Fifteen Years After TX: A Look Back at High Assurance Multi-Level Secure Windowing

Jeremy Epstein
webMethods, Inc.
jepstein@webMethods.com

Abstract[1]

Research in the late 1980s and early 1990s produced a prototype high assurance multi-level secure windowing system that allowed users to see information of multiple classifications on the same screen, performing cut & paste from low to high windows. This retrospective discusses the motivations for the project, reviews the architecture and implementation of the prototype, discusses developments in the intervening years, and concludes with lessons learned.

1. Introduction

The 1980s brought wide-spread availability of relatively inexpensive workstations[2] with bitmapped graphics displays, mostly running the UNIX[3] operating system and the MIT-developed X Window System (X). The expectation in the early 1990s was that this would be followed by widespread usage of multilevel secure (MLS) versions of these workstations, which would require MLS windowing systems.

Compartmented Mode Workstations (CMW[4]) showed that a windowing system can provide functionality and assurance which meet the class B1[5] requirements of the Trusted Computer System Evaluation Criteria [TCSEC]. For some applications, assurance beyond B1 is necessary. These applications might include system administration or security officer functions on a high assurance Multilevel Secure (MLS) system as well as general operations in non-benign environments (i.e., not a compartmented mode environment where all users are clearable for all information on the system). The guidelines in [Yellow85] require a B3[6] level of assurance for such applications.

Our goal was to design and build a prototype system, which we call Trusted X (TX). We did not attempt to build a system which is B3 certifiable *per se*, but rather focused our attention on those issues that must be resolved if a product development of TX targeted at B3 is to succeed. These include the development of a formal security policy, an architecture that satisfies the B3 structuring and minimization criteria and a way to secure the X protocol that avoids covert channels with minimal impact on X applications. This approach allows us to tackle the difficult, high risk technical problems first to demonstrate that the problem is solvable. The prototype effort ignored some of the B3 evaluation requirements that would be needed in a product, such as configuration management and extensive security testing and documentation.

TX is a trusted application, not a complete computing system. As such TX needs a B3 or better operating system as its host. We used the TMach 2.5 [TMach90] prototype from Trusted Information Systems as our base. For the prototyping effort, we ignored the issue of a B3 network.

The paper continues with a description of the assumptions and the environment of the time when the TX prototype was built, provides a brief introduction to the X architecture, then describes the TX security policy, system architecture, operation, and architectural limitations. We then describe how our system is similar to and different from other approaches to building MLS

---

[1] This paper is an updated and expanded version of *A High Assurance Window System Prototype,* Journal of Computer Security, Vol. 2, No 2&3, 1993. The work described in this paper was performed while the author was employed by TRW, Inc., and was sponsored by the Defense Advanced Research Projects Agency under Contract No. MDA 972-89-C0029. The updates and historical perspective presented in this paper were not sponsored by DARPA or performed while the author was affiliated with TRW.

[2] Low end Sun 3 workstations were priced around $10,000.

[3] Trademarks: UNIX and Motif are registered trademarks of The Open Group. X Window System is a trademark of the Massachusetts Institute of Technology.

[4] A CMW is a workstation which meets the requirements defined in [CMWREQS87] or [CMWEC91]. The CMW functionality requirements include an operating system and windowing system with TCSEC [TCSEC85] B1 features, plus some additional features from B2 and B3 such as access control lists and trusted path. For the purposes of this paper, we will consider only the windowing system aspect of the system, just as we will generally ignore the operating system which underlies our Trusted X prototype.

[5] TCSEC assurance levels included both specific feature requirements as well as assurances. B1 was the lowest level of assurance that also included MLS features. The assurances in B1 are roughly comparable to Common Criteria EAL4.

[6] The assurances in B3 are roughly comparable to Common Criteria EAL6.

windowing systems, including both those available at the time and research since then.

2. Background & Requirements

For most users then (and even today) who need to access data of multiple classifications, the common practice is to have multiple computer systems each operating at a single level[7], each with an independent keyboard, mouse, and display. This causes a number of problems: desk real estate, hardware cost, heat dissipation, noise, and usability (especially the inability to cut & paste text or graphics between machines). As a result, the mid 1980s saw the first experiments with MLS windowing.

The target audience for MLS windowing was analysts working with data of multiple classifications, typically to synthesize reports. While workstation users had graphic display screens, most usage was text. Because data of multiple classifications was expected to be on the screen, proper visible labeling was critical to avoid accidental misclassification, just as classified paper documents are labeled with page and paragraph markings. While graphics displays were physically large (19" was common), they had fewer pixels than today's laptops (1024 x 952 was a common resolution), so users frequently resized windows. Hence, the ability to dynamically change window size and position was considered a requirement.

It was widely accepted that X was a flexible platform for windowing, but one built without security as a primary driver. An extensive discussion of trust issues in X can be found in [Epstein91].

The state of the art for MLS windowing in 1989, when the TX project started, was CMWs. The facilities of the commercial CMWs from DEC, IBM, Sun, and others influenced the research – we felt that in order for a high assurance solution to be considered a reasonable alternative, it must offer comparable features. Thus, we quickly ruled out options that did not allow for windows of different classifications to be displayed on the screen at the same time. Such "screen switching" technology was (and is) cost effective, but prevents the user from seeing information from multiple classifications at the same time, hence eliminating much of the value of the windowing system.

Graphics hardware, as provided in workstations, was relatively primitive compared to today. In particular, most workstations had simple framebuffers[8] without hardware accelerators. Operating systems had not yet fully integrated the concept of graphics hardware, and "root" permission was typically required to access the framebuffer or the graphics hardware. Additionally, the graphics hardware was not virtualized by MLS operating systems (unlike memory and other resources), so it could not be shared among untrusted software operating at different levels without risk of information leakage.

MLS operating systems were being developed by several companies; our DARPA contract required use of TMach so we did not consider any alternatives. Building custom hardware was out of scope for the project.

In developing our architecture for TX we had several goals dictated by concerns for both B3 certifiable security and acceptable X functionality and performance. From the standpoint of secure functionality the primary requirement of TX is that it displays correctly labeled data to the user in an unspoofable fashion and allows the user to run untrusted applications without any potential for violation of the operating system's security policy. With the exception of the functionality required to assure correct labeling and trusted path interactions, we had no claims about the veracity of data displayed by clients. As we frequently described it, from a trust standpoint, we didn't care if circles were drawn as squares or *vice versa*.

The goal of the TX project, then, was to build a highly assurable MLS windowing system that would run on standard hardware, on the TMach operating system.

3. The X Window System

The following description is of the X Window System as it existed in the early 1990s. There have been numerous modifications since that time, including to improve security, but this section is largely as originally published, as it reflects the state of the world at the time of the prototype.

The X architecture is based on the client/server model of distributed computing. The *X server* manages the screen(s), keyboard, and pointing device (typically a) mouse), as well as graphical resources such as windows and properties on behalf of the *X clients*. The X server also manages global resources (such as the search path for fonts and the keyboard and pointer characteristics) that clients may change but not destroy.

X clients and the X server communicate via the X protocol [Protocol88]. Clients send *requests* to the server over a bi-directional communications channel using any

[7] For purposes of this discussion, we do not differentiate between information of different classifications (e.g., Secret vs. Unclassified) and information in different categories (e.g., NOFORN vs. RELNATO). This distinction is only relevant for TX in the context of cut and paste.

[8] A framebuffer is an area of memory shared between the CPU and the graphics card where each bit (in the case of the simplest black & white displays) represents the contents of a single pixel on the screen. (For more sophisticated graphics hardware, such as grayscale or color displays, one or more bytes is needed per pixel.) Modifying the display is as simple as modifying the bits in memory. The graphics card causes the display to render appropriate images based on the contents of each bit in the framebuffer. More modern graphics chips have framebuffers, but most drawing uses dedicated graphics hardware.

reliable byte-stream protocol (e.g., TCP/IP), and receive *events* and *responses*.

Protocol requests include administrative requests, requests to create and destroy resources, and drawing requests.

Applications can be written at a number of levels of abstraction, but all of these reduce to X protocol requests to the X server. Consequently, use of libraries and toolkits such as Motif [Motif90] are invisible to the server.

X has no concept of privilege, and a minimal notion of protection. Protection is provided at connection time only. The X server maintains a host access list which identifies those computers from which connections will be accepted. In addition, an optional authentication mechanism allows the server to demand some form of authentication from the client (e.g., an MIT magic cookie or a Kerberos [Kerberos88] authentication ticket). Once a client has connected to the server, it may perform any request, including a request to turn off authentication for clients that attempt to connect in the future. Clients can also directly impact other clients (e.g., by killing them), although such behavior is considered undesirable (see [ICCCM89]).

Management of windows on the screen is performed by a *window manager*. There are many existing window managers, each of which provides a different look-and-feel. Because there is no notion of privilege in X, the window manager is simply another client. The conventions described in the X *Inter-Client Communication Conventions Manual* [ICCCM89] are used to define an environment where "well-behaved" (ICCCM compliant) clients can interact cooperatively.

4. Concepts & Architecture

This section presents the TX architecture. We first provide a high level view, introducing each of the trusted and untrusted components of the system. Next, we describe the size and internal structure of each component. The remainder of the section describes the facilities provided by each component and the component interactions.

The target hardware platform for TX was the Sun 3 line of workstations, because those were the primary TMach platform. The Sun 3/50 and 3/150 were both based on the Motorola 68020 processor, operating at about 16MHz[9] (the 3/50 was slightly slower), and were typically equipped with 4MB of memory. Later versions of TMach and TX ran on Gateway desktops with 33MHz Intel 486 processors. The hardware limitations compared to today's systems were not foremost in our minds, but impacted the resulting system.

[9] Or about 1000 times slower than the typical laptop computer of 2006.

4.1. Overlapping Windows & Visible Labeling

The goals of the visible labeling policy were that the labels must be easily readable and clearly associated with the window contents. We also wanted the visible labeling mechanism to be "spoof free" – it should not be possible for a malicious client to draw something that looked to the user like a classification label. Additionally, compatibility with the core X look and feel was a must. Because there is no universally acceptable solution, we chose a policy that balances utility with assurance.

