Securing a Remote Terminal Application with a Mobile Trusted Device

Alina Oprea[1], Dirk Balfanz[2], Glenn Durfee[2], D. K. Smetters[2]

[1] *Carnegie Mellon University*
Pittsburgh, PA
alina@cs.cmu.edu

[2] *Palo Alto Research Center*
3333 Coyote Hill Road
Palo Alto, CA 94304
{balfanz, gdurfee, smetters}@parc.com

Abstract

Many real-world applications use credentials such as passwords as means of user authentication. When accessed from untrusted public terminals, such applications are vulnerable to credential sniffing attacks, as shown by recent highly publicized compromises [20].

In this paper, we describe a secure remote terminal application that allows users possessing a trusted device to delegate their credentials for performing a task to a public terminal without being in danger of disclosing any long-term secrets. Instead, the user gives the terminal the capability of performing a task temporarily (as long as the user is in its proximity). Our model is intuitive in the sense that the user exposes to the untrusted terminal only what he sees on the display, and nothing else. We present the design and implementation of such a system. The overhead – in terms of additional network traffic – created by introducing a trusted third party is a moderate 12%.

1. Introduction

From 2000 to 2002, Queens resident JuJu Jiang collected over 450 online banking passwords from unsuspecting Kinko's customers [20]. Jiang accomplished this by installing keyboard-sniffing software on public terminals in thirteen Kinko's stores in Manhattan, thus learning the GoToMyPC [15] account passwords of hundreds of people that had used those terminals. Armed with those GoToMyPC passwords, Jiang could connect to, and gain full control of the home PCs of his victims.

This story teaches us two lessons: (1) Computer users want to access their home computing environment from public terminals. (2) Those public terminals cannot be trusted with passwords or other credentials that give full access to the users' home computing environment.

How can we reconcile these two issues, and provide the user any secure access at all to their home computing environment from an untrusted terminal? Clearly, if the terminal were *completely untrusted*, then users would refrain from using it. On the other hand, if they could completely trust the public terminal, then no precautions against keyboard sniffers and the like would be necessary. In practice, the truth is usually somewhere in the middle – such terminals may often be trusted to perform the functions they advertise, but not be trustable with long-term secrets. We call a terminal that is not fully trusted "untrusted", even though we trust it to a certain extent (*e.g.*, not to deny service).

If users want to access their sensitive home computing environment from an untrusted terminal, it would be nice if they could control what information that terminal gains access to – in particular, the terminal should only obtain information about those parts of the home computing environment that the user chooses to access via the terminal, and gain neither access to other parts, nor to the user's long-term secrets (*e.g.*, passwords). This would allow the user to decide what information to expose to a given terminal on a case-by-case basis. For example, I may have no problem having a friend's PC access individual email messages in my Inbox (as long as I have control over which email messages get accessed), while I may not let a public terminal at a hackers' conference access my Inbox at all.

This sounds like we are asking users to make sophisticated security decisions – *i.e.*, "decide what to expose to an untrusted host", something end-users are usually considered ill-equipped to do. We believe instead that an effective focus on usability in system design can make such things not only possible, but easy and intuitive. In designing this system, our primary goal was to make it very easy for users to specify which parts of their home computing environment

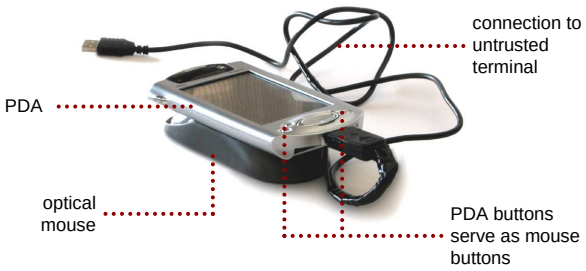

Figure 1. Our augmented PDA

are exposed to the untrusted terminal, and which are not. The simple analogy we have used here is "seeing" – the untrusted terminal gains access to any information, and only that information, that it "sees" – that the user chooses to display via the monitor. We use the same desktop metaphor of traditional user interfaces, and simply *extend* the familiar click-to-open semantics of such interfaces. In our system, a mouse click (or double-click) that normally means "open this item and display it" takes on an additional, yet intuitive, meaning: "expose this component of my computing environment to the untrusted terminal I'm currently at".

If a simple mouse click can expose a resource in our home computing environment to an untrusted terminal, we better make sure that only we, not the untrusted terminal, can issue such mouse clicks (or other input events). In other words, we provide the untrusted terminal *read-only* access to our home computing environment, and provide the input necessary to interact with that environment via a separate, *trusted input path* protected from the untrusted terminal.

At the same time, we would like the interaction with the untrusted terminal to be as natural as possible. Our solution to this problem is to use a modified PDA that the user plugs into the untrusted terminal, and which can be used as a mouse (see Figure 1). The PDA will authenticate all mouse (and other input) events to the home computing environment. It will withhold the credentials necessary to do this authentication from the untrusted terminal, thus making it impossible for the untrusted terminal to issue its own input events. At the same time, the PDA will temporarily issue (and update) credentials to the untrusted terminal allowing it to display representations of those parts of the home computing environment that the user decided to open up to it. Once we unplug the PDA from the untrusted terminal, the last credential issued expires after a timeout, leaving the terminal unable to even display the user's home computing environment.

