Implementing a multi-hat PDA

Matthew Johnson and Frank Stajano

University of Cambridge

Abstract. We describe our work in progress aimed at implementing a multi-hat PDA. Our current prototype is based on SELinux and KDE and accepts a proximity token, in the form of a Bluetooth cellphone, as an alternative authentication method. We analyse in detail the suitability of several alternatives for the graphical environment and underlying OS and we discuss a variety of interesting implementation issues that arose during development.

1 Introduction

At the previous Security Protocols workshop one of us (Stajano [13]) described a "multi-hat" PDA as a design reconciling security and usability.

The central idea of that work was that, although the PDA is the archetypal single-user machine, it may be useful for its owner to be able to assume several roles (referred to as "hats"), each with specific security requirements. Hats map naturally to operating-system-level user-ids, each with its own password—or, more generally, access credentials.

The multi-hat PDA supports several concurrent graphical sessions, only one of which is accessible at a time. It is possible to switch from one session to another without closing the first session, so long as one presents the credentials of the hat owning the second session; on coming back to the first or any other suspended session, though, one must again present the credentials for that session.

There is a special "null" hat which can be accessed without any credentials. It corresponds to a "guest" user-id or to an anonymous account. Being able to switch to the null hat session at any time with no hassle to access stateless applications or briefly to lend the machine to others is one of the features that improve the usability of the multi-hat PDA without compromising its security.

The work in progress described in this paper aims to build a multi-hat PDA. Ideally, our projected implementation target is a machine from the Sharp Zaurus SL-C family, a series of StrongARM powered devices that natively run Linux and Qtopia. In that perspective, our multi-hat PDA will have to be able to run native unmodified Zaurus applications. At the prototype stage, however, for convenience of development we have chosen to target a standard desktop or laptop PC. We are also refraining from committing prematurely to a particular graphical environment such as Qtopia, since only the experience gained while implementing the prototype will allow us to choose the technically most suitable alternative.

2 Operating System Alternatives

2.1 Windows XP

Microsoft Windows XP was initially the closest to what we are aiming for because of its "Fast User Switching" feature [2]. This functionality—having multiple users running at the same time and being able to switch between them without having to log out or close the applications that other users are running—had been available in Linux for some time at the console level, while Richardson et al [9] extended multiple users to graphical logins in the context of the X-based Teleport System and, later, VNC [10]. The innovation introduced by Windows XP was to integrate this with the login manager, screen-saver and desktop. This has many of the features that we require such as allowing multiple users with differing permissions to run sessions simultaneously and providing a central method for changing between them. However, there is no easy method to hook our own events, such as lid closing or proximity events, into the switching system.

To implement a session for the null hat we need a system either supports users without a password, or allows logging in without having to provide the password. Windows XP supports password-less users. If the user without password is the only one then the login screen will not even be displayed: the user will be logged in directly. When there are multiple users one must still be explicitly selected, but no password will be prompted for if the user has none. Windows allows for some credentials other than passwords (for example smart cards), but the range supported is fairly small and adding support for other credentials is non-trivial.

Windows has a fairly comprehensive system of file permissions which is reasonably expressive and fine-grained—it will certainly allow us to restrict the less privileged hats as desired. We also, however, need to limit the network access of each hat. Windows XP has a built in firewall, but this is designed for a different problem. The firewall can restrict which ports and applications can access the network, but not which user can access the network, which is what we need to be able to restrict. The user and group policy editor allows an administrator to restrict many of the operating system features on a per-user basis, including remote access of the computer over the network, but cannot restrict which users can make outgoing connections.

The ultimate goal of this project is to produce a system on a PDA. Microsoft provide a version of Windows to run on PDAs, but this is Windows CE, rather than Windows XP, and doesn't support the permissions and Fast User Switching features we need. This, combined with the other problems, means that while possibly being the closest existing system to what we need, Windows cannot easily be used as the base for our initial prototype.

