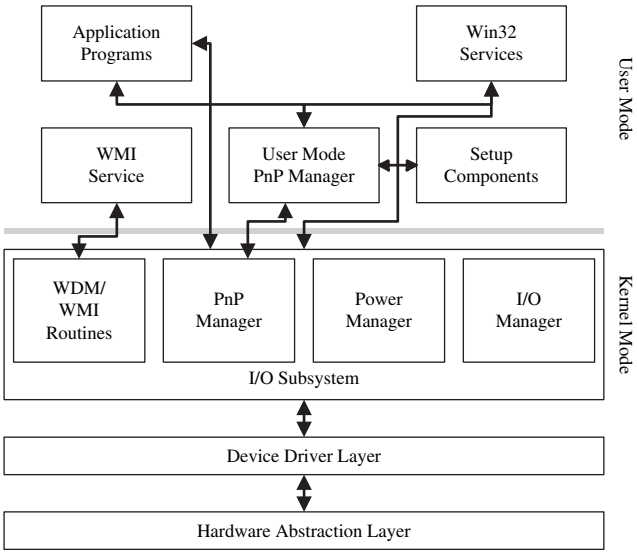

Fig. 1 – Windows I/O architecture.

In this model, I/O requests originating from environmental subsystems are routed through the native system services API and the kernel-level I/O manager. The I/O manager dispatches I/O request packets (IRP) to device drivers that are registered with it (exceptions to this model exist, but are of no significance to the following); ultimately the device drivers interface with the hardware abstraction layer (HAL) which provides access to the physical ports and memory areas required for interfacing with the devices proper. While monolithic drivers exist, the device driver layer is typically subdivided into several devices. General I/O processing for a class of devices is provided by class drivers while processing for types of ports such as USB are frequently handled by port drivers. Specific device instances within a type of port are then handled by miniport drivers relying on the support of port drivers.

Moreover, the IRP-based I/O structure permits the insertion of a type of driver called filter drivers into arbitrary positions of this device driver stack and to require processing both in the control flow direction towards the hardware as well as in the reverse direction. This enables both pre- and post-processing steps for each IRP; the latter are also possible within individual drivers in the form of I/O completion routines that are called by the I/O manager after the processing of an earlier step; the potential for exploiting this mechanism for defensive purposes is discussed in Section 4.

If a device requires the attention of the operating system apart from possible regular polling intervals, it is required to post an interrupt (there may be several types of interrupt depending on bus and device type, some of which are handled at the bus interface). If such an interrupt is raised, the appropriate device driver registered for the given interrupt enters an interrupt service routine (ISR) in which the operations required for servicing the device are either performed directly in trivial cases or, more frequently, transformed into a deferr ed procedure call (DPC), which then completes the required operations at a lower interrupt privilege level, avoiding blocking the remainder of the system. All of these processing steps must be handled by filter drives as well. As can be seen from the above discussion, inserting filter drivers into the processing stack permit the insertion of instrumentation points for monitoring and auditing as well as for access and other behavior-based controls for existing, loaded device drivers; the typically modular structure of the device drivers (class, port, and miniport drivers) permits the efficient interception of several generic device interfaces without requiring knowledge of device drivers for individual device models.

Additional device driver functionality that can also be made subject to interception includes I/O cancellation routines that are called whenever an I/O operation is canceled either explicitly or by termination of the thread that caused the original IRP to be issued; since such operations may not only result in cleaning up of data structures but also involve operations on physical devices, they must be intercepted and made subject to security policies as well.

3. Threats

Regardless of the operating system type used, several broad threat categories can be identified, some of which depend on the type of interface and level of dynamism in the respective operating system, which must be considered in addition to threats against the mobile devices themselves (Susilo, 2002).

While all of these attacks (with the exception of some wireless attacks such as via Bluetooth that are not covered explicitly here and bridging physical interface protocols over wireless links) require physical presence or at least proximity to the target system, they nevertheless represent a significant subversion and industrial espionage threat that is particularly easy to conduct for insiders.

