TrustedPalm Patch for the PalmOS

By:
Jeff Doering
John Dunagan
Lila French

Project:
6.033 Design Project II
4/30/97
RI/TA: Karger/Fu

Abstract

We provide a patch to the PalmOS (the TrustedPalm patch) as well as an application for the management of security information that allows secure communication of data using both existing conduit technology and general sockets. Section I introduces our patch and the rationale behind it. The next two sections cover authorization. Section II introduces our security model – it requires familiarity with the Palm concepts of databases and records. Section III describes the implementation of authentication lists and permission checks. Section IV summarizes how other applications can leverage and build upon our design. Section V covers authentication and the networking issues involved. This section requires familiarity with the current implementation of conduits and networking on the PalmPilot, and it makes reference to some IETF drafts. In Section VI we delve into the user interface which allows the current generation of applications to benefit from our TrustedPalm patch. In Section VII we outline the behavior of our design in specific scenarios.

I. Introduction

As the 3Com PalmPilot has grown enormously in popularity, so have the needs of its user community. Critical to the success of the 3Com PalmPilot has been the ability to reconcile Pilot data with data on the user's desktop computer. Aside from direct user input, this technology has provided the principal means for interaction between the Pilot and the world.

The Palm III has recently been introduced with an IR port and PalmOS support for TCP/IP. As the ability to communicate with clients other than the desktop computer has grown, so has the need for authentication and authorization. Previously, security was very simple; the user either entered the data on the Palm directly, or checked the data's authenticity when downloading it to his desktop computer. In an increasingly networked world, the ability to communicate with clients other than a secure desktop computer will greatly expand the market for and the utility of the PalmPilot.

To realize this goal, our team has designed a patch for the PalmOS that we call the TrustedPalm patch. This patch incorporates support for authentication and authorization in such a manner that these new capabilities can be easily integrated into existing software. The TrustedPalm patch supports authenticated communication over all of its external interfaces. In addition, our patch supports both the establishment of a two-way channel and the receipt of signed messages - two different styles of communication with very different security parameters. The patch supports all the current functionality of the Conduit API. Principals may be granted different types of access at several levels of granularity. Lastly, the TrustedPalm patch comes with an application for easily managing both access permissions and authentication data.

Given that many applications will be security blind and that all current applications are security blind, one might question the value of this project. To this we respond that our solution demonstrates the utility of tackling this problem. The TrustedPalm patch provides such valuable functionality through a simple mechanism that we believe the user community will rapidly adopt it. Additionally, the TrustedPalm patch provides a base for the development secure applications and room for future growth.

II. Authorization Model

At the highest level, the design goal of authenticating and authorizing data access requests appears quite simple. Certain principals are allowed to perform certain operations. Likewise, the lowest levels of security involve straightforward concepts such as using public/private key pairs for authentication. Yet, the mapping of high level goals to low level methods is rarely simple. It is the abstract security model that bridges the gap between high level goals and low level methods.

The security model for the TrustedPalm patch defines several abstract entities. These include principals, groups, permissions, permission sets, and authorization lists. These entities aid in constructing a security model that allows authorization of requests while providing a manageable method for granting these authorizations.

The authentication and authorization provided by the TrustedPalm patch occur when the PalmPilot is in secure mode. In this mode the PalmPilot exposes the Secure Conduit API to its external interfaces. All Secure Conduit API requests are processed using the TrustedPalm security model. It is important to note that the TrustedPalm security model does not preclude insecure access to the PalmPilot. HotSyncing the PalmPilot in insecure mode is explicitly allowed. This might be done because the user wishes to use an older conduit that is somehow incompatible with the Secure Conduit API, or perhaps the user wishes to perform a full system backup without regard to security policy. It is left to the user to weigh the benefit of using the old access mode versus the hazard of insecure HotSyncing.

The TrustedPalm patch defines access permissions and identifies principles in order to provide an authorization mechanism for data requests. The use of a set of permissions allows assignment of various degrees of data access to different principles or groups of principals. A principal refers to a unique individual distinguished by a cryptographic signature. Groups are simply named sets of principles.

The TrustedPalm uses authorization lists to store permission sets for various resources. These resources are items in the PalmOS storage hierarchy such as records or databases. Each entry in an authorization list consists of a permission set paired with either a principal or group. A principle or group may appear only once in a particular authorization list. This simplifies permission enumeration at the authorization list level.

Clearly, a permission set is only meaningful in relation to its associated resource. Although the primary use of the TrustedPalm patch is authorization for record-level access requests, authorization lists can actually refer to several types of resources. While the simplest security model would only provide authorization lists at the record level, this would waste storage space and make specification of access policies far more cumbersome. Instead the TrustedPalm implements a security hierarchy.

