Security Analysis of the Palm Operating System and its Weaknesses Against Malicious Code Threats

Kingpin and Mudge

@stake, Inc.
196 Broadway
Cambridge, MA 02139
{kingpin,mudge}@atstake.com

Abstract

Portable devices, such as Personal Digital Assistants (PDAs), are particularly vulnerable to malicious code threats due to their widespread implementation and current lack of a security framework. Although well known in the security industry to be insecure, PDAs are ubiquitous in enterprise environments and are being used for such applications as one-time-password generation, storage of medical and company confidential information, and e-commerce. It is not enough to assume all users are conscious of computer security and it is crucial to understand the risks of using portable devices in a security infrastructure. Furthermore, it is not possible to employ a secure application on top of an insecure foundation.

Palm operating system (OS) devices own nearly 80 percent of the global handheld computing market [11]. It is because of this that the design of the Palm OS and its supporting hardware platform were analyzed. The presented research provides detail into specific scenarios, weaknesses, and mitigation recommendations related to data protection, malicious code, virus storage, and virus propagation. Additionally, this work can be used as a model by users and developers to gain a deeper understanding of the additional security risks that these and other portable devices introduce.

*This paper has been published by The USENIX Association in the *Proceedings of the 10th USENIX Security Symposium*, Washington, DC, August 13-17, 2001, pp 135-151, ISBN 1-880446-07-3.

†Palm OS and HotSync are registered trademarks of Palm, Inc. Other product and company names may be trademarks of their respective owners.

1 Introduction

A new threat model exists for malicious code and virus attacks on portable devices. These threats are no longer contained to common desktop environments. Portable devices employing custom electrical circuit design, product-specific capabilities, and embedded operating systems are commonplace in corporate infrastructure. It is increasingly common for vendors to introduce these devices to an environment before the security ramifications have been examined. PDAs are now being deployed by corporations for security-related applications. Added functionality of wireless technologies, such as infrared (IR) and radio frequency (RF), increases risk areas. New classes of malicious code attacks exist that cannot be detected or contained by current methods long deployed in desktop environments. In addition, the notion of cross-architecture pollination very quickly becomes a mainstream concern. [5] provides an overview of some malicious threats to PDAs and can be read in parallel with this text.

Many users do not recognize that the information stored on their PDA is open to compromise by unauthorized users, and hence do not treat the data stored on their handhelds with the same care as they do on their desktop. Our research discusses the underlying problem that security is not properly designed into the Palm OS platform. Although Palm OS is not presented as a secure operating system, if the device is being used for security purposes, which is becoming prevalent in corporate environments, there are a number of risk areas to be concerned with.

For example, Palm OS offers a built-in Security application which is used for the legitimate user to

protect and hide records from unauthorized users by means of a password. In all basic built-in applications (Address, Date Book, Memo Pad, and To Do List), individual records can be marked as "Private" and should only be accessible if the correct password is entered. Another example is the "Beam Bit" flag contained in every application database, which is used to prevent the information from being transferred, or "beamed", to another device via IR. Honoring the state of the Beam Bit is purely voluntary by the executing application. These simplistic mechanisms lull the user and perhaps some developers into a false sense of security. There should be strong warnings by the vendor that these mechanisms are trivially bypassed (as in §4, §5, and with [14]), so users and developers can plan for and workaround the lack of security. Security-based applications exist on the Palm OS, such as software authentication tokens, cryptographic key storage, and encryption products, all that require a secure operating system in order to be properly implemented. Without proper protection mechanisms in place, applications that rely on the secure storage of secret components are severely at risk of compromise.

The properties of malicious code, particularly viruses, can be distilled into four stages: Infection, Storage, Triggers, and Actions. In this paper, the design of Palm OS is analyzed with respect to each of these stages. A number of weaknesses and attack vectors have been identified from both classical and new technology areas and we offer insight into addressing these problems in design and usage. In no way is this text exhaustive in enumerating attacks. Rather, an attempt is made to educate the reader on the design flaws and new threats that exist on portable devices.

In §2, we provide a summary of the various types of malicious code: viruses, Trojan horses, and worms. §3 describes the typical design and architecture of a PDA, focusing on the Palm OS software and hardware platform. §4 and §5 detail the risks of weak system password storage and backdoor debug modes inherent in Palm OS. §6 through §9 address the four stages of the virus lifecycle with respect to Palm OS.

We conclude that current state-of-the-art portable devices are not equipped for the threat of viruses or other malicious code components. In addition, it becomes apparent that threat models and attack vectors these devices introduce are not yet taken into account by product designers and anti-virus vendors[1]. Hopefully, the various sections of this paper can act as a road map towards the future design of these devices and aid in security awareness for existing deployments.

2 Summary of Malicious Code Types

For the purposes of clarity, we will classify malicious code into three areas [23]:

- **A Virus** is a self-replicating code segment which must be attached to a host executable. When the host is executed, the virus code may also execute. If possible, the virus will replicate by attaching a copy of itself to another executable. The virus may include an additional "payload" that triggers when specific conditions are met.
- **A Trojan horse** is malicious code masquerading as a legitimate application. The goal of the code is to have the user believe they are conducting standard operations or running an innocuous application when in fact initiating its ulterior activities. There are many ways this attack manifests with the most frequent being reliance upon user naivety. A Trojan horse is similar to a virus, except a Trojan horse does not replicate.
- **A Worm** is a self-replicating program. It is self-contained and does not require a host program. The program creates the copy and causes it to execute; no user intervention is required. Worms commonly utilize network services to propagate to other computer systems [19].

3 Palm OS Device Architecture

At the highest level, the architecture of the Palm OS device, and most other PDAs, can be broken down into three layers (Figure 1): Application, Operating System, and Hardware.

Use of the Palm OS Application Programming Interface (API) provides the application developer

[1] Anti-virus software for PDAs is available from a number of vendors, including, but not limited to: Central Command, F-Secure, McAfee.com, Symantec, and Trend Micro.

