The Cross Domain Desktop Compositor: Using hardware-based video compositing for a multi-level secure user interface

Mark Beaumont and Jim McCarthy
Defence Science & Technology Group
Edinburgh, SA, Australia
mark.beaumont@dsto.defence.gov.au

Toby Murray
University of Melbourne and Data61
Melbourne, VIC, Australia
toby.murray@unimelb.edu.au

ABSTRACT

We have developed the *Cross Domain Desktop Compositor*, a hardware-based multi-level secure user interface, suitable for deployment in high-assurance environments.

Through composition of digital display data from multiple physically-isolated single-level secure domains, and judicious switching of keyboard and mouse input, we provide an integrated multi-domain desktop solution. The system developed enforces a strict information flow policy and requires no trusted software. To fulfil high-assurance requirements and achieve a low cost of accreditation, the architecture favours simplicity, using mainly commercial-off-the-shelf components complemented by small trustworthy hardware elements.

The resulting user interface is intuitive and responsive and we show how it can be further leveraged to create integrated multi-level applications and support managed information flows for secure cross domain solutions.

This is a new approach to the construction of multi-level secure user interfaces and multi-level applications which minimises the required trusted computing base, whilst maintaining much of the desired functionality.

1. INTRODUCTION

High assurance systems require greater rigour in their design, development, implementation, and verification to ensure they correctly satisfy certain safety or security critical properties [9]. Examples include avionics [24], or national security infrastructure systems [8], where the consequences of failing to enforce these properties can be grave. As such, systems targeted for operation in these environments are usually subjected to strict evaluation and accreditation requirements before they are put into service [3, 21]. Often the accreditation requirements either unduly constrain a system, or limit its use to operation in lower assurance environments. Typically, a formal analysis of the system reasoning about the critical properties is needed [23] prior to deployment. As a consequence, high assurance systems require a balance between complexity, security or safety properties, and usability to ensure a system is suitable for the intended environment.

In a computing environment, separate security domains enforce strict isolation to prevent data leakage and maintain system integrity and availability. This is especially important for government, national security, and other sensitive networks (e.g., financial, medical), where data compromise, data loss, or system down-time can have severe impact [10]. Using multiple of these single-level secure systems in parallel can be cumbersome and inefficient, often resulting in replicated infrastructure and multiple user interfaces.

Extant high assurance hardware solutions allow user interface infrastructure to be shared [20], however the user still interfaces with each domain independently. Extant software solutions do combine the user interfaces for multiple domains onto the same desktop [1], however these rely on large trusted computing bases comprising hypervisor, security domain software, and drivers – making them too complex to evaluate and too risky to accredit for high assurance use. Software solutions fail to address the increasing risk of compromised hardware [22], implicitly incorporating many hardware components into the trusted computing base.

Protecting the integrity of isolated security domains, and preventing information leakage between the domains, precludes the use of existing software solutions on our most sensitive networks. We instead present an approach that integrates the user interface from multiple computing domains, external to the domain computing infrastructure. We sacrifice some functionality and loss of semantic information for a smaller trusted computing base and a verifiably more secure solution – attempting to combine the security of existing hardware solutions, with the cognitive integration benefits of the software solutions.

The Cross Domain Desktop Compositor (CDDC) is a multi-level secure (MLS) user interface that uses hardware-based composition to simultaneously display and decorate graphical output from multiple single-level-secure domains on a single computer monitor. There is no need to trust any software residing on the individual domains, or the hardware platforms on which they exist. Secure keyboard and mouse switching dynamically directs user input between domains to provide a seamless user experience and maintain confidentiality between domains.