The security hierarchy allows the authorization of information with varying degrees of granularity. The PalmPilot system is the root of the hierarchical tree. One level down are all of the Pilot's databases, and below each database are its header information and categories. Categories contain data records, while database headers contain header information. For security purposes header information and data records are treated in the same manner. Figure 1 shows a simple example of a security tree for records that are not flagged as secret. Secret records are described later. A given resource in the tree is influence by its immediate authorization list as well as the authorization lists of its ancestors. This means that the authorization information for a single resource might actually be spread across several authorization lists. For example, the authorization list for record A might specify permission sets for principal X and principal Y. However, if the parent database includes a permission set for Principal Z, complete authorization information for the record requires examining the authorization links of both the record and the parent database. For simplicity the TrustedPalm allows a given principle or group to have only one effective permission set for a particular resource. The requirement that a given principle or group appear only once in a given authorization list enforces this restriction when only one list applies to a resource. However, if multiple lists apply to a resource, a given principle might appear in several of the lists. This means that identifying permission sets requires more than just the union of all applicable authorization lists.

In order to identify exactly one effective permission set for a given principle or group, only the permission set stored in the "nearest" authorization list is used. In this case proximity refers to levels traversed in the hierarchical tree. Permissions applied to a specific record would have a distance of zero from that record, while default permissions for the whole PalmPilot would always have a distance of three from any record. Because the tree allows each resource to have only one parent no two resources can be the same distance from a common descendant. This method for choosing a nearest authorization list prevents any ambiguity in deciding which permission set to apply. By allowing low level authorization lists to completely override higher level lists, the security model allows for the addition or subtraction of permissions. This is accomplished by using higher levels in the security hierarchy for either more or less restrictive permission sets.

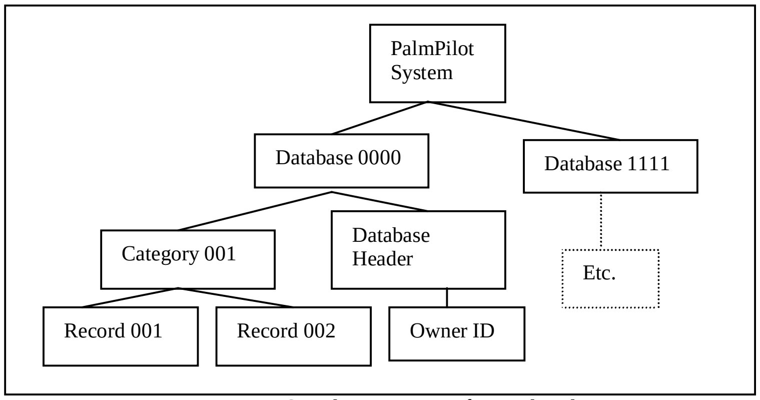

Figure 1. Sample security tree for a PalmPilot with two databases (information is abbreviated, does not include secret records).

The TrustedPalm considers secret records a special case. Because the secret attribute exists in all versions of the PalmOS, existing application are aware of this feature. In order to preserve the protection expected for secret records, the TrustedPalm disallows all access to these records.

The security tree provides the model for assigning and reading a particular principal's or group's access rights. However, a principal's ability to perform data access is not just a matter of the permission set for that principal. The principal might also be a member of one or more groups with access rights to the particular resource.

In this case access rights are additive. A principal's ability to perform a particular data access requires that the principal or any of the groups the principal belongs to has the appropriate access right. This means that an access request requires that the TrustedPalm examine both the permission set for the principal and that for every group that includes the principal. However, in practice the TrustedPalm simply examines authorization lists until it finds the necessary permission (e.g. permission to read). Because permissions are additive, there is no need to continue searching after a permission is found. If examination of all appropriate authorization lists fails to locate the needed permission the request is denied. In particular the absence of any authorization information results in failure, as the default is no access.

For example, if Alice wishes to read record two from database zero in the PalmPilot from figure 1; the system will search for a permission set for Alice starting at database zero, record two. Assuming that the record is not secret, the search progresses as follows until a permission set is found: first the authorization list for record two, next the list for category two, then in the list for database zero, and finally in the list for the whole system. If the search finds a permission set, it checks for the read permission. If read permission is allowed the security system allows Alice to read the record. On the other hand, if the read permission is not allowed the system repeats the search procedure for each of the groups which include Alice. If the search fails to find an applicable permission set that allows the necessary access it rejects the access request.

The security tree model above explains the application of permission sets. However, application of the model requires an understanding of the permission sets themselves. The TrustedPalm patch defines the following security permissions: read, write, add, and delete. The read, write, and delete permissions refer to record level operations. These permissions can be specified at any level in the security hierarchy. The add permission refers to creating new records. Thus, it can only be applied at the category level or higher in the security tree. Each permission applies to certain types of Secure Conduit API calls. For example enumerating records requires the read permission while creating categories requires write permission on certain database header fields.

