Assessment of Palm OS Susceptibility to Malicious Code Threats

Tom Goovaerts, Bart De Win, Bart De Decker, and Wouter Joosen

DistriNet Research Group, Katholieke Universiteit Leuven,
Celestijnenlaan 200A, 3001 Leuven, Belgium
{tomg, bartd, bart, wouter}@cs.kuleuven.ac.be

Abstract. The Palm OS operating system for Personal Digital Assistants (PDAs) and mobile phones has a weak security architecture, which introduces all sorts of security problems. This paper specifically targets the problem of malicious code. The main contribution of this work is the in-depth analysis of different vulnerabilities in Palm OS and the ways in which they can be exploited by malicious code. Furthermore, the key reasons for this problem are discussed and some suggestions for improvement are formulated.

Keywords: malicious code, worms, mobile operating systems, Palm OS.

1 Introduction

PDAs have become very popular devices that appear in day to day life. They have evolved from simple electronic agendas to powerful connected mini-computers that use wireless communication technologies such as IrDA, Bluetooth, WiFi and GSM. These devices can hold sensitive information such as confidential documents and passwords or they can participate in commercial transactions. However, it has already been documented that the operating systems found on PDAs do not pay much attention to security [1,2].

One important class of security problems is malicious code such as viruses, Trojan horses, worms or backdoors. Except for the weak security architecture of PDA operating systems, there are two reasons why these devices are at risk of being targeted by malicious code. First of all, PDA applications are easily downloaded from the Internet and exchanged between devices. This encourages passive infection strategies, and it leads to a situation that is somewhat comparable to the old PC era where viruses were propagated on floppies. Secondly, new wireless communication technologies introduce new threats for actively propagating forms of malicious code. Imagine a worm that wirelessly propagates from PDA to PDA in rush hour traffic or in a full auditorium. When PDAs are being used for critical tasks such as mobile payment or as advanced security tokens, the malicious code problem is a risk that is completely unacceptable [3,4]. So far, a few PDA viruses have effectively been reported [5,6,7].

In this paper, the security problems of Palm OS with respect to malicious code are discussed. We have carried out an in-depth vulnerability assessment of

J. Dittmann, S. Katzenbeisser, and A. Uhl (Eds.): CMS 2005, LNCS 3677, pp. 240–249, 2005.

Palm OS and discuss how vulnerabilities in the operating system can be exploited by malicious code. All of these vulnerabilities have been tested in practice in a prototypical worm. Some suggestions are proposed to tackle the problems and the security architecture of the future version of the operating system is evaluated shortly. The implementation and verification of the vulnerabilities has been carried out in a master's thesis [8].

The rest of this paper is structured as follows. Section 2 gives a short introduction to Palm OS. Section 3 discusses the vulnerabilities and explains how malicious code can exploit them. In Section 4, the proof of concept implementation is discussed that was developed as a test bed for the vulnerabilities. Section 5 reflects on the security problems of Palm OS, proposes some suggestions and gives a short overview of the improvements in the future version of Palm OS. Finally, Section 6 concludes the paper.

2 Palm OS

Currently, the Palm OS operating system comes in two flavors: Garnet [9] and Cobalt [10]. Palm OS Garnet (version 5) is the version that is found on new Palm OS powered devices today. In the future, Palm OS Garnet will become the version for less powerful devices such as cell phones, while Palm OS Cobalt (version 6) will target PDAs. This paper focuses on Palm OS Garnet.

Palm OS is a single tasking operating system for ARM-based processors running on speeds from 100MHz to 400MHz. Palm OS based devices do not have nonvolatile memory. Memory is used for execution as well as storage and is partitioned in two logical regions: the *dynamic memory* is used for dynamic allocations and maintaining the execution state of programs and the *storage memory* stores data that has to be preserved. Data as well as applications are kept in structures called *records* and records are grouped in *databases*. Because data is stored in RAM, a device has to maintain a minimal voltage over the memory at all times. Resetting the device will only erase the dynamic memory, but it is possible to erase all memory by doing a *hard reset*.

Palm OS supports multiple wireless communication technologies including IrDA, Bluetooth, WiFi and GSM allowing users to 'beam' data or applications from one device to another. Palm OS implements a TCP/IP stack for networking wirelessly or by modem and newer Palm OS devices support mobile phone technology to make voice calls or send text messages.