2.2 Linux

The free Linux operating system was the next system we looked at. Linux has the advantage that, due to its open source nature, it can be extended very

easily. We looked at several methods to achieve our aims in Linux—most basic distributions are unsuitable in a couple of ways, but fortunately these have been supplemented by other projects that build on Linux. Firstly we needed to improve the permissions model that is in use by Linux. Linux uses a fairly coarse-grained POSIX discretionary access control mechanism. We looked for solutions that provided more fine-grained control over the permissions granted to the various hats and that allowed us to define other restrictions such as per-user network access, things which it is not possible to do in normal Linux. Secondly, the multi-user support is not very well integrated—there should be a centralised way to manage sessions and multiple users.

Xen The Xen Virtual Machine Monitor [3] provides several virtual machines with strong separation between them. Each hat would be given its own virtual machine and Xen would provide the strong protection properties between hats. The idea is to offer essentially a separate machine to each hat, with a separate operating system and application suite. Network access can be restricted by using a firewall in the controlling domain. Xen would fulfil the security requirements, but has several practical disadvantages in this case. Because Xen is providing a different machine for each hat, the physical resources must be partitioned a priori. This is particularly a problem because Xen uses a static memory partitioning scheme, while PDAs usually do not have very much memory. Having an entirely separate operating system for each hat implies substantial overhead in disk, CPU and memory usage because there is very little which can be shared. It reduces the effectiveness of techniques such as shared libraries as these must be loaded separately for each domain. Xen is also currently only available on the Intel x86 and AMD-64 architectures, whereas we are intending to produce a prototype on an ARM-based PDA. Porting Xen would require significant work.

SELinux SELinux [8] is an extra security layer for Linux, developed by the NSA and distributed in the form of patches to the Linux kernel. SELinux is a Mandatory Access Control System that uses Role-Based Access Control.

The feature we are most interested in is the role-based access control. This means that a user may have several roles, which could be shared by other users and the access permissions are defined in terms of roles. These roles are inherited even when the UNIX user may not be, which means a process started in one role always has that role associated with it when checking permissions. The idea of roles maps very closely onto the idea of hats we are using.

SELinux provides hooks for restricting the right to view or write arbitrary files, execute programs, bind to network ports and send or receive network packets. These permissions can be granted based on the user, role, program used and object type and can represent any of the protections we wish to implement. Several users can have the same role for permissions which are based on the role, while having other permissions distinct per user. These permissions are all enforced using the general SELinux mandatory access controls.

The overhead for SELinux is a lot smaller than that of Xen. Because when using SELinux all sessions use the same operating system instance, there is no need to statically partition the RAM between the security domains, the normal operating system dynamic memory allocation is sufficient. There are also no processes which are duplicated between hats, such as operating system daemons, which all need to use CPU time for what is essentially duplicated effort. In addition, read only code pages can still be shared between domains which can execute the same processes.

The permissions system is also sufficiently flexible that nothing needs to be duplicated on disk because each process and user can be given access to exactly what they need.

Because SELinux is implemented as a security layer internal to Linux and uses generic extended attributes to store all the extra policy and label data, it is completely independent of the architecture and hardware that it is run on. This allows us to use SELinux on any system on which standard Linux will run. The PDAs we are using to build this system already run a port of Linux which makes SELinux an ideal candidate.

KDE The permissions system used to govern the access given to each hat is fairly independent of the user interface used to control sessions for the hats and the method used to control switching between the sessions.

Session switching has only recently made its way into the UNIX world in a centralised and integrated way through the K Desktop Environment (KDE). The facility is functionally similar to that in Windows XP. KDE uses multiple X Window System sessions and provides integrated GUI management of the sessions. These can be queried and controlled via the K Display Manager (KDM) and widgets on the desktop allow the user to easily change between the available sessions, or create a new one. KDM provides a remote interface by means of a FIFO and that can be used to control the creation of and switching between sessions.

KDE also provides a comprehensive remote interface to all the GUI features within each session. Each application registers methods which can be called remotely with the Desktop Communications Protocol (DCOP) server, which allows these methods to be queried and executed. DCOP calls can be used to send commands to any application and also perform session management, screen blanking and locking and so on.

As regards support for a public user without credentials, Linux makes it difficult to create a user with a blank password—the standard tools for setting passwords reject a blank password. If the password is manually set to nothing, however, then logging in will not prompt for a password. We would rather not do this in general because there may be some remote interface to the PDA and only local access should be without credential. KDM provides an interface to create a session for a user which can be provided with a password. This allows our system to create the public session without requiring any credentials, but also protect the system in other ways.