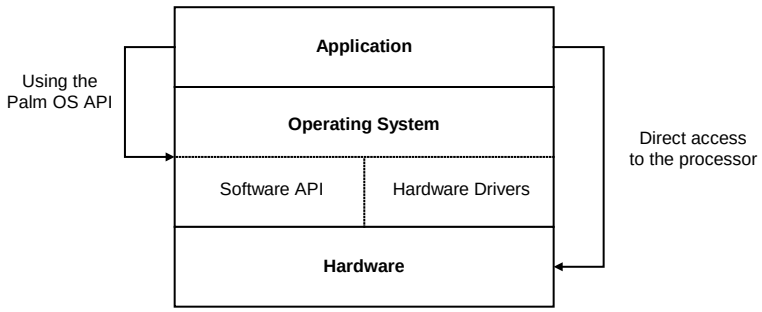

Figure 1: Typical layered architecture of a PDA

with a notion of hardware independence and provides a layer of abstraction. If the API is used properly, recompiling of the application is all that is necessary in order to run on Palm OS devices based on different hardware. Therefore, it is important to examine weaknesses and attack vectors that can be found at the programming interface to the operating system.

Directly accessing the processor by avoiding the interface put forward by the operating system allows the developer to have more control of the processor and its functionality. A risk of legitimate use of direct processor access is the loss of compatibility for future models. For example, older Palm OS devices did not support a grayscale LCD palette through the Palm OS API, even though the underlying hardware possessed this capability. Bypassing this interface and tapping into the functionality of the processor directly will remedy this [13]. Ideally, to provide some semblance of access control and security, only the operating system should have access to the underlying hardware. Allowing applications to directly access hardware provides an avenue for malicious attack (as discussed in §9.2).

3.1 Operating System

Palm OS was designed to be open and modular to support application development by third-parties. The notion of layer- or file-based access control is notably absent. It is not surprising that all program code and data can be accessed and modified by any user or other application. In such uniform memory access scenarios, it is difficult to differentiate between legitimate and malicious applications solely from memory read/writes and system calls.

[20] offers the following overview on file system and application structure:

- Palm OS does not use a traditional flat file system. Data is stored in memory chunks called "records", which are grouped into "databases". A database is analogous to a file. The difference is that data is broken down into multiple records instead of being stored in one contiguous chunk.
- Palm OS applications are generally single-threaded, event-driven programs. Only one program runs at a time. Each application has a `PilotMain` function that is equivalent to `main` in C programs. To launch an application, Palm OS calls `PilotMain` and sends it a launch code. The launch code may specify that the application is to become active and display its user interface (called a "normal launch"), or it may specify that the application should simply perform a small task and exit without displaying its user interface. The sole purpose of the `PilotMain` function is to receive launch codes and respond to them. Future versions of the Palm OS may allow third-party applications to be multi-threaded.
- Applications can send launch codes to each other, so an application might be launched from another application or it might be launched from the system. An application can use a launch code to request that another application perform an action or modify its data.

3.2 Hardware

All Palm OS devices, including those by Handspring, Sony, IBM, Kyocera, QUALCOMM,

Franklin Covey, TRG and Symbol Technologies, currently use the Motorola DragonBall MC68328-family of microprocessors which are based on the Motorola MC68EC000 core[2]. The DragonBall processors are inherently low-speed, ranging from 16MHz to 33MHz depending on the type (MC68328, 'EZ328, or 'VZ328). ARM Limited's microprocessor architecture, employed in many consumer, wireless, and security products, will be used as the core of future DragonBall processors [2] and is planned to be implemented in Palm OS devices in 2002.

Palm OS and other handheld embedded devices use battery-backed Random Access Memory (RAM) to store application and user data. The operating system and other non-transient components are often stored in Read-Only Memory (ROM). However, newer devices are moving towards Flash memory for static components such as the operating system. Flash memory is non-volatile and the data stored in it will remain intact even with loss of battery power or a hard reset. The Palm OS is restarted from its ROM or Flash storage area upon system reset.

4 Retrieval of Passwords

It is possible, via a number of methods, to extract data from portable devices by reading raw memory or from the host system after such data has been backed up. These attacks can retrieve files containing potentially valuable data such as passwords, financial, medical, or other company or personal information. In officially sanctioned scans, the authors found that the passwords chosen by users to protect data on their PDAs were the same as those being used for critical corporate assets.

One example of a high-security application is medical data, which is increasingly being stored on portable devices by doctors in order to have immediate access to patient information. Recent situations have occurred in which hospital intruders have beamed extensive amounts of unprotected patient data off of Palm OS devices. This could have been avoided with the proper use of passwords, encryption, and access-control on the device.

[2]Motorola's *MC68328 DragonBall Integrated Processor User's Manual* describes the programming, capabilities, and operation of the MC68328; the *M68000 Microprocessor User's Manual* provides instruction details for the 'EC000 core.

History has shown the weaknesses of poorly chosen or stored passwords, as in [17] and with the Morris Worm [19]. Users of portable devices, especially those that have no keyboard and require character input with a pen, oftentimes choose short, easily guessable passwords, placing convenience over security. Leveraging this, the scenario presents itself where malicious code determines the user's password on the local device and, upon connection to a network or other system, attempts to gain access to other systems using the user name and now-known password. This type of attack ends up being disconcertingly successful.

As it happens, an encoded block is stored on the Palm OS device in the Unsaved Preferences database that contains a reversible obfuscation of the user's system password [15]. The block is not only readable by any application on the actual device, but is also transmitted over the serial cable, airwaves, and networks during a HotSync operation. This problem is verified to concern Palm OS versions 3.5.2 and earlier.

4.1 Password Decoding Details

The password is set by the legitimate user with the Security application. The maximum length of the ASCII password is 31 characters. Regardless of the length of the ASCII password, the resultant encoded block is always 32 bytes. Two methods are used to encode the ASCII password, depending on its length. For passwords of four characters or fewer, an index is calculated based on the length of the password and the string is XORed against a 32-byte constant block. For passwords of more than four characters, the string is padded to 32 bytes and run through four rounds of a function that XORs against a 64-byte constant block. By understanding the encoding schema, it is possible to essentially run the routines in reverse to decode the password.

The Palm desktop software makes use of the Serial Link Protocol (SLP) to transfer information between itself and the Palm device. Each SLP packet consists of a packet header, client data of variable size, and a packet footer [20]. During the HotSync negotiation process, one particular SLP packet's client data consists of a structure which contains the encoded password block (Figure 2).

```
struct {
  UInt8 header[4];
  UInt8 exec_buf[6];
  Int32 userID;
  Int32 viewerID;
  Int32 lastSyncPC;
  time_t successfulSyncDate;
  time_t lastSyncDate;
  UInt8 userLen;
  UInt8 passwordLen;
  UInt8 username[userLen+1];
  UInt8 password[passwordLen+1];
};
```

Figure 2: Structure sent during the HotSync process containing encoded password block

Passwords of 4 Characters or Less: By comparing the encoded password blocks of various short passwords (example in Figure 3), it was determined that a 32-byte constant (Figure 4) was simply being XORed against the ASCII password block.