In 2006, the notion of overlapping windows (where one window partially obscures another) is obvious. In 1990, we considered whether they were mandatory, and whether a windowing system that required non-overlapping windows was acceptable. Additionally, we considered whether a system was acceptable that provided a simple switch to move between screens each at a single level, much as KVMs allow switching between computers. In order to meet our goal of compatibility with the core X look and feel, we needed to allow overlapping windows, and a simple screen switch was insufficient.[10]

Figure 1 shows window layout in a system without restrictions (so each window is individually labeled), with region tiling (windows of each classification in a reserved area of the screen), with window tiling (non-overlapping individually labeled windows), and screen switching.

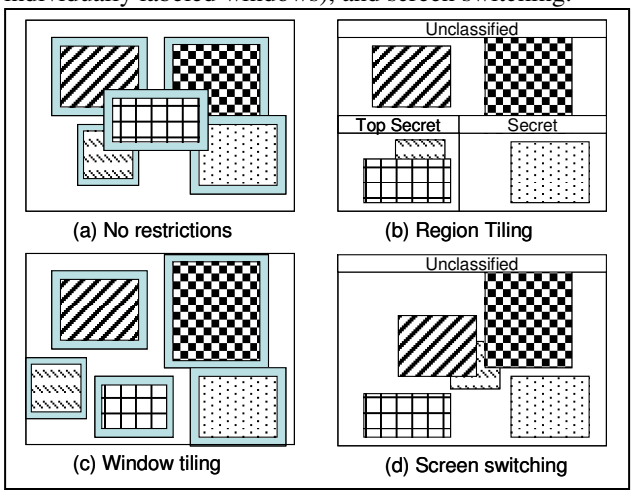

Figure 1. Window Layout Alternatives.

Once we determined the need for overlapping windows of multiple classifications, we then considered the *visible labeling* requirements[11]. Visible labeling marks windows on the display so the user can clearly determine the

[10] Section 7.3 discusses a windowing system without overlapping windows.

[11] Sections 7.2 and 8.1 describe systems with overlapping windows, but without visible labeling.

maximum potential classification of data in a window[12]. As overlapping windows may make it difficult to determine the classification of data in a window, we determined that a "point to identify" capability was required, so the user could know the classification of any particular area on the screen. We considered whether the classification for a window should be displayed on the top of each window, or around all four sides (and what to do with non-rectangular windows). We also prototyped what happens if windows are overlapped in such a way that portions of the data are not continuous to any label; we called these "orphan" areas. Our conclusion was that labels on all four sides of a window were desirable, and that non-rectangular windows would not be supported. Figure 2 shows the appearance of labels on the screen. Further details on these studies can be found in [Epstein90] and [Epstein93a].

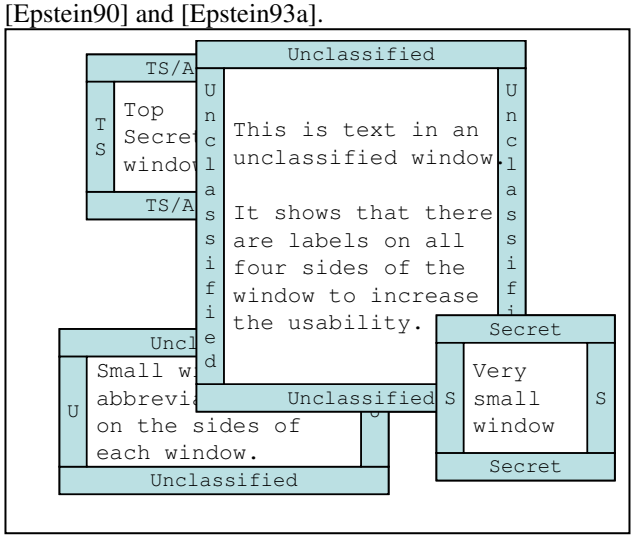

Figure 2. Overlapping windows and labels.

4.2. System Overview

When we initially approached the concept of a trusted X Window system, we considered the possibility of securing the X protocol and the server that interprets it. After a substantial effort that included formal modeling and covert channel analysis of some of the X protocol requests as well as substantial study of several X server implementations, we reached the conclusion that a high assurance MLS X server was beyond our resources and that problems inherent in the X protocol might preclude such an implementation even with unlimited resources. The "server per level" or "polyinstantiated subject" idea was an obvious alternative, which we adopted.

The TX TCB is composed of four sets of trusted processes that mediate client access to TX and control multilevel interactions among clients. The modular structure of the TCB and the specialized functionality assigned to each module simplifies analysis of the modules and strengthens the assurance arguments.

- **Input control and trusted path functions**. This functionality is provided by three trusted modules: the Input Manager (IM) distributes keyboard and pointer inputs and coordinates activities of the trusted Mini Server (MS) and the Trusted SHell (TSH) which provides the user with operational control of TX.
- **Display management and window labeling functionality.** The Display Manager (DM) composes the physical display from the virtual framebuffers of the single level servers and adds the necessary visible labels to top level windows.
- **Initiation of the task set for each sensitivity level**. The single level servers and their associated supporting clients are started by two trusted modules; the Server Initiator (SIT) for starting the servers and the Client Initiator (CIT) for the clients.
- **Multi-level information sharing functionality**. The Property Escalator (PE) supports cut & paste operations among single level clients.

Each of the trusted subjects has its own internal security policy which must be shown to enforce the underlying information flow policy inherent in the TMach access control policy. In particular, the security policies of MS and TSH are minimal, because they have no interfaces to non-TCB software. DM, IM, SIT, and CIT all implement a policy of read-equals and write-equals as defined by Bell-LaPadula [Bell75]: a given subject is only provided information about subjects and objects at its own sensitivity level. PE's policy is read-down, write-equals to allow pasting of lower level information at a higher sensitivity level.

Each instance of an untrusted X server (SLS) is supported by two additional untrusted clients: the Selection Emulator (SE) and the Window Manager (WM). An untrusted X client interacting with other untrusted X clients at its level sees the full X functionality and is unaware of the existence of clients at other levels.

There is no notion of a security policy *other* than the MLS policy – within a single instance of the X server, the policy is the standard X policy (i.e., no limits).

Figure 3 shows the TX subjects and their interactions.

[12] The actual information in the window may be *less* classified than the label indicates, but cannot be *more* classified. For example, opening your grocery list in a window marked Secret does not automatically make the list Secret. Information Labels (ILs) in CMWs are an attempt to indicate to the user the *actual* classification of information, rather than the *maximum* classification.

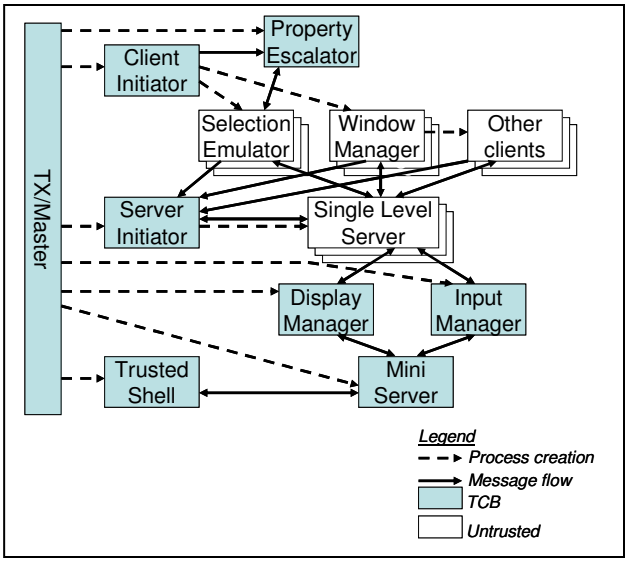

Figure 3. TX Architecture.

To allow the individual X servers to share the screen, each X server maintains a *virtual framebuffer*, which is a memory area set up exactly as a (hardware) framebuffer. Thus, the polyinstantiated X server draws in its virtual framebuffer in exactly the same way as it would use a hardware framebuffer, thus minimizing changes to the untrusted software.

4.3. TCB Minimization and Modularization

One of the key difficulties in building a high assurance X is to minimize and modularize the TCB to conform to the architectural requirements of the TCSEC and general principles of minimization. While the MIT provided code is generally well-written and well-structured, design documents do not exist, and it is far too large (roughly a 100,000 lines of code in the server alone[13]) to meet high assurance goals.

By contrast, the TX prototype contains about 30K LOCC (or 8K statements) in its TCB. This compares favorably with other high assurance TCBs: the SCOMP TCB with about 21K statements[14]; the XTS 200 TCB (SCOMP's successor) with about 20K statements[15]; and the TMach TCB with about 100K statements[16]. Comparable numbers for the GEMSOS TCB were unavailable[17].

The individual trusted subjects range from a few hundred to a few thousand LOCC. The encapsulation and limited roles[18] make their evaluation less difficult than comparable portions of an operating system kernel.

Subject	Trusted?	LOCC	Statements
M	Yes	400	150
IM	Yes	1,600	400
DM	Yes	5,500	1,600
MS	Yes	2,300	450
TSH	Yes	2,700	1,000
SIT	Yes	600	150
CIT	Yes	200	50
PE	Yes	400	100
SLS	No	103,000	39,700
SE	No	1,500	500
WM	No	58,000	11,000

Figure 4. Approx. Lines of C Code for TX Subjects.

The total LOCC for trusted components (including the library) is approximately 18K (5K statements), while the untrusted portion of the system contains approximately 414K LOCC (or 130K statements)[19].

The following sections describe each of the components in turn.

4.4. Master

The Master (M) synchronizes the initialization of the TCB components. Once the system reaches its initial configuration, M does not take an active part in its operation.

M does not communicate with untrusted subjects.

4.5. Input Manager

The Input Manager (IM) routes keyboard and pointer (typically a mouse) input, checks for the secure attention key (SAK) sequence, and for inactivity timeouts. It can also provide information about the input hardware configuration, information that is bound at TX initialization time and not subsequently changed. At any given time, there is at most one "current" TX server which may be a SLS, the MS, or *none*. In normal operation, the current server receives all input. In X, these are low level operations representing key press and release events, pointer motions, etc. rather than ASCII characters and pointer coordinates.