Mac OS X Recent Apple operating systems have a similar fast user switching feature to Windows. Since the Panther release you have been able to switch between simultaneously running users (with suitable Apple eye-candy), being prompted for a password as necessary. Despite this we haven't considered Mac OS X a good candidate. Mac OS is a proprietary operating system and Apple do not provide any version of it on a PDA. This may be because their graphical environment is heavily dependent on powerful graphics hardware, which is generally not found in PDAs. The lack of portability to our desired platform makes OS X a bad choice.

The system we have, therefore, chosen as the basis for the initial prototype is Linux with the NSA Security-Enhanced patches running KDE with KDM. This will allow us to run standard X-based applications. Most Linux-based PDAs don't run standard X, however, but instead run an embedded environment like Qtopia or GPE. To run applications for these PDAs unmodified we will have to produce a system which will work with those environments. This is discussed later in section 6.3.

3 The Hats

The multi-hat system is introduced primarily in order to allow easy access to the null hat that holds no secret data. A minimal multi-hat system will just have the null hat and another hat holding some secret data. The main distinction is therefore between the null hat and the others. Any further differentiations between hats will be implementation-specific and our security policy allows for this.

3.1 The Null Hat

The null hat is publicly accessible and provides services for which no credential is needed. These applications and services are ones for which the Big Stick security model [12, p. 96] is appropriate. Typical PDAs contain several applications like this, including a calendar (not diary), calculator and world time function, as well as most games. These applications are all stateless and do not use any resources which may be scarce or costly (for example, Internet access). The null hat can, therefore, always be accessed without presenting any credentials and one should be able to do this easily and simply. The default session to which the system changes if the user does not have any valid credentials is that of the null hat.

3.2 Restricted Hats

The security policy defined here will be flexible enough to allow any number of restricted hats, each of which can be given a different set of permissions. In our implementation of this we give an example set, although this is by no means

canonical or the most that can be done. These hats will have access to resources that the null hat will not be able to use, such as communication over the network, or access to files on the PDA. Any stateful applications, particularly those which store credentials for other systems such as email clients, should be in a restricted hat but it is up to each implementation to assign the desired security policy for what each hat is allowed to do.

We provide as an example two forms of restricted hat. The first one contains particularly secure personal data and always requires a credential to change into. In our implementation this credential is a password. The other restricted hat contains applications with network access and that can store some state, but do not have access to the private data. This will have a more relaxed security policy which may not always require explicit passwords to access.

4 The Multi-Hat Security Policy Model

The multi-hat idea is described concisely but completely in the rules of the following security policy model.

Rule 1. Hats. The machine supports a finite number of hats[1]. One of them is the special case known as the null hat[2]. Every hat, except the null hat, is associated with some credentials.

Rule 2. Sessions. The machine supports several simultaneous sessions[3], each belonging to a hat. Each session can be either active or locked[4] . At any one time, at most one session is active. The user can only interact with the active session. When the machine is in sleep mode, in hibernation mode or off, no session is active.

Rule 3. Session Unicity. For each hat there is at most one session[5].

Rule 4. Hat selection. There is a convenient way[6] to select any of the hats of the machine for the purpose of switching to (or launching, if necessary) the session of that hat.

---

[1] Hats may be considered equivalent to roles or to OS-level user-ids.

[2] The null hat, reachable without credentials, corresponds to an anonymous "guest" account.

[3] A session corresponds to a graphical login or "desktop". Within a session, the private data of the corresponding hat is accessible. The null hat has no private data.

[4] When a session is locked, all the data that is private to the hat owning that session is inaccessible to anyone without the credentials for that hat. This may be implemented by encrypting the data of a locked session with a key derived from the hat's credentials. Note that the null hat session has no private data and therefore locking it has little practical effect.

[5] This rule is not logically necessary (one could conceive opening several sessions for the same hat) but it makes the model simpler and easier to understand—and therefore more usable.

[6] This may be implemented with dedicated buttons, a scroll wheel, a top level menu or any other appropriate user interface device.

Rule 5. Switching sessions. To make a new session active, the credentials of the hat owning that session must be presented.