3.1. *Application control*

The underlying paradigm used for allocating various devices attached via interfaces such as serial (RS-232C), parallel (IEEE 1284), USB, FireWire (IEEE 1394), and Bluetooth in most currently dominant COTS operating systems including various Unix derivatives, Linux, and also Microsoft Windows permits the association of application programs with devices and interfaces without subsequent intervention. This model is generally limited to access control mechanisms operating on the device interface as the sole entity (e.g. access control to a device object under Microsoft Windows or device special files under Unix derivatives). While some mechanisms may impose certain dynamism such as temporal restrictions of device access, there are few limitations imposed on communicating through an interface once access has been granted.

As a result, the enforcement of any security policy (e.g. access controls) typically devolves to the area of responsibility of the application program or, beyond this, the user of the application program. Thus, if an operating system otherwise enforces security policies with regard to other external interfaces such as network interfaces and storage interfaces including ones for removable media, this leaves a gap in the enforcement mechanism suite that could be exploited by both malicious users and external threats.

An example of such an operation is the use of data synchronization mechanisms for PDAs or similar devices; it is

trivial to initiate a synchronization (i.e. transfer) of otherwise protected data from a workstation to an untrusted device; typically all that is needed to circumvent the rudimentary authentication policy is to present a completely blanked-out device in which only the user name of the victim has been entered. Default policies will assume the attacker's device to be a legitimate one whose memory has been erased (e.g. through loss of power) and will proceed to upload all data designated for synchronization on the user's desktop. Such an operation can, depending on the interface used, be performed clandestinely within seconds while the legitimate user of the workstation does not notice or is absent from the workstation's console (which can, e.g. occur in a populated area such as an airport lounge). Moreover, the fact that such a data transfer has occurred may not be readily apparent and visible only in audit data that is likely to be used infrequently if at all. Typically, no authentication (e.g. password) is required, and the user name is easily guessed. The culprit here is clearly that the application program (i.e. the synchronization software) implicitly assumes that any device physically present is authorized to receive and transmit data. Similar problems arise with other storage and communication devices whose policies are based on similar assumptions.

For storage devices as well as for synchronized devices (e.g. smart phones or PDAs), the threats are not merely the extraction of information but also the possibility of injecting malware into systems that may not have virus scanning and detection active at all times.

3.2. *Identification and authentication*

Both common operating systems and application programs typically do not identify and authenticate devices and application programs (or users), regardless of whether the device is configured statically or dynamically (see Section 3.3). In many cases the underlying devices do not provide for such mechanisms themselves (e.g. in case of human interface devices attached via radio or infrared interfaces, these commonly employ only limited disambiguation), which can lead to undesirable interactions at both functional and security levels. In other cases, the identification and authentication mechanism does not, in addition to potential weaknesses, e.g. in the strength of cryptographic mechanisms, establish the identity of the communicating entities at the semantic level required. As an example, the Bluetooth pairing mechanism[1] establishes only knowledge of the PIN code, not the identity of a device or even of a subject controlling such a device. Frequently, the devices attached to a host system do not identify individual users but rather assume that the possession of the device either implicitly identifies the user or that, conversely, the user identity on the host system pre-determines the ownership of the attached device.