[13] By comparison, Microsoft Windows XP is about 40 *million* lines of code, and current versions of Linux are millions of lines of code. Not all of this must be trusted, but the relative size is instructive.
[14] Conversation with Les Fraim, MITRE.
[15] Conversation with Chuck Bonneau, HFSI.
[16] A preliminary estimate by Homayoon Tajalli, TIS.
[17] Conversation with Don Brinkley, GEMINI.

[18] The TX trusted subjects are trusted in a limited sense. Each has limited interactions with several untrusted clients at different sensitivity levels and must be shown not to pass information among them in unacceptable ways. TX trusted subjects *do not* have unlimited access to TMach resources and cannot, for example open arbitrary files, manipulate the memory management, etc.
[19] Including 252K LOCC (78K statements) in common libraries shared by all clients.

The SAK sequence is not necessarily a single key press event, but may represent a combination of events that are watched for by the IM.

If the SAK is detected, the current server is set to be MS, which then receives all input until it directs otherwise. This is the trusted path.

If IM does not detect any keyboard or pointer activity for a (configurable) period of time, it sets the current server to *none* (even if the server is MS). In this mode, all inputs except the SAK are discarded. This allows implementation of a screen lock facility (described in section 5.6). Once locked, the screen can only be unlocked through the trusted path. The current server is also set to *none* if the receiving server terminates. Otherwise, IM changes its current server only when instructed to do so by MS.

IM notifies a SLS when it becomes the current server and when it loses this status. This notification allows the SLS to inform the window manager (or any other client), which might take actions such as raising all windows to the top on activation, or clearing its focus indicator on deactivation. This is an operational consideration, and we believe that no covert channel is introduced because the notification is the direct result of user action.

IM uses Mach threads (lightweight processes) to achieve parallelism. One thread constantly monitors the keyboard and pointer (sending the data to the current server), a second thread responds to requests for hardware configuration information, and a third thread watches for inactivity timeouts.

Because TX provides a graphical interface, it might appear logical to use a pointer click on an icon (rather than a typed sequence of keys) for the SAK. To implement such a facility would require the inclusion of pointer handling code as well as substantial portions of the graphics display code in the TCB; that is something that we wish to avoid. In the present architecture, the physical cursor position on the screen is not known by IM since each SLS is free to "warp" the cursor position[20] as it sees fit and IM would be unable to determine when the cursor is pointing to the icon.

4.6. Display Manager

The Display Manager (DM) controls the physical display, composing the displayed image from the pieces that are provided by the individual SLSs and the MS in their virtual framebuffers. It also is responsible for ensuring that the individual windows on the display have proper visual labels. Figure 5 shows this image merging. Windows may overlap each other regardless of level; in this example, unclassified window C overlaps top secret window H, but H overlaps unclassified window D.

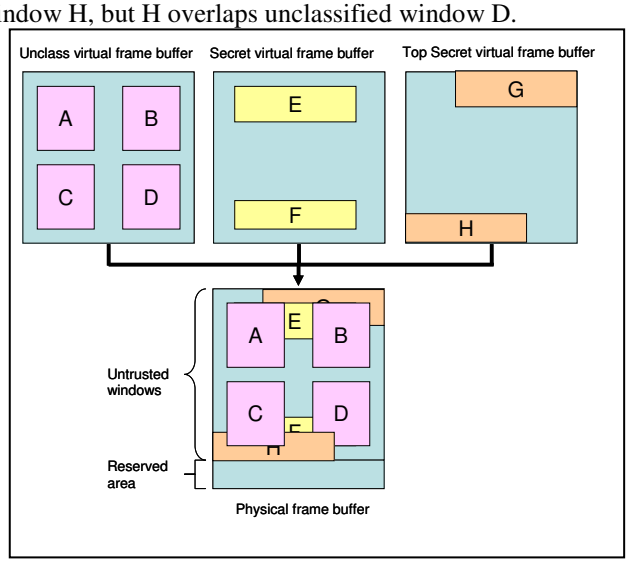

Figure 5. Display Composition from Virtual Framebuffers.

DM also provides a service to draw "helping lines" to aid window managers (described below).

DM maintains general hardware and configuration information that can be queried by any SLS. As with the IM, this information is bound at TX initialization time and cannot be changed thereafter.

As is the case with the IM, the DM supports a notion of a current server which is *none*, a SLS or MS. The current server is set by the MS. The display is divided into two regions, a small region for the exclusive use of MS, and the large majority for use by all servers including MS. The MS reserved area is used for displaying status information such as the visual labels for the current operating level and the high water mark level[21] and for interactions with TSH.

Within DM, certain artifacts of the display state are polyinstantiated and maintained on a per level basis. These include a list of installed colormaps, the position, image, and color of the cursor, the positions, sizes, contents, and stacking order of top level windows, and size and location of the "helping lines."

Each SLS has two TMach ports to DM. The "nonhold"' port can be used by the SLS to update the contents of currently mapped top level windows, and to query the hardware configuration (which is read-only). To update window contents, the SLS provides a replacement virtual framebuffer including the new window image, which DM clips using the boundaries of the given window and any overlapping windows. The "hold" port can be used to map and unmap windows,

[20] Cursor warping is typically used when a dialog window is brought up, and the cursor image is automatically moved to the default choice.

[21] The "high water mark" is the upper bound of all windows on the screen, including both classification levels and categories.

change their stacking order, update the cursor position and image, draw "helping lines" and perform various other administrative functions. Internally, DM identifies windows by both their SLS provided window ID and the identity of the SLS which owns them. Each SLS can only refer to windows which it owns.

DM processes requests from all nonhold ports as they are received, but only processes the hold port belonging to the current server. The distinction between the hold and nonhold ports is to prevent another SLS from interfering with the current server by mapping windows arbitrarily. As is the case with input notification, this is an issue of functionality rather than security and is a partial solution to potential problems that could arise because single level window managers are not aware of windows at other levels and cannot take action to avoid obscuring them.

MS can update the contents of the "reserved area" by providing a replacement virtual framebuffer with the updated image, just as SLSs provide replacement images for windows. SLSs cannot draw in or examine the reserved area and are not aware that it exists[22].

When the current server changes, DM installs the colormap(s) belonging to it, and displays its last cursor image in the proper position. Polyinstantiation of cursor position and colormap values allows complete flexibility in colormap and cursor handling without the covert channels which would normally exist. DM also brings up the last set of helping lines (if any) provided by the server. DM visibly labels each mapped window using a visual representation of the sensitivity level of the SLS which owns the window. The labels appear on all four sides of the window outside any window manager decorations. They are controlled entirely by the DM and are not accessible to any SLS. The visible labels are the only graphics performed by DM other than copying virtual framebuffer contents to the real framebuffer and drawing helping lines.

Many window managers assist users in placing windows on the screen by drawing rectangular boxes called "helping lines" directly on the screen background[23]. Unfortunately, there is no way for the X server to distinguish between a window manager drawing helping lines and a client performing arbitrary drawing on the screen background. We extended the X protocol to provide an explicit request for the drawing of helping lines. A SLS passes this request to DM through its hold port. DM only allows one set of helping lines on the screen at any time, namely those belonging to the current server. Each request by a server to draw helping lines cancels any previous request it has made. We do not consider helping lines to be information requiring a visible label. Note that helping lines cannot be detected by clients or SLSs, so no covert channel is introduced.

DM also uses Mach threads for parallelism. One thread processes requests from MS, plus requests for connections from new SLSs. A second thread handles all nonhold ports from all SLSs, and a third thread handles the hold port from the currently selected SLS. The first thread has highest priority, so MS requests will generally be processed first.

The DM is the largest and most complex component of the TX TCB. The majority of the code in this component is to handle determination and copying of overlapping windows. This portion could be implemented in a high assurance hardware facility, which could increase both performance and assurance [Epstein96].

4.7. Mini Server

The Mini Server (MS) coordinates the activities of IM, DM, SIT, and TSH. It also provides a very limited set of graphics facilities for use in drawing the screen during trusted path interactions and for maintaining the visible labels for the current operating level and high water mark in the reserved area of the screen.

Based on its interactions with the TSH, MS directs the IM and DM to set the current server and update the current operating level, accompanying this with an appropriate image (including labels) for the reserved area of the screen. It also directs the SIT to start new single level servers as needed.

X allows arbitrary fonts, and has primitives for drawing lines, rectangles, polygons, circles, and other shapes using a variety of different styles (e.g., line width, mitering algorithms). By contrast, MS provides primitives for TSH to clear an area, draw vertical and horizontal lines, and draw text using a single fixed width font. These restrictions allow the size of the MS drawing code to be several orders of magnitude smaller than that of an X server. MS draws in a virtual framebuffer, which is sent to DM for actual display either in the reserved area or on the general screen. MS also handles input in a very restricted fashion.

Cursor movement is tracked internally, and the cursor position is passed to TSH only when a pointer button is pressed or released. Key press and release events are converted into text strings for transmission to the TSH.

4.8. Trusted Shell

The Trusted Shell (TSH) provides an interface through which the user can perform certain administrative and

[22] For example, as part of startup the SLS is told that the display is 1024x700 pixels rather than 1024x768, and any attempt to write to pixel rows 701-768 is ignored as off the virtual screen.

[23] Helping lines are a dashed outline of a window, used when moving or resizing windows on the screen. They were used to avoid the computational cost of moving the actual image around during the resize or move operation.

security functions necessary for the operation of TX. These functions are:

1. starting a new SLS,
2. selecting the current operating level,
3. displaying the level of a window on the screen,
4. locking the screen,
5. unlocking the screen after either a manual or automatic lock,
6. changing the user's password[24], and
7. exiting TX.

TSH uses the drawing primitives provided by MS. Its user interface is based on a simple menu displayed in the reserved area of the screen. Providing this functionality via interactions with the TX display and input devices avoids the need for a separate trusted interaction facility; however, if one wished, all but the third item in the list above could be accomplished through interactions with a much less complicated device than the bitmapped X display. We believe that there is a firm requirement for obtaining the sensitivity level of a displayed window through trusted path interactions, and see no way to avoid at least some trusted display functionality.