In this policy we do not specify what will qualify as appropriate authentication credentials, as we believe this should be both implementation-specific and also subject to change. In our prototype system, authentication credentials mainly consist of traditional user-name and password combinations, but we look forward to using other more convenient authentication methods. Some of these are discussed next.

4.1 Authentication by proximity token

One of the methods of authentication we have investigated is the use of a locality-based token. The credential is deemed to have been presented whenever the token is within a certain distance of the PDA. We have a working prototype of a proximity-based system that tracks the position of a 'master token' using Bluetooth. The master token can be any Bluetooth-capable device: we use a mobile phone. This is done by tracking the signal strength of the Bluetooth connection between the PDA and our token. This is similar to the system implemented by Corner and Noble [6], but does not require custom hardware or protocols because it uses the existing Bluetooth protocol found in many devices.

Bluetooth provides some authentication and encryption in the protocol itself. A shared secret (PIN) which is input manually into each of the two devices the first time an encrypted connection is requested. The PIN is used, along with nonces sent in the initial handshake, to generate a shared secret. This secret is used in future connections to authenticate the devices to each other and to secure the connection. If an attacker can read the traffic during the initial handshake then the security is entirely reliant on the strength of the PIN. Ideally the initial pairing operation should be done somewhere the communications cannot be snooped. For any extra security another protocol would have to be run on top of the Bluetooth protocol, which we decided not to do because we wanted our system to be compatible with all Bluetooth devices which wouldn't need to be programmable. With recent devices such as smart phones it would be possible to implement a separate protocol using a more secure system.

This still doesn't address the Man in the Middle (MITM) attack which is possible in this system. We could envisage an attacker having stolen the PDA relaying the Bluetooth protocol to an agent standing within range of the token to cause the PDA to unlock. This is more subtle than the conventional MITM attack on key agreement protocols, because the attacker is not able to read any data. He can, however, convince the PDA that it is close to the token. Bluetooth uses pre-shared keys in the form of the PIN and this shared secret is used to bootstrap the authentication. However, the attacker does not need to be able to read the traffic, all he is interested in is relaying the authentication challenge from the PDA to the authentication token and sending the replies back. This is one of the reasons we have not used the proximity token as authentication for

all of the restricted hats—even with this attack access would not be gained to personal or particularly sensitive data.

A number of people have suggested solutions to this problem. Brands and Chaum [5] proposed to calculate the maximum distance between prover and verifier given the propagation speed of the communications channel. More recently, Sastry et al [11] proposed a solution based on ultrasound echos and measuring the response time. A different solution to this problem is given by Alakassar [1]. He points out that being able to snoop the communication channel is required to perform this MITM attack and proposes a probabilistic channel hopping scheme which would require an attacker to be able to relay a large spectrum and be technically infeasible. Most of these systems have the same disadvantage as Corner and Noble's proposal because they require custom hardware.

An approach which doesn't necessarily require custom hardware was suggested in 1990 by Thomas Beth and Yvo Desmedt [4]. Their protocol requires each participant to reply at a fixed time interval agreed between the two. If the attacker requires a non-zero time to forward the signal, then the attack will be detected because responses will take too long. This protocol, unlike many others, can allow computation by both the prover and the verifier because the time delay can be set appropriately. The technical requirements for this approach are merely that the transmission time of the communications medium is already known by the participants and that the jitter in transmission times is smaller than the expected time the attacker needs to forward the signal. Unfortunately, the link layer echo times recorded vary between 26.38ms and 48.85ms at close range and up to 60.8ms at longer range. In comparison the time to transmit a packet over a fixed wired network can be as little as 0.1ms and even for a wireless network only a few ms (average 3ms in tests). Such a large variance makes this approach not applicable to a Bluetooth-based strategy.

4.2 Automatic Switching

The security policy above governs what happens when a change of hat is attempted. This can be caused by a user selecting a new hat using a keyboard combination or a menu item, or by a system event triggering a transition.

Events such as selecting sleep mode and turning the PDA off will cause active sessions to be locked following Rule 2. Other events, such as closing the PDA, might trigger other actions, for example switching to the null hat session. We have produced an extensible and configurable event handler to manage these events and associate them with actions. The event handler can also be used to manage the presence or absence of our authentication token.

Table 1 illustrates how these events cause actions to be executed on each of the sessions in our sample implementation.