We emphasize that the PDA in question is a general-purpose PDA, which can be used as such by its users. It just happens to also be a mouse that provides a trusted input path to the user's home computing environment when connected to an untrusted terminal. Figure 1 shows our initial prototype of such a PDA. A finished product would have the same form factor as a PDA, perhaps with an LED and optical sensor such as those found in optical mice embedded in the back of it. Alternatively, the trusted device could be a cell phone or two-way pager (*e.g.*, a RIM Blackberry™), or a special-purpose device with a form factor similar to "travel mice" currently sold.[1] This has the advantage that the device is much smaller (*e.g.*, could be put on a key chain), but has the disadvantage that it presents yet another special-purpose device to be carried by its user that cannot be used for other applications. For the rest of this paper, we will assume that the trusted mobile device is a general-purpose PDA that has optical mouse capabilities built in, or at least provides "touchpad"-like functionality through its touch screen.

In this paper, we describe a particular implementation of this idea. The graphical representation of the home computing environment is, in fact, a copy of the home PC's GUI desktop, using any one of the standard "remote desktop" protocols in common use today (for our implementation we chose VNC [18]). We give the user full access to a home PC of their choice.[2] They can open applications or documents, using their PDA as a mouse (and also as a keyboard). The desktop environment is displayed on the untrusted terminal, but only as long as the PDA is present. The terminal loses all credentials to access the home environment after a specified timeout has elapsed since the user walked away with the PDA. The end result is that the user interacts with the untrusted terminal (almost) the same way he would with his home PC, or with current remote desktop software. The advantage is that the user can be certain that only those items that he "exposes" to the untrusted terminal are accessible to the untrusted terminal.

We describe the goals of our system in Section 3, after looking at some related work in Section 2. In Section 4 we explain the design, implementation and performance evalu-

1 These are very small optical mice, similar in size to a car key fob.

2 While we describe our application in terms of giving a user access to a "home" PC, the same approach can obviously be used to access any network-accessible machine supporting the chosen remote terminal protocol.

ation of our particular VNC-based application. We conclude in Section 5.

2. Related Work

Traditional approaches to securing network access – *e.g.*, the use of `ssh` [5] – leave the user still vulnerable to password-sniffing attacks. Even the use of one-time passwords or tokens, while potentially preventing the untrusted terminal from accessing the user's resources in the future, does not protect the user from malicious manipulation of his home environment by the untrusted terminal while he is present and authenticated. These approaches provide the untrusted terminal total access to the home computing environment, while protecting that access from malicious eavesdroppers on the network. As we want to protect the home computing environment from a potentially malicious access terminal, this is insufficient. Instead, we must be able to delegate to that terminal very limited capabilities to access the home environment.

Delegation of credentials has been the focus of much previous work, ranging from the theoretical to the more applied. For example, in the ABLP access control logic [1], delegation plays a central role in form of "speaks-for" statements. A principal Alice can delegate her privileges to a principal Bob by announcing that Bob speaks for Alice. Refinements of that approach (*e.g.*, [2] or [11]) allow finer-grained delegation, *e.g.*, Alice could announce that Bob speaks for her only for the next 5 minutes. SPKI [12] has a form of certificates that allows principals to delegate a subset of their rights to other principals.

Compared to these efforts, we take a rather pragmatic approach: we simply send short-lived keys to the delegatee, which allow it to decrypt messages meant for the delegator for a short period of time. The advantage of our approach is that we can easily augment existing software (in our case, VNC) without any of the heavy lifting required by some of the cited approaches (*e.g.*, [2] requires clients to come up with proofs of authorization; other systems require some sort of shared infrastructure in which every host must participate). In fact, our system doesn't even require the untrusted terminal to possess a private key.

Remotely-Keyed Encryption [6, 7] is closer to our model: here an untrusted host is able to encrypt and decrypt data only with the help of a trusted smartcard (analogous to the PDA in our system). The remotely-keyed encryption protocol is designed to guarantee that the smartcard must be present and involved to encrypt or decrypt any data, while not actually sending all of that data over the low-bandwidth link to the card. One disadvantage of remotely-keyed encryption is that the host can decrypt only entire messages – handling streaming data, for example, poses a problem. One could get around this by dividing the data stream into smaller packets which are encrypted and decrypted separately, but then the smartcard has to interact with the host for every packet received. In our system, the frequency with which the PDA and untrusted terminal interact is determined by the expiration time of delegated keys, not by the frequency of packets sent.

Zero-Interaction Authentication [10, 9] uses an authentication token to store credentials needed to use a host computer. If you take the token away from the host computer, the host computer will forget all keys needed to read and write files, and will essentially become unusable. The similarity to our work is that the token issues keys to the host computer, which it then uses to get useful work done. The difference is in the trust model. In Zero-Interaction Authentication, the host computer is trusted. It voluntarily forgets all keys once it senses that the token is no longer in proximity. In our system, the host computer is untrusted. It constantly needs updated keying material from the PDA to access the user's home PC desktop.