In the example described in the preceding section, common PDA software performs an identification verification on initiation of a data synchronization process by the user, but does not authenticate this information. As a result, no additional user intervention (e.g. if a legitimate user is absent and has locked the console) is required, and an attacker can trivially prepare a PDA with the requisite user identity derived heuristically or from unrelated communication. Similar threats arise from devices in bus or broadcast configurations taking over unauthenticated device identifiers without causing reconfiguration to take place (see Section 3.3); while such operations may be detected in case of simultaneous operations of both the legitimate device and the attacker's device, human interface devices are particularly susceptible to this type of attack since the shared medium (bus or wireless broadcast) is rarely contended. Taking over an unused or temporarily dormant identifier can also be used in an attack, e.g. for eavesdropping on USB bulk data transfers between a host and a legitimate device such as those provided e.g. by the KeyGhost[2] product. The configuration mechanisms of the Windows platform (see also Section 3.3) make even forensic analysis problematic as the system does not retain information on device usage if, e.g. a device with the same vendor and device ID as one that has already been encountered is attached; this can allow an adversary to either use a similar device or mimic such a device at the protocol (e.g. USB) level to avoid being detected while retaining the ability to also perform additional actions such as (USB) bus snooping.

3.3. *Dynamic configuration*

A crucial feature permitting attacks described above and one that is potentially problematic in its own right is the dynamic and automatic configuration mechanism for new devices and devices instances integrated into operating systems. In most Unix derivatives this is somewhat limited, and although, e.g. the Sun Solaris USB framework includes nexus drivers supporting mass storage profiles and hence also has full volume manager support for USB-based mass storage and removable media, most Unix systems and Linux depend on individual device drivers to support one or more dynamically (or Plug-and-Play, PnP) configured devices.

The Windows NT family of operating systems, however, includes extensive support for PnP (beginning with the Windows 2000 release) and therefore faces several threats that do not exist in the previously mentioned systems. Here, devices found both during the boot process and identified at runtime are activated. This occurs regardless of the privilege level of the user or users currently logged in. If, for a given device, no device driver is currently loaded, the PnP manager will attempt to install a driver for such an identified device. It is particularly noteworthy that if the system contains the setup components for the device (which is frequently the case since the Windows NT family by default includes a repository of device drivers and setup components on installation), even this reconfiguration will occur regardless of the currently active users (see Fig. 1).

Such device drivers, even if they are not Trojan horses installed by an adversary, may cause undesirable interactions with existing components or permit the attachment and

[1] Establishing a shared secret (a PIN code) for symmetric channel encryption and authentication using an out-of-band mechanism.

[2] A family of products by KeyGhost Ltd., Christchurch, NZ, which perform purely hardware-based human interface device action logging, corporate URL: http://www.keyghost.com.

operation of devices along with application programs and system services automatically installed along with the device driver by the setup components (which operate at administrative privileges) that contradict security policies in effect for a given system. It is therefore possible to induce a demonstrably insecure system state by having a system recognize an additional or new device without requiring the presence and actions of an authorized user or even elevated privileges. Depending on the device class of the newly installed device (e.g. a networking component), this can – together with other default settings such as the automatic association with the wireless network access point providing the strongest signal – also result in transitive threats such as the possibility of other devices joining in an ad-hoc network that violates security policies and is not adequately protected by the organization's firewalls.

3.4. *Direct memory access*

A threat class related to those described in Section 3.2 can be identified in semi-autonomous communication mechanisms that are not captured within the framework of network communications and are hence not protected by firewalling mechanisms that have become both mandatory and ubiquitous for network links.

The specifications of both the USB and particularly the IEEE 1394 (also known as FireWire or iLink by some vendors) device interfaces includes the requirement for a direct memory access (or DMA) capability. This mechanism is intended to allow transfers to and from main memory to be initiated and controlled by the device without intervention by the central processing unit of the host computer and is primarily intended to offload I/O processing requirements for time-sensitive applications such as video capturing or streaming where jitter induced by the CPU not servicing interrupt-based I/O is unacceptable.

However, similar to the problems discussed in Section 3.2, there is an implicit assumption of trust built into the device drivers which faithfully provide DMA capability, the assumption being that the host system will initiate such transfers and set up the appropriate memory mappings before commencing data transfers in a controlled fashion. This, however, is merely convention and not covered by the interface specification. It is therefore possible to initiate a DMA transfer from the device side and have direct access to arbitrary memory locations within the host computer.