Event	Effect on Session: Null Hat	Limited Hat	Restricted Hat
Power Off/On	Create Session and Activate	Create and Lock Session	Create and Lock Session
Sleep	Activate		Lock Session
Lid Close	Activate		Lock Session
Token Out Of Range	Activate	Lock Session	Lock Session
Token In Range		Unlock and Activate Session	

Table 1. Events and Actions on Sessions in Prototype

5 The Prototype

The prototype we have working at the moment is implemented on an Intel-based laptop using Linux and the K Desktop Environment as a basis. This prototype demonstrates the above switching policy at work controlling access to three hats.

As per our specification we have a stateless null hat offering a null hat session which has no network access. Certain events, such as closing the lid and suspending the machine, cause switching to the null hat session and locking all the other sessions.

The hats are implemented using three different users which have corresponding different SELinux roles. The null hat user/role is given permission to run only a small set of binaries, which are essentially all stateless applications. No network access is allowed, and file-system permissions are restricted to only read the programs and libraries required to run the stateless applications.

The other two hats have users/roles with progressively more permissions allowed, Both are given network access, but have separate home directories and may not read each other's. Most applications can be executed by either, but a few can be launched only by one.

We have produced an event-handling system which uses the exposed interfaces in KDE and KDM to do session management. It can switch sessions, lock sessions and automatically log a user in on a given display. The event handler has configuration files for each session, for each event the system should be able to handle and for each action that can be performed on a session. Individual event handlers can be configured to call any of the actions on any of the sessions. The event handlers get passed all of the state of the system and can query the authentication mechanisms.

The interface is managed via KDM and KDE. On boot an event is triggered which logs in each user to a new session and locks the sessions for all but the null hat. KDE has menu options which list all the open sessions (via a call to KDM) and will allow the user to change between them. Our event handler makes sure that when the session is changed all the other sessions are locked. Various other events (mainly triggered by ACPI events such as suspend and lid-closing) are

passed to the event handler which may cause the computer to activate a different session (usually that of the null hat).

Checking of credentials is done by our session-switching logic which can use DCOP calls to unlock the session without providing a password if other credentials were sufficient.

Section 3.2 describes our sample policy with two types of restricted hat. This is partly to demonstrate what can be done with the system, but also to provide access to common programs using an easier authentication method than passwords while still protecting more sensitive data. The way that system events affect the sessions for the restricted hats can be seen in table 1.

The Bluetooth proximity token uses a background demon process which maintains an open connection to the token. Each Bluetooth device has a 48bit unique public identifier. Initially the user manually gives the computer the identifier of the chosen token and pairs the two when the first connect. The demon constantly monitors the signal strength to the token and when the signal crosses a user-defined threshold the demon triggers an event in the event handler. The demon can also be queried to return the current connection state. The Bluetooth monitoring demon will be released as a stand-alone program under the GPL soon.

The prototype was implemented on a laptop rather than a PDA due to ease of use and hardware availability. We have, however, been very careful to use only elements which are going to be easy to port to a PDA. We have already said that the class of PDAs we are aiming this system at have existing Linux ports and many of them support the X Window System as an option for display. Both SELinux and KDE are independent of the underlying hardware and simply work on top of Linux. Bluetooth is also hardware that is often found in modern PDAs.

6 Further Work

The prototype demonstrates that this system is possible, but there is a lot more work for us to do in this area. Our current prototype is on an Intel IA32 laptop rather than a PDA which is the ultimate aim of the project. Because of the components we have chosen it will be comparatively easy to port the current system, but just that does not fulfil our goals because we still cannot run the original PDA applications. Section 6.3 contains remarks on using graphical environments other than the X Window System.

6.1 Authentication Methods

We have in general so far only talked about "Authentication Methods" without going into more specific details. A traditional candidate for this is, as we have said, passwords. We are, however, trying to get away from using passwords as they are inconvenient for the user to remember and enter. Therefore, we continue to explore alternative types of credentials. Our system uses a proximity-based token but it would be simple to integrate a secure contact-token like the Dallas

iButton, or a more traditional smart-card. Most PDAs come with a compact-flash slot which can be fitted with a smart-card reader.