People have proposed to use PDAs as touchpads before (*e.g.*, [17]), but mostly for reasons of convenience. In our case, we use the PDA as a touchpad to create a *trusted input path*. Furthermore, we have augmented a PDA with (parts of) an optical mouse (see Figure 1). This allows the PDA to be used as a mouse, not just as a touchpad.

Balfanz and Felten [3] point out that PDAs can provide a trusted input and output path when used with an untrusted host. We also use a PDA as a trusted input device, and find it more trustworthy than a public terminal for the same reasons given in [3]. They, however, use the PDA to provide cryptographic functions that the host cannot be trusted with. This method would completely fail in our case – the amount of framebuffer data that needs to be decrypted by the untrusted host is much too large to be handled by the PDA. Consequently, we *do* let the untrusted host handle its own decryption, albeit with rapidly expiring keys.

Finally, our work has some similarity to SSL-Splitting [16]. There, untrusted proxy servers are given only encryption, but not MAC keys to serve out Web content over an SSL connection. Here, we give the untrusted terminal decryption *and* MAC keys to receive data over an SSL connection. There, clients connect to untrusted proxy servers while requiring assurance from the server that the content served out is authentic. Here, the clients *are* the untrusted entities, and it is the *server* that requires assurance that the input events are authentic.

3. Preliminaries and Goals

The goal of our system is to allow a user in the possession of a small, trusted device to access sensitive information stored on his home computer in a secure manner. We

provide that access in the form of a "remote desktop" application, giving the user any access he desires to his home computing environment. The user holds certain capabilities on the trusted device and delegates some of them, temporarily, to an untrusted host that displays the sensitive information from the home environment. We emphasize that in our system, the input and output paths are clearly separated: all the input comes from the trusted device, whereas the output goes to the untrusted host. In other words, the untrusted host has "read-only" access to just that part of the home computing environment that the user chooses to display. The untrusted host is trusted to some extent to display correctly the information it receives.

3.1. Model

The players in the secure remote terminal application are: the home computer (HC) holding sensitive information, that acts as the remote terminal server; the untrusted terminal (UT) to which the user delegates the capability of acting as the remote terminal output; the user with its trusted PDA from which the input to the remote terminal server is sent.

We are assuming that all the devices from the user home network (all the devices that are trusted to the user and form the trusted computing base) are given certain credentials. In our particular implementation, the credentials take the form of certificates signed by a root home certification authority. The root home certification authority might be the PDA itself or any other trusted device. Any two devices from the home network use these certificates to mutually authenticate. The communication between devices from the home network (in particular, between the PDA and the home computer) is always done over SSL with client authentication enabled.

The PDA and the untrusted terminal communicate through a secure channel. In our prototype, we physically plug the PDA into the untrusted terminal, but we could also imagine a wireless connection authenticated through a location-limited channel [4] such as infrared.

Our security goals are the following:

1. All the input events (mouse and keyboard) to the remote terminal server come from the trusted device on a secure communication channel.
2. All the output goes to the untrusted host on a secure communication channel.
3. The untrusted host can access sensitive information only when the trusted device is in its proximity.

We need to clarify an aspect related to our third security goal: there is a timeout between the moment in which the PDA stops delegating the credentials and the time the host computer is actually denied access to the sensitive information. We think that values on the order of seconds, and even minutes, are reasonable for the timeout. Such a timeout is important, as the user may choose to display on the host computer time-varying information whose future contents may be sensitive. For instance, the user may open a window showing the contents of his most recent e-mail, that automatically updates as new messages arrive. Or, the user may display the current output of his home video surveillance system. It is important, therefore, that the untrusted host's access to the home environment is time-limited and tied to the presence of the user, as evidenced by the presence of his trusted device.

A straightforward solution that one might think of deploying for the delegation of capabilities is the following: the trusted device hands its certificate (and private key) to the untrusted host. There are two problems with this approach: (1) The certificate is likely a coarse-grained capability, *i.e.*, it authorizes its bearer for all kinds of access to the home network and (2) the certificate may not expire soon enough, and leave the untrusted computer with the ability to access the user's home network long after the user has walked away. Therefore, we need to find lower-privilege credentials to delegate (*i.e.*, credentials that do not give full access to the user's home network), and we need a mechanism to ensure that the delegation is *temporary*, *i.e.*, just for the period when the user is in the proximity of the untrusted device.

Another, perhaps even simpler solution to this problem would be to avoid the delegation of capabilities at all: simply provide secure remote access to the home environment to just the trusted device, which can then forward any subset of the information it receives to the untrusted terminal. Unfortunately, this places unreasonable bandwidth and computation demands on the trusted device, which we would like to allow to be as small and inexpensive as possible.

3.2. Overview of SSL

SSL (Secure Socket Layer) [14] is a transport level security protocol that creates a secure transparent tunnel between a client and a server. The protocol consists of two phases: an initial handshake initiated by the client and a data transfer phase. In the handshake phase, the server authenticates to the client (client authentication is optional); then the parties establish the cryptographic algorithms used for confidentiality and data integrity(ciphers, MAC algorithms); finally, the client and server derive the cryptographic keys.