A = ASCII password
B = 32-byte constant block
C = encoded password block

The starting index, j, into the constant block where the XOR operation should begin is calculated by:

```
j = (A[0] + strlen(A)) % 32;
```

The encoded password block is then created:

```
for (i = 0; i < 32; ++i, ++j)
{
   // wrap around to beginning
   if (j == 32) j = 0;

   C[i] = A[i] XOR B[j];
}
```

```
56 8C D2 3E 99 4B 0F 88 09 02 13 45 07 04 13 44
0C 08 13 5A 32 15 13 5D D2 17 EA D3 B5 DF 55 63
```

Figure 3: Encoded password block of ASCII password 'test'

Passwords Greater Than 4 Characters: The encoding scheme for long length passwords (up to 31 characters in length) is more complicated than for short length passwords, although it, too, is reversible.

```
09 02 13 45 07 04 13 44 0C 08 13 5A 32 15 13 5D
D2 17 EA D3 B5 DF 55 63 22 E9 A1 4A 99 4B 0F 88
```

Figure 4: 32-byte constant block for use with passwords of length 4 characters or less

A = ASCII password
B = 64-byte constant block
C = encoded password block

First, A is padded to 32 bytes in the following fashion:

```
j = strlen(A);

while (j < 32)
{
   for (i = j; i < j * 2; ++i)
      // increment each ASCII value by j
      A[i] = A[i - j] + j;

   j = j * 2;
}
```

The resultant 32-byte array, A, is then passed through four rounds of a function which XORs against a 64-byte constant (Figure 5). k is an index that begins at {2,16,24,8} for each of the four rounds.

```
j = (A[k] + A[k+1]) & 0x3F; // 6 LSB
shift = (A[k+2] + A[k+3]) & 0x07; // 3 LSB

for (i = 0; i < 32; ++i, ++j, ++k)
{
   // wrap around to beginning
   if (j == 64) j = 0;
   if (k == 32) k = 0;

   temp = B[j]; // xy
   temp <<= 8;
   temp |= B[j]; // xyxy

   temp >>= shift;

   C[k] XOR= (unsigned char) temp;
}
```

The resultant 32-byte encoded password block (example in Figure 6) does not have any immediately visible remnants of the constant block as the short length encoding method does. However, it is still reversible with minimal computing resources.

```
B1 56 35 1A 9C 98 80 84 37 A7 3D 61 7F 2E E8 76
2A F2 A5 84 07 C7 EC 27 6F 7D 04 CD 52 1E CD 5B
B3 29 76 66 D9 5E 4B CA 63 72 6F D2 FD 25 E6 7B
C5 66 B3 D3 45 9A AF DA 29 86 22 6E B8 03 62 BC
```

Figure 5: 64-byte constant block for use with passwords greater than 4 characters

```
18 0A 43 3A 17 7D A3 CA D7 9D 75 D2 D3 C8 A5 CF
F1 71 07 03 5A 52 4B B9 70 2D B2 D1 DF A5 54 07
```

Figure 6: Encoded password block of ASCII password 'testa'

4.2 Recommendations

Palm OS 4.0, due to be released at the end of 2001, appears to have resolved the issue of weak password obfuscation. However, it is highly recommended that a thorough analysis of OS 4.0 takes place before a security-critical application is deployed.

In the current state, it is recommended that Palm OS devices should not be trusted to store any critical or confidential information. In lieu of this, users and vendors are encouraged to adhere to the following guidelines for increased password security:

- **Engage a challenge/response mechanism.** These mechanisms will minimize the potential for adversaries to glean passwords through passive monitoring of the transport medium. The transfer of a secret component, even if it is encoded or obfuscated, over accessible buses (e.g., serial, IR, wireless, or network) is a risky design decision. Unfortunately, it's common practice that applications choose to simply obfuscate passwords instead of using encryption.
- **Encrypt and salt credentials stored on systems.** Simple obfuscation and reversible transforms lull the user into a false sense of security and simultaneously show a lack of concern about security from the vendor. The use of a salt, such as the Palm user name, user ID, or unique serial number of the Palm device, minimizes the possibilities of a password being represented on multiple systems with the same hash.
- **Implement policy to lock and encrypt data on the device.** The Palm OS Security application provides "system lockout" functionality in which the Palm device will not be operational until the correct password is entered. This is meant to prevent an unauthorized user from reading data or running applications on the device. Although this protection can be bypassed as discussed in §5, it provides an additional layer of security for particular deployments. The encryption of data can be achieved with a number of third-party applications, though care should be taken to verify secure storage of the encryption components.
- **Implement an alternative password scheme.** Third-party solutions exist which provide power-on and data protection by requiring a handwritten signature, physical button taps, or other form of password before allowing access to the device. Ths use of graphical passwords on PDAs is studied in [12].

5 Backdoor Debug Modes

Designed into the Palm OS is an RS232-based "Palm Debugger", which provides source- and assembly-level debugging of Palm OS executables and the administration of databases existing on the physical device [21].

Entering a short keystroke combination [21], the Palm OS device enters one of two interfaces provided by the Palm Debugger and monitors the serial port for communication. "Console mode" interacts with a high-level debugger and is used mostly for the manipulation of databases. "Debug mode" is typically used for assembly- and register-level debugging. A soft-reset of the Palm device will exit debug mode, leaving no proof of prior use.

The Palm Debugger can be activated even if the Palm OS lockout functionality is enabled (which is currently assumed by most users to be a sufficient

protection feature, because a password is required before the device becomes operational). This problem is verified to concern Palm OS versions 3.5.2 and earlier.

Aside from the specific attack of retrieving the obfuscated system password block by using `export 0 "Unsaved Preferences"` and decoding as detailed in §4.1, it is possible to access all database and record information on the entire Palm OS device [16]. For example, using the `import` console command, one can load a Palm OS application into the device, therefore side-stepping any HotSync or beaming operations and logging mechanisms. A complete listing of console and debug commands can be found in [21].