4.9. Server Initiator/Terminator

The Server Initiator / Terminator (SIT) performs two main tasks: SLS creation and connecting clients to the appropriate SLS. SIT starts a SLS at the request of MS or when a client requests a connection to TX at a sensitivity level for which no SLS exists. This avoids a need for preconfiguration. When SIT creates a new SLS, it also requests CIT to create a new window manager and selection emulator at the sensitivity level of the SLS.

SIT also connects untrusted clients to the correct SLS. It would be preferable for each SLS to make itself known to the TMach name server and for clients to connect to the appropriate SLS through TMach. Since TMach does not provide polyinstantiation of its name space, each SLS would have to pick a unique name, and X clients would need to know how the unique names were generated (which would also provide a potential covert channel). To avoid this, SIT registers as the point of contact for all TX connection requests. When a client asks to connect, SIT forwards the request to the appropriate SLS, starting a new one if necessary. Once the connection is established between the client and the SLS, SIT is no longer involved and TX clients converse directly with their SLS.

4.10. Client Initiator/Terminator

The Client Initiator / Terminator (CIT) starts window managers and selection emulators as requested by SIT. CIT and SIT could be folded into one task. The reason for their separation is to maintain the distinction between clients (managed by CIT) and servers (managed by SIT). This distinction allows CIT and clients to run on a different machine from the server tasks, which are usually on the same machine as the physical hardware.

4.11. Property Escalator

TX supports multilevel cut and paste in accordance with the ICCCM selection based protocol. The operation requires interactions between untrusted clients called Selection Emulators (see 4.13) and the Property Escalator (PE), which can be seen as a primitive MLS database allowing read-down. The Property Escalator (PE) provides primitives to SEs to write data (write-equals) and to read data provided by other SEs (read-down). Read requests always provide the most recent request which meets the format criteria. The SE which wrote the data is not informed that the data has been read, nor can it discern that a read took place. This avoids the covert channel inherent in the handshakes that are a part of the X protocol operations used in the ICCCM protocol. This assurance comes at the price of support for limited conversion formats and multiple conversions. We feel that this is a reasonable price to pay for a high assurance system, but we realize that there is only limited experience in this area.

Because the PE is the only trusted subject whose purpose is to support interclient communications between sensitivity levels, its internal security policy deserves discussion. The subjects of the PE security policy are the untrusted SE clients with which it communicates. The objects of the PE security policy are the cuttings. The access modes are *cut* and *paste*. The objects are polyinstantiated at all sensitivity levels at which the PE may operate. Under the policy, all SE's have cut access to the selections at their sensitivity level. This gives them the ability to ask the PE to replace the previous value of the cutting with a new one. In addition, SEs have paste access to those cuttings whose sensitivity level they dominate. Under the policy, a paster can have paste access to numerous cuttings. From a security standpoint, it does not matter which one is pasted; from a functionality standpoint, it does and we choose the most recent cutting as satisfying the usual model of cut and paste interactions. This policy provides the information flow protection desired and is consistent with the TMach access control policy.

A side effect of this policy is that an instance of SE can perform a Denial of Service (DoS) attack against other instances of SE by periodically sending data to be pasted (perhaps once a second). This would prevent the user from pasting any information other than that provided by the malicious SE, unless they are operating at a level such

[24] This function is provided so the user need not exit TX and use the TMach TSH. It adds about 100 LOCC (50 statements) to the TCB.

that they cannot read the data from the malicious SE instance[25].

4.12. Single Level Server

The Single Level Servers (SLS) are modified versions of the MIT X server. Whenever possible, we have avoided modifying the MIT code to avoid introducing bugs or incompatibilities between TX and X. Our changes involve device handling code which initializes input and output devices, input and output specific code to replace device specific code, replacing UNIX operating system dependent code with TMach code (for receiving connections from clients and reading andwriting X protocol to and from clients using TMach ports rather than UNIX sockets), and disabling requests to change global settings such as the keyboard mapping.

Because the SLS does not have control of the physical input and output devices, device initialization code is not necessary and these functions have been transferred to the IM and DM. The SLS receives its input from IM via a TMach port. This change has minimal impact on the X server. For output, the SLS allocates a virtual framebuffer of the same size as the physical framebuffer (without the reserved area, which is unknown to the SLS). All drawing is performed using this virtual framebuffer, which is then sent to DM as TMach messages. Mach and TMach send messages copy-on-write, so the virtual framebuffer is not actually copied. Rather, the SLS and DM share the same framebuffer, except when it is being updated by the SLS. This minimizes the additional memory required, and the overhead of copying.

Window mapping and unmapping requests result in messages to DM. Rather than adding "move window" and "resize window" primitives to DM, the SLS unmaps a window and remaps it in its new size and position. Cursor and colormap modifications are similarly modified to send messages to DM. Note that the SLS is unaware of labels placed on windows by DM.

All interpretation of input (i.e., determining which client(s) receive the keystrokes and/or pointer events) is performed by the SLS.

Each SLS *could* map the keyboard differently (e.g., the *secret* SLS could use a QWERTY keyboard mapping, while the *confidential* SLS could use a Dvorak keyboard mapping). This is a side-effect of the polyinstantiation of the SLSs that may appear at first to be a flaw in the system: why would one want to have different keyboard mappings for different sensitivity levels? Consider, however, a system which provides specialized function keys, some of which may only apply to interactions with data at one sensitivity level. Under TX, these keys would automatically be mapped appropriately to the sensitivity level of the interaction window because each SLS performs its own mapping from the physical keys to the logical values. We feel that any benefit of allowing this flexibility are outweighed by potential problems for the user. Thus, our SLS ignores requests to remap the keyboard (along with certain other administrative requests) not as part of our security policy, but simply to avoid confusion.

4.13. Selection Emulator

Cut and paste in X is performed according to the selection based conventions described in the ICCCM [ICCCM89]. Summarized, the cutting client announces (by asserting ownership of an X entity called a *selection*) that it has data available and provides (upon request) a list of formats in which it can present the data. The pasting client requests the data in one or more of the advertised formats which the cutting client then makes available. This mechanism allows the cutting and pasting clients to negotiate an acceptable format (e.g., text, formatted graphics, Postscript). Because it uses "lazy" evaluation, this mechanism avoids using CPU cycles for conversion until the data (and format) is to be pasted.

The disadvantage to this mechanism is that the communication between the cutting and pasting clients is bidirectional. Because the bidirectional communication is required by the protocol, we hesitate to call it a covert channel. In any event the potential capacity of the channel is so large that it cannot be constrained without severely limiting functionality.

In our approach, an untrusted client called the Selection Emulator (SE) listens for announcements by cutting clients. When one is received, it immediately asks for the data in *all* of the advertised formats. SE then passes the data to the Property Escalator (PE) which retains a database of available data. SE also listens for requests by pasting clients. When one is received, it queries PE for the most recently cut data available in the specified format. PE passes the data to SE, which then makes it available to the client. Thus, an SE is paired with each SLS. A cut and paste operation involves two SEs: one at the sensitivity level of the cutting client, and one at the sensitivity level of the pasting client.

Figure 6 shows the interaction of the different components in a cut and paste cycle. The slightly simplified[26] steps are as follows:

0. User requests a cut operation via keyboard or mouse actions.

[25] For example, if the malicious SE is running at Secret/A/, then any paste operations at Top Secret/A/, Top Secret/A,B/, or Secret/A,B/ will use the data from the malicious instance. Paste operations at Unclassified or Secret/B/ or Top Secret/B/ will not see the data from the malicious SE, since they cannot read down to Secret/A/.

[26] Omitted steps include negotiation at both the high and low level as to available and preferred formats for the cut/paste operation.

1. Cutting client informs SLS it has something available by asserting selection ownership.
2. SLS notifies SE (which had previously had selection ownership) that a client has cut available.
3. SE asks SLS to get cut data in all possible formats.
4. SLS passes SE request to cutting client.
5. Cutting client passes data in all formats to SLS.
6. SLS passes data in all formats to SE.
7. SE passes data in all formats to PE.
8. User switches to the higher level and requests a paste operation via keyboard or mouse actions.
9. Pasting client asks for data to be pasted.
10. SLS notifies SE that a request for a paste has occurred.
11. SE asks for most recent data in all formats from PE.
12. PE sends cut data to SE.
13. SE sends data to SLS.
14. SLS sends data to Pasting client, which performs the paste operation and updates the screen (if appropriate).

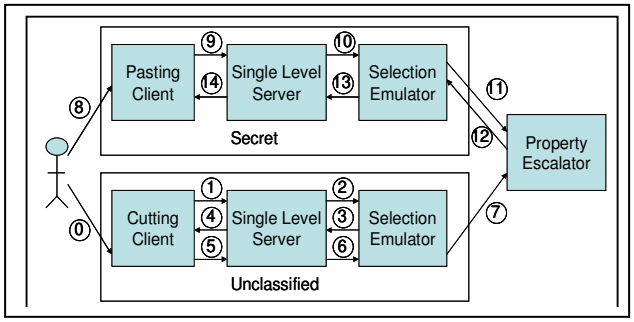

Figure 6. Cut and Paste Sequence.

Through this mechanism, compatibility with the ICCCM is maintained without covert channels or loss of flexibility. The price paid is lower performance, as we use "energetic" evaluation (i.e., the opposite of "lazy" evaluation).

4.14. Window Managers

Any X window manager can be modified to be a Window Manager (WM). The only change required is in the drawing of "helping lines" to assist the user in placing windows on the screen. For the reasons discussed in section 4.6, this must be done via the X protocol helping-lines extension. Each WM manages windows only at its own sensitivity level. Thus, the *secret* window manager could not be used to move a *confidential* window, as the *secret* WM would not have any knowledge of the *confidential* window. An interesting side effect of this architecture is that different window managers can be used at different sensitivity levels (i.e., run Motif at *secret* and OPEN LOOK at *confidential*), although the probable user confusion makes this undesirable[27]. The prototype supports three window managers: mwm (the Motif window manager), olwm (the OPEN LOOK window manager), and twm (the MIT provided window manager).

5. TX Operation

This section describes some of the more interesting aspects of the TX operations. The discussions that follow emphasize the advantages that both the TMach base and the architectural structure of the system provide in meeting trust and assurance requirements.