After verifying the server's certificate, the client and server agree on a shared secret – the "pre-master secret". If the server authenticates with an RSA public key, the client simply generates a random value to be the pre-master secret, and encrypts it with the public key of the server. From

this pre-master secret, both the client and server can derive a master secret. All the cryptographic keys are generated from the master secret using cryptographically secure hash functions MD5 and SHA-1. The keys generated by the client and server are: (1) the client write MAC key (the key used by the client to compute MACs), (2) the server write MAC key (the key used by the server to compute MACs), (3) the client write key (the key used by the client to encrypt) and (4) the server write key (the key used by the server to encrypt). After the key computation, both the client and server compute a MAC of all the handshake messages and verify that the corresponding MAC computed by the other party is correct.

In the data transfer phase, SSL breaks the data sent in both directions into SSL records of variable size. The records are encrypted with the write key of the party that sends the message. The sender of the message computes a MAC for each record, using its secret MAC key and adds a header specifying the length of the packets and the SSL version used. The use of SSL is transparent to the application level, so any application running over TCP can be modified to run over SSL.

3.3. Virtual Network Computing (VNC)

Virtual Network Computing [18] is an open-source remote display system developed by AT&T that allows a remote terminal (VNC viewer) to access the graphical interface of a VNC server. The protocol is platform-independent and is designed such that the viewer is a lightweight application that can run on various hardware configurations.

VNC is based on the Remote FrameBuffer (RFB) protocol [19], a protocol for encoding screen images as rectangles. It supports several encodings of these images, and negotiates the one to be used in a particular connection in the initial phase of the protocol. The input to the VNC server comes from the client, which encodes keyboard and mouse events and transmits them to the server. The protocol is adaptive in that an update is sent by the server only when explicitly requested by the client . Thus, the update rate can be adjusted dynamically by the client according to its capabilities and network characteristics. This means that a VNC client sends two kinds of messages to the server – one containing traditional input (keyboard and mouse) events, the other containing simple requests for display update. An application such as ours which wishes to separate the display and input components of the VNC client must therefore provide the display component a channel back to the VNC server, over which it can only request display updates.

VNC authentication is password-based. The VNC server is configured by the administrator with a password and the VNC viewer has to prove knowledge of that password when it initiates a connection. The proof is done via a challenge-response protocol: the server sends a challenge, and the viewer replies with a DES encryption of the challenge with a key derived from the password supplied by the user.

4. A Secure Remote Terminal Application

We present an overview of our system and then explain the roles of the three parties (untrusted terminal, PDA, and home computer) in the remote terminal application. We then report on some issues that we encountered when designing our system, and describe a specific implementation before evaluating its performance.

For our prototype, we chose to modify an existing open-source remote desktop application, VNC [18] (see Section 3.3 for details of the VNC protocol). We secure all communication between trusted device, home computer, and untrusted terminal using SSL/TLS.

4.1. The Three-Party Secure Remote Terminal Protocol

Below we describe the secure remote terminal protocol.

- I: The PDA contacts the home computer (PDA $\leftrightarrow$ HC)

  The PDA initiates an SSL session with the home computer in which they both authenticate using the certificates signed by the home root certification authority. They also compute the master secret ms of the SSL session and negotiate the length of the time interval t.

  In our implementation, the PDA needs to be connected to the untrusted terminal to perform this step (the untrusted terminal provides network connectivity). If the PDA has its own network connection, then this step can be performed before coming in contact with the untrusted terminal.

- II: The PDA contacts the untrusted terminal (PDA $\leftrightarrow$ UT)

  The PDA sends to the untrusted terminal the name of the home computer, the home certification authority's root certificate, and the VNC password derived from ms.

  Again, in our implementation, the destination of this information is implicitly given by the fact that the untrusted terminal and PDA are connected by a wire. In general, a PDA that has its own network connection could learn the identity of the untrusted terminal through a location-limited handshake as in [4].

- III: VNC connection initiation (UT $\leftrightarrow$ HC)

  The untrusted terminal starts an SSL session with the home computer in which client authentication is disabled . Over this session, it starts a VNC connection to the home computer with the password provided

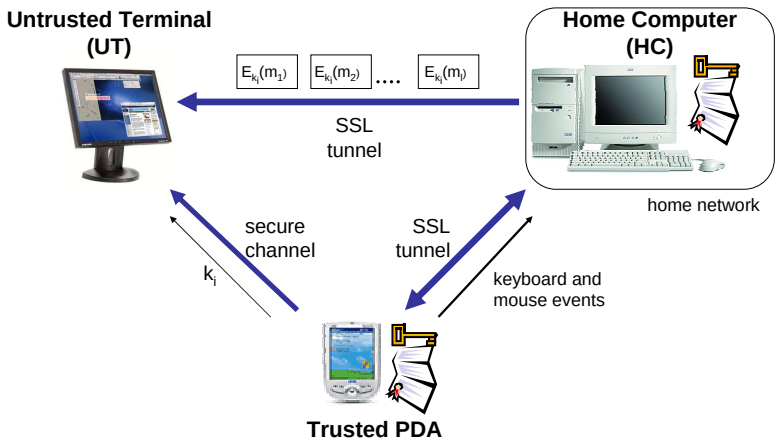

Figure 2. Three-party VNC protocol

by the PDA. The input of this VNC client (*i.e.*, mouse and keyboard events sent by the untrusted terminal) is discarded by the home computer.