III. Implementation

The TrustedPalm uses existing PalmOS features to implement its security model. This implementation minimizes the impact of operating system changes on legacy applications. Further, by using standard PalmOS features, existing tools can manage security system data. For example, existing backup conduits can operate on the security information without modification.

The description of the TrustedPalm security internals is provided to explain the current security implementation. However, applications should not directly access the security databases. Instead, Section IV describes the supported API for obtaining and setting security information. The TrustedPalm PalmOS itself uses this API when performing authentication, authorization, and security policy management. The TrustedPalm creates two PalmOS databases to store the security information needed for the TrustedPalm security model. The first database manages authorization lists and the information that links them to resources in the security hierarchy. The second database manages principal and group information. This information includes the private key for the PalmPilot owner, public keys for known principals, and group membership lists.

The first database is labeled *tpal* for TrustedPalm Authorization Lists. The use of all lowercase letters in the name indicates that it is a reserved PalmOS database. The *tpal* database defines the following categories for its records: record, category, database, database-header, and system. The categories correspond to various resource

Tpal Record ID	Category	Database ID	Record/Category Number	Authorization List Pointer
000	Record	0001	002	44FF
001	Database	0002	000	2237
0002	Column	0001	001	44FF
0003	System	0000	000	12A4
0004	Database-Header	0002	004	33EE

Figure 2. Sample Entries for *tpal* database. Note that the Authorization List Pointer refers to memory in the PalmPilot's address space.

types. The variable portion of each record entry points to a data structure that identifies the exact resource from the resource hierarchy and the associated authorization list. Resource identification is accomplished through several fixed fields. Although the fields have meaningful names, the actual purpose varies depending upon the category of the record. The first field is a four-byte database identifier. For all resources at or below the database level in the hierarchy, the database identifier identifies which database the resource is descended from. The only resource higher in the tree is the system. For security records categorized in this category, the database identifier is not used and defaults to zero. The next field uses three bytes to represent the record identifier, category identifier, or header identifier. In the case of records, the combination of the database field and this field uniquely identify the record. Likewise, database and category identifiers combine to uniquely identify any category. While the PalmOS already includes the concept of record and category identifiers, the use of identifiers for database header information is new. The information fields in database headers are simply numbered sequentially starting with one. Because the header format is the same for all databases, one numbering system suffices without attaching any additional information to the database headers themselves. Again the combination of this numbering system with database identifiers allows unique specification of any header information. This identifier is unused for system records. The final field for all security records is a four-byte pointer to the authorization list. By using a pointer, several security records can share the same authorization list in order to save space. Because applications might apply the same security permissions to several records, categories, etc. this sharing has the potential to substantially reduce memory usage. An authorization list is simply a series of one-byte permission sets each followed by a three-byte principle identifier. The permission byte uses the individual bits from lowest to highest to represent read, write, delete, add, and add pre-packaged. The three highest order bits are unused and always set to zero. The three-byte principle identifier specifies the appropriate principle or group by referring to a specific record in the principle table.

While the *tpal* database stores the information needed for authorization, a second database is needed to support authentication. This database is called *tpid*, for TrustedPalm Identifier. Three categories are used in *tpid*: private, public, and group. The private category exists to identify the *tpid* record that stores the PalmPilot owner's private key. The structure of this record is simple. It consists of the owner's name followed by the private key encrypted with the TrustedPalm password. This password is used for entering and exiting secure mode. Because the owner enters the password, it is not stored anywhere in the PalmPilot. This means that the PalmPilot can only access the private key while in trusted mode. This prevents a lost PalmPilot from compromising the owner's private key. If the PalmPilot is lost while in trusted mode, only the Secure Conduit API is available which will not allow any analysis of the PalmPilot memory and thus the private key. On the other hand, if the PalmPilot is lost while in unsecured mode, an attacker can examine the memory contents of the PalmPilot. However, in this case the private key is encrypted and the PalmPilot has no information to aid in decryption. While a lost PalmPilot might result in total compromise of the owner's data, a compromise of his private key is much worse. Private key compromise could result in forgery and future data interception. Thus, protecting the private key is a critical design choice.