A fundamental aspect of TX operation is that there is at most one current server. In normal operation, all input is routed to the current server, a SLS at the current operating level, and certain output operations can only be performed by the current SLS. For example, if the current sensitivity level is *secret* then all input is routed to the *secret* SLS, and only the *secret* SLS can map and unmap windows from the screen. Each SLS can update windows that it has mapped, no matter what the currently selected server is. The operating level will not refer to a SLS if the active SLS dies, the system is locked, or the user has invoked the trusted path. In these cases, no input crosses the TCB boundary and we claim that the notion of an operating level is inappropriate.

5.1. TX Startup

TX is started from the TMach Trusted Shell. M starts all of the trusted tasks (IM, DM, CIT, SIT, PE, MS, and TSH). Initially, IM and DM have no current level because they have no SLS with which to communicate. Any input is discarded by IM, and DM has no hold or nonhold ports to read from. When it starts, TSH makes drawing requests to MS to display the current operating level (which is *none*) in the reserved area.

When initialization is complete, the internal communications paths shown in Figure 3 have been established by giving the trusted subjects the appropriate rights to TMach ports. Because further propagation of these rights can be controlled, this pattern of communication cannot be changed, even if one of the subjects wished to do so. The SIT registers its connection request port with the TMach name server which then

[27] At the time of this research, the "UNIX wars" were in full swing, with one camp (primarily HP, IBM, and DEC) supporting the Open Software Foundation (OSF) which put out Motif (among many other technologies), and the second camp (primarily AT&T and Sun) supporting their unified effort which included Open Look. The ability to support both options was considered a significant advantage over CMWs, which heavily relied on the window managers to provide visible window labeling and to enforce other MLS policies, and hence were tied to one of the two camps.

mediates requests for connections to TX in accordance with the TMach MAC and DAC policies. At this point, clients wishing to connect to TX must pass their requests through the TMach name server to SIT.

5.2. Single Level Server Startup

SLSs are normally started by the user through the TSH. They are also automatically started if a client sends a connection request to SIT and there is no SLS at the sensitivity level of the client. To start a single level server, SIT creates a new untrusted SLS task at the requested sensitivity level and notifies CIT that a new SLS has been created. CIT creates new WM and SE clients at the same sensitivity level as the new SLS. The WM and SE clients send requests to SIT to be connected to the new SLS. SIT holds the requests.

The new SLS sends messages to IM and DM asking for connections and for the hardware configuration data. IM replies to the SLS by providing information about the keyboard and pointer and DM replies to the SLS by providing information about the display (e.g., the screen size, and whether it is color or black and white). This information is considered to be system low allowing the request to be honored identically at all sensitivity levels. IM and DM inform MS that the new SLS has connected. MS in turn informs TSH.

The new SLS sends a message to SIT that it is ready to accept connections. SIT forwards the client connection requests it is holding to the new SLS giving it the send rights to the reply port provided by the client. The new SLS replies directly to the clients (e.g., WM and SE) and they are then connected.

At this point, the SIT is effectively out of the loop. It does not retain the ability to communicate with the untrusted clients of the SLS and is not in a position to compromise them. It has send rights to the SLS connection request port, but that is all.

5.3. Client Connection

Clients are typically started by the window manager or an existing client. The client sends a connection request to SIT having obtained send rights to its request port from the TMach name server. If a SLS at the client's sensitivity level is not already running, one is started as described in the previous section. In this case, the client's request is passed to the new SLS along with the requests from the SE and WM. Otherwise, SIT forwards the request to the appropriate SLS and the SLS responds directly to the client. As noted above, the SIT retains no connection to the client.

5.4. Normal Operation

Once clients are connected to the SLS, the system is in normal operation. Figure 8 shows the connections used in this state, while Figure 7 shows the appearance of the screen. This section describes a few common operations.

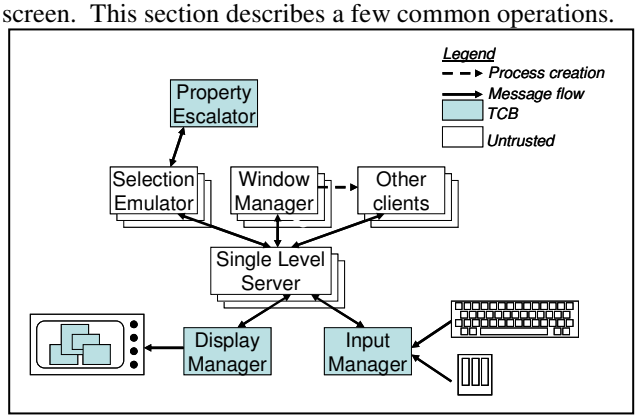

Figure 8. TX Normal Operational State.

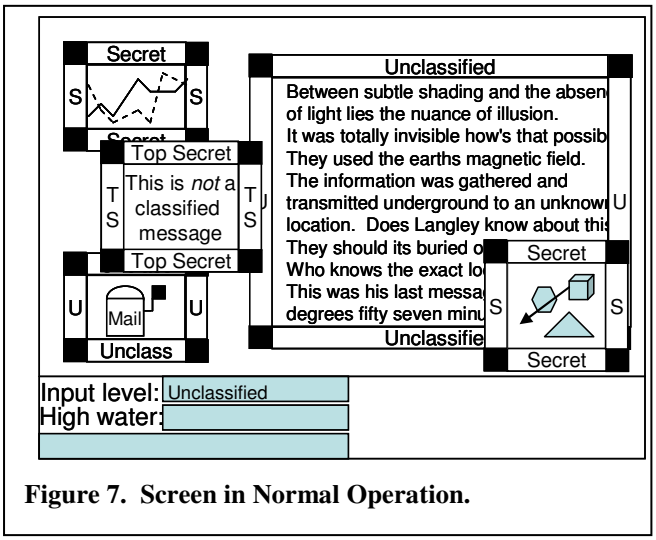

Figure 7. Screen in Normal Operation.

The DM operates on top level windows (children of root in the X vernacular). Within such a window, all operations are performed by the SLS which is required to make its entire contents (bitmap) available to the DM unless it is obscured by another top level window of the same SLS. When a client requests mapping of a top level window, the SLS sends a message to DM using its "hold" port. If the SLS is the current server, DM will immediately process the request and send back an acknowledgment to the SLS.

If the SLS is not the current server, then the message will remain queued until the user selects its sensitivity level, at which point it will be processed. Once DM acknowledges the request, the SLS may provide contents for the window in the form of a bitmap.

When the DM maps a window for a SLS, it creates a suitable labeled border around the window. This happens for all windows including those claimed to be transient by the client.

Moving, resizing, and unmapping top level windows is handled similarly. Note that mapping, unmapping, moving, and resizing of non-top level windows is entirely internal to the SLS, except that this may cause the contents of a top level window to change.

Reparenting a window (an operation performed by window managers to add "decorations" to the window) is simply unmapping the old top level window, followed by mapping the new top level window in its place.

When a client draws in a window, the SLS performs the drawing in its virtual framebuffer. When the drawing is complete, the SLS sends its virtual framebuffer to DM along with a list of windows and areas changed. DM receives the contents update request on its nonhold port from the SLS. After clipping the new window contents relative to other windows on the screen DM updates the visual display. The clipping confines the updating to the unobscured interiors of windows belonging to the SLS in question. This prevents the untrusted SLS from being able to affect the display outside of areas surrounded by proper visible labels.

When the user moves the pointer, clicks buttons, or types on the keyboard during normal operation, IM sends the input to the SLS that is the current server. The SLS performs the ordinary X rules for routing input to its clients and is oblivious to other SLSs which might exist.

5.5. Trusted Path

Trusted path operations are initiated by the user when the secure attention key sequence is invoked. If the current server is a SLS, then IM notifies it that it is now deactivated and notifies MS that trusted path was invoked. IM begins sending input to MS. MS notifies DM and TSH that trusted path has been invoked. DM sets its current server to *none*, blocking processing of "hold" port requests. TSH sends commands to MS to draw the menu of commands, and to change the current operating label displayed in the reserved area to "Trusted Computing Base". MS performs the drawing operations in its virtual framebuffer, and forwards the framebuffer to DM for display.

TSH then waits for the user to click in one of the menu boxes. Note that all pointer motion is interpreted by MS, and TSH is only notified (and given pointer coordinates) when a click occurs.

The details of TSH command processing are too lengthy to describe here. As an example, consider the case where the user has asked to change the current sensitivity level to another one for which a SLS exists. TSH updates the current label in the reserved area to be the newly selected sensitivity level by sending drawing requests to MS. Again, MS performs the drawing in its virtual framebuffer and forwards the virtual framebuffer to DM for display.

TSH notifies MS of the new value for the current operating level. MS notifies IM and DM of the new value for the current level. IM notifies the newly selected SLS that it has been selected as the current server and begins sending it pointer and keyboard input. DM begins processing requests from the "hold" port belonging to the newly selected SLS.

Figure 9 shows the screen while the trusted path is in use. Additional information can be found in [Epstein93b].

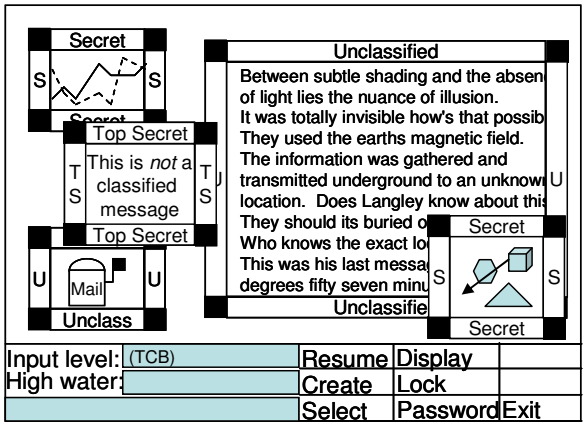

Figure 9. Screen in Trusted Shell Mode.

5.6. Screen Lock

Automatic screen locking occurs when IM detects that a period of time has elapsed without any input. The goal of automatic locking is to cover the working area (that portion of the screen which is not the reserved area) with an opaque pattern, and not to remove the cover until the user unlocks the screen. Manual screen locking is invoked through the trusted path. It is initiated when the user clicks on the "lock" menu entry in the reserved area, and is otherwise identical to automatic locking.