It is also possible to use some sort of biometric authentication. There are a number of companies selling mice and even PDAs with built in fingerprint readers. When the user tries to change to a restricted hat, the PDA could prompt them to touch the fingerprint reader. It would even be possible to have the system monitor the fingerprint reader and cause the PDA to automatically change to the session of a different hat.

This leads to an interesting new problem with the authentication. Ideally we would simplify the user interaction as much as possible, which is one reason for introducing tokens and biometrics. This reduces the steps required to change hat to two—select session and present token—or even to one step if the action of presenting the token causes the system to change hat. Automatic switching like this is good from the point of view of simplifying the interaction, but produces the problem of knowing which session to change to. In the system we have outlined here it is possible to have several hats which require the same credentials. Therefore, if those credentials are presented, there needs to be a method of selecting which session should be activated. Also, in general it is not desirable to authenticate by taking a password (or other credential) and then searching the possible users to see which (if any) the credential matches. We have taken the view here that presenting a token like this causes an event to occur which can be configured by the user. One of our future research areas will be into how best to handle this case.

6.2 Filesystem Protection

Just restricting logical access to data through the operating system is not enough to secure the data. Because the PDA is a small, portable device we are expecting the attacker to have physical access to it for at least a small amount of time. If the attacker only has a few minutes of physical access there is not a lot which he can do to bypass the operating system security, assuming the boot loader and BIOS are protected; but, if the PDA has been stolen, the attacker has a lot more time in which to work. In that case it is not hard to remove the permanent storage from the machine and connect it to another system to read the data directly.

Because of this we are planning to use an encrypting file system which can be integrated with the authentication system, similar to that in Corner and Noble [6]. In this case rather than just linking the encryption to a proximity token we will integrate it with the switching and authentication logic. A proximity token is just one option which may be used.

6.3 Decoupling From X

The system that we have demonstrated uses multiple X sessions, one for each hat. This is a bit wasteful on resources but, more importantly, there are several methods of changing session such as the Alt-Control-F key combination

which changes between virtual terminal and are handled at a much lower level than the one we are dealing with. These are hard to trap and mediate without modifying the underlying operating system. There are various approaches which can be taken to ensure that any non-active sessions are kept locked and require credentials to change into but these are all inherently fragile and susceptible to being bypassed. Directly preventing this requires a lot of low-level alteration of the Linux kernel. A much better system would be one where we control all the methods of changing between the various sessions. Modelling the sessions as different virtual desktops within the same X server would be a better approach, the window manager in this case would have to mediate the desktop-switching and check for credentials. However, if there are several applications in the same X server then they have a lot of control over each other. This is not compatible with our security policy which says that applications with different hats cannot access each other.

One possible solution to this was proposed by Kilpatrick et al [7]. This would be a system by which all the X protocol operations are mediated by SELinux-based permissions. This would allow several users to have windows in the same X server without also being able to access each other's windows. This would enable multiple users without requiring multiple X servers. Some work would have to be done on this, however, since the model that Kilpatrick is trying to implement is separating individual applications from each other—all of which may be seen by the user—and not coping with a user who may change role during the session.

The Kilpatrick approach is a very general solution with per-application granularity, which we don't necessarily need. There are some current research projects involving X proxy servers which could provide several hats and only allow one of them at a time to communicate with the X server. In both these cases we would delegate the access control to a trusted applet which would run all the time and switch between the hats.

Running in a single session would also help us to remove the dependency on X entirely. Most of the current Linux-based PDAs, while supporting X, are designed to use embedded environments such as Qtopia or GPE. Our goal is to support the original applications written for our chosen PDA and therefore we need to be able to support these environments.

Generic solutions which can use several environments need a server/client which can display on all of them. One such solution is Virtual Network Computing (VNC) [10]. VNC is a system which can run a graphical session unconnected to an actual display and has clients on most operating systems which can connect to the session and display it in a window. There are VNC clients for all of the main systems used on PDAs and therefore a system built around VNC sessions would be feasible. In this case we would have a small switching applet written for the specific platform which would switch between different VNC connections. VNC, however, introduces a lot of latency even on local connections and, because the protocol has to cope with the lowest common denominator, a lot of optimisations which can be done with the windowing system tend to be lost.