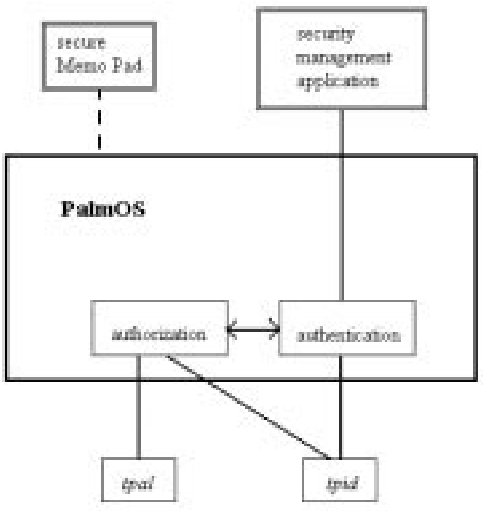

Figure 3. TrustedPalm System Components

The public and group categories support authentication. The public category includes all records that identify individual principals. These records consist of the principal's name, public key, and a list of the groups to which the principal belongs. This information is sufficient to perform authentication as well as authorization. The principal "Unknown" is reserved and has a Public key set to zero. The unknown principal is used for unrecognized or unauthenticated requests while the PalmPilot is in secure mode. The group category stores group information. Each record consists only of the group's name. While storing members in a list would speed up enumeration of list membership, it would also increase storage requirements. The typical TrustedPalm use of groups involves enumeration of a principal's memberships for the purpose of authorization checking. This is accomplished easily by examining the principal's record. On the other hand, only administrative tasks need to see all the members of a given group. This requires searching each of the principals in the *tpal* database. Although this is not an optimal solution, the storage savings justify the mechanism.

The implementation of the security model significantly influences the storage and performance costs of the security system. Storage of public keys in the *tpid* database requires significant space. Assuming one kilobyte keys, PalmPilot storage which is on the order of one-half to two megabytes significantly restricts the number of principals on pilot can recognize. While offline storage of key would eliminate this restriction, it would severely limit the utility of a mobile PalmPilot. The storage of authorization lists requires much less space. Each list has a fixed cost of seven bytes plus four bytes for each entry in the list. These costs do not account for standard database requirements common to all PalmOS databases. Because each entry in an authorization list requires extra space, assignment of groups rights can save memory over assignment of individual rights. Thus, use of groups is strongly encouraged.

All access to *tpid* and *tpal* will store records in sorted order. Existing PalmOS calls allow such sorting. This prevents O(n) lookup time on databases of length n. Without such sorting authorization lookups would be unnecessarily slow. Use of categories to identify resource types also improves performance. The PalmOS can already enumerate records of a particular category using database indexes. This implementation clearly addresses performance issues with maintaining simplicity.

IV. Internal API

Applications which wish to make decisions about security have a simple API to call upon. This API allows the implementation of an alternative security manager by a third party, as well as the implementation of different security schemes that reuse some part of our security model. As an example, another application could make use of our public key encryption routines, and our public key/principal pairings in some context other than a secure conduit, perhaps implementing a PGP-like email program. Similarly, another application could also make use of our access control lists, perhaps using these to decide what data is sensitive or not and what should be exported to a file which is going to be printed on an insecure printer.

The access control lists can easily be manipulated by any security-aware application (not just the security manager). This would allow some future security aware-version of the memo pad application to allow security modifications of memos without switching to the security manager application. This functionality is quite desirable because the application fully understands all of the data in its database. This makes it much more qualified than the PalmOS to make security decisions.

The authentication library is similarly straightforward. Our API supports registering and deleting principal public key mappings, establishing aliases to principals, and listing all the principals. It also supports all of the cryptographic primitives one might need: signing, certifying, encrypting, decrypting, and hashing messages, as well as a good pseudo-random number generator.

Secure conduits are built on top of these foundations. This enforces a desirable degree of modularity between levels in our system. If our current public key system were compromised tomorrow, we could simply substitute a new public key library and a new set of public keys to correct the system. This ability to grow is useful even in the event that no such disaster occurs. In the normal course of things, we can continually upgrade our libraries to support longer public keys and new versions of public key algorithms.

One of the strong elements of our design is that we provide a solid foundation for the development of other secure applications. Although we have only implemented secure conduits, our implementation of authenticated, secure sockets and public key/principal mappings through a documented API allows any future application writer to reuse our work. The developer of a banking program could, with a few function calls, establish a secure, authenticated socket to a host elsewhere on the network.

V. Secure Conduits

Our secure conduit design allows two levels of authentication. The first is interactive while the second is not. The interactive version, which we call Authenticated Sessions, allows the full power of the Conduit API, within the restrictions imposed by our authorization checks. The second version, which we call Authenticated Messages, makes a minimal subset of the Conduit API available to untrusted principals delivering trusted data. Put another way, the non-interactive mode is useful when you cannot talk to the trusted principal, but you can see that a particular message was from the trusted principal.

Traditionally, cryptography has been more concerned with the former scenario, where there is two-way communication with a trusted principal. This scenario is the simpler case. However, the second scenario corresponds to many real world situations. For example, an MBTA schedule transmitter in the local subway station may be entrusted with a signed piece of information, but not with the power to sign arbitrary pieces of information, or with access to the trusted principal. In this case, the PalmPilot cannot know anything from communicating with the MBTA's proxy except that this single piece of information is legitimate. Because this type of communication is inherently vulnerable to replay attacks, some of the conduit API should be unavailable to requests made through this type of communication. For example, we do not want to allow replay of delete requests. Our support for Authenticated Messages allows this inherently less secure (although useful) communication, within a secure framework.