Because the debug modes communicate with the host via the serial port, it would be possible to create a Palm OS-based application to emulate the required commands and, with a modified HotSync cable, be used for the retrieval of passwords or other data in a mobile fashion. When the possibility exists to retrieve data from a portable device while "in the field" and not requiring the use of a desktop computer, the threat of physical attacks increases greatly.

5.1 Recommendations

Solutions for this class of attack can be remedied with minimal changes to the Palm OS. If the device has been placed in the system lockout mode, the Palm Debugger functionality should be disabled. Palm OS 4.0 appears to have removed the activation of debug functionality during the "system lockout" mode. In an ideal situation, although a disadvantage to application developers, all debugging functionality should be removed in production devices.

Additionally, logging all Palm Debugger actions, especially with time stamping, aims towards forensics readiness and will aid in post-attack analysis.

If access control features are implemented in future Palm OS versions, as they should be, it should be noted that the permissions remain intact during debug sessions and that global memory accessibility is not allowed.

6 Infection Techniques

Common to most virus applications, and intrinsic to worms, is the notion of self-replication. Through self-replication and propagation, the malignant code can infect programs, devices, users, or combinations thereof. Hence, it is important to look at avenues available to such programs to better understand the risks at hand and determine areas to analyze for solutions.

Generic applications can be loaded in a number of different fashions. They can even execute without user knowledge or interaction. Any method of loading data onto the Palm OS device can act as an entry point for virus or malicious code infection. Four major entry points for the Palm OS devices are: HotSync operations, serial ports, infrared beaming, and wireless radio. Additionally, applications can be loaded using the Palm Debugger as described in §5.

Possibly more threatening and intriguing is the potential for cross-architecture pollination and infection. As with biology, the life cycle of a pathogen may involve more than one species of host. A virus could easily be designed to infect a desktop PC and contain a secondary payload for the Palm OS device. Alternatively, a virus on a Palm OS device could contain a payload aimed to compromise a desktop PC.

6.1 Application Installation Procedure

The current installation procedure for loading third-party applications onto a Palm OS device is simplistic in nature and was not designed with security in mind. The Install Tool, provided with the Palm Desktop software, copies the desired application into the `/Palm/<user>/Install` directory on the desktop PC. Upon the next HotSync operation, the contents in this directory are automatically loaded onto the Palm OS device. This is one example of cross-architecture pollination as the virus effectively transfers itself to the new platform.

No confirmation or authentication mechanisms exist during the HotSync operation. This shows the integrity and security of the host PC as an integral component in this chain of actions. If the host PC is compromised, the PDA can be considered com-

promised, as well.

6.1.1 Recommendations

Since the user places each individual program in the directory or otherwise intentionally labels the applications to be uploaded, user verification at synchronization to confirm the applications should be a trivial solution. This could be achieved by automated prompting on the host PC or by manually inspecting the contents of the `/Palm/<user>/Install` directory. However, many users have a learned behavior to simply accept system prompts without careful examination.

Cryptographic signing of applications by the vendor then verified by the user or Palm device will also reduce the chances of illegitimate code being loaded or executed on the device.

6.2 Desktop Conduits

"Conduits", in the form of Dynamic Link Libraries (DLLs), interface with the HotSync Manager program on the desktop PC. They enable the transfer of data between the Palm OS device and a specific desktop application during the HotSync process.

The standard conduits for Palm OS transfer Address, Date Book, Memo Pad, and To Do List data to the Palm Desktop software. Palm Expense data interfaces directly with Microsoft Excel. Third-party conduits exist which replace the standard conduits and will route data to Microsoft Outlook or Exchange, Lotus Notes, Novell GroupWise, or other Personal Information Manager (PIM).

Conduits are an extremely likely entry point for the cross-architecture transfer of malicious code. Aside from virus infection (such as a macro virus through the use of Microsoft Word or Excel macro functionality), malicious code transferred from the Palm device to the desktop through a conduit could exploit a known security problem in the destination desktop application. This could lead to compromise of the desktop machine (such as the execution of arbitrary code, theft or erasure of data, or elevation of privilege).

6.2.1 Recommendations

Cross-architecture infection risks exist for any portable device that employs data transfer or synchronization capabilities to other devices. Proper security practices should exist in the desktop environment consisting of, but not limited to, disabling macros, scripting, and the unprompted execution of code. Anti-virus software running on the desktop should scan the incoming data before passing it to the destination application. Once the malicious code has successfully been transferred to the destination application, it poses the same threats as if a user executed such a file directly.

6.3 Creator ID Replacement

Applications running on the Palm OS make use of a 4-byte Creator ID for identification purposes. If the Creator ID of a malicious application is defined to be the same as one of the built-in applications, it will be executed in place of the built-in application. Launching a Trojan program in this manner will appear transparent to the user until it is too late and the malicious action has occurred. Creator IDs of the basic built-in applications are listed in Table 1.

This behavior has characteristics of a list created in a Last In First Out (LIFO) fashion. Upon addition of a new piece of software to the system, its Creator ID is pushed onto the list. When a program is launched, a traversal of the list occurs to find the entry point to the program. When the first match on the Creator ID is found, the list traversal exits.

Application Name	Creator ID
Address	`addr`
Calculator	`calc`
Date Book	`date`
Expense	`exps`
HotSync	`sync`
Mail	`mail`
Memo Pad	`memo`
Preferences	`pref`
Security	`secr`
To Do List	`todo`

Table 1: Creator IDs of the basic Palm OS built-in applications

6.3.1 Recommendations

Vendors can prevent this problem by monitoring the Creator IDs at the operating system layer and disallowing duplicates. Furthermore, a complete traversal of the list could take place upon each application launch and if duplicate Creator IDs are found, neither application is executed and user intervention would be required. While this opens a window for denial-of-service-style attacks, it closes an obvious Trojan horse attack which is potentially much more damaging.

6.4 Wireless Communications

6.4.1 Infrared

For point-to-point, close quarters communications, infrared is typically the model of choice. In a standard IR beaming session, the Palm OS will send a `sysAppLaunchCmdExgAskUser` launch code to the receiving application. Typically, applications do not have custom handlers for this launch code, in which case the default response is to present the user with a dialog box prompting for acceptance or rejection of the request. If, however, the application handles the launch code, as detailed in §8.1, and sets the `result` flag to `exgAskOk`, the application will send a `sysAppLaunchCmdExgReceiveData` launch code and always receive the incoming data without displaying a dialog box or requiring user intervention.