While it is not trivial to target specific memory locations since the DMA transfer can only operate on physical memory addresses and cannot immediately access the virtual memory page tables required for resolving the VM addresses used by the host operating system, it is nevertheless possible to circumvent all memory protection mechanisms and access arbitrary memory locations both for read and write access, including memory used by the kernel. An attacker can, e.g. using a portable media player such as an Apple iPod, take a full memory dump of the host system and then either search for credentials or other data items of interest. Subsequently the attacker may then, manipulate stored credentials (e.g. by substituting hashed passwords or private keys for PKI certificates), change security settings for alternative entry routes, or install code (e.g. a root kit). Even a passive memory dump is obviously a significant threat since it contains a snapshot of all applications and their loaded credential data, even if the respective memory has been released since the respective ''dirty'' VM pages may not have been reclaimed.

3.5. *Device driver quality*

An emerging indirect threat can also be observed in that attackers are increasingly targeting flaws and limited error handling in device drivers to subvert host systems. This can be accomplished by sending malicious protocol data units or exploiting insufficient parameter validation mechanisms on the part of the device drivers.

Given that drivers are required to run with SYSTEM privileges, this allows attackers to subvert host systems directly without network-based intrusion detection and prevention systems being able to detect this type of attack. Besides requiring an interface matching with the one the device driver on the host is servicing, this class of attacks simply requires the attacking device to be capable of running general-purpose code and also to be able to modify protocol data units. In this, it is similar to the attacks discussed in Section 3.4. Since performance requirements for the attacks are generally very modest or non-existent, a simple portable media player or smart phone is generally sufficient to conduct this type of attack. The attack itself (like others described above) can, e.g. be conducted during an unobserved moment in a meeting or when the targeted system is otherwise unattended and unobserved for 1 or 2 min (Me, 2005).

4. Countermeasures

While it is possible to disable some interface types depending on application profiles (e.g. serial and parallel legacy ports are typically not used in contemporary environments, and few applications explicitly require the availability of an IEEE 1394 interface for normal operations) and such capability is provided in part by the operating system or can be effected by physically blocking or disabling the respective ports, this is not always possible. For some interface types such as the USB bus, this would assume that a given user or all users for a given system will have no legitimate use for the interface and all devices that may be attached to such an interface. This is clearly not the case since at least some human interface devices will need to be attached unless these are integral to the computer system (which may not be admissible based on health and safety regulations in any case for frequently used systems).

Another possible countermeasure is the selective granting of elevated privileges for accessing devices to certain applications or processes; an example of this approach is the device allocation mechanism commonly found in MLS systems (e.g. Sun Microsystems' Trusted Solaris[3]) for removable media and similar constraints which will presumably be part of the Microsoft Vista operating system. While such approaches permit, e.g. the handling of removable media, dynamically

[3] http://www.sun.com/software/solaris/trustedsolaris/.

configured bus systems are subject to similar constraints as in the previously described countermeasure. Given the limitations of such static protection mechanisms, it appears that a security mechanism for dynamic devices and device interfaces should also be itself dynamic and adaptive and be able to enforce any security policy an individual or an organization might have with regard to the admissibility and use of such devices. The following sections briefly describe some of the requirements imposed on effective security mechanisms that counter at least some of the threats described in Section 3.

4.1. *Static device interface security*

Group policies under the Microsoft Windows operating system allow the fine-grained control over device drivers and applications and can be administered centrally. They are, however, limited by their static nature and therefore must be set in advance, anticipating the possible needs of all users of a given system. Since this list of possible access types can be quite extensive despite being needed only in rare circumstances, this violation of the *principle of least privilege* (Saltzer and Schroeder, 1975) is highly undesirable.