7 Conclusions

Our proof of concept prototype demonstrates that a multi-hat PDA can be implemented by combining existing software subsystems without the need for extensive modifications. Other subsystems to be integrated may include an encrypting file system and alternative mechanisms for handling the graphical sessions.

Further challenges will include reproducing the prototype's functionality on an actual PDA and being able to run native PDA applications under the multi-hat system. This porting activity may look conceptually straightforward but, as already happened during the development of the prototype, we expect that it will highlight new interesting research issues and open new avenues for investigation.

References

1. Ammar Alkassar, Christian Stüble and Ahmad-Reza Sadeghi. "Secure object identification—or: solving the Chess Grandmaster Problem". In "NSPW '03: Proceedings of the 2003 workshop on New security paradigms", pp. 77–85. ACM Press, 2003. ISBN 1-58113-880-6.
2. Anonymous. "Windows XP Technical Overview White Paper", May 2001. `http://www.microsoft.com/technet/prodtechnol/winxppro/evaluate/xptechov.mspx`.
3. Paul Barham, Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris, Alex Ho, Rolf Neugebauer, Ian Pratt and Andrew Warfield. "Xen and the art of virtualization". In "SOSP '03: Proceedings of the nineteenth ACM symposium on Operating systems principles", pp. 164–177. ACM Press, 2003. ISBN 1-58113-757-5. `http://www.cl.cam.ac.uk/netos/papers/2003-xensosp.pdf`.
4. Thomas Beth and Yvo Desmedt. "Identification Tokens—or: Solving the Chess Grandmaster Problem". In A. J. Menezes and S. A. Vanstone (eds.), "Advances in Cryptology—CRYPTO '90", vol. 537 of *Lecture Notes in Computer Science*, pp. 169–176. Springer-Verlag, 1991, 11–15 Aug 1990.
5. Stefan Brands and David Chaum. "Distance Bounding Protocols". In Tor Helleseth (ed.), "Advances in Cryptology—EUROCRYPT 93", vol. 765 of *Lecture Notes in Computer Science*, pp. 344–359. Springer-Verlag, 1994. ISSN 0302-9743. `http://link.springer-ny.com/link/service/series/0558/papers/0765/07650344.pdf`.
6. Mark D. Corner and Brian D. Noble. "Zero-Interaction Authentication". In "The Eighth ACM Conference on Mobile Computing and Networking", ACM Press, Sep 2002. `http://mobility.eecs.umich.edu/papers/mobicom02.pdf`.
7. Doug Kilpatrick, Wayne Salamon and Chris Vance. "Securing The X Window System With SELinux". Tech. Rep. 03-006, NAI Labs, Mar 2003. `http://www.nsa.gov/selinux/papers/X11_Study.pdf`.
8. Peter Loscocco and Stephen Smalley. "Integrating Flexible Support for Security Policies into the Linux Operating System". In "The 2001 USENIX Annual Technical Conference", USENIX Association, 2001. `http://www.nsa.gov/selinux/papers/freenix01-abs.cfm`.
9. Tristan Richardson, Frazer Bennett and Andy Hopper. "Teleporting in an X Window System Environment". *IEEE Personal Communications Magazine*, **1**(3):6–12, Nov 1994. `http://www.uk.research.att.com/pub/docs/att/tr.94.4.ps.Z`.

10. Tristan Richardson, Quentin Stafford-Fraser, Kenneth R. Wood and Andy Hopper. "Virtual Network Computing". *IEEE Internet Computing*, **2**(1):33–38, 1998. ISSN 1089-7801. http://www.uk.research.att.com/pub/docs/att/tr.98.1.pdf.
11. Naveen Sastry, Umesh Shankar and David Wagner. "Secure Verification of Location Claims". *CryptoBytes*, **7**(1):17–29, Spring 2004.
12. Frank Stajano. *Security for Ubiquitous Computing*. John Wiley and Sons, Feb 2002. ISBN 0-470-84493-0. http://www-lce.eng.cam.ac.uk/~fms27/secubicomp/.
13. Frank Stajano. "One user, many hats; and, sometimes, no hat—towards a secure yet usable PDA". In "The Twelfth International Workshop on Security Protocols", April 2004. http://www-lce.eng.cam.ac.uk/~fms27/papers/2004-stajano-hats.pdf. To appear.