Using the Exchange Manager functionality in this manner, it is trivial to transmit and receive applications and data over the infrared communications channel. With collusion on the receiving end, as would be possible with an infected system, IR functionality creates a viable conduit for propagation of virus and other malicious applications.

The scenario of beaming business cards at conventions comes quickly to mind as a potential hostile environment that previously might not have been considered as such. Consider a scenario where an adversary, posing as a conference attendee, beams malicious code or other payload, in the form of a business card object, to another individual. The malicious code could then spread from this individual to trusted parties during seemingly innocuous business card transfers.

6.4.2 RF

While infrared beaming is workable in close quarters, other mechanisms must be engaged for wide distance communications. The wireless technology space, particularly RF, has become a primary driver for portable devices. Internet and e-mail connectivity can be obtained through numerous providers, including Novatel Wireless, SkyTel, and Sprint PCS. Wireless Application Protocol (WAP)-capable PDAs and phones are becoming commonplace. Symbol Technologies' family of Palm OS devices integrate a Spectrum24 wireless local-area-network module for enterprise connectivity. The Palm VII employs a radio modem to communicate with the "Palm.Net" service on the Bell South Wireless Data network.

6.4.3 Recommendations

As with any other ingress or egress point on PDAs, wireless technologies create a new vector for possible infection through such means as application transfer or the transmission of intentionally faulty data packets. The design of properly secured wireless networks is beyond the scope of this paper, but it should be noted that if the portable devices are not sufficiently protected, they become a weak link in the transaction process. Consideration should particularly be placed on the storage of secret components (e.g., encryption keys), user authentication, and data transfer mechanisms.

Care should be taken when running server applications on a portable device, particularly when using RF technology (which has a wide operating range). These applications allow other devices to connect inbound to the server device thereby increasing the potential for malicious code to be transferred or for other malicious action (e.g., theft of data) to take place.

Global system functionality that would always prompt for user input and display the applications requested for data reception or transmission would diminish wireless infection. The addition of logging mechanisms for post-mortem analysis would also assist. As these are two suggestions that require vendor intervention, it behooves the user of the device to be cognizant of their surroundings and assess the threat before accepting beamed information from unknown people.

7 Storage and Payload Hiding

A key trait of virus code is the ability to remain invisible to casual scrutiny. This is often accomplished by storing program contents in non-standard areas. While the various methods of encrypting or otherwise obfuscating the payload of a virus program to avoid detection from anti-virus software is beyond the scope of this paper, areas in which code may be attached or stored in Palm OS devices is addressed.

7.1 Preferences and Databases

In the Palm OS API, Preferences and Data Manager functions offer several avenues for data storage. System and application preferences are accessible via the `Pref{Get,Set}Preferences` and `Pref{Get,Set}AppPreferences` function calls. Similarly, any system or application database can be attached to and used to store malicious content. `DmOpenDatabase`, `DmWrite`, `DmResizeRecord`, and `DmSetDatabaseInfo` are all common database manipulation functions that, due to the lack of protection and ownership of individual records, become conduits for attachment.

Unused fields in records are commonly used as covert channels. Databases on the Palm OS device are no exception. For example, the Application and Sort Info Blocks are optional fields in each database that can be used to store application-specific information. Common data stored in this block includes category names or database version numbers. However, it is not necessary for this field to be populated and often times it is not. Traversing the existing database records on the device and checking the `appInfoID` or `sortInfoID` parameter for a `null` pointer will yield a location for the attacker to store the handle (pointer to a location) of their payload. This would not affect the legitimate application's usage in any way.

7.2 Flash Memory

Palm OS devices incorporating non-volatile Flash memory currently use it solely for the storage of the operating system code. Depending on the family of Palm OS device, there remains between 440kB and 824kB of unused memory space.

Utilities exist, such as [27], which make use of the unused memory areas to backup applications and databases. These utilities are OS- and device-specific and use functionality outside of the Palm OS API. This is a perfect example of payload storage and is identical to how a malicious application would utilize Flash memory for such a purpose.

Data could also be stored on the Flash memory outside of the address space that is used by Palm OS, but within the valid memory map as specified in the DragonBall `Group-Base Address` registers. In doing so, applications running on Palm OS using only API functions will not be able to access nor see the data stored in this region.

Recommendations to minimize the risks of improper Flash memory usage are discussed in §9.3.1.

8 Execution Triggers

Viruses do not always execute immediately after infecting a target device. There is often an "incubation period" in which the virus sits dormant, waiting for a specific time, key sequence, or other preordained initiator. The inclusion of an incubation period increases the difficulty of determining exactly how or when the system was infected. As more system activity takes place over time, the ability to backtrack to the point of infection becomes difficult if not impossible.

8.1 Launch Codes

Particular launch codes sent by Palm OS are received by all applications on the Palm device. This becomes a prime candidate for incubation or virus execution, since code segments defined in handling routines are executed without the user's knowledge or intervention. Full details of the launch codes can be found in [22]. A casual perusal of the documentation for launch codes uncovers several obvious events that will likely be used for incubation of malicious code. Our speculations on these are listed in Table 2.

Launch codes are handled in `switch`-style constructs within the `PilotMain` function. An application checks each code that it receives to determine if a handler exists. If one does exist, execution is

Launch Code	Potential Incubation Method
`sysAppLaunchCmdSystemReset`	This launch code signifies that a system reset has just occurred. No user input is allowed during this launch code. As Palm OS devices are not reset at regular intervals, this provides a random timing for the launch of malicious code.
`sysAppLaunchCmdSyncNotify`	When a HotSync operation has been completed or an application has been successfully beamed and received by the device, this launch code is sent to application. This could signify that the malicious code has successfully propagated to the target device and can perform its payload hiding or destructive actions.
`sysAppLaunchCmdAlarmTriggered`	A most probable launch code for malicious use. Malicious code could set an alarm for a future time. Upon receipt of the alarm, the desired code would be executed.

Table 2: Selected application launch codes and theorized incubation methods

handed off to the appropriate functions. The launch code of `sysAppLaunchCmdNormalLaunch`, sent when an application is normally executed, would most often vector to legitimate code. This provides an appearance of normalcy while malicious payloads remain dormant until their specific launch code is seen.