Orchestration of the composited graphical output regions and the keyboard/mouse switching allows a unified user interface to be constructed. Through this interface a user can

ACSAC '16, December 05 - 09, 2016, Los Angeles, CA, USA

ACM ISBN 978-1-4503-4771-6/16/12…$15.00
DOI: http://dx.doi.org/10.1145/2991079.2991087

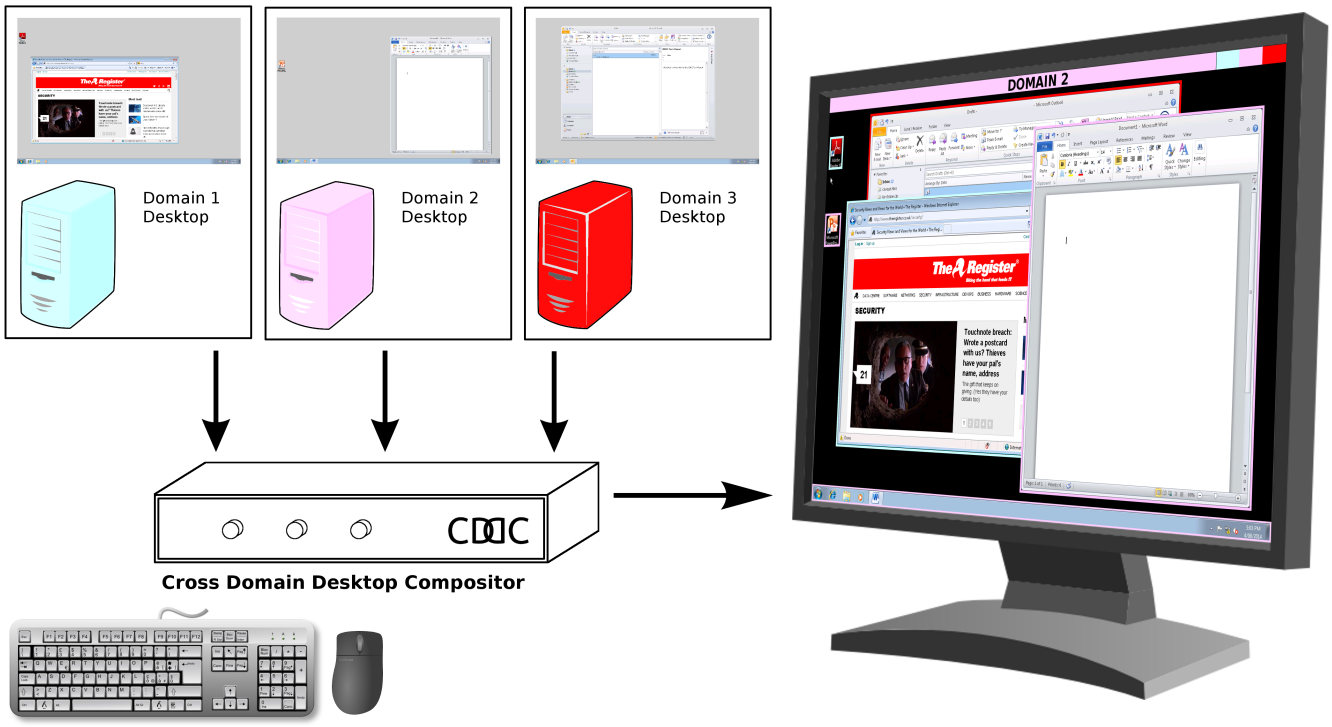

Figure 1: The Cross Domain Desktop Compositor

interact with the domains as though they were part of a single desktop environment. At any point in time, one domain is designated as the *active* domain which has its windows composited foremost and exclusively receives all user input.

Figure 1 shows three domains being composited and DOMAIN 2 is currently active.

Creating an MLS user interface for use in high assurance environments is difficult: the CDDC achieves this and creates a converged desktop for modern operating systems using simple, hardware-based compositing and intelligent trusted switching. Once in use, the CDDC can be leveraged to facilitate the deployment of true MLS-like applications without many of the usual requirements for evaluation and accreditation of such solutions.

In this paper, we position and analyse the CDDC in the field of related solutions, describe the threat model and architecture of the CDDC, cover the implementation of a hardware-based demonstrator, and detail the operation of domain-side software. Consideration is given to the operation of the trustworthy elements within the design and a formal analysis of the security properties of the CDDC is presented.We analyse potential covert channels and steps that can be taken to mitigate information leakage. We then look at deploying integrated cross-domain applications on the architecture and examine the benefits, as a generalised approach for application delivery across multiple domains. We conclude with some remarks on the architecture and forward looking research.