Synchronization with the desktop is realized by *Hotsync*. This is an application that consists of a device component and a desktop component that lets the user synchronize his/her calendar and contacts between the applications on the device and those on the desktop. Hotsync is also used to install new software and to take backups.

A number of security features is supported in Palm OS. It is possible to *lock* the device, which will disable it until the user authenticates him/herself with a password.[1] Locking can be activated manually or automatically, e.g. when the

[1] Since a PDA is supposed to be a single user device, a username is not required.

device is turned off. Furthermore, some protection mechanisms for databases are provided. For example, database records can be made 'secret', meaning that the user must enter his/her password to be able to read or write the information in the record. Palm OS also has a cryptographic library that contains implementations of a limited number of algorithms for encryption, secure hashing and signing.

3 Vulnerabilities

We have assembled and verified a set of new and known vulnerabilities for Palm OS Garnet. Approximately half of the vulnerabilities comes from the work of Kingpin and Mudge [1]. The discussion of the vulnerabilities is structured according to the vulnerable operating system aspects: database management, event mechanisms, application and process management, communications, built-in security mechanisms, desktop components and the GUI. For each vulnerability, the problem is first explained and then the possible exploitations by malicious code are discussed.

3.1 Database Management

Full access to data [1]. Palm OS has no access control mechanism for databases. Direct memory access into the storage heap is disallowed, but when the database API is used, every application has full access to all databases stored on the device. It is possible to read and write information into database records, append or remove database records or create and remove entire databases. Even records that have been marked as secret (see Section 2) or databases that have been marked read-only can be deleted without restrictions.

Since Palm OS devices can contain sensitive information such as business cards, important documents and passwords, the lack of access control towards data poses an important threat. Malicious code can execute numerous malicious actions such as erasing, sending or modifying important data, storing information in existing databases or erasing log entries.

Program Infection or Destruction [1]. The fact that every application has full write access to the code of other applications opens the platform for a wide range of virus infection techniques.

A malicious program that is running on the device can replicate itself by writing its own code into another application by means of the database API. Or it could also simply destroy other applications by writing data in their code. We have experimented with program infection by means of a simple proof-of-concept infection program that replaces the first occurrence of the `RET` instruction of every user-installed program with the machine code of an infinite loop. This causes the execution of these applications to block the system.

Preventing the Deletion of Databases. Databases can be protected by means of the `DmDatabaseProtect` operation. Internally, Palm OS keeps a per-database

counter with the number of times the database has been protected. Calling the protection function increases this counter by one. A database with a positive protection counter cannot be deleted.

The protection mechanism is easily circumvented because the protection function can also decrease the counter until the database can be deleted again. Beyond this obvious threat, the database protection mechanism can also be abused by malicious programs to protect themselves against removal. A malicious program can protect its own database and detect[2] when someone tries to remove it. In response to the removal attempt, the program could reprotect itself. This way, the user is unable to delete the database, even with applications such as Filez [11] that give the user much more low-level control over databases and their attributes than the default database management functionality of Palm OS.

3.2 Event Mechanisms

Palm OS applications are event-based and the operating system provides a number of event sources and event handling mechanisms:

1. *Launch codes* are sent by the operating system when certain events take place that are of interest to all applications, for example the completion of a Hotsync synchronization.
2. There is a publish-subscribe mechanism called *Notifications* that allows applications to register themselves with a notification service for a certain event and receive notification when the event takes place. An example of a notification is the launching of a particular application.
3. Through the *Alarm Manager*, applications can set a number of alarms. At a given time the application will get a warning for the alarm and can execute code in response to it.

All these event mechanisms allow malicious code to trigger their malicious actions with a high precision [1]. When a malicious program is installed on the device, it will start receiving launch codes. Once it has received its first launch code, it can register itself for notifications and set alarms, and become active on numerous occasions.

Hiding of Noticeable Actions. It is possible that malicious activity cannot be completely hidden for the user. Therefore a malicious program could carefully wait for a moment at which the user is not using or watching his PDA for a while. To choose such a moment, the following notifications could be used: the `sysNotifyIdleTimeEvent` notification is sent whenever the device is inactive for a short period, the `sysNotifySleepNotifyEvent` notification is sent whenever the device is put into standby mode.

User Interface Tracking. By registering for the `sysNotifyEventDequeuedEvent` notification, the application gets a notification whenever a user interface event

[2] By means of a Notification, see Section 3.2.