8.1.1 Application Transfer

Through the use of launch codes sent by the Palm OS during the loading of an application (via the HotSync process or IR beaming), it is possible to have an application self-execute after it has been transferred to the target device. Using an infection technique such as described in §6.1, it would be trivial for malicious code to be loaded and executed on a Palm device with the legitimate user having no knowledge of the event.

A typical sequence to execute an application by transfer is as follows:

The newly transferred application will first receive a `sysAppLaunchCmdSyncNotify` launch code from the OS to specify that the device has successfully received the application. If the handling of this launch code sets an alarm for an immediate or future time, the application will be started again with a `sysAppLaunchCmdAlarmTriggered` launch code when that time is reached. The `AppLaunchWithCommand` API function can be called with a `sysAppLaunchCmdNormalLaunch` launch code in order for the application to begin normal execution.

8.1.2 Recommendations

While it is difficult to determine if programs being introduced to the system are malicious in nature, it is possible to sweep existing applications to determine if new launch code handlers have been inserted since the application's original introduction. The modification of an existing program to execute new code at launch would be endemic of viral activity and noticeable through these scans.

8.2 Trap Patching

Well-known to the virus writing community is the notion of "trap patching". When a system function is called, the operating system performs a look-up on the trap dispatch table to determine where in memory the desired function is located. In patching a system function, this address is replaced in the table with an address pointing to new code. Oftentimes, the new code will hand execution off to the original routine after it has served its purpose. In such a scenario, the patch appears invisible to the end user, as the original functionality still succeeds.

Trap patching has many uses beyond that of virus design. For Palm OS devices, trap patching has been made popular with HackMaster [13]. Any native functions in the Palm OS are potential vectors that can be trapped and exploited. This is not only the case for exported user programming interfaces, but includes those that are defined for system-use only.

To help in understanding trap patching as a vulnerability, consider a trivial denial-of-service event:

When a `penUpEvent` event is detected in the writing area, `SysHandleEvent` hands control over to the `GrfProcessStroke` API function. `GrfProcessStroke` is located in the trap dispatch table and the Program Counter starts execution at the address returned. If the `GrfProcessStroke` routine were replaced with a stub that returned immediately after entry, which is to say that the routine does nothing, the attack would result in characters being prevented from entering into the key queue.

Obviously, this constitutes a much more benign attack than ones that might be introduced with greater functionality.

8.2.1 Recommendations

Solutions for this class of problem have been historically difficult [7, 25]. Rollback, in particular, makes the tracking of potentially legitimate patching problematic. For example, take a natural scenario as shown in Figure 7.

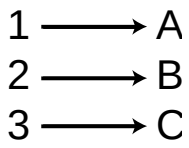

Figure 7: Functions {1,2,3} with corresponding Addresses {A,B,C}

Assuming that the structure in the trap dispatch table for Function 1 is modified to point to a new Address, D (Figure 8), it would be up to the program that introduced the modification to keep track of the original value.

If yet another patching program is introduced, it would note the native location of Function 1 as Address D. In this case, the second program has no way of knowing that it did not store the original address of Function 1. Upon the first program returning Function 1 to Address A, the second program can still rollback, pushing the return location back to that of Address D.

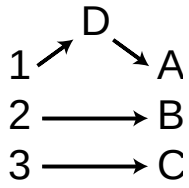

Figure 8: Function 1 patched to point to Address D. Address D hands off to the original location, Address A, upon completion.

Potential exists for periodic checks against vendor-published hash tables to avoid the rollback scenario. It is envisioned that vendors would publish and cryptographically sign a list of the entry points to the various functions. Checks could be made on the portable devices themselves. The Palm OS could also create a list of entry points of newly installed applications and, upon execution, check the stored values against the live values noting discrepancies. A message box or other user alert would be shown should the necessity arise. A cryptographic coprocessor, such as [8, 26], could assist in the secure storage of these entry points.

9 Malicious Actions

9.1 Application Deletion

Without memory protection, it is trivial to create applications capable of deleting program code or database information. The `Palm.Liberty.A` Trojan horse, detected in August 2000 and claimed to be the first known Trojan for the Palm OS platform, did just this in erasing all databases on the device. With complete and unrestricted memory access, the malicious application simply iterates through the linked list of databases and unlinks each one as it proceeds.

9.1.1 Recommendations

There are several preventive approaches for this type of attack. Trapping operating system calls at the API level has been employed in certain scenarios [18]. The calls are often patched to alert the user of a particular action or to disallow an action alto-

Register(s)	Potential Effects
`Phase-Locked Loop (PLL) Control` `Power Control`	System can be halted.

`Group-Base Address`
`Group-Base Address Mask`
`Chip-Select`
	Corrupted memory maps making code and data fetches impossible.

`LCD Controller Module`	Affect LCD functionality. It may be possible to cause LCD hardware damage by modifying the refresh frequency or by improper power cycling.

Table 3: Selected registers and theorized effects of improper modification

gether. Placing the onus of allowing or disallowing certain functions on the user can be problematic as, more often than not, the user is not security-conscious and will improperly configure, circumvent, or completely ignore the protection mechanisms due to their complexity. Security processes need to be in place at the operating system level that are undetectable and inescapable.

While this technique of trapping operating system calls has enjoyed some amount of success, it has the drawback that applications legitimately creating and erasing their own databases are often hindered. One remedy to this situation is to have the operating system enforce rules that only allow modification to databases with the same Creator ID as the application performing the actions. In this case, the Creator ID would need to be non-modifiable by the user.

9.2 Register Manipulation

While attacks using the Palm OS API are a major threat, lack of compartmentalization in the operating system allows the user to target the underlying hardware controlling the device. The DragonBall allows direct control of its registers via memory-mapping. Direct control of these registers allows an attacker to control many low-level aspects of device operation. An application simply has to define a pointer to the specific memory location representing the target register.

By examining the DragonBall registers, we have determined particular registers that, when improperly modified, can lead to disruptive events or physical damage to the Palm OS device. Our theorized effects are listed in Table 3. It should be noted that while these examples focus on the DragonBall processor, other embedded microprocessors exhibit similar vulnerabilities. These attacks are comparable to the desktop computer environment in which malicious programs would change the synchronization rate of a monitor or over-drive and manipulate hard drive heads.