2. RELATED WORK

Historically, the development of an MLS desktop experience has been tackled in a variety of manners from multilevel secure workstations [18] to virtualised desktops [6], secure graphical user interfaces (GUIs) and secure keyboard-video-mouse (KVM) switches. The goal of these approaches is to provide a user experience where multiple computing domains can share a common interface and a user can interact with all applications in a common environment.

Modern systems that virtualise access to desktops include: AFRL's SecureView [1, 15], which runs multiple environments in logically isolated Virtual Machines and provides secure software based compositing of different level windows. SecureView provides similar window decoration and input switching as the CDDC; and Raytheon's Trusted Thin Client [16] which uses a customised Centos operating environment to support the delivery of remote desktops from multiple domains across a single wire.

Increasingly, the trusted element in these solutions is a hypervisor such as Xen [2]. Often a relatively small secure domain will contain additional trusted code to further support the required user interface functionality. Some examples include TrustGraph [13] which implements a trustworthy graphics subsystem, and Qubes OS which implements a secure GUI virtualisation subsystem [17].

Similarly, secure GUIs and trusted windowing systems operate on top of trusted operating systems, hypervisors, or microkernels. Examples include: Nitpicker [5] which provides secure window buffers for different applications to write into, and then be displayed and moved around on the screen; and Trusted X [4] which secures client interaction with the X server, preventing applications accessing each others' display data. Although these models are useful, they often do not operate or interact well with the complexities incorporated in modern desktop user interfaces – the windowing environments and API constructs are too complex.

The above solutions all have a software trusted comput-

ing base and also assume for a large part that the underlying hardware mechanisms can be unconditionally trusted. Two issues with the software trusted computing base arise: first, the size of the code is often too large and unwieldy to formally analyse; and second, the software is often vulnerable to many well-known attacks, as well as zero-day attacks.

Conversely, the CDDC does not rely on trusting any software or any commercial-off-the-shelf hardware. Instead, a simple trusted computing base is constructed in hardware and can be retrofitted to existing multi-desktop environments, removing any vulnerability to software-based attacks and making it more amenable to formal security evaluation. The computing domains themselves remain untrusted, pushing the trust boundary into the small, well controlled external hardware, which both strengthens the security guarantees, and increases the performance of the solution, whilst making it easier to accredit for high-assurance environments.

The K424F-SH from Smart Security Labs [20, 19] is a secure KVM switch that allows multiple domains to be simultaneously viewed on a single screen. The domains are each presented within their own decorated window that can be moved or scaled on the screen in either a tiled or cascaded type of display. The keyboard and mouse are switched between the domain through mouse interaction.

The CDDC architecture is similar to the K424F-SH, taking advantage of isolating input video paths, and isolating and proxying Human Interface Device (HID) paths. Visual desktop integration in the K424F-SH occurs at a coarse level; where the entire graphical user interfaces for each domain are presented separately and a user still interacts with each domain separately. In contrast, the CDDC presents individual windows from each domain together on a single graphical desktop interface – providing a user with the cognitive benefits of operating within a single desktop environment.

The draw-back of operating in hardware is the loss of semantic information – the CDDC operates purely on the video output from a desktop, with no implicit notion of windows, widgets, or other interface elements. This issue is mitigated by the aid of untrusted domain-side software.

3. THREAT MODEL

The single-level secure computing infrastructure from each domain is untrusted. This includes the hardware platforms, operating systems, and application software; even if that software supports the operation of the CDDC.

The display is trusted, more so than for a single level display as it is relied upon to accurately reflect the multi-level secure state of the user interface. The keyboard and mouse are also trusted. The consequences of a malicious keyboard and mouse are discussed in Section 7.

The CDDC device is trusted to operate correctly. A deeper analysis of the trusted computing base is provided in Section 4.4.

4. THE CDDC ARCHITECTURE

Desktop computers from separate single-level secure domains are connected to the CDDC via their digital display output (e.g., displayport) and USB input to accept HID input, i.e., keyboard and mouse data – Figure 2.

The main components of the architecture are: unidirectional forcing elements on the input video streams and the HID outputs; video processing blocks to extract display regions to composite from each input video stream; a compositing engine that generates the composited output in real-time; and a trusted switch that directs the key strokes and cursor position to the correct domain.