(such as the tapping of a button) is handled. This way, malicious programs can track tapped buttons and the information he/she enters. The security consequences are twofold: first, it allows for a fine grained selection of triggers. For example, a self-propagating program can track when the user presses the 'beam' button to send information to another device and react by sending itself instead, hoping that the receiving user will accept the transfer. Second, malicious programs can intercept confidential information entered by the user.

3.3 Application and Process Management

Hiding from the Application List. The database of each application has a bit that, when set, will hide the application from the list of launchable applications. When doing so, the application will still be listed in the dialogs for copying, moving and deleting applications.[3] Malicious applications could set their hidden bit for trivial hiding purposes.

Hiding Entirely From the User Interface. Every database has a type and an identification number, called a *creator id*. Databases that contain application code have the type `appl`. When the type of an executing applications' database is changed into a non-executable type (for example the `data` type or a custom type), it will no longer be seen as an executable but it will still receive launch codes and notifications as long as the device is not reset. Doing so will hide the program from the launch screen. Moreover, when the creator id of the executing application is also changed to another applications' creator id, the application will be regarded as a database of the other application and will also be hidden from the copy, move and delete dialogs. The executing code can only be deleted by removing the application whose creator id was taken. When the creator id of a built-in application is taken, the executing code cannot be deleted at all because the delete dialog only allows the removal of user-installed applications.

Thus, a malicious program can hide itself completely from the user interface and render itself unremovable while it can remain executing in the background. Third party database managers such as Filez will still allow the removal of the database.

Application Replacement. Besides a type and a creator id, every application on Palm OS has a name and a version number. Two vulnerabilities in the handling of these attributes exist that can be exploited to replace existing applications:

1. Installing an application that has the same creator id but a higher version number than a built-in application will replace the built-in application by the new one[1].
2. When an application is installed with the same name as an application installed by the user, the old application will be physically removed from the device and is completely overwritten by the new application.

[3] A user can visually notice installed applications either in the launch screen or in the copy, move and delete dialogs.

Consequently, malicious code can replace all applications that are installed on the device. The replacement of an application poses an obvious threat of making existing applications unusable, but these vulnerabilities can also be exploited to exhibit Trojan horse behavior. The functionality and appearance of existing applications can be mimicked for secretly executing malicious actions.

Occupying the Processor. Since Palm OS is a single tasking operating system without any process management capabilities, a malicious program can perform a denial of service attack by simply executing and not releasing the processor. An infinite loop is enough to block the entire system and force the user to reset.

3.4 Communications

Backdoor for Beamed Data [1]. The Exchange Manager is a library that allows data to be sent to another device using a high level transport-independent API. Data sent by the exchange manager typically travels over wireless protocols such as IrDA, Bluetooth or GSM protocols. An application can be registered to handle the reception of data of a certain type, for example contacts, meetings or applications. When data of the application type is received, by default a dialog will be shown asking the user for confirmation.

Every application can register itself to receive all beamed applications and can override the confirmation dialog, always accepting the received code. This way, a malicious application can open a backdoor that allows other self-replicating malicious programs to transfer themselves silently onto the system.

Detection of Devices in the Proximity. Low-level communication libraries such as the IrDA or Bluetooth library pose less of an infection threat than the Exchange Manager mainly because no servers are running for these protocols. They can however be used by malicious code to detect other devices that are in the proximity. For example, the IrDA library can be used to poll for other devices periodically and trigger a propagation with the Exchange Manager when a potential victim has been detected.

Detection of Communication Facilities. Self-propagating malicious code uses communication channels and network connections to copy itself onto other devices. PDAs typically only have short-lived connections, for example when data is sent over IrDA or Bluetooth. Sometimes the communication hardware is not always present. A number of notifications exist that are useful propagation triggers: the `sysExternalConnectorAttachEvent` is sent whenever an external device such as a WiFi adapter or a Hotsync cradle is attached and the `sysNotifyLibIFMediaEvent` is sent whenever a network interface is activated.

3.5 Built-in Security Mechanisms

Brute Force Password Guessing. Since entering a long password on a PDA can be an annoyance to the user, passwords are often chosen to be very short. The

authentication mechanism of Palm OS is implemented by simply comparing the MD5 hash of the user-provided password and the original password stored in a (freely accessible) system database. Since passwords on a PDA are often significantly shorter than normal passwords, the MD5 hash of the password is vulnerable for a brute force attack.

Experiments on a device with a 144MHz processor have shown that all alphanumerical passwords of length 3 can be found within 15 seconds.