9.2.1 Recommendations

Direct register access is not detected by existing anti-virus software. Current software in this field only watches for improper usage of the Palm OS API function calls (such as the `DmEraseDatabase` function).

Discerning a legitimate application from a malicious application is challenging when direct register access is involved. One solution is to prevent any third-party application from direct register access. While this would hinder legacy applications that did not adhere to the published API, the minor loss in backwards compatibility would most likely be deemed acceptable for the increase in security.

9.3 Memory Corruption

Devices using Flash memory supporting field-upgradeable operating systems have inflection points that ROM-based devices do not. Malicious code is capable of taking advantage of the field-upgradeable capabilities of the Flash device to modify or destroy data. Through this, they can patch the operating system with custom code or completely overwrite it. [9, 10] provides details of performing operating system upgrades in the Flash memory of Palm OS devices.

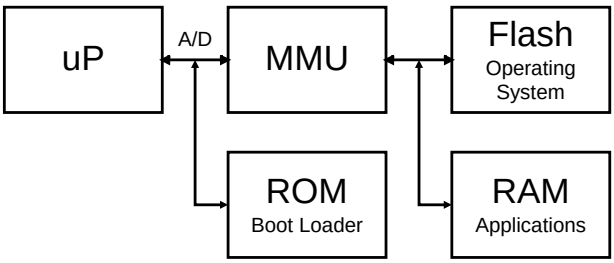

Figure 9: Possible design configuration for a secure PDA

Successful attacks on Flash can be crippling for the Palm OS device. The critical boot loader functionality for controlling field-upgrades is often stored in Flash. If this area is not properly protected using the Software Protection and Boot-Block locking features provided by the Flash memory device, it can be altered. Complete erasure of the boot loader prevents field-reprogramming of the operating system and will require the device to be returned to the factory for replacement. Any data not stored in protected areas of Flash memory is subject to erasure or modification, often without detection.

9.3.1 Recommendations

Current implementations of Palm OS devices do not use any Flash memory for application data storage and is used solely to store the operating system itself. All applications and data reside on battery-backed RAM. Therefore, a trivial solution for security-critical deployments would be to use devices that store the OS in ROM (such as the PalmPilot family) or guarantee that the entire Flash device is read-only. A similar scenario (Figure 9) would be to use a ROM device for all boot loading and Flash memory upgrade routines, still leaving the actual operating system in Flash. This would allow the critical routines to be protected and still allow the OS to be upgraded. It is apparent that the current PDA model places convenience of OS upgrades of greater importance than security.

A disadvantage to using Flash memory for the storage of applications and other often-modified data is the low amount of write-cycles (typically ≈10,000) guaranteed during the memory's lifetime. Given that RAM has no such limitation, it is still a natural choice for this type of data storage.

The Boot-Block areas of Flash memory could be used to implement a secure boot process similar to [1], which will guarantee the integrity of the system.

Implementing a hardware-based memory management unit (MMU) will aid in supplying memory isolation and preventing applications from unauthorized access to external memory. The MMU, commonly designed into embedded microprocessors, is not available in the DragonBall core. For purposes of Palm OS devices, this unit could be implemented in an application-specific IC (ASIC) or programmable logic device. It is hoped that an MMU is designed into the ARM core for future DragonBall processors. The MMU is located on the address and data buses between the microprocessor and the external memory. If the address requested for read/write access is outside of a legal, pre-defined range, the MMU can either prevent the operation outright or respond back to the processor in some manner.

It should be noted that solely implementing an MMU is not enough for proper memory protection. If the Palm OS is modified by an adversary, it may still be possible to access "restricted" areas of Flash. Using [1] in conjunction with an MMU implementation will work nicely, as there is integrity to guarantee that the operating system and underlying components are trusted and there is hardware-based memory protection for fault isolation. Figure 9 is one possible design configuration. The ROM and the MMU could be internal to the CPU, depending on its type. The MMU will monitor the address and data buses as described previously. The entire configuration could be designed as an ASIC or as a secure cryptographic coprocessor, along with the proper tamper-response and physical protection systems as recommended in [4, 6].

Another solution to the problem of accessible Flash

memory and risks of intentional corruption would be to introduce hardware jumper protection. This would physically allow or prevent writing to the Flash device. In order to accomplish this, a user would typically have to place a jumper or depress a button to enable or disable writing to Flash memory areas. Such a jumper could be connected to the Chip Enable, Write Enable, or Output Enable line of the memory device. Alternatively, it could enable circuitry that would connect the required address lines between the processor and memory device. When enabling field-upgradeable functionality, some modicum of due diligence must be taken to ensure integrity and authorization for such actions. Even if the hardware jumper was only active for the regions storing the base operating system, this would increase the security of the system. If applications are stored in Flash in future devices, the same scenario would exist and the user would have to physically "approve" each application as it is loaded into their device. This, however, is tedious for the user and could easily be bypassed with simple modifications to the hardware.

Secure coprocessors, such as [8, 26], enable secure distributed applications by providing safe havens where an application program can execute, free of observation and interference by an adversary with direct physical access to the device [26]. Designing such a configuration into the underlying Palm OS hardware will greatly enhance the security of the device and may minimize enough risk to be a suitable platform for security-based applications. It is possible that smartcards can serve as interim cryptographic coprocessors for portable devices [28]. Additionally, [3] proposes a software-based solution of using PDAs as cryptographic tokens.

Currently, Palm OS devices are extremely vulnerable to Flash memory attacks and have no protection mechanisms as described in this section. This is quite possibly the case for other PDAs and portable devices, as well.

10 Conclusions

In this paper, we analyzed the design of the Palm OS and hardware platform with respect to data storage issues, improper security design, and malicious code threats. Vulnerable and at-risk areas were identified that could be taken advantage of for such attacks. It has been pointed out that a variety of problems exist that can be exploited at both the operating system and hardware levels. Specific changes to Palm OS and its associated hardware were recommended and would be required to begin to properly implement preventive measures.

For solutions, it becomes apparent that implementing layer-based access control may be necessary to allow the application level to communicate only with the operating system. Conjunctively, these access control mechanisms would allow the operating system only to communicate with the hardware. The current design of the Palm OS software and hardware is not laid out in this fashion. As a result, many of the attacks discussed in this paper remain extremely difficult to defend against with third-party software running at the application layer. If future versions of Palm OS allow third-party applications to run as multi-threaded, anti-virus applications could essentially run in the "background" and use monitoring techniques as proven useful in desktop environments. Additionally, it may be possible to emulate a virtual machine that provides integrity and memory protection. Virtual memory areas of RAM used during cryptographic operations can be encrypted similar to [24] to protect temporarily stored plaintext.