A number of third-party vendors have therefore developed software packages which allow the local (as well as centrally administered) reconfiguration of group policies for device access based on explicit requests and circumstantial information (e.g. the use of a certified application requiring access to a specific device interface, running under the account of a user explicitly authorized to perform such operations). This approach counters the threats described in Sections 3.1 and 3.2, to the extent this is possible without direct control of both the application and the attached device, but is limited in its effectiveness against the lower-level attack types also discussed in Sections 3.3–3.5.

4.2. *Dynamic device interface security*

To remedy the remaining deficiencies in the approach described above, it is necessary to modify the operating system as briefly discussed in Section 2. The pertinent control and information flows are depicted schematically in Fig. 2 (Dekker and Newcomer, 1999; Russinovich and Solomon, 2004).

Since the IRP mechanism generally does not carry information as to the identity of the user causing the I/O request, it is necessary to intercept the relevant I/O Manager data structures; this allows the back-tracking of the process owner (i.e. generally either a system service or an application on behalf of a user) and to inter-link the IRPs generated in the process.

At the same time it is also possible and desirable (if only for performance reasons) to resolve the target device at the same logical level; while the environmental subsystem and most of the Windows kernel will operate on file handles, these need to be resolved ultimately to *device objects*, which can be obtained from the handles by way of the internal *file objects*, which refer to *driver objects*. Several device objects can then be associated with each driver object (i.e. a driver may serve more than one device), so disambiguation, typically device- or bus-specific is required on this level as well. While access control decisions can be made at this level (e.g. regarding individual processes

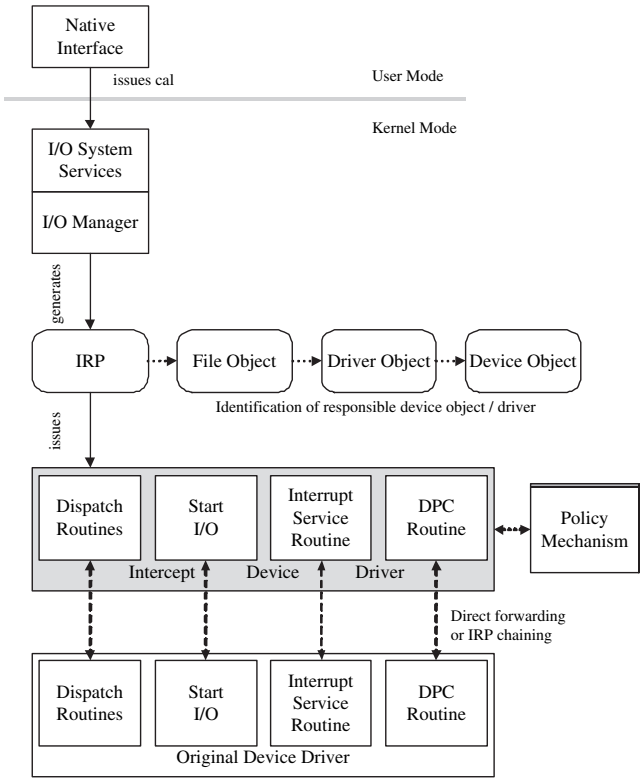

Fig. 2 – Processing of device I/O.

and specific devices), the level of control is still too coarse for a number of security objectives.

The additional interception mechanisms required to regain full control over the operating system's use and handling of devices are shown in a shaded box in Fig. 2. Each intercepted device driver's components need to be filtered for potentially malicious activity. This involves four main components: the command to start a programmed I/O operation or an interrupt service routine initiates an I/O request and will typically require further processing. Some activity that is not in accordance with active security policies can be terminated or modified at this level (e.g. through plausibility checks for parameters or back-links to user and process identification). For more complex operations, however, it is also necessary to monitor and intercept operations performed by device drivers in their respective DPC processing. Finally, some functions such as general housekeeping is typically handled by the dispatch routines of the device driver; these may also require interception. From the description above it should be obvious that while some generic mechanisms for protecting and controlling device drivers exist, finer-grained controls such as detailed parameter validation are dependent on the drivers to be protected.