Password Removal. As mentioned earlier, a record can be protected by setting its secret bit. This requires the user to enter his/her password upon viewing or changing the information in the record. When a user has forgotten the password, he/she can remove the password. This deletes all records that have their secret bit on. However, there is an API call (`PwdRemove`) that removes the password without any further consequences.

This API call effectively undermines almost all security mechanisms of Palm OS. By removing the password, a malicious program can open the device and the information carried by it to persons that can physically approach the device.

3.6 Desktop Components

Surviving a Hard Reset. Every kind of malicious code has to reside in RAM, so when the user does a hard reset it would be erased. By setting the 'backup bit' of a database, it will be copied to the desktop upon synchronization. Malicious code can transfer its own database to the PC in order to survive a hard reset. After a hard reset, the next synchronization will restore all backed up databases.

Infection via Hotsync [1]. The Hotsync component on the desktop is the gateway to the PDA for all applications. By placing a copy of an application in a certain directory on the desktop and calling a function on a Hotsync library, the Hotsync installation program will install the file on the device.

Cross-platform malicious code that carries a Palm OS payload can exploit this weakness for infecting the device.

3.7 GUI

Hiding Dialogs. Two API calls exist for turning off the screen and for stopping its redrawing. Malicious code can abuse these calls for hiding noticeable actions from the user.

A possible technique is to first detect when the user wants to put the device into standby mode and then turning off the screen, but leaving the device on. The user cannot make a visual distinction between its device in this state or in standby mode. When the screen is turned off, a malicious program can operate without being detected by the user.

When malicious code wants to perform certain actions such as beaming itself to another device, some dialogs appear on the screen. A technique to hide these dialogs is to freeze the screen just before the dialog appears and release it when the dialog is gone, given that this latter moment can be determined.

4 Proof of Concept Implementation

To verify and experiment with these threats in practice, we have created a generic proof of concept implementation of one specific kind of malicious code: a worm. We have chosen a worm because it is a good vehicle for testing the full spectrum of discovered vulnerabilities and techniques. Malicious code typically combines a number of exploits. Therefore, the main purpose is not only to allow easy experimentation with isolated vulnerabilities but also with combinations thereof. The implementation is written in C and can flexibly combine exploits at compile-time. It consists of the skeleton of a worm and is structured along three phases: the *activation* phase hides and protects the worm and activates the other two phases, the *propagation* phase actively propagates the worm to other systems and the *execution* phase contains the payload. Table 1 shows the phase(s) to which each of the discussed exploits belongs.

Table 1. Vulnerabilities and the phase(s) to which they belong: Activation, Propagation or Execution

	A	P	E		A	P	E
Full access to data			•	Backdoor for beamed data		•	
Program infection or destruction		•	•	Detection of devices in the proximity	•		
Preventing the deletion of databases	•			Detection of communication facilities	•		
Hiding of noticeable actions	•			Brute force password retrieval			•
User interface tracking	•		•	Password removal			•
Hiding from the application list	•			Surviving a hard reset	•		
Hiding entirely from user interface	•			Infection via hotsync		•	
Application replacement			•	Hiding dialogs	•		
Occupying the processor			•				

We have found that it is very easy to use combinations of these exploits to create a proof of concept worm that is quasi invisible *and* virtually impossible to delete. We have used our implementation to create a self-propagating security utility with a benign payload. This utility uses a number of exploits discussed above to hide and protect itself. In the mean time, its benign payload tries to determine the user's password, warning him/her when it is not strong enough. As can be seen in Table 1, currently the vulnerabilities for the activation and execution phases are the main threats to Palm OS. The simplicity and relatively small size (less than 2000 lines of code) of our proof of concept implementation confirms that Palm OS is an easy target for malicious code.

5 Discussion

We have seen that Palm OS is vulnerable for numerous malicious code threats. The reason for this lies within Palm OS's security model: it is a secure operating system under the assumption that all applications can be fully trusted.

This assumption is reflected in two ways. First of all, the responsibility for enforcing various security rules is pushed up towards the application layer. A trusted benign application will behave well and will implement the enforcement logic, but an application with malicious intentions is completely free to ignore this enforcement request. Examples of security rules to be enforced by the applications are the read-only bit for a database and the secret bit for a record. Secondly, as illustrated extensively in Section 3, simple use of Palm OS's API can seriously harm the system and can be a large threat. Trusted applications will not misuse the API, but malicious applications are only limited by the creativity of their writers in the malicious actions they can perform. The fact that in the Palm OS world, new applications are frequently installed only worsens the untrusted code problem.