At this point, the home computer has two open SSL connections: one to the PDA, which is authenticated using client authentication; the other to the untrusted terminal, which is not authenticated on the SSL level (but has used a one-time VNC password). The packets that come in from the authenticated connection (the PDA) are fed into the input-event queue of the home computer. As a result of these input message (mouse movement, key strokes, *etc.*), the display of the home computer changes, and the display updates is sent through the other SSL connection (to the untrusted terminal).

- IV: Start Timer (HC $\leftrightarrow$ PDA)

  The home computer sends a "Start Timer" message to the PDA. Both the home computer and the PDA (on receiving the message) start a timer with timeout t and every time period, they compute $K_i = \text{SHA1}(ms\|i)$.

- The protocol

  The home computer uses K_i as the server encryption key in the SSL session with the untrusted terminal over which it sends RFB update messages. At the start of each interval i, the PDA sends K_i to the untrusted terminal for as long as the PDA wishes to delegate to it. The PDA encodes mouse and keyboard events and sends them to the home computer over the SSL session already established. Figure 2 shows a graphical representation of the secure remote terminal protocol.

4.2. Design Issues

In the design of our system, we tried to minimize the computation involving the PDA and the communication going through it, as we consider the PDA a resource constrained device compared to both the VNC server and viewer. While ultimately arriving at the design choices presented above, along the way we considered several challenges and potential alternative designs discussed below:

TWO CONNECTIONS VS. ONE CONNECTION. At a first glance, it might seem that the SSL connection from the PDA to the home computer is superfluous. Perhaps the PDA could instead inject messages into a single SSL connection that is set up between the untrusted terminal and the home computer (the untrusted terminal, of course, would not have the necessary keying material to inject those messages itself). This way, the VNC server on the home computer would have only needed to deal with one incoming connection (which would have required fewer changes to the original code base).

However, this design overlooks the fact that the untrusted host needs to also send messages, not only to receive messages from the home computer. The reason is the design of the RFB protocol, which is the underlying protocol for VNC (see Section 3.3). First, in the RFB protocol, there is an initialization phase in which the server and client establish some parameters that the connection uses: the protocol version, the type of encoding and the pixel format. Secondly, during the VNC session, the viewer responds to FramebufferUpdate messages (these denote changes in the screen since the last message) from the server with a FramebufferUpdateRequest message. The server sends a new FramebufferUpdate only if there is a screen update and the client has sent a FramebufferUpdateRequest message. This mechanism is useful for allowing the client to regulate the rate at which it receives FramebufferUpdate messages, depending on its characteristics and network connection. Taking this into account, we need to provide the untrusted terminal with the means of sending messages to the home computer, *i.e.*, with the necessary MAC and encryption keys.

This means that the input (mouse and keyboard) messages need to be authenticated with different keys, which we accomplish by opening a separate SSL connection to the home computer.

CAPABILITIES. In our protocol, time is divided into equal time intervals and capabilities are issued by the trusted device to the untrusted host at the beginning of each time interval. We chose the capability for time interval i to be a session key used by the home computer to encrypt the communication to the untrusted terminal. It is derived from the master secret ms of the SSL session between the PDA and the home computer and from the time period index i as $K_i = h(ms\|i)$, where h is a cryptographically secure hash function such as SHA-1. The overhead for the PDA is the computation of a hash function every time interval.

The PDA also needs to provide the untrusted terminal with an initial VNC password and CA root certificate for the home network so it can connect to the home computer (see paragraph "VNC password" below). This can be viewed as an initial capability (at the initial time 0).

Alternatively, we considered a second instantiation for the capabilities: the capability for time interval i is a certificate signed by the PDA valid for that time period. At the beginning of every time period, the home computer requests a proof of possession of the certificate for that period and closes the connection if the untrusted terminal is unable to provide such a proof. We decided against this approach as it increased the computational burden on the PDA (to sign a certificate every time interval). In addition, it would have added additional communication between the untrusted terminal and home computer and would require much greater changes to the VNC and SSL protocols to incorporate a challenge-response exchange between the untrusted terminal and home computer every time interval.

RESTRICTING DELEGATION. The communication between the home computer and the untrusted terminal needs to be restricted only to the time when the PDA is in the proximity of the untrusted terminal. In our solution, the mechanism for achieving this is encrypting all the communication between the home computer and the untrusted terminal with a key provided by the PDA to the untrusted terminal.

LENGTH OF THE TIME INTERVAL t. We recommend a fairly large value for the parameter t (around 1 minute). We believe that allowing the untrusted terminal to continue to display the home computing environment for at most one more minute after the PDA has left from its proximity is reasonable. From the perspective of the PDA, computing a hash function every minute is a negligible overhead.

We emphasize that we only require fairly weak time synchronization in our protocol. The home computer starts a timer for generating the capabilities K_i when the untrusted terminal connects to it, and then sends a "start timer" message to the PDA. When receiving this message, the PDA also starts a timer, which is delayed with the time it takes the packet to reach the PDA (typically under a second). All the messages sent from the home computer to the untrusted terminal contain a key identifier so that the untrusted terminal knows which key it has to use to decrypt that packet. If the untrusted terminal has not yet received the key from the PDA, it waits an amount of time equal to double the length of the time interval. If the timeout has elapsed, the untrusted terminal concludes that the PDA is not in its proximity anymore. This works as long as the clocks on the PDA and home computer run at roughly the same speed. If the clock speed differs significantly, we could add a resynchronization message from HC to PDA to our protocol. In our tests, we did not find this to be necessary.

