A Prototype B3 Trusted X Window System

Jeremy Epstein*, John McHugh†, Rita Pascale*, Hilarie Orman†, Glenn Benson†, Charles Martin†, Ann Marmor-Squires*, Bonnie Danner*, Martha Branstad†

* TRW Systems Division, Fairfax Virginia;
† Trusted Information Systems, Glenwood Maryland;
‡ University of North Carolina at Chapel Hill, Chapel Hill, North Carolina

Abstract

Multilevel secure windowing systems are a key technology for the 1990s. We have spent the past 20 months designing and implementing a prototype of a multilevel secure X Window System as a proof of concept vehicle for our software engineering process model for the development of trusted systems. The prototype is targeted to B3 evaluation criteria. In the early stages many doubted that B3 was achievable for a windowing system (especially X); we believe that our prototype demonstrates that B3 is achievable.

This paper describes our goals, the architecture of our system, and some of the trade-offs we made to achieve our goals. It also contrasts our work with existing Compartmented Mode Workstations (CMW) windowing systems.

1 Introduction and Goals

The 1980s brought widespread availability of relatively inexpensive workstations with bitmapped graphics displays. As the move to industry standards continues, the UNIX® operating system and the MIT-developed X Window System (X) have become the basis for most of these workstations. The 1990s are bringing multilevel secure versions of these workstations.

There are many problems associated with building a trusted X Window System, whether aimed at B1, CMW, or B3. The underlying problem is that X has only a very limited notion of protection. X is designed to encourage cooperation among clients in an atmosphere of openness and mutual trust and provides support for this via numerous direct and indirect communication mechanisms. This is the antithesis of secure computing. An extensive discussion of trust issues in X can be found in [2].

The Compartmented Mode Workstation (CMW) [9] project (sponsored by the Defense Intelligence Agency) has shown that a windowing system can provide B1 functionality and assurance. For many 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). We feel that the guidelines in [12] require a B3 level of assurance for such applications.

Our goal in this initial phase of our research project is to design and build a prototype system, which we call Trusted X (TX). We are not attempting to build a system which is B3 certifiable per se, but are focusing our attention on those issues that we feel 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 current phase of our work ignores 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 are using the TMach 2.5 [1] prototype from Trusted Information Systems as our base. This paper discusses several of the key Mach and TMach features used by our prototype. For our current effort, we have ignored the issue of a B3 network. When TMach provides appropriate network capabilities, network access to TX can be accommodated.

In addition to developing a TX architecture suitable for B3 evaluation, the focus of this paper, our research addresses the development of formal policies.
for trusted applications and the composition of these policies with those of the supporting operating system base. We are also investigating visible labeling of windows, a key aspect of trusted windowing systems not present in operating systems or networks. Some of our early thoughts on visible labeling are found in [3]; an update on this aspect of TX will be described in a forthcoming paper.

The paper continues with a brief introduction to the X architecture, then describes the X architecture, TX operation, architectural limitations, and a comparison of TX with CMWs. We conclude by summarizing the current status of the project.

2 X Architecture

The X architecture is based on the client/server model of distributed computing. As shown in Figure 1, the X server manages the screen(s), keyboard, and pointing device (typically a mouse).

The X server manages X 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 [4]. Clients send requests to the server over a bi-directional communications channel using any reliable byte-stream protocol (for example, TCP/IP or DECnet), 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 OSF/Motif[5] 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 [6] 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 [7]).

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 not simply another client. The conventions described in the X Inter-Client Communication Conventions Manual [7] are used to define an environment where "well-behaved" (ICCCM compliant) clients can interact cooperatively.

3 TX Architecture

In developing our architecture for TX we had several goals dictated by concerns for both B3 certifiable security and acceptable X functionality and performance. We were also interested in using TX as a proof of concept vehicle for our software engineering process model [11] that describes our approach to the development of trusted systems. From the standpoint of secure functionality the primary requirement of TX is that it displays correctly labeled data to the user in an unspookable fashion and allows the user to run untrusted applications without any potential for violation of the TMach security policy. With the exception of the functionality required to assure correct labeling and trusted path interactions, we make no claims about the veracity of data displayed by clients.

The TMach security policy is a Bell-LaPadula [10] style policy. Under TMach, objects are Mach ports which always have a single security level. Subjects are processes (or groups of processes having identical security characteristics including the associated user id) that operate over a range of security levels from alter min to view max. For untrusted subjects, the two are the same. For trusted subjects, view max dominates alter min. Propagation of port access rights is the action subject to mediation under the policy.