The cryptographic code signing of applications has been used in many ActiveX scripts and Java applets for a number of years. Portable devices should employ such methods to verify the integrity of trusted applications. Ideally, the code signing routines and resultant signatures would be stored in ROM along with the Certificate Authority (CA) public key of the product vendor. It may be possible to store signatures in Secure Digital (SD) external memory cards (which are planned to be designed into Palm OS devices in late 2001) or Handspring's Springboard modules.

In lieu of any operating system upgrades or hardware re-designs, there are a number of simple and immediate precautionary measures a user can exercise to reduce the risk of data theft or malicious attacks:

- Be aware of what applications are being loaded onto the portable device. If an application comes from an untrusted source, extra care must be taken. This may entail using an existing anti-virus package on the PC to scan the

file for known threats or testing the application functionality on a spare device.

- Monitor the HotSync Log and Last HotSync Operation date to verify that there were no unauthorized HotSync operations performed.
- Disable the "Beam Receive" functionality in the System Preferences panel. Enable this feature only when necessary. This prohibits anyone from beaming information to the Palm OS device.
- Be aware of the physical location of your Palm device at all times. Attaching a belt clip or lanyard will reduce loss, misplacement, or theft.

Because Palm OS devices account for the majority of the PDA market, it is hoped that the research in this paper is used to create a more secure computing environment in the short term. It is also hoped that the analyses and ideas provided in this paper will be used in future work to design more secure products.

In the current state, caution should be taken when employing portable devices for security purposes. In a War College-style approach, it is believed by the authors that oftentimes the simple knowledge of a vulnerable area is enough to help steer the user towards more security-conscious use.

Acknowledgments

The authors would like to thank @stake's Research Labs, especially Brian Carrier, for constructive criticism and interesting discussions.

References

[1] W. Arbaugh, D. Farber, and J. Smith, "A Secure and Reliable Bootstrap Architecture," *IEEE Security and Privacy Conference*, May 1997.

[2] ARM, Ltd., "Motorola's DragonBall Processor Portfolio to Include ARM Architecture in 2001," Press Release, December 11, 2000.

[3] D. Balfanz and E. Felten, "Hand-Held Computers Can Be Better Smart Cards," *8th USENIX Security Symposium*, Washington, D.C., August 1999.

[4] D. Chaum, "Design Concepts for Tamper Responding Systems," *Advances in Cryptology: Proceedings of Crypto '83*, 1984.

[5] E. Chien, "Malicious Threats to PDAs & Prototype Solutions," *Virus Bulletin Conference 2000*, September 2000.

[6] A.J. Clark, "Physical Protection of Cryptographic Devices," *Advances in Cryptology: EUROCRYPT '87*, 1988.

[7] I. Goldberg, D. Wagner, R. Thomas, and E. A. Brewer, "A Secure Environment for Untrusted Helper Applications," *6th USENIX Security Symposium*, San Jose, California, July 1996.

[8] P. Gutmann, "An Open-Source Cryptographic Coprocessor," *9th USENIX Security Symposium*, Denver, Colorado, August 2000.

[9] T. Harbaum, "Flashlib," April 1999, `http://bodotill.suburbia.com.au/flashy/flashy.html`.

[10] T. Harbaum, "OS Flash," September 2000, `http://bodotill.suburbia.com.au/osflash/osflash.html`.

[11] IDC, "Market Mayhem: The Smart Handheld Devices Market Forecast and Analysis, 1999-2004," Report 22430, June, 2000.

[12] I. Jermyn, A. Mayer, F. Monrose, M. Reiter, A. Rubin, "The Design and Analysis of Graphical Passwords," *8th USENIX Security Symposium*, Washington, D.C., August 1999.

[13] E. Keyes, "Hacking the Pilot: Bypassing the Palm OS," *PDA Developers 4.6*, November 1996.

[14] Kingpin, "Palm OS Beam Bit Modification Tool," January 1999, `http://www.atstake.com/research/tools/beamcrack.zip`.

[15] Kingpin, "Palm OS Password Retrieval and Decoding," *@stake Security Advisory*, September 26, 2000, `http://www.atstake.com/research/advisories/2000/a092600-1.txt`.

[16] Kingpin, "Palm OS Password Lockout Bypass," *@stake Security Advisory*, March 1, 2001, `http://www.atstake.com/research/advisories/2001/a030101-1.txt`.

[17] D. Klein, "Foiling the cracker: A survey of, and improvements to, password security," *2nd USENIX Security Workshop*, August 1990.

[18] McAfee.com, "Increased Protection for Wireless Users in Wake of Recent PDA Trojan Discovery," Press Release, September 5, 2001.

[19] United States General Accounting Office, Report to the Chairman, Subcommittee on Telecommunications and Finance, Committee on Energy and Commerce – House of Representatives, "Virus Highlights Need for Improved Internet Management," *GAO/IMTEC-89-57*, June 1989.

[20] Palm, Inc., *Palm OS Programmer's Companion*, DN 3004-003.

[21] Palm, Inc., *Palm OS Programming Development Tools Guide*, DN 3011-002.

[22] Palm, Inc., *Palm OS SDK Reference*, DN 3003-003.

[23] W. T. Polk and L. E. Bassham, "A Guide to the Selection of Anti-Virus Tools and Techniques," National Institute of Standards and Technology Computer Security Division, *SP 800-5*, December 1995.

[24] N. Provos, "Encrypting Virtual Memory," *9th USENIX Security Symposium*, Denver, Colorado, August 2000.

[25] B. Schneier, "The Trojan Horse Race," *Communications of the ACM*, Volume 42, Number 9, September 1999.

[26] S.W. Smith and S.H. Weingart, "Building a High-Performance, Programmable Secure Coprocessor," *Computer Networks (Special Issue on Computer Network Security)*, 31: 831-860, April 1999.

[27] TRG Products, Inc., "FlashPro," `http://www.trgnet.com/cat-flashpro.htm`

[28] University of Michigan, "Smart Card Research At CITI," `http://www.citi.umich.edu/projects/smartcard`.