VNC PASSWORD. We chose not to change the password-based authentication mechanism used to authenticate the untrusted terminal to the home computer (we note that we *do* use strong SSL client authentication to authenticate the PDA to the home computer). The untrusted terminal (unlike the PDA) has no *a priori* relationship with the home computer, so strong authentication makes less sense . Therefore, we found it unnecessary to replace the password-based authentication mechanism inside the VNC server with something stronger.

The password we use to authenticate the untrusted terminal to the home computer is a one-time password derived both by the PDA and the home computer from the SSL master secret negotiated in their SSL handshake. This gives the home computer some assurance that the untrusted terminal connecting is, in fact, connected to the trusted PDA.

TRUSTED OUTPUT. In our system, we assume that the untrusted terminal faithfully renders the frame buffer data sent to it by the home computer. Although the untrusted terminal cannot send output events to the home computer (it has read-only access to the home environment), it is theoretically possible for the untrusted computer to manipulate what it displays to the user in a way that tricks the user into opening up resources he did not intend to open up.

We briefly experimented with a counter-measure that consisted of displaying a snapshot of the screen (a small portion around the location of the mouse pointer) on the PDA. The user could move the PDA around, and thus see faithful representations of different parts of the home computer's screen, and compare them to the display of the trusted terminal. Apart from the questionable usability properties of this approach it turned out that the performance of the RFB protocol and VNC software on our PDA was too poor to make this approach work efficiently.

Spoofing the output is therefore currently a vulnerability in our system. We do point out, however, that using this vulnerability to launch a targeted attack on a home comput-

ing environment (*e.g.*, learning the content of a file of the attacker's choice) would require sophisticated software that interpreted the frame-buffer data, drew conclusions about the objects displayed, and then intelligently spoofed the display. While possible, we consider this threat mostly theoretical. Users need to be aware, however, that our system does not currently guarantee a trusted output path.

TRUSTED KEYBOARD INPUT. While it is quite natural to use our augmented PDA as a mouse, generating keyboard events on the PDA is somewhat awkward. Writing a whole email message, for example, can be quite cumbersome. On the other hand, we cannot let the untrusted terminal generate arbitrary keyboard events, as this would unduly expose the home computer. As a middle ground, we considered the following: The untrusted terminal could be allowed to send certain restricted keyboard events to the home computer, such as pure ASCII characters that are not accompanied by any Ctrl or Alt keys. All other keyboard events (function keys, those accompanied by Ctrl or Alt keys, *etc.*) would be ignored by the home computer (unless they come directly from the PDA). Now, one could use our PDA/mouse to open up, say, an email client, and compose a message using the untrusted terminal's keyboard. One would not be able to switch applications (using Alt-Tab), or send the message (using Ctrl-S) using the untrusted terminal's keyboard. To do this, the user would still have to use the PDA/mouse. While this "sandboxing" is not part of our current implementation, we believe that it would be a useful addition to future versions of our system.

SOFTWARE REQUIREMENTS. Our proof-of-concept prototype requires adaptation of the VNC software on both the untrusted terminal (the VNC client) and the home computer (the VNC server), and a special-purpose client on the trusted device to perform its part of the protocol. Requiring software modifications on all three components of the system seems a strong barrier to adoption. But, in order to provide a trusted secure remote access solution, it is clear that all three components of the system must be "aware" of the three-party nature of the interaction.

Of these, modifications to the untrusted terminal seem the most difficult to accomplish in practice, as users currently install new server software on their home computers in order to be able to engage in new remote access protocols (*e.g.*, VNC itself, Microsoft's remote desktop software, or GoToMyPC [15]), and are likely to install client software on their PDA that makes it easier for them to securely access their home environment. We can actually deploy our system without requiring any software to be pre-installed on the untrusted terminal, by using a Java or ActiveX-based viewer (*e.g.*, in our case, VNC client) program delivered via a web browser. This is in fact the solution adopted by GoToMyPC [15].

4.3. Implementation

For our implementation, we have used the Visual C++ 6.0 and eMbedded Visual C++ 3.0 environments. Our prototype consists of three applications.

The home computer application, developed for Microsoft Windows in Visual C++ 6.0, uses and modifies the VNC server and OpenSSL sources. First, the VNC server is modified to run over SSL sockets instead of standard sockets. The VNC server is also changed to send output packets to a different host than the one from which it receives input packets. Whenever it receives an input message from the PDA (mouse or keyboard event), it updates the screen and sends the update to the untrusted terminal. The VNC server discards any keyboard or mouse input packets received from the untrusted terminal after the initialization phase of the VNC protocol is over.

Several modifications had to be made to the OpenSSL library. We needed to change the server encryption key periodically (without changing the master secret or any other keys) and OpenSSL does not provide an interface for this functionality. We also changed the SSL record header to include a key identifier, such that the SSL client (UT) knows which key to use to decrypt that packet.