When the PalmPilot tries to HotSync while in secure mode, the order in which protocols are attempted is simple and backward compatible. First, the protocol that both parties wish to use is announced. Both parties can then agree on the most advanced protocol each desires. For high security conduits, this will be the Authenticated Sessions protocol. For the simple delivery of a signed message, this will be the Authenticated Messages protocol. If neither is supported or neither is desired the old conduit protocol will be used, with the exception that all conduit calls will be checked by the TrustedPalm under the assumption that they originate from an unknown principal.

In each case, the TrustedPalm simply sees a traditional conduit call along with an associated external principal ID. The TrustedPalm then decides whether to honor this request or not based on the applicable permission settings. This uniformity of format is made possible by the use of a default unknown principal. Whenever a request originates from an unrecognized source or from a source that does not support authentication, the source is presented to the TrustedPalm as the unknown principal. The unknown principal will typically have the most restricted access rights.

The secure conduit design is quite modular. Public key cryptographic functions and library lookups occupy the lowest level. On top of these functions we construct the second level of our protocol, the calls which interact with the outside world. The second supports an interactive and a non-interactive protocol as follows.

Authenticated Sessions (Interactive)

Authenticated Sessions offer authentication, encryption, and a high level of security against active adversaries. Use of two-way authentication, encryption, one-time session keys, and sequence numbers limits computationally bounded adversaries to denial-of-service attacks. Both replay and man-in-the-middle attacks are prevented.

Encryption is required to support authorization of read requests. If we only wish to make information visible to a given user, then we do not want to broadcast the response to an authorized read request as clear-text. Additionally, encryption incurs almost no extra overhead. Because we two-way authenticate the selection of a shared session key, encryption with that key accomplishes authentication as well. Assuming we can encrypt as fast as we can sign, authentication with encryption costs no more than authentication alone.

To achieve our goal of a secure, reliable, authenticated communications channel, we have adopted a well-known standard, the Secure Socket Layer Protocol (see References section for information on SSL IETF draft). This protocol provides authentication, encryption, nonces, and other features needed in a secure communication protocol. Additionally, using a well-known, extensively studied protocol provides a strong assurance of security. The only interesting detail of our is the layer of indirection we impose between the external principal ID and public key used by the protocol versus that presented to our authorization layer. The SSL protocol may use a public key/principal pair that is unrecognized by our internal database of such pairs. In this case, the authorization layer will receive communications as though they came from the unknown principal even though the SSL protocol will continue to authenticate and encrypt communications with the client.

The external principal ID associated with this session is established at the beginning of the authenticated session. The authenticated session encapsulates the conduit concept of a session, making integration relatively simple. Every time a message is received from the outside and decrypted (and thus authenticated), the conduit API call is presented to the authorization layer with the same external principal ID established at connection time.

A different model is required for Authenticated messages. The chief complication in the implementation of authenticated messages is the need to present different external principal IDs to the security manager during the lifetime of the session.

Authenticated Messages (Non-Interactive)

Authenticated Messages offer a much lower degree of security; this necessitates restriction of the conduit API. Since conduits are designed around record level access, a signed message to change a particular record, without any other communication with the trusted principal makes no sense**.** A Conduit API call to change a record (a write call) requires knowledge of PalmPilot record identifiers. These identifiers can only be obtained through a read call. Thus a write call should always follow a read call. Since, by assumption, we are not engaged in two-way communication with the trusted principal, we only treat signed add-record calls as originating from the trusted principal. Although any Conduit API call may be submitted with a signature, only add-record calls are presented to the authorization layer with information regarding the trusted principal. All other calls, under this protocol, are presented as originating with the unknown principal.

Since this protocol is designed to give a much lower level of security, the implementation details are much simpler, and we can use a more lightweight structure than for the Authenticated Session protocol. The authenticated messages protocol allows the mixing of signed and unsigned messages (there is no point in signing a message that will be interpreted as coming from the unknown principal). Every request must come across the channel as either {message, externalPrincipalID, publicKey, signature} or {message}, where {message} is the traditional conduit data representation, {externalPrincipalID} refers to the principal who signed the message, {publicKey} refers to the public key of the external principal, and {signature} is the result of our public key signature algorithm applied to the {message, externalPrincipalID, publicKey} string.

If a request is of the form {message} it is presented to the authorization layer as originating from the unknown principal. If a request is of the form {message, externalPrincipalID, publicKey, signature}, then a more complex protocol is followed. First, the signature is checked with the given public key to ensure that it is authentic. Then the external principal ID is looked up in the *tpid* database. If the external principal ID is not found, then the message is presented as a request from the unknown principal. If the external principal ID is found, then we perform the additional check that the public key in the communication matches the internal record of the principal's public key. If the two public keys do not match, then the message is presented to the security manager as a request from the unknown principal. In the case that everything matches, we finally check the request type. If the request is of any type other than add record, then we present the request to the authorization layer as a request from the unknown principal. If the request is of the add-record type, then we present the request to the security manager with the given external principal ID.