The external TX security policy is identical to the TMach policy. External TX objects are the ports through which it communicates with its clients. TX subjects are TMach subjects. These are divided into two classes: internal processes, trusted and untrusted, that are part of the TX system and external processes,
the untrusted⁴ clients that use TX. Each of the trusted subjects that make up the TX TCB (Trusted Computing Base) has its own internal security policy which must be shown to enforce the underlying information flow policy inherent in the TMach port access control policy.

One of the key difficulties in building a high assurance X is to minimize and structure the TCB to conform to the architectural requirements of TCSEC 3.3.3.1.1 [8]. The X portion of existing CMW TCB implementations is typically 200K to 450K lines of ‘C’ code (LOCC), or 100K to 200K statements.⁵ While the MIT provided code is generally well-written and well-structured, design documents do not exist, unless written by individual vendors. We feel that a TCB of this size and structure is fundamentally limited to B1 assurance.

We rely on the TMach base to satisfy requirements for a separate execution domain, protection from tampering, etc. as well as to provide assured labeling of subjects and external objects. Our TCB consists of a small collection of trusted subjects which communicate with TMach, the TX untrusted subjects, TX clients, and each other over TMach ports. Each of the trusted clients has a well defined function and a carefully defined and limited interaction with its peers and untrusted users. As a trusted application, TX is constrained to deliver a predefined functionality in a trustworthy fashion.

The prototype will contain about 20K LOCC (or 6K statements) in its TCB. This is less than 10% of existing CMW TCBs, and compares favorably with other B3/A1 TCBs: the SCOMP TCB with about 21K statements;⁶ the XTS 200 TCB (SCOMP’s successor) with about 20K statements⁷; and the TMach TCB with about 100K statements⁸. Comparable numbers for the GEMSOS TCB were unavailable⁹.

The individual trusted subjects range from a few hundred to a few thousand LOCC and we feel that their encapsulation and limited roles¹⁰ make their evaluation less difficult than comparable portions of an operating system kernel.

3.1 TX Organization

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 even 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. During one of our risk mitigation meetings, the “server per level” or “polynomial subject” idea surfaced. The present architecture is a second generation version of this approach. The TX architecture is based on polynomially, single level untrusted X servers (TX/SLS) whose virtual screens share the physical screen through a trusted subject, the Display Manager (TX/DM), and who share the keyboard/pointer inputs through a trusted subject, the Input Manager (TX/IM). In addition, a limited multilevel cut and paste capability is provided through a trusted subject called the Property Escalator (TX/PE). Additional trusted subjects are responsible for initiating single level servers (TX/SIT) and the untrusted clients (TX/CIT) associated with them such as selection emulators (TX/SE) and window managers (TX/WM). Trusted path interactions between the user and TX are supported by a minimal trusted server (TX/MS) and a trusted shell subject (TX/TSH). 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. Figure 2 shows the TX subjects and their interactions.

To minimize our TX, we have included virtually the entire X server outside the TCB. The entire window manager, and all of the X libraries (e.g., the widget libraries and toolkits) are also outside the TCB. We have relatively few special clients, and they do not rely on any of the X libraries. Minimizing the TCB without significantly impacting functionality requires polynomialization of the X server. At each sensitivity level for which a connection is requested, we create a single level X server to perform all X protocol requests, a window manager to manage windows at that sensitivity level, and a client to assist with multilevel cut and paste operations, all untrusted. An untrusted X client interacting with other untrusted X clients at its sensitivity level sees the full X functionality and is unaware of the existence of clients at other sensitivity levels. This creates a defacto policy of client isolation by level (with the exception of cut and paste discussed below), a policy chosen because almost all interactions between X clients and the server either require or have a potential for the transmission of information to other clients.

⁴ At the present time, we do not explicitly support trusted (multilevel) clients. If such a client were to be supported by TMach, TX could support it via multiple single connections.
⁵ This assumes that the entire X server, window manager, widget libraries, toolkit libraries, protocol libraries, and special clients are in the TCB. We believe that all existing CMWs include these items in the TCB. These estimates of TCB size are based on conversations with CMW developers. The LOCC value is simply a count of lines in source files (including comments and blank lines), while the statement count is the number of semicolons, which is a slight overestimation of actual statement counts.
⁶ Conversation with Les Fraim, MITRE.
⁷ Conversation with Chuck Bonneau, HFSI.
⁸ A preliminary estimate by Homayoon Tajalli, TIS.
⁹ Conversation with Don Brinkley, GEMINI.
¹⁰ 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 man-
3.3 Input Manager

The TX/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 either a TX/SLS or the TX/MS. 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.