The untrusted terminal application is also developed in Visual C++ 6.0 for Microsoft Windows. It is based on the VNC client and OpenSSL sources. The VNC viewer is modified to run over SSL and to receive the initial VNC password and root certificate from the PDA. It also keeps track of the 10 most recent encryption keys received from the PDA and the index of the current key identifier. Whenever it receives an SSL packet, it checks its key identifier. If the key identifier is greater than the current one, it waits for the key from the PDA; if it receives the key, it updates the client read key in SSL; otherwise, if a timeout has expired, it concludes that it lost the connection with the PDA and the application triggers an exception. If the key identifier is less than the current one, then it retrieves the key from the key table and changes the client read key in SSL.

The PDA application is written in eMbedded Visual C++ 3.0 for Windows CE. From the VNC viewer sources, it uses only those files that process keyboard and mouse events. In our implementation, the untrusted terminal provides the PDA with the necessary network connectivity once the PDA is plugged into the terminal[3]. In particular, the connection from the PDA to the home computer flows through the untrusted terminal. Remember, though, that this connection is end-to-end SSL-protected.

3 The WindowsCE connectivity software for Windows (ActiveSync) acts as a NATing gateway, sharing the untrusted PC's network connection with the PDA.

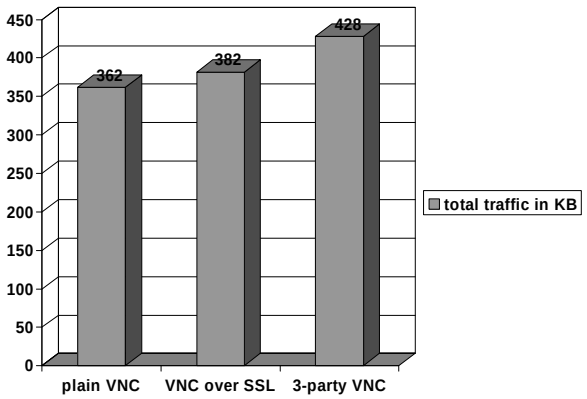

Figure 3. Overhead comparison of plain VNC, SSL-secured VNC, and three-party VNC

The application consists of a full-screen window, which receives mouse events from the PDA stylus, thus acting like a touchpad. The keyboard and mouse events are encoded and sent over the SSL tunnel to the home computer. In addition, we augmented the PDA with (parts of) an optical mouse (see Figure 1). In the current implementation, the communication between the PDA and the untrusted terminal takes place over a (USB) wire, but we could also use a secure wireless connection authenticated through a location-limited channel [4]. In the latter scenario, rather than plugging our PDA/mouse into the terminal, we require that it has its own network connectivity, and we would merely point the PDA/mouse at the untrusted terminal we wish to use.

4.4. Performance Evaluation

We evaluated the overhead that our three-party secure remote terminal introduces compared to a two-party remote terminal application. We considered three protocols: plain VNC, VNC over SSL and the three-party VNC. Using Ethereal [13], we measured the number of bytes sent in the three-party VNC for a benchmark in which we connected to the home computer, opened an email application, and composed and sent an e-mail message. From this, we estimated a lower bound on the traffic generated by the plain VNC, and the VNC-over-SSL protocols.

We considered bandwidth overhead (rather than, say measurements of latency) as the appropriate metric for our system for several reasons. First, our benchmark application, email composition, takes an amount of time primarily dependent on how long it must wait for the user to enter the message – it is not CPU bound. Second, timing measurements for our protocol will depend heavily on both the processor characteristics of the trusted device, and the bandwidth of its network connection. Measurements of bandwidth overhead are independent of these. Finally, latency and timing requirements can be estimated from measurements of bandwidth overhead using the characteristics of the trusted device under consideration, the bandwidth of its connections to the UT and HC, and existing benchmarks showing the proportion of time of SSL exchanges spent in cryptographic computation and other activities [8].

Let us detail our analysis. We measured the number of bytes sent on the links HC $\leftrightarrow$ PDA and HC $\leftrightarrow$ UT for the three-party VNC protocol[4] and the results were:

S_1	HC $\rightarrow$ UT	:	361.450 KB
S_2	UT $\rightarrow$ HC	:	31.815 KB
S_3	HC $\rightarrow$ PDA	:	14.470 KB
S_4	PDA $\rightarrow$ HC	:	20.611 KB

The total number of bytes sent in the three-party VNC protocol is $S_1 + S_2 + S_3 + S_4$ = 428.346 KB. In the two-party VNC over SSL protocol, the packets to be sent are at least those on the links S_1 and S_4, which are VNC output and VNC input packets, respectively. The packets on the links S_2 and S_3 are mostly ACK packets and some of them may be sent in the VNC over SSL protocol. Thus, a lower bound on the number of bytes sent in the VNC over SSL protocol is $S_1 + S_4$ = 382.061 KB. In the worst case, our three-party VNC protocol introduces an overhead of 12.1% compared to the two-party VNC over SSL protocol.

We can evaluate how much overhead the SSL protocol introduces in terms of the number of bytes. On the links S_1 and S_4, there were a total of 792 (SSL) packets sent. Without SSL, each packet would have been 25 bytes smaller (SSL adds to each record a 5 bytes header and a 20 bytes MAC for data integrity), which means that in total, SSL introduces an additional total of 19.8 KB compared to (the estimated lower bound of) plain VNC, which corresponds to an overhead of 5.5%. Figure 3 summarizes our findings: Compared to plain VNC, we estimate that an SSL-based version of VNC would add 5.5% of traffic overhead, and our three-party-based protocol adds an additional 12.1%.