If IM detects a timeout without any input, it notifies MS. If the current server is an SLS, then IM notifies it that it is now deactivated. MS notifies DM to change its current sensitivity level to *none*, thus causing it to stop processing its current "hold" port (if any).

MS notifies TSH of the timeout. TSH notifies MS to map a window over the entire user portion of the screen. TSH then draws a pattern on this window using MS drawing primitives. TSH also sends messages to MS to set the current operating label displayed in the reserved area to *none*, and to display a message directing the user

to invoke the secure attention key to unlock the screen. After drawing the pattern, MS sends the virtual framebuffer to DM for display.

TSH notifies MS to change the current operating level to *none*. MS notifies IM and DM to change the current operating level to *none*[28]. IM discards input, and DM ceases to process its "hold" port until the user unlocks the screen through the trusted path.

6. Architecture Limitations and Issues

The architecture described here takes a very complex problem and makes it relatively simple. Tradeoffs have been made to achieve trust and simplicity. This section describes some of the positive and negative aspects of these tradeoffs.

6.1. The Price of Polyinstantiation

Polyinstantiation of servers has a price. For example, while the TMach security policy allows write-up and read-down, the TX policy does not allow either of these operations for X resources. That is, if a low client sends an untrusted high client an X resource ID (using TMach mechanisms, not TX), that ID will not be useful to the high client, because the high client has no means of using the resource ID in its low context. Similarly, the X operation to get a list of windows in the system, **XQueryTree** will only return those windows at the label of the caller, and none of the windows dominated by the caller. Existing CMW implementations do not polyinstantiate servers, so they can allow general read-down (and some also allowing write-up).

Our architecture could be extended by adding an additional trusted server which would pass information between SLSs in accordance with the TMach security policy. Each SLS could pass its resources (windows, colormaps, etc.) to that server, which would pass the resources to all servers which dominate the sensitivity level of the resources. Because resource IDs are polyinstantiated (as part of the server polyinstantiation) the X protocol would need to be extended so clients would be able to include a level as well as resource ID, or there would need to be some form of mapping between the actual and virtual resource IDs.

Other effects of polyinstantiation include problems with managing screen real estate. A tiling window manager[29] could successfully tile the windows at each level, but windows at different levels would overlap, because the window manager at one level has no way of knowing about operations at other levels. Each window would be correctly labeled, but the effect might not be what the user anticipated.

6.2. DAC and Information Labels

Many people who work with trusted X systems believe that some form of discretionary access control over X resources is desirable [Epstein91]. Without making the SLSs and window managers trusted (which would vastly increase our TCB size), DAC at the X resource level cannot be added to this architecture. Restricting a TX server to clients belonging to a single X user (or to users that the X user is willing to trust) at one time is a tradeoff to minimize the TCB.

While this project is clearly not aimed at replacing CMWs, we rejected adding information labels[30]. Once again, the SLSs and window managers would have to be trusted to provide useful information labels. Whether clients that use DAC or information labels also require a degree of trust remains an open question.

Hybrid solutions are possible. TX as a B3 windowing system could have less trusted single level servers and window managers to provide DAC or somewhat more trusted SLSs and WMs to provide information labels (but less trusted than the overall system). This allows a high degree of assurance that the system is trusted and enforces the overall system security policy, with lesser assurance that information labels are properly maintained and that the DAC policy is properly enforced.

6.3. Trusted Graphics

A major difference between the TX architecture and commercial CMW implementations is the question of what is trusted. CMWs provide trusted graphics: the CMW evaluation should provide assurance that the graphics drawing is correct. In our architecture, we provide no assurance that the SLS drawing code is correct, though we have no particular reason to suspect it either. For example, if an application asks to draw a circle, existing CMWs guarantee that the circle will be drawn in the correct location with the correct attributes. In TX, we guarantee that *if* the SLS draws a circle correctly in its framebuffer and correctly passes the framebuffer to DM, DM will properly display it in an appropriately labeled window. Thus, we have traded the functionality of trusted graphics for a much smaller TCB.

A close analogy can be drawn to an untrusted file server encapsulated within a trusted system. In this case,

[28] If an automatic lock occurs, IM is already in the *none* state. This extra notification is required when the user manually locks the screen.

[29] A window manager that arranges windows automatically so they do not overlap each other, but rather form a tiled pattern on the screen.

[30] An information label represents the sensitivity of the information contained in a subject or object. Information labels are required in CMWs, and used for labeling information which is believed to be of a lower sensitivity than the object (or subject) from which it came. Information labels are advisory and are not used for access control.

the overall trusted system makes no guarantee that the data will be stored or retrieved faithfully, only that it will be labeled correctly.

6.4. Graphics Hardware Usage

Our architecture presumes a "dumb" framebuffer (i.e., one where the graphics hardware simply maps bits in memory to the screen). Intelligent graphics hardware now perform many functions, such as drawing polygons, filling regions, and 3-dimensional operations using hardware, rather than using software in the X server. However, without special provisions our architecture is unable to take advantage of intelligent graphics hardware. The problem is that the SLSs cannot be allowed to use the graphics hardware directly, and DM only performs simple region copying operations. If graphics hardware can be encapsulated so that SLSs can use it with their virtual framebuffers, then it could be used in this architecture. Unless the encapsulation is reentrant, this probably means bringing graphics hardware into the TCB, a questionable undertaking. An approach to using polyinstantiated graphics hardware is described in [Epstein96].

6.5. Performance

Performance of the prototype TX server was not studied in detail. Some of the standard benchmarks provided with X were run, with a typical result that TX is half the speed of ordinary X, when the underlying hardware and software base were kept constant. Interactive usage was much slower; one user compared it to typing on a 300baud dialup terminal. Performance measurement showed that TMach message passing and context switching times dominate the system throughput.

6.6. User Interface

The TX user interface requires the user to invoke the trusted path to change the current operating level. This contrasts with switching between windows at the same sensitivity level, where the user typically just moves the pointer to place the cursor in the new window, or moves the pointer to place the cursor in the new window and clicks. While unsure how users will react to this requirement, we were unable to devise any other mechanism which would be both unspoofable and not require large amounts of trusted code.

As noted above, colormaps are polyinstantiated, and are reinstalled whenever the user changes the current level. One of the side effects of this change can be "colormap flicker": because individual pixels in the framebuffer reference a colormap entry, not a particular color, switching colormaps may cause the windows associated with other levels to change colors.

For example, consider running two instances of SLS, with the colormaps as shown in Figure 10. Assume the foreground of an Unclassified window uses colormap entry 2 (i.e., a pixel value of 2) and the background uses entry 4, while the foreground of a Secret window uses colormap entry 3 and the background uses entries 4 and 5. If the current input level is Unclassified, then the first window will be a red foreground on a purple background and the Secret window will be a green foreground on a purple and orange background. Switching the current input level to the Trusted Shell will cause the windows to become black, while switching the input level to Secret will cause the Secret window to appear in its natural pastel colors (lavender foreground with orchid and sky blue background) while the Unclassified window will appear with a pink foreground and orchid background.

While this may be a bit shocking to users, we believe the color shifts are more desirable than allowing a covert channel, as would exist if there were a shared colormap among the SLS instances.

Colormap Entry #	Mini Server	Unclass SLS	Secret SLS
0 (reserved)	Black	Black	Black
1 (reserved)	White	White	White
2	Black	Red	Pink
3	Black	Green	Lavender
4	Black	Purple	Orchid
5	Black	Orange	Sky blue

Figure 10. Sample Colormap Entries.

More subtle compromises were made with respect to cursors and the screen background. Cursors were limited to 32x32 pixels (a limitation permitted by the X protocol) to eliminate any chance of the cursor spoofing a window. The screen background (the area not inside any window) is only writeable by the SLS with the lowest classification (i.e., system low), and is limited to a tiled image no more than 32x64 pixels, again to preclude use of the screen background as a window spoof.

6.7. Non-X Implementations

While the purpose of the TX project was a platform to build trusted X, there is no fundamental reason why the SLS could not implement a window protocol other than X. For example, a SLS could provide the graphical portion of a Microsoft Windows system, rather than an X server. This is a convenient mechanism to allow running different windowing systems simultaneously on the same display, with minimal modifications to the windowing

systems. Such an approach might have utility even in an environment where trust is not required [Pascale92]. As an example, the Sun version of the prototype supported both X and MGR [Uhler88], a freely available windowing system from Bellcore. No changes to the TX TCB were required to support MGR, which indicates that our basic architecture is flexible enough to support a variety of windowing systems.

6.8. Extensions

Various extensions have been developed for X, including 3-dimensional graphics (PEX) and video (VEX). In the TX architecture, many extensions can be added without requiring any trusted code, and hence without consideration to the security implications. For example, we expect that PEX could be implemented as part of the SLS without any change to the TCB, providing it does not use the graphics hardware directly. This ability to extend the SLS (or replace it entirely) without changing the TCB is a major advantage of our architecture over other architectures.

7. Contemporaneous Related Work

Two other contemporaneous approaches for high assurance windowing system are a paper design for a system without any trusted code by Mayer and Padilla, and a patented architecture using hardware for high assurance by Sherman, Dinolt, and Hubbard. This section describes some of the advantages and disadvantages of each approach relative to the revised TX architecture, as well as a comparison to existing CMW implementations.

7.1. Compartmented Mode Workstations

All existing CMW implementations have very similar architectures, including a monolithic trusted server, a trusted window manager, and a few trusted clients to assist with visual labeling and cut and paste operations. However, there is no reason why other architectures (including the one described here) could not be adapted to the CMW requirements. The descriptions of CMWs are based on numerous conversations with CMW designers and developers.

The TX TCB is much smaller than that of the existing CMWs. We expect our TCB to be less than 10 percent of the size of existing CMW X TCBs. As previously noted, CMWs provide trusted graphics, DAC, and information labels which we do not. While we do not feel these are major limitations, the hybrid approach described in section 6.2 is a possible solution. That approach provides a high level of assurance on the overall system, with assurance equal to that of the CMWs for graphics, DAC, and information labels.