Integration into PalmOS

The discussion up to this point has dealt exclusively with conduits as an abstraction which magically connects our PalmPilot to an outside source. At this point we will specify how our design integrates into the PalmOS and existing hardware.

The PalmOS currently supports two different external communication mechanisms. The first two generations of the PalmPilot only have a serial port through which to communicate to the outside world. The Conduit API has been implemented to communicate over this serial port. The most recent generation of the PalmPilot, the Palm III, also has an IR port. Communication through the IR port is supported through Berkeley Sockets, which are layered on top of TCP/IP and industry-standard IR protocols. Thus, there has been a bifurcation in communication protocols for the PalmPilot.

We will similarly support secure conduits over both the bare hardware represented by the serial port and through the socket mechanism, representing a network connection over TCP/IP. Sockets have now been implemented over both the serial port (for use when the serial port is connected to some network device, such as a modem) and over the IR port. Additionally, conduits have been implemented to use sockets layered on top of TCP/IP.

Since we support HotSyncing over a socket, and since multiple socket connections may be open at any given time, the initiation of HotSync allows a choice between opening a socket to a remote machine over the IR port, opening a socket to a remote machine over the serial port, or just talking to the machine which we are connected to over the serial port. Support for other devices, such as modems connected to the serial port, should be easy to implement.

When a secure HotSync is initiated over the IR port, the secure protocol is performed by reading and writing to a Berkeley Socket. A socket is the appropriate abstraction to interface with since anything below this layer has to deal with possible packet loss, and all the vagaries of transmitting over an IR port. The same is true for any network connection. The only time we do not want to layer something on top of TCP/IP is when we are communicating over the serial port to the conduit directly, in which case the serial connection will probably not exhibit any packet loss. After establishing a connection, the secure protocols take over. The protocol translates every communication into the appropriate message, and decodes communications back before passing them to the TrustedPalm.

VI. User Interface

In the current generation of the PalmOS, HotSync is initiated by pressing a physical button located on the “cradle” of the Palm Pilot, that is, the container used to hold the Pilot when connected to a host PC. The addition of our PalmOS patch requires the addition of virtual HotSync button on the screen of the Pilot. This will allow HotSyncing over the IR port, when the Pilot is not connected to the host PC.

After installing our PalmOS patch, pushing either button will prompt the user whether they want to HotSync securely or insecurely. (In compliance with the current HotSync technology, an insecure HotSync gives the conduit access to all the data on the PalmPilot. This provides valuable backward compatibility, as we want the user to be able to HotSync insecurely to old conduits if they so desire.) If the user is already in secure mode, the secure HotSync will be run by default, using the permissions specified in *tpal*.

During the installation of our PalmOS patch, we need to initialize the elements we have added. First, we need to set the private key/public key pair for the PalmPilot. The host PC is more than capable of generated numbers for this purpose. Immediately after the private key is selected, the user will be prompted to select a password. This is because the private key is encrypted with the user's password in storage. Next, we have to initialize the new databases, *tpal* and *tpid*. As *tpal* contains the authorization lists of the PalmPilot, this database will initially be empty. *tpid*, on the other hand, will contain the user's own private key (encrypted with the password) and 3Com's public key. Having 3Com's public key will allow 3Com to send the user authenticated messages containing the keys of trusted third parties. The new key information can then be directly added to the *tpid* database. This allows 3Com to prepackage signed keys for certificate authorities. Once a user has an authority's key, he can enable the authority by giving it add rights to the *tpid* database.

Besides obtaining keys from 3Com, the user can more simply enter a key manually after seeing it in a secure forum, such as 6.033 lecture. Using the default security management application, one can write down an 18 character string, representing an 1116 (=18x62) bit string. Thus, it is feasible to distribute some keys by hand.

The security management application will also be used when setting permissions for information in current, security-blind applications. Because these applications lack the interface to allow the user to specify permissions, record-level settings are not supported, since the security manager knows nothing about specific records. Database and category-level permissions, however, are supported, since this higher-level information is available to outside applications. New security-aware applications will incorporate permissions at whatever level of granularity is desirable for the particular application. So, for example, in a security-aware version of the schedule application, a user will be able to select the record containing "9AM- execute 'I love 6.033' hack" and modify its permissions accordingly within the application.

The PalmOS patch makes no changes to the current user interface, as the security manager is an independent application. However, new security-aware applications should facilitate the use of permissions by providing an interface for their modification.