The SAK sequence is not necessarily a single key press event, but may represent a combination of events that are watched for by the TX/IM. If the SAK is detected, the current server is set to be TX/MS, which then receives all input until it directs otherwise. This is the trusted path.

If TX/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 TX/MS). In this mode, all inputs except the SAK are discarded. This allows implementation of a screen lock facility (described in section 4.6). Once locked, the screen can only be unlocked through the trusted path. The current server is also set to none if the receiving server dies. Otherwise, TX/IM changes its current server only when instructed to do so by TX/MS.

TX/IM notifies a TX/SLS when it becomes the current server and when it loses this status. This notification allows the TX/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 cover channel is introduced because the notification is the direct result of user action.

TX/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 TX/IM since each TX/SLS is free to "warp" the cursor position as it sees fit and TX/IM would be unable to determine when the cursor is pointing to the icon.

3.4 Display Manager

The TX/Display Manager (TX/DM) controls the physical display, composing the displayed image from the pieces that are provided by the individual TX/SLSs and the TX/MS. It also is responsible for ensuring that the individual windows on the display have proper visual labels. In addition, it provides a special service, the drawing of "helping lines" to aid window managers. TX/DM maintains general hardware and configuration information that can be queried by any TX/SLS. As with the TX/IM, this information is bound at TX initialization time and cannot be changed thereafter.

As is the case with the TX/IM, the TX/DM supports a notion of a current server which is none, a TX/SLS or TX/MS. The current server is set by the TX/MS. The display is divided into two regions, one for the exclusive use of TX/MS, the other for use by all servers including TX/MS. The TX/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 and for interactions with TX/TSH.

Within TX/DM, certain artifacts of the display state are polyninstantiated and maintained on a per level basis. These include a list of installed colormap(s), the position, image, and color of the cursor, the position, sizes, contents, and stacking order of top level windows, and size and location of the "helping lines."

Each TX/SLS has two TMach ports to TX/DM. The "nonhold" port can be used by the TX/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 TX/SLS provides a replacement window image, which TX/DM clips using the boundaries of the given window and any overlapping windows. The "hold" port can be used to map and unmap windows, change their stacking order, update the cursor position and image, draw "helping lines" and perform various other administrative functions. Internally, TX/DM identifies windows by both their TX/SLS provided window ID and the identity of the TX/SLS which owns them. Each TX/SLS can only refer to windows which it owns.

TX/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 TX/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. TX/MS can update the contents of the "reserved area" by providing a replacement image, just as TX/SLSs provide replacement images for windows. TX/SLSs cannot draw in or examine the reserved area and are not aware that it exists.

When the current server changes, TX/DM installs the colormap(s) belonging to it, and displays its last cursor image in the proper position. Polyninstantiation of cursor position and colormap values allows complete flexibility in colormap and cursor handling without the covert channels which would normally exist. TX/DM also brings up the last set of helping lines (if any) provided by the server. TX/DM visibly labels each mapped window using a visual representation of the sensitivity level of the TX/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 TX/DM and are not accessible to any TX/SLS. Policies for avoiding spoofing of visible labels will be described in a forthcoming paper. The visible labels are the only graphics performed by TX/DM other than copying virtual window contents to the real frame buffer 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. Unfortunately, there is no way for the X server to distinguish between a window manager drawing helping lines and a client program using a transparent drawing on the screen background. We extended the X protocol to provide an explicit request for the drawing of helping lines. A TX/SLS passes this request to TX/DM through its hold port. TX/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.

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

The TX/DM is the largest and most complex component of the TX TCB. We expect that experience with it in the prototype will enable us to simplify its design in subsequent versions of TX. As hardware
costs, especially graphics hardware costs continue to
decline, it may be possible to provide a hardware based
DM at a reasonable price.

3.5 Mini Server

The TX/Mini Server (TX/MS) coordinates the ac-
tivities of TX/IM, TX/DM, TX/SIT, and TX/TSH. It
also provides a very limited set of graphics facili-
ties 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 TX/TSH, TX/MS
directs the TX/IM and TX/DM to set the
current server and thus the current operating level,
accomplishing this with an appropriate image (includ-
ing labels) for the reserved area of the screen. It also
directs the TX/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, metering algorithms). By contrast, TX/MS
provides primitives for TX/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 TX/MS drawing code to be several or-
ders of magnitude smaller than that of an X server.
TX/MS draws in a virtual frame buffer, which is sent
to TX/DM for actual display either in the reserved
area or on the general screen. TX/MS also handles
input in a very restricted fashion. Cursor movement
is tracked internally, and cursor position passed to
TX/TSH only when a pointer button is pressed or
released. Key press and release events are converted
into text strings for transmission to the TX/TSH.