Compatibility with untrusted X is a major goal for both TX and CMWs. Because of our architecture, we are able to offer a much higher degree of compatibility with X. For example, we are able to run untrusted window managers, which is impossible with CMWs. We require no special privilege mechanisms, unlike CMWs. While we constrain what clients can do, our system imposes *fewer* limits than CMWs, which is a counter-intuitive result.

Finally, our architecture allows addition of extensions to the X server, or even replacement of the X server (e.g., with a new release) without modifying the TCB. This allows TX to keep up with developments in the broader X marketplace more easily than CMWs.

7.2. Mayer/Padilla Design

A paper design for a high assurance windowing system is proposed in [Mayer92]. Their MLS windowing system (henceforth referred to as MP) does not include any trusted code. M argue that the TX architecture is not minimal, as their strawman architecture is clearly smaller.

In the MP architecture, the IM functions of detecting trusted path and routing input are presumed to be handled by the operating system. They have a DM equivalent which runs at system high, accepting requests from window system servers (such as X servers) but never acknowledging the operations.

While this architecture avoids trusted code, the cost is usability:

- Windows are not labeled with sensitivity labels, since the Display Manager equivalent is not trusted. We that visible labeling is essential to usability of multi-level windowing systems.
- The trusted path is non-graphical, meaning the user's windowing environment is destroyed and the user is dumped into terminal mode. We feel that users want graphical interfaces as much as possible. CMW implementations have carried this to an extreme, providing highly stylized color interfaces.
- The trusted path provides no capability to point to a window and determine its sensitivity label. If visible labels are not to be provided, we feel that this is a minimal function. However, the MP architecture cannot support this functionality.
- MP assumes that the operating system trusted path facilities are sufficient to start new servers and change input levels. Since this is not typical of operating systems, at a minimum it would require changing the existing trusted path.

As Mayer and Padilla have not implemented their proposed system, we are unconvinced whether it is possible to build a system which works as they describe.

The issue at the heart of the difference between TX and the MP architecture is the definition of the security policy. Because windowing systems exist to provide human interfaces, we consider that the security policy includes not only the information flows, but also the human interface.

The TX architecture uses trusted code to provide what we consider to be those functions which need to be trustworthy: input routing, trusted path detection, trusted path features, and visible labeling. As such, we feel that TX is minimal when both trust and usability are considered. A trusted windowing system without at least trusted visible labeling is a "secure brick."

TX is a balance between the high functionality and large TCB of the CMW approach, and the low functionality and completely untrusted system of the MP architecture.

7.3. Sherman/Dinolt/Hubbard Design

In [Loral91] Richard Sherman, George Dinolt, and Frank Hubbard describe a very high assurance (beyond A1) multilevel windowing system using hardware, henceforth referred to as SDH.

The SDH architecture (which predates TX, but was unknown to us during our design phase due to the patent process) is not predicated around any particular window system. SDH dedicates an untrusted processor to each *imaging generator*, which are analogous to TX single level server processes isolated using a high assurance operating system. The SDH *display generator* uses hardware to merge the results of imaging generators together and place them on the screen, performing a subset of the functions provided by the TX Display Manager.

Because of its reliance on hardware separation, SDH provides high assurance and high performance. However, in order to keep the hardware simple, SDH does not allow for overlapping windows of varying sensitivity labels. Rather, it partitions the screen into horizontal bands, each of which has a unique sensitivity label. Within each band, windows can overlap.

Use of hardware as an isolation mechanism gives higher performance than TX, but requires adding additional hardware for each new sensitivity label to be used. Additionally, the set of sensitivity labels to be used must be predefined in SDH. This special purpose hardware with fixed labels is in contrast to the general purpose hardware with a trusted operating system used in TX, where servers can be dynamically created and destroyed as necessary.

Because the SDH uses non-overlapping bands, the input management and trusted path concepts are much simpler than in TX [Dinolt92]. Applications in SDH can cause the position of the pointer to move, but only within the current band. When the user moves the pointer to another band, the user's input is then labeled at the new band's sensitivity label by the RMMI which serves a function similar to the TX Input Manager. That is, the RMMI switches the input level by moving the pointer, rather than by invoking trusted path as in TX. The ability to switch by a pointer movement was a goal in TX, but one was not realized because of the massive covert channels present due to overlapping windows and the X protocol.

The size of the window bands can also be adjusted using the pointer. Since bands cannot overlap, spoofing is not a concern as it is in TX. Other trusted path facilities necessary in TX, such as creating new window system servers, are not necessary, since the set of sensitivity labels in use is fixed.

A concern with using the mouse position for determining input classification is human engineering: if the user accidentally knocks the mouse so that it points into a different window, the input classification may be changed without the user noticing.

One feature present in SDH but not in TX is that SDH is designed to allow trusted input (such as video) to be routed directly to the screen in a trusted manner.

In summary, SDH trades off functionality in the areas of overlapped windows, the ability to create windows which cover the whole screen, and dynamically created untrusted window system servers for a high performance special-purpose hardware solution which has a simple input paradigm.

8. Later Related Work

This section describes several projects which were influenced by or built on the TX research.

8.1. Starlight

Starlight Interactive Link [Anderson96] was built to solve many of the same problems as TX, namely to provide high assurance windowing. The relative complexity of TX (as seen by the number of servers and interactions, plus the requirement for an MLS operating system) steered the designers towards a hardware-based solution[31].

Starlight's concept is multiple untrusted X servers, each on a physically separate computer system. The X clients can run anywhere so long as they can connect to

[31] Private communication with Mark Anderson.

the appropriate X server[32]. The "low" servers are replaced by *proxy servers* that run on the low systems, each of which feeds the protocol requests via a one-way communication link to a "high" *proxy client* which receives input from all low proxy servers. The proxy servers each maintain sufficient state that they can provide responses to the clients at their level, since they do not receive any feedback from the high server. The proxy client on the high system simply passes requests to the high server, discarding any responses. The high server handles overlapping windows, but does not provide any visible window labeling.

Input is handled by connecting the keyboard and mouse to a high assurance n-way switch, with the output of the switch physically connected to all of the servers. The user selects the input level using a knob on the switch, which causes the input to be redirected to the appropriate X server.

As with TX, a separate window manager is used at each level.

The trusted portions of the Starlight architecture are the one-way transfer used for low to high transfers, and the input switch.

The advantages of Starlight include:

- No trusted software, and simple trusted hardware.
- Allows overlapping windows (unlike [Loral91]).
- No requirement for an MLS operating system.
- No modification to the X server, even to use a virtual framebuffer.

The disadvantages of Starlight include:

- No visible window labeling.
- Requires one hardware system for each level of data to be processed (i.e., no MLS).
- There must be one "high" system to which all the others can send their data.

The Starlight system is now marketed in the U.S. by Tenix America[33] as the Interactive Link Data Diode (IL-DD) Keyboard Switch (IL-KBS), and has been evaluated as meeting Common Criteria EAL7 requirements.

8.2. Eros Window System

The Eros Window System (EWS) [Shapiro04] takes a "clean sheet" approach to designing a trusted windowing system, without respect to compatibility with existing systems such as X. Their paradigm is one of isolation by default, rather than sharing by default as in X, and which much of the TX design is focused on controlling. The focus on isolation also solves a limitation of TX, namely that within a given level, TX clients can interfere with each other.

Using the fresh approach, EWS moves all of the drawing logic into the window system clients (rather than requiring an X server to do so), and uses shared memory to provide the image for rendering (similar to TX's use of Mach messaging to share the virtual frame buffer). EWS uses a cut & paste paradigm far simpler than that in X that avoids the downgrade problems described above, without requiring the "energetic" evaluation described in section 4.13.

EWS is designed to allow for multi-level windowing support, although this is not part of the prototype described in [Shapiro04]. Their design does not provide for visible window labeling, which we considered important. Regardless, their ability to reduce the trusted portion of the system to only 4500 lines of code is impressive.

Finally, the published work on Eros includes an original artwork by a precocious artist using the EWS. The author is pleased to offer this first independent citation of Wesley Vanderburgh's artwork.

8.3. Nitpicker

Nitpicker [Feske05] is an effort to address many of the same issues as TX, although their focus is on preventing theft of input by malicious clients, rather than an MLS protection architecture. Their prototype does not provide window labeling, but does provide limited trusted path capabilities. The Nitpicker TCB is 1500 lines of code, significantly smaller than that of TX.

There are many similarities between Nitpicker and TX, including the use of virtual frame buffers, the effort to exclude as much code as possible from the TCB, and input processing.

8.4. IBM cut & paste patent

Carson *et al* [IBM96] address the cut and paste issues in a Compartmented Mode Workstation, in an effort to provide mediation for cross-level operations, including limited downgrade actions. Their focus is on using the existing X protocol operations, and to have the user decide via pop-up windows whether to allow the operation.

8.5. Other efforts

An effort to build an MLS version of X for SELinux [Kilpatrick03] rejects the concept of server polyinstantiation, and focuses on defining permissions necessary for each type of object managed by the X server, and how to tie those into the SELinux policy model.

[32] There is no presumption of separate networks for each level, although that is the most common usage.

[33] http://www.tenixamerica.com/images/white_papers/TenixIL_KBS.pdf

Ganapathy *et al* [Ganapathy05] used semi-automated tools to identify where authorization decisions should be made in legacy code. Their test case was the X server, and they compare their results to the manual results of [Kilpatrick03].

Both of these approaches are roughly equivalent to that of the CMW vendors, in that they do not address assurance or TCB minimization. Neither addresses visible window labeling.

Sun's Solaris 10 Trusted Extension will include MLS extensions to X based on Sun's CMW effort[34]. This is the first instance of an MLS windowing system in a general-purpose commercial product.

9. Lessons Learned

Looking back after 15 years, the primary lesson learned is that the architectural tradeoffs change with time, some of which have made this architecture more desirable, and others have made it less feasible.

Among those that are more feasible:

- Faster CPUs with significantly more memory have made the penalty exacted by polyinstantiation less onerous.
- The availability of virtual machine monitors such as VMWare have made MLS itself less important than it was. The TX architecture could be implemented in a VMWare environment, with the trusted servers running in one VMWare partition and other VMWare partitions each supporting a single SLS, SE, and clients.