VII. Scenarios

Our decision to allow this single type of authenticated communication in the context of a protocol that is inherently vulnerable to replay attacks was motivated by several design project requirements. As a first example, take the following case:

Later that day, a student who has a Palm III (which has an infrared port) is walking through Harvard Square and sees an Palm III-compatible infrared transmitter that claims to be transmitting MBTA bus schedules as PalmPilot memo items. The student would like to add this information to her PDA, but would like to verify that the information does indeed come from the MBTA rather than being a trick played by the local cab drivers (who want you to miss your bus and have to take a cab).

Our Authenticated Message protocol clearly allows this type of exchange. It authenticates and authorizes the interaction. However, it cannot prevent replay. It is acceptable for the student to receive multiple copies of this information. We are forced to presume here that the application is not dumb enough to crash if data is duplicated. In the case of the PalmPilot memo application, this assumption is certainly true; in fact, we are unaware of any PalmPilot applications that would crash if they received two copies of the same record. It is not too great a restriction to impose on future Palm application writers that they make their applications "multiple copy of identical data" safe.

As a second example take the scenario:

Consider a 6.033 student with a PalmPilot and a public PC in her dorm. The student connects from the PC to a web site that claims to have the latest 6.033 quiz schedules and locations encoded as a PalmPilot memo. The *student uses a conduit to add the memo to the PalmPilot. However, the student would like to be sure that the information does indeed come from the 6.033 staff.*

This case also uses the Authenticated Message protocol. Here, it is also fine if multiple memos are sent to the PalmPilot. If a malicious third party retransmitted an old quiz schedule, this would not replace the current quiz schedule. Instead, both memos would now reside on the PalmPilot. The student could manually check which one is newer, something the 6.033 staff would presumably have included in the memo itself. By leaving it up to the student to choose the newer information based on some clue within the memo itself (such as a date, or a handout number), we avoid any need to modify the PalmPilot database format to include timestamps for the purposes of data reconciliation.

Finally, we give an example where an intelligent conduit could make use of publicly readable information on the student's PalmPilot in its decision of what authenticated information to transmit. Consider the following scenario.

Then, the student uses the new 6.033 automatic recitation scheduling conduit on Athena. It talks to the student's PalmPilot via the serial port on the Athena workstation to read the student's schedule and automatically figures out what recitation to assign to the student. The student wants to make sure that the recitation schedule item (with time and room number) that the software adds to her PalmPilot comes from the registrar rather than being a spoofed recitation assignment. In addition, the student would also like to make sure that the software is not able to read the schedule item that indicates that the student is planning to do a hack by putting an "I love 6.033" sign on top of the great dome. (That extracurricular schedule item should only be accessible to the people who are helping her with the hack.) Furthermore, the student does not want the 6.033 scheduling software to overwrite the "Sleep" entries in her schedule with "Study for 6.033".

First, let us assume that the recitation scheduling conduit cannot communicate with the trusted 6.033 principal itself, and thus the recitation scheduling conduit must use the Authenticated Messages protocol. In this case, there is an obvious set of permission settings that will completely fulfill the requirements of the above scenario. If the user has set all the records that occur from nine am to five p.m. to be publicly readable (that is, readable by the unknown principal), and all the other items to be private, then the recitation scheduling conduit will be able to decide what recitation best fits the student's schedule without being able to read anything about the planned hack.

By only allowing items with the registrar's signature (or the 6.033 staff's signature) to be added, the student can be sure that any recitation assignment which is added is indeed legitimate. Although the recitation scheduling conduit could add a ``study for 6.033" entry at 4am in the morning, this would require the collusion of the 6.033 staff, and the student would notice that there are now two entries for 4am in the morning, ``study for 6.033" and ``sleep". It is reasonable to presume that the student can figure out what is going on here, and make the correct choice of what schedule item to follow.

This discussion presumes that the scheduler's internal representation for all entries is akin to a ``sparse matrix" representation. That is, instead of having an entry for every date and time, entries are added only as appointments are added. The latter is clearly the intelligent way to write this software on a client (like the PalmPilot) with very limited resources.

Next, let us consider the case that the recitation scheduling conduit just serves as an intermediary between the PalmPilot and a trusted principal, such as a secure server run by the registrar. In this case, the authenticated session protocol would be feasible, and the user could gain even more knowledge from the conduit. Not only could the conduit schedule a recitation for the student, but the conduit could also communicate what recitation was scheduled to the registrar, and additionally convince the student that the registrar (and hence the 6.033 staff) knew this. This situation is ideal. The situation that we can communicate directly with the registrar is mentioned as an example of the power possible when communicating directly with a trusted principal.

Another case, involving dealing with a merchant, has been brought up during design requirement discussions. It was suggested that in order to update the balance on a PalmPilot after a user had made a purchase, the merchant could communicate to the bank the fact that the user had made a purchase and get a signed note from the bank stating the user’s new balance. We suggest that this is a bad model. If the user really cares what the balance on her PalmPilot is, maybe because he is using it for some type of electronic commerce, then the user would feel much better if they talked directly to the bank, something the user is allowed to do under our model, and which the merchant could allow or prevent, but not corrupt using our model for authenticated sessions.