3.6 Trusted Shell

The TX/Trusted Shell (TX/TSH) provides an in-
terface through which the user can perform certain ad-
iministrative and security functions necessary for the
operation of TX. These functions are:
• creating a new TX/SLS,
• selecting the current operating level,
• displaying the sensitivity level of a window on the
  screen,
• locking the screen (to allow walking away from
  the screen while logged in, without risk of some-
one else walking up and using it),
• unlocking the screen after either a manual or au-
tomatic lock,
• changing the user’s password,\textsuperscript{12}
• exiting TX.

TX/TSH uses the drawing primitives provided by
TX/MS. Its user interface is based on a simple menu

\textsuperscript{12}This function is provided so the user need not exit TX and
use the TMach TSH. It adds about 100 LOC (50 statements)
to the TCB.

displayed in the reserved area of the screen. Provid-
ing this functionality via interactions with the TX dis-
play 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 accom-
plished through interactions with a much less compli-
cated device than the bitmapped X display. We be-
lieve 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. Again, we expect
experience with the TX prototype to provide insight
in this area.

3.7 Server Initiator/Terminator

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

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

3.8 Client Initiator/Terminator

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

3.9 Property Escalator

TX supports multilevel cut and paste in accordance
with the ICCCM selection based protocol. The oper-
ation requires interactions between untrusted clients
called Selection Emulators (see section 3.11) and the
TX/Property Escalator (TX/PE). The TX/Property
Escalator (TX/PE) provides primitives to TX/SEs to write data (write-equals) and to read data provided by other TX/SEs (read-down). Read requests always provide the most recent request which meets the format criteria. The TX/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. We expect that the prototype will aid in determining an appropriate level of support for cut and paste functions.

Because the TX/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 pasters 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.

3.10 Single Level Server

The Single Level Servers (TX/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 and writing 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 we are using the Sun version of TMach for our prototype, we have based the TX/SLS on the Sun version of the X server.

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

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

All interpretation of input (i.e., determining which client(s) receive the keystrokes and/or pointer events) is performed by the TX/SLS. Each TX/SLS could map the keyboard differently (e.g., the secret TX/SLS could use a QWERTY keyboard mapping, while the confidential TX/SLS could use a Dvorak keyboard mapping). This is a side-effect of the polyinstantiation of the TX/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 TX/SLS performs its own mapping from the physical keys to the logical values. We feel that any benefit of allowing this flexibility is outweighed by potential problems for the user. Thus, our TX/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.

3.11 Selection Emulator

Cut and paste in X is performed according to the selection based conventions described in the ICCCM [7]. 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 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 TX/Selection Emulator (TX/SE) listens for announcements by cutting clients. When one is received, it immediately asks for the data in all of the advertised formats. TX/SE then passes the data to the TX/Property Escalator (TX/PE) which retains a database of available data.

TX/SE also listens for requests by pasting clients. When one is received, it queries TX/PE for the most recently cut data available in the specified format. TX/PE passes the data to TX/SE, which then makes it available to the client.

Note that there is a TX/SE running for each TX/SLS. A cut and paste operation involves two TX/SEs: one at the sensitivity level of the cutting client, and one at the sensitivity level of the pasting client.

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).

Should the overhead of conversions become excessive, compromises are possible. Multilevel cut and paste can be restricted to only a few formats with the usual lazy conversion rule used to support additional formats on a single level basis. While the selection based ICCCM protocol is recommended for new applications, older X clients may use "cut buffers," "clipboards," or other communications mechanisms. Within a single sensitivity level, these may be used, but they are not supported across levels. We suspect that some or all of these mechanisms could be supported, at a price, given sufficient need.

3.12 Window Managers

Any X window manager can be modified to be a TX/Window Manager (TX/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 above, this must be done via the X protocol helping-lines extension. Each TX/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 TX/WM would not have any knowledge of the confidential window. An interesting side effect of this architecture is that different window managers could be used at different sensitivity levels (i.e., run OSF/Motif at secret and OPEN LOOK at confidential), although the probable user confusion makes this undesirable. Our first TX/WM will be a modified version of OSF/Motif.