To tackle these problems, a number of changes to Palm OS could be implemented. First of all, memory protection should be introduced. Without it, no other security mechanism could be implemented securely. Furthermore, all existing security enforcement (e.g. for read-only databases and secret records) should be pulled down to the operating system. A number of new security mechanisms could also be added to Palm OS. The introduction of a code authentication mechanism would eliminate a great deal of the discussed malicious code threats. Unauthenticated code could be disallowed or be executed with limited permissions. In addition, an access control mechanism could be implemented that can limit access at least to databases but preferably also to API calls.

As can be seen in Table 1, most of the discovered vulnerabilities either belong to the activation phase or to the execution phase. Propagation phase techniques that actively propagate code to other devices are currently much less of a threat to Palm OS than they are to classical desktop operating systems. This can be explained by the fact that, except for the built-in exchange manager, Palm OS devices do not run server programs. Furthermore, Palm OS devices normally only have short-lived connections. Finally, Palm OS devices are less prone to buffer overflow attacks than most desktop operating systems because the address of the central application stack can not be determined easily. Actively propagating forms of malcode are most likely to propagate on the desktop platform and install their payload through Hotsync.

Palm OS Cobalt [10] introduces a number of interesting security-related changes. Most importantly, code can be signed to guarantee its integrity. Furthermore, memory protection is implemented, so no more direct memory writes can be made. Another novelty is the introduction of secure databases. These contain encrypted information and are only accessible through a configurable authorization manager that controls which operations may be executed on the database. Finally, programs that do not respond can be terminated by the user. Beside these positive evolutions, not all issues are solved. For example, the enforcement of database attributes (e.g. the read-only bit) is still not done by the operating system and secure databases are sent over the wire in plaintext when performing a synchronization. Unfortunately, code authentication is also not mandatory. Cobalt is a step in the good direction, but is certainly not perfect.

6 Conclusion

We have studied the vulnerabilities of Palm OS and have given potential malicious code threats that result from them. These threats and vulnerabilities were tested and verified in a proof of concept implementation. It was found that the platform is extremely vulnerable to malicious code. The reason is that Palm OS fully trusts all code. Although Palm OS devices are more and more equipped with wireless technologies, currently the main threat comes from passively propagating forms of malicious code. Suggested improvements are memory protection, system-level security enforcement, access control and code authentication.

Further study has to show the impact of Cobalt on the security of the platform in general and to the problem of malicious code in particular. Furthermore, a comparison will be made of this problem and the solutions on other operating systems for both PDAs and desktops.

References

1. Kingpin, Mudge: Security analysis of the palm operating system and its weaknesses against malicious code threats. In: Proceedings of the 10th USENIX Security Symposium, USENIX (2001) 135–152
2. Murmann, T., Rossnagel, H.: How secure are current mobile operating systems? In: Proceedings of the Eighth IFIP TC-6 TC-11 Conference on Communications and Multimedia Security, IFIP (2004) 47–58
3. Ghosh, A.K., Swaminatha, T.M.: Software security and privacy risks in mobile e-commerce. Communications of the ACM **44** (2001) 51–57
4. Pfitzmann, A., Pfitzmann, B., Schunter, M., Waidner, M.: Trustworthy user devices. In: Multilateral Security in Communications, Addison-Wesley (1999) 137–156
5. Symantec Security Response: The WinCE.Duts.A virus for Windows CE. `http://securityresponse.symantec.com/avcenter/venc/data/wince.duts.a.html` (2004)
6. Symantec Security Response: The SymbOS.Cabir worm for Symbian. `http://securityresponse.symantec.com/avcenter/venc/data/epoc.cabir.html` (2004)
7. Symantec Security Response: The Palm.Phage.Dropper virus for Palm OS. `http://securityresponse.symantec.com/avcenter/venc/data/palm.phage.dropper.html` (2000)
8. Vanhoof, J., Goovaerts, T.: Studie van de wormproblematiek op het Palm OS platform (Dutch). Master's thesis, Katholieke Universiteit Leuven (2004)
9. PalmSource: Palm OS Garnet. `http://www.palmsource.com/palmos/garnet.html` (2004)
10. PalmSource: Palm OS Cobalt 6.1. `http://www.palmsource.com/palmos/cobalt.html` (2004)
11. Software, N.: Filez 6.7. `http://www.nosleeep.net` (2005)