Our analysis of these cases supports our belief that the TrustedPalm patch will be of significant benefit to the PalmPilot community. With this patch, the users will be able to safely regulate access to their Pilots while maintaining the benefits available from operating in a networked environment.

VIII. Performance Analysis

While the TrustedPalm patch provides an excellent security framework, it does incur performance costs. However, analysis suggests that these costs will have little impact on the typical user. The slowest operation performed by the TrustedPalm is message signing and encryption using public/private key pairs. This operation occurs during the handshake stage of Authenticated Sessions and during all Authenticated Message transfers. However, the encryption library specifies speeds of 1 KB/ .1 sec. or equivalently 10 KB/sec. Although this is slower than the typical 56KB/sec. Supported by the PalmPilot's various serial interfaces, the limited storage capacity of the PalmPilot makes the very marginal. For example the most memory supported by a current PalmPilot is 2MB. This could be transferred in roughly three and half minutes using the slowest mode. Further, most transfers of large amounts of data will use Authenticated Sessions. While Authenticated Sessions use Public/Private keys during protocol negotiation, they use a shared private key the rest of the time. Encryption with a shared private key typically takes far less time. Thus typical Authenticated Session transfers could move data at rates almost as fast as the raw interfaces.

IX. Conclusion

A final analysis of the security provided by the TrustedPalm patch warrants comments regarding several issues. One of the biggest weaknesses of the security provided is the inability to prevent replay attacks on signed messages. Unfortunately, the one-way nature of such communications makes such prevention unrealistic. With one-way communication the only way to prevent replay would be to include a message identifier with each message. The system would then have to keep track of all of these identifiers forever. However, this solution is unacceptable for a device such as the PalmPilot with already limited memory. Another security weakness is the vulnerability to denial of service attacks. However, most security systems are vulnerable to such attacks. Further, denial of service against the PalmPilot is relatively benign. The most obvious attack would be to replay a message over and over in order to fill the PalmPilot's memory. Yet, the PalmPilot's owner could simply delete the new records to free the memory.

A few useful features were considered for our design but ultimately rejected. One option was to allow authenticated messages to be downloaded then stored for later verification. This type of functionality would help in the case where the PalmPilot does not have the principal's public key. The message would be held until the public key was obtained from a reliable source. Unfortunately, this type of option would add significant complexity to our system. Thus, we decided that the feature would not be incorporated into the TrustedPalm patch. The greatest omission from our security system is probably the lack of an auditing system. While our current design allows authorization of data access, it provides no record of who performed what actions. This lack of auditing means that users should be very careful in assigning permissions to public groups or questionable principals.

One particularly notable feature we chose to omit was some method for version stamping records. The design project specifically asks about reconciling old and new versions of a phone number. However, this type of reconciliation is not an operating system function in the PalmOS. The PalmOS is ignorant regarding the data it stores in application databases. It can read category information, header information, some basic status flags, and not much else. Therefore reconciliation of various versions of application data is an application level task. If the PalmPilot is using an Authenticated Session to talk to the server conduit, the server can manage data constistency on its own using application specific data structures. If the user trusts the conduit to change the databases, the user had better trust the conduit to manage versioning properly. On the other hand, Authenticated Messages cannot accomplish this. However, we have already explained that Authenticated Messages can only add records anyway. In this case an application might end up with multiple versions, but no version can get overwritten or deleted.

The current TrustedPalm patch supports key security features without overburdening the simple PalmOS. By incorporating mechanisms that do not alter the core operating system, the patch adds functionality without threatening the already popular design. The APIs added by the TrustedPalm provide opportunity for future enhancement of the system.

X. References

1. A. Freier, P. Karlton, P. Kocher, “The SSL Protocol Version 3.0,” [Online document], 1996 Nov (SSL 3.02), [cited 1998 Apr 30] Available HTTP: http://home.netscape.com/eng/ssl3/draft302.txt

2. MIT Course 6.033 Staff, “Design Project #2: Authenticated PalmPilot Conduits,” [Online document], 1998 Apr, [cited 1998 April 30] Available HTTP: http://web.mit.edu/6.033/www/handouts/h19-dp2.html

3. Palm Computing Division, U.S. Robotics, “Corporate Background: HotSync Technology,” [Online document], 1998, [cited 1998 April 30] Available HTTP: http://www.palmpilot.com/newspromo/corporate/hotsync.html

4. Palm Computing Division, U.S. Robotics, “Developing Palm OS Conduits,” [Online document], 1996 Aug, [cited 1998 April 30] Available HTTP: http://www.palmpilot.com/devzone/tools/sdk30.html