4 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 operating level. In normal operation, all input is routed to the current server, a TX/SLS at the current operating level, and certain output operations can only be performed by the current TX/SLS. For example, if the current sensitivity level is secret then all input is routed to the secret TX/SLS, and only the secret TX/SLS can map and unmap windows from the screen. Each TX/SLS can update windows that it has mapped, no matter what the current operating level is. The operating level will not refer to a TX/SLS if the active TX/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.

We rely heavily on TMach for the establishment and control of communications paths among the trusted and untrusted subjects that make up TX. Some of the control mechanisms are evolving as we develop the prototype since TMach is in development also. When the TMach and TX efforts mature, we believe that TMach will provide the precise forms of restricted communications implied in the discussions that follow. These features include non-transferable port rights, one time send rights, and delivery to subjects with specific user ids.

4.1 TX Startup

TX is started from the TMach Trusted Shell. TX/M starts all of the trusted tasks (TX/IM, TX/DM, TX/CIT, TX/SIT, TX/PE, TX/MS, and TX/TSH). Initially, TX/IM and TX/DM have no current level because they have no TX/SLS with which to communicate. Any input is discarded by TX/IM, and TX/DM has no hold or nonhold ports to read from. When it starts, TX/TSH makes drawing requests to TX/MS to display the current operating level (which is none) in the reserved area. When initialization is complete, the internal communications paths in the figure 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 propagated, even if one of the subjects wished to do so. The TX/SIT registers its connection request port with the TMach name server which then 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 TX/SIT.

4.2 Single Level Server Startup

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

The new TX/SLS sends messages to TX/IM and TX/DM asking for connections and for the hardware configuration data. TX/IM replies to the TX/SLS by providing information about the keyboard and pointer and TX/DM replies to the TX/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. TX/IM and TX/DM inform TX/MS that the new TX/SLS has connected. TX/MS in turn informs TX/TSH.

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

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

4.3 Client Connection

The client sends a connection request to TX/SIT having obtained send rights to its request port from the TMach name server. If a TX/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 TX/SLS along with the requests from the TX/SE and TX/WM. Otherwise, TX/SIT forwards the request to the appropriate TX/SLS and the TX/SLS responds directly to the client. As noted above, the TX/SIT retains no connection to the client.

4.4 Normal Operation

Once clients are connected to the TX/SLS, the system is in normal operation. Figure 3 shows the connections used in this state. This section describes a few common operations.

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

When the TX/DM maps a window for a TX/SLS, it creates a suitable labeled border around the window. This happens for all windows including those claimed to be transient by the client. The visible labeling policy to be followed when one window obscures the labels of another is the subject of a forthcoming paper.

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 TX/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 TX/SLS performs the drawing in its virtual frame buffer. When the drawing is complete, the TX/SLS sends its virtual frame buffer to TX/DM along with a list of windows and areas changed. TX/DM receives the contents update request on its nonhold port from the TX/SLS. Afterclipping the new window contents relative to other windows on the screen TX/DM updates the visual display. The clipping confines the updating to the unobscured interiors of windows belonging to the TX/SLS in question. This prevents the untrusted TX/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, TX/IM sends the input to the TX/SLS that is the current server. The TX/SLS performs the ordinary X rules for routing input to its clients and is oblivious to other TX/SLSs which might exist.

4.5 Trusted Path

Trusted path operation is provided to permit the user to change security sensitivity levels and perform other security related administrative functions. Trusted path operations are initiated by the user when the secure attention key sequence is invoked. If the current server is a TX/SLS, then TX/IM notifies it that it is now deactivated and notifies TX/MS that trusted path was invoked. TX/IM begins sending input to TX/MS.

TX/MS notifies TX/DM and TX/TSH that trusted path has been invoked. TX/DM sets its current server to none, blocking processing of "hold" port requests. TX/TSH sends commands to TX/MS to draw the menu of commands, and to change the current operating label displayed in the reserved area to Trusted Path. TX/MS performs the drawing operations in its
virtual frame buffer, and forwards the frame buffer to TX/DM for display.

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

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

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

4.6 Screen Lock

Automatic screen locking occurs when TX/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.

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

TX/MS notifies TX/TSH of the timeout. TX/TSH notifies TX/MS to map a window over the entire user portion of the screen. TX/TSH then draws a pattern on this window using TX/MS drawing primitives. TX/TSH also sends messages to TX/MS to set the current operating label displayed in the reserved area to none, and to display a message informing the user to invoke the secure attention key to unlock the screen. After drawing the pattern, TX/MS sends the virtual frame buffer to TX/DM for display.