Among those that have made this approach less desirable:

- The lack of desktop MLS operating systems means that the approach is currently infeasible, other than using a virtual machine monitor.
- The reduced cost, size, power, and heat of desktops have made multiple hardware approaches, such as those in Starlight, more feasible (although the management of multiple computer systems is no easier than it was).
- The lack of direct access to graphics hardware in the TX architecture is much more limiting than it was at the time.
- Inexpensive KVMs can be used to address the requirement in some environments, where there is no requirement for simultaneously viewing windows of different classifications.

Another major class of lessons learned is not to focus exclusively on the MLS aspects of the system, to the extent of ignoring attacks on the X clients. For example flaws in the xterm terminal emulator client [CERT93] provide a local user the ability to obtain "root" privilege, and [CERT97] describes local vulnerabilities in the libXt library used by nearly all X clients that can allow a user to gain "root" access if the client program is setuid-root. In our architecture, these attacks would be limited to a single level and a single SLS, and perhaps further constrained by TMach. However, overlooking this type of flaw missed a key aspect of X security.

The lesson from Eros is that starting with a clean design can yield a much simpler system than we achieved with our goal of compatibility with X. In particular, the cut & paste mechanism in Eros is far simpler than ours, yet maintains the desirable properties of allowing cross-level cut and paste without covert channels. Of course, this lesson is true for all types of software – considering security from the start is far simpler than bolting it on afterwards, as we attempted to do with TX.

A final lesson relates to the selection of TMach: while our contract required use of TMach, both TMach and the Mach system underneath were quite immature. Significant effort went into overcoming the limitations of the port model. Additionally, the lack of shared memory forced us to pass virtual frame buffers from SLSs as very large messages to the DM, which contributed substantially to the performance limitations of the prototype. While our colleagues at TIS who were building TMach were quite helpful, building a prototype (TX) on top of a prototype (TMach) on top of a prototype (Mach) substantially increases risk.

No research project is complete without a catchy acronym. While we considered several names (e.g., TRIX – TRusted Interactive X[35]), we never settled on one, and the project became known just as "Trusted X". As this is also a generic term, citations of this research frequently use the term "TRW Trusted X". Had we anticipated the emergence of Google, perhaps we would have put more energy into selecting a better name, to make the project easier to find for future generations of researchers.

10. Conclusions

When this research started in 1989, the general reaction in the security community was that a high assurance MLS windowing system (especially X) was impossible, and we occasionally had doubts as well. Our design and development has shown that the early pessimism was unfounded. Our minimal, modular TCB implementation, combined with a simple security policy show that high assurance MLS X is feasible.

[34] Christoph Schuba, personal communication.

[35] Perhaps the "fruity flavors – raspberry red, lemony lemon, orangey orange, wildberry blue, grapity purple and watermelon" in the breakfast cereal could have been used to represent different classifications, a welcome change from the steady diet of "Secret/NOFORN/" and similar labels used in most MLS papers.

11. Acknowledgements

The TX project involved contributors from TRW, Trusted Information Systems, and the University of North Carolina, Chapel Hill. The author acknowledges the many contributions of Rita Pascale[36], Marvin Shugerman[37], Bonnie Danner[38], and Ann Marmor-Squires[39] (all from TRW), Marty Branstad[40], Hilarie Orman[41], Homayoon Tajalli[42], Doug Rothnie[43], Steve Padilla[44], and Glenn Benson[45] (all from Trusted Information Systems), and John M^{c}Hugh[46] and Charlie Martin[47] from the University of North Carolina, Chapel Hill.

Many of the ideas that went into the TX project grew out of discussions in the Trusted Systems Interoperability Group (TSIG), a working group of vendors working on CMWs and related systems. The author particularly appreciates his interactions with Jeff Picciotto from The MITRE Corporation and Glenn Faden from Sun Microsystems.

Finally, the author thanks Dan Thomsen for starting the "Classic Papers" session at ACSAC, and to Tom Haigh for the opportunity to present this paper.

12. References

[Anderson96] Mark Anderson, "Starlight: Interactive Link", *Proceedings of the 12th Annual Computer Security Applications Conference*, December 1996.

[Bell75] *Secure Computer Systems: Unified Exposition and Multics Interpretation*, MTR 2997, The MITRE Corporation, D. E. Bell, L. J. La Padula, July 1985.

[CERT93] CERT advisory CA-1993-17 xterm Logging Vulnerability, www.cert.org/advisories/CA-1993-17.html

[CERT97] CERT Advisory CA-1997-11 Vulnerability in libXt, www.cert.org/advisories/CA-1997-11.html

[CMWEC91] R.D. Graubart, J.L. Berger, J.P.L., Woodward, *Compartmented Mode Workstation Evaluation Criteria, Version 1*, MTR 10953, The MITRE Corporation, Bedford, MA, June 1991 (also published by the Defense Intelligence Agency as document DDS-2600-6243-91) [Not in the public domain].

[CMWREQS87] *Security Requirements for System High and Compartmented Mode Workstations*, DIA Document Number DDS-2600-5502-87, John P. L. Woodward (MITRE), November 1987.

[Dinolt92] Private conversation with George Dinolt, October 2, 1992.

[Epstein90] Jeremy Epstein, "A Prototype for Trusted X Labeling Policies" in *Proceedings of the Sixth Annual Computer Security Applications Conference*, December 1990.

[Epstein91] Jeremy Epstein and Jeffrey Picciotto, "Trusting X: Issues in Building Trusted X Window Systems -or- What's not Trusted About X?", in *Proceedings of the 14th National Computer Security Conference*, October 1991.

[Epstein93a] Jeremy Epstein, et al, "A High Assurance Window System Prototype, in *Journal of Computer Security*, Vol. 2, No 2&3, 1993

[Epstein93b] Jeremy Epstein and Rita Pascale, "User Interface for a High Assurance Windowing System" in *Proceedings of the Ninth Annual Security Applications Conference*, December 1993.

[Epstein96] Jeremy Epstein, "A High-Assurance Hardware-Based High-Performance Trusted Windowing System, in *Proceedings of the 19th Annual National Information System Security Conference*, October 1996.

[Feske05] N. Feske and C. Helmuth. "A Nitpicker's guide to a minimal complexity secure GUI", in *Proceedings of the 21st Annual Computer Security Applications Conference*, December 2005

[Ganapathy05] Vinod Ganapathy, Trent Jaeger, and Somesh Jha, "Retrofitting Legacy Code for Authorization Policy Enforcement", in *Proceedings of the 2005 IEEE Symposium on Security and Privacy*, May 2005.

[36] Now a Montgomery County (MD) math teacher.
[37] Deceased.
[38] Now with Northrup Grumman.
[39] Now with Northrup Grumman.
[40] Retired.
[41] Now with Purple Streak, Inc.
[42] Now with Symantec, Inc.
[43] Now a freelance videographer.
[44] Now with Washington DC Temple of the Church of Jesus Christ of Latter-Day Saints.
[45] Now with JP Morgan Chase.
[46] Now with Dalhousie University.
[47] Now with Sun Microsystems.

[IBM96] Mark Carson, Mudumbai Ranganathan, Janet Cugini, and Khalid Asad, *Integrity Mechanism for Data Transfer in a Windowing System*, U.S. Patent 5,590,266, issued December 31 1996.

[ICCCM89] *Inter-Client Communication Conventions Manual*, Version 1.0, MIT X Consortium Standard, 1989.

[Kerberos88] Jennifer Steiner, Clifford Newman, and Jeffrey Schiller, "Kerberos: An Authentication Service for Open Network Systems" in *Proceedings of the Winter USENIX 1988 Conference*.

[Kilpatrick03] Doug Kilpatrick, Wayne Salamon, and Chris Vance, "Securing the X Window System with SELinux", Technical Report 03-006, NAI Labs, March 2003. www.nsa.gov/selinux/papers/x11/t1.html

[Loral91] Richard Sherman, George Dinolt, and Frank Hubbard, *Multilevel Secure Workstation*, U.S. Patent 5,075,884, issued December 24, 1991.

[Mayer92] Frank Mayer and Steve Padilla, "A Straw Man Design for an MLS Window System: An Example Complex Application for High Assurance Systems", in *Proceedings of the 15th Annual National Computer Security Conference*, October 1992.

[Motif90] *OSF/Motif Programmer's Reference*, Open Software Foundation, Prentice-Hall, 1990.

[Pascale92] Rita Pascale and Jeremy Epstein "Virtual Window Systems: A New Approach to Supporting Concurrent Heterogeneous Windowing Systems" in *Proceedings of the 1992 USENIX Summer Conference*, July 1992.

[Protocol88] X Window System Protocol, MIT X Consortium Standard, X Version 11, Release 4, Robert Scheifler, 1988.

[Shapiro04] J. S. Shapiro, J. Vanderburgh, E. Northup, and D. Chizmadia, "Design of the EROS trusted window system", in *Proceedings of the 13th USENIX Security Symposium*, August 2004

[TCSEC85] National Computer Security Center, Fort Meade, MD, *Trusted Computer Systems Evaluation Criteria*, DoD 5200.28--STD, December 1985.

[Uhler88] Stephen Uhler, *MGR - C Language Application Interface*, Bell Communications Research, 1988.

[Yellow85] National Computer Security Center, Fort Meade, MD, Computer Security Requirements - Guidance for Applying the Department of Defense Trusted Computer System Evaluation Criteria in Specific Environments, CSC-STD-003-85, June 1985.

13. Glossary

CIT	(TX) Client Initiator Terminator
CMW	Compartmented Mode Workstation
DM	(TX) Display Manager
ICCCM	Inter-Client Communications Conventions Manual
IM	(TX) Input Manager
KVM	Keyboard Video Mouse (switch)
MLS	Multi Level Secure
MS	(TX) Mini Server
PE	(TX) Property Escalator
SAK	Secure Attention Key
SE	(TX) Selection Emulator
SIT	(TX) Server Initiator Terminator
SLS	(TX) Single Level Server
TCSEC	Trusted Computer System Evaluation Criteria
TSH	(TX) Trusted Shell
WM	(TX) Window Manager