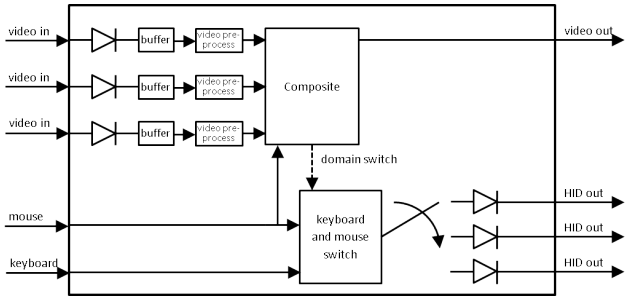

Figure 2: Basic CDDC architecture

The CDDC is differentiated from a secure KVM in the manner it identifies and operates on sub-regions of each video stream and consequently how the user interacts with the composited output.

4.1 Window Identification

Modern operating environment GUIs are often constructed using rectangular windowed regions of the display; e.g., application windows, dialog boxes, and desktop icons. These rectangular elements combine together to form the familiar desktop interface. The CDDC constructs an MLS user interface by compositing these windowed regions from multiple isolated domains onto a single graphical output.

There are various methods to identify window regions, e.g., chroma-key information or image processing algorithms. These methods can be complex and unreliable and result in inadequate composition. In the prototype CDDC, windows are identified by untrusted domain-side software and sent in-band within the raw video stream, encoded as pixel data.

This approach has two benefits, firstly the location of all windows is accurately known, and secondly the reported locations are implicitly associated with each and every frame of the video stream received. We address the trustworthiness of the window locations in Section 7.1.

4.2 Composition

The CDDC receives a separate video stream containing a full graphical desktop from each domain. The CDDC identifies window regions from each desktop and composites them based on a dynamically defined domain ordering, outputting only the pixel value from the foremost domain window region at any specific pixel location. Necessarily, window regions within any one domain are kept in the same order, as the CDDC only has access to the raw video stream, and hence can only manipulate already drawn window regions.

Domain ordering prescribes which content to display when multiple window regions are identified at the same pixel location. Domain ordering is controlled by the CDDC and updated based on user interaction with the domains. The *active* domain has the highest priority and its content is output in preference to all other domains.

The CDDC augments the composition by generating and rendering its own content. CDDC generated content is MLS content, it is used to identify windows from different domains by decorating them with a unique coloured border. CDDC

generated content is output in preference to other content at any pixel location, and cannot be modified by any individual domain. CDDC generated MLS content is trusted. Figure 3 shows a typical converged desktop with composited and decorated windows from three separate domains.

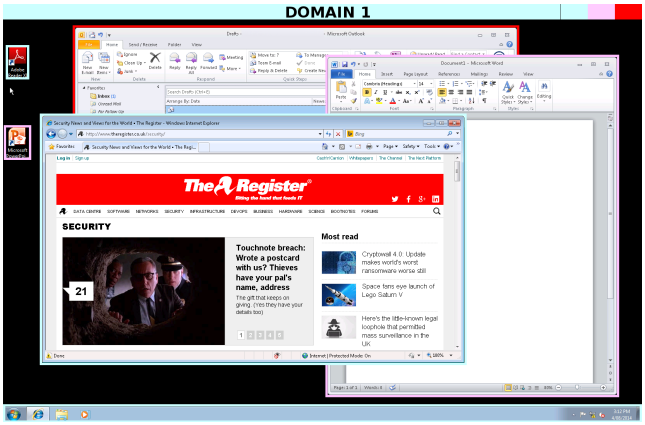

Figure 3: Composited Desktop view for three domains. DOMAIN 1 is the active domain.

Undecorated regions of the composited display are treated according to a predetermined security policy, e.g., rendering as a static background colour, or rendering a greyed-out version of the unidentified content from the currently active domain. Rendering this content unaltered may encourage certain spoofing attacks.

4.3 User Interaction

To facilitate secure user interaction, the CDDC renders a cursor, as the highest priority, on the composited output. The position of this cursor is dictated solely by the movements of the mouse as interpreted by the CDDC. The cursor position and keyboard input are exclusively directed to the active domain. The user is alerted to the currently active domain by a non-maskable CDDC generated banner rendered across the top of the composited output. It is large and easily distinguishable from other elements, to aid correct user interaction and understanding of the current context.