In addition to device-specific drivers described above, it is also necessary to intercept and filter the relevant bus drivers (e.g. USB and IEEE 1394). While these drivers are typically provided by the operating system and of higher quality than some third-party drivers, enforcing some aspects of security policies such as monitoring and eliminating malicious handcrafted protocol data units (PDUs) or filtering out the DMA

control PDUs must be accomplished at this level before the data packets are handled by additional, higher-level device drivers. This mechanism can also be used effectively to control configuration changes of the bus, preventing among other effects, the commencement of Plug-and-Play activity.

Finally, another control that can be imposed (also at higher levels described above) is precise access control based on device identity and characterization; in most modern interface classes (excluding, e.g. legacy serial and parallel interfaces), devices will identify themselves by providing at least a device class (e.g. human interface devices), a vendor, and a product identification. Some devices can also provide further information such as a unique serial number (this is an optional element, e.g. at the USB level, so it cannot be used for some mandatory controls) and information on the capabilities provided (e.g. the profiles found in the Bluetooth protocol stack), which may also factor in policy decisions.

Several commercial products exist that cover aspects of the countermeasures described above, although no product currently covers all of these elements (Wolthusen, 2003). Moreover, effective protection requires not only the use of interception mechanisms as described in this and the preceding section but also must be configured and maintained to reflect the configurations and security policies of a given organization, which may call for considerable efforts to be expended.

5. Conclusions

This paper has briefly described several classes of threats and risks associated with devices and device interfaces beyond the network devices forming the focus of most security efforts. The potential for damage must be considered significant for environments in which subversion or the injection and removal of data from computer systems exposed (even briefly) to external individuals or to internal users wishing to subvert systems or to gain unauthorized privileges. However, current operating system protection mechanisms, particularly in the Microsoft Windows family of operating systems do not adequately allow the enforcement of security policies that would mitigate these threats. We have therefore briefly sketched the mechanisms which can be used by add-on mechanisms to provide these added capabilities, some of which can already be found in commercially available products, although several of the more intricate threats are not yet addressed by these products.

Observations of the behavior of attackers in recent years clearly demonstrate a migration to newer types of attacks whenever the relative cost for detection and exploitation of a class of vulnerabilities becomes unattractive. Attacks on devices and device drivers are only beginning to draw the attention of attackers, and it is to be hoped that at least in this case a widespread deployment of countermeasures can be achieved before the attacks are in widespread and systematic use.

REFERENCES

Arce I. Bad peripherals. IEEE Security & Privacy 2005;3(1):70–3.

Dekker EN, Newcomer JM. Developing Windows NT device drivers. Reading, MA, USA: Addison-Wesley; 1999.

Me G. Exploiting buffer overflows over Bluetooth: the BluePass tool. In: Proceedings of the second IFIP international conference on wireless and optical communications networks (WOCN 2005). Dubai, UAE: IFIP; March 2005. p. 66–70.

Oney W. Programming the Microsoft Windows driver model. 2nd ed. Redmond, WA, USA: Microsoft Press; 2002.

Russinovich ME, Solomon DA. Microsoft Windows internals. 4th ed. Redmond, WA, USA: Microsoft Press; 2004.

Saltzer JH, Schroeder MD. The protection of information in computer systems. Proceedings of the IEEE 1975;63(9):1278–308.

Solomon DA, Russinovich ME. Inside Microsoft Windows 2000. 3rd ed. Redmond, WA, USA: Microsoft Press; 2000.

Susilo W. Securing handheld devices. In: Proceedings of the 10th IEEE international conference on networks (ICON 2002). San Diego, CA, USA: IEEE Press; August 2002. p. 349–54.

Wolthusen SD. Goalkeeper: close-in interface protection. In: Proceedings 19th annual computer security applications conference (ACSAC'03). Las Vegas, NV, USA: IEEE Press; December 2003. p. 334–41.