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

5 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.

5.1 The Price of Polyinstantiation

Polyinstantiation of servers has a price. For example, while the TMac security policy allows write-up and read-down, the TX policy does not allow either of these operations for the resources. That is, if the client sends an untrusted high client an X resource ID (using TMac 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, XQueryWindowTree will only return those windows at the label of the caller, and none of those 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 TX/SLSs in accordance with the TMach security policy. Each TX/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 refer to include a level as well as resource ID.

Other effects of polyinstantiation include problems with managing screen real estate. A tiling window manager 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.

5.2 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[2]. Without making the TX/SLSs and window managers trusted (which would vastly increase our TCB size), DAC at the X resource level cannot be added to this architecture. We feel that 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 an acceptable tradeoff to minimize the TCB.

While this project is clearly not aimed at the Compartmented Mode Workstation community, the question of whether information labels could be added to

---

13 If an automatic lock occurs, TX/IM is already in the none state. This extra notification is required when the user manually locks the screen.

14 An information label represents the sensitivity of the information contained in a subject or object. Information labels are...
the architecture has been raised. As with DAC, the answer is no. Once again, the TX/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 C2 trusted single level servers and window managers to provide DAC or B1 trusted TX/SLSs and WM's to provide information labels. 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.

5.3 Trusted Graphics

A major difference between the TX architecture and existing CMW implementations is the question of what is trusted. Existing 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 TX/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 TX/SLS draws a circle correctly in its frame buffer and correctly passes the frame buffer to TX/DM, TX/DM will properly display it in an appropriate window. Thus, we have traded the functionality of trusted graphics for a much smaller TCB. This notion is no different from a file system as a component of a trusted system which does not guarantee that the data will be stored or retrieved faithfully, only that it will be labeled correctly.

5.4 Graphics Hardware Usage

Our architecture presumes a "dumb" frame buffer (i.e., one where the graphics hardware simply maps bits in memory to the screen). Intelligent graphics boards 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 boards. The problem is that the TX/SLSs cannot be allowed to use the graphics board directly, and TX/DM only performs simple region copying operations. If graphics hardware can be encapsulated so that TX/SLSs can use it with their virtual frame buffers, 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. It is likely that the economics of hardware will make a reasonable level of graphics hardware polyninstantiation feasible in the near future permitting a different approach to the problem.

5.5 Performance

Performance of the prototype TX server is difficult to predict accurately at this time. Our implementation is not yet far enough along to run any benchmarks. However, several observations can be made:

- TMach message passing time will be a major cost, especially for passing the virtual frame buffers using copy-on-write. Benchmarks of message passing in Mach are contradictory, so we are not yet sure how this will fare.
- Context switching times will be an even more critical factor in TX than in X, since most operations will require the involvement of multiple servers (e.g., TX/IM and the TX/SLS for input, TX/DM and the TX/SLS for output) instead of just the X server.
- On a multi-processor system, the potential for speedup is large, as TX is already partitioned. For example, on a four processor system, TX/IM could run on one processor, TX/DM on a second, the secret TX/SLS on a third, and the confidential TX/SLS on a fourth. Thus, without any multi-threading of the TX/SLS, significant speedup can be achieved.

5.6 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. We are unsure how users will react to this requirement, however we have been unable to devise any other mechanism which would be both unsophisticable and not require large amounts of trusted code.

6 Comparison with CMWs

This section compares some of the key aspects of TX with existing CMW implementations. 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. 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 5.2 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 will allow us to keep up with developments in the broader X marketplace more easily than CMWs.

7 Status and Conclusions

Our implementation has been underway for about six months. As of this writing (September 1991), TX/M, TX/IM, TX/MS, TX/TSH, and TX/SIT are complete, and a monochrome version of TX/DM is working. Changes to TX/SLX have been completed, and development of TX/PE and TX/SE are underway. We expect to complete the remainder of the black and white system by the end of 1991.

When this phase of the project started 20 months ago, the general reaction in the community was that a B3 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, combined with a simple security policy and a formal model indicate that B3 X is feasible. Our experience thus far with implementation indicates that our analysis was correct, and a working prototype is strong evidence of such.

8 Acknowledgements

The authors appreciate the assistance of Steve Padilla (formerly with TIS) and Marvin Shugerman (formerly with TRW), both of whom assisted during various phases of the architecture development, and Doug Rothnie (with TIS), who has assisted with the implementation of TX/SE and TX/PE. We also appreciate many valuable comments from reviewers.

References