4 The traffic on the link PDA $\leftrightarrow$ UT is negligible.

5. Conclusions

Accessing a home computing environment from an untrusted public terminal is currently a risky endeavor, as well-publicized security compromises show. In this paper we have presented a system that lowers that risk considerably: First, the public terminal never learns credentials that allow it to gain full access to the user's home computing environment. Second, the access that we do grant to the untrusted terminal is read-only – it cannot manipulate the home computing environment in any way. Third, which part of the home computing environment is exposed to the untrusted terminal is entirely in the hands of the user. This unique combination of trusted input device and read-only access for an untrusted terminal allows for a natural and safe interaction with the user's home computing environment while away from home. It introduces only moderate overhead compared to insecure, or merely traffic-protected, remote terminal applications, as we have demonstrated in our prototype.

6. Acknowledgments

We would like to thank Paul Stewart for hacking an optical PS/2 mouse so that it can be used with the serial port of a Compaq iPAQ, as well as creating the necessary cables to connect (the remnants of) an optical mouse to the PDA, and the PDA/mouse to the untrusted terminal. We would also like to thank Mike Reiter for useful discussions concerning the verification of the output displayed by the untrusted terminal.

References

[1] M. Abadi, M. Burrows, B. Lampson, and G. D. Plotkin. A calculus for access control in distributed systems. *ACM Transactions on Programming Languages and Systems*, 15(4):706–734, September 1993.

[2] A. W. Appel and E. W. Felten. Proof-carrying authentication. In *Proceedings of the 6th ACM Conference on Computer and Communications Security*, Singapore, November 1999.

[3] D. Balfanz and E. Felten. Hand-held computers can be better smart cards. In *Proceedings of USENIX Security '99*, Washington, DC, August 1999.

[4] D. Balfanz, D. Smetters, P. Stewart, and H. C. Wong. Talking to strangers: Authentication in ad-hoc wireless networks. In *Proceedings of the 2002 Network and Distributed Systems Security Symposium (NDSS'02)*, San Diego, CA, February 2002. The Internet Society.

[5] D. J. Barrett and R. E. Silverman. *SSH The Secure Shell*. O'Reilly, 2001.

[6] Blaze, Feigenbaum, and Naor. A formal treatment of remotely keyed encryption. In *EUROCRYPT: Advances in Cryptology: Proceedings of EUROCRYPT*, 1998.

[7] M. Blaze. High-bandwidth encryption with low-bandwidth smartcards. In *Proceedings of the Fast Software Encryption Workshop*, number 1039 in Lecure Notes in Computer Science, pages 33–40. Springer-Verlag, 1996.

[8] C. Coarfa, P. Druschel, and D. S. Wallach. Performance analysis of TLS web servers. In *Proceedings of Network and Distributed System Security Symposium, NDSS '02*, February 2002.

[9] M. D. Corner and B. D. Noble. Zero-interaction authentication. In *Proceedings of the eighth Annual International Conference on Mobile Computing and Networking (MOBICOM-02)*, pages 1–11, New York, Sept. 23–28 2002. ACM Press.

[10] M. D. Corner and B. D. Noble. Protecting applications with transient authentication. In *The First International Conference on Mobile Systems, Applications, and Services (MobiSys '03)*, 2003.

[11] J. DeTreville. Binder, a logic-based security language. In *2002 IEEE Symposium on Security and Privacy*, Oakland, CA, May 2002.

[12] C. M. Ellison, B. Frantz, B. Lampson, R. Rivest, B. M. Thomas, and T. Ylonen. *SPKI Certificate Theory*, September 1999. RFC2693.

[13] Ethereal Project. Ethereal. `http://www.ethereal.com`.

[14] A. O. Freier, P. Karlton, and P. C. Kocher. *The SSL Protocol Version 3.0*. IETF - Transport Layer Security Working Group, The Internet Society, November 1996. Internet Draft (work in progress).

[15] GoToMyPC. GoToMyPC. `http://www.gotomypc.com`.

[16] C. Lesniewski-Laas and M. F. Kaashoek. SSL splitting: Securely serving data from untrusted caches. In *Proceedings of the 12th USENIX Security Symposium*, Washington, D.C., August 2003.

[17] B. A. Myers, J. O. Wobbrock, S. Yang, B. Yeung, J. Nichols, and R. Miller. Using handhelds to help people with motor impairments. In *Proceedings of the Fifth International ACM SIGCAPH Conference on Assistive Technologies; ASSETS 2002*, Edinburgh, Scotland, July 2002.

[18] T. Richardson, Q. Stafford-Fraser, K. R. Wood, and A. Hopper. Virtual network computing. *IEEE Internet Computing*, 2(1):33–38, 1998.

[19] T. Richardson and K. Wood. *The RFB Protocol*. ORL, Cambridge, January 1998.

[20] The Register. Guilty plea in Kinko's keystroke caper. `http://www.theregister.co.uk/content/55/31832.html`.