A user can interact with all visible windows through normal move and click cursor actions. The CDDC reacts appropriately, fordwarding the HID actions to the correct domain. The active domain is changed by: a mouse click when the cursor is positioned above content from a different domain; a physical button press on the CDDC; or a click on a virtual button rendered by the CDDC. The domain order, top banner, and keyboard and mouse switch are all consistently updated with the active domain. The set of window regions from the newly active domain are composited foremost. Figure 3 shows the trusted banner across the top of the composited display and the virtual buttons in the top right corner.

The CDDC can render arbitrarily complex user interfaces that allow a user to communicate with and control aspects of the CDDC.

4.4 Trusted Computing Base

Our design philosophy was to minimise the size and complexity of the trusted components, allowing us to focus on their trustworthiness, and enforce an appropriate information flow policy – non-interference between the separate domains. The minimal trusted computing base of the CDDC consists of a composition module and a switch for the HID data. We also use replication and isolation of components, and enforced unidirectional data flows to help achieve the desired non-interference property.

Minimising this trusted computing base, limits the attack surface for both malicious software and hardware attacks. Limiting the trusted component to a single FPGA allows specific Hardware Trojan countermeasures to be applied.

4.4.1 *Trusted Composition*

The correct composition and decoration of window regions is critical for the integrity of the CDDC. A user must be able to discern which domain they are currently interacting with. By decorating all identified content from every domain, correctly ordering the domains, and displaying the active domain banner a user can always be sure of the current context. Whilst window decoration is not critical to ensuring this, it aids the user against potential spoofing attacks, and as such it must operate correctly.

The position of the cursor must also be accurately rendered to ensure a users actions are correctly enacted.

Failure of the decoration, active banner or cursor rendering can result in a compromise of the CDDC's security. A user may either unwittingly perform incorrect actions, or perform actions in an incorrect domain – risking both the integrity and confidentiality protections afforded by a correctly operating CDDC.

The window position inputs are not trusted, however the identified window positions are always correctly decorated. Attacks against these positions could cause user confusion. The active domain banner combined with the decoration should alert users to potential issues. Possible attacks are explored later in Section 7.3.

The trusted code for the firmware to implement the composition, including the on-screen display and cursor rendering is around 150 lines of VHDL for the prototype described in Section 5.2.

4.4.2 *Trusted Switch*

The HID switch is trusted to direct the key strokes and cursor position to the currently active domain. Failure of this switch could compromise the confidentiality of data associated with the underlying domains. The trusted code for the firmware to implement the switch is only a couple of lines of VHDL.

4.4.3 *Failsafe Architecture*

We maintain separation between the inputs and replicate video processing up until the data is composited – meaning data is only mixed in the trusted components. We enforce undirectionality constraints on the input video streams and the output HID data in the hardware. Whilst there are no generic processing elements that could take advantage of bidirectional channels, these hardware-based data diodes provide failsafe mechanisms in the architecture, and also provide primitives upon which we can model and reason about the security of the CDDC.

The underlying unidirectional nature of the input and output streams means there is little impact of these data diodes. The dynamic information flow policy we desire is: informa-

tion will only flow from the keyboard or mouse input to a single (active) domain at any one time; and no information will flow between domains.

The integrity of the CDDC relies on the compositor being trusted to correctly: apply domain decorations, maintain domain ordering, render the active domain banner, maintain and render cursor position, and interpret some cursor interactions, e.g., window-based domain switching.

When correctly operating, the CDDC can protect the integrity and confidentiality of user actions to an equivalent level of a single-level secure system.

5. PROTOTYPE

5.1 Domain Software

Untrusted domain-side software identifies graphical windows to be composited by using standard *Windows* API calls. This list includes the application windows, task bars, pop-up windows, dialogue boxes, menus, desktop icons, and tool tips. The list is processed to remove duplicates and some items fully enclosed within other windows (e.g., some tool-tips, dialogue boxes and menus).

The domain software reserves the top portion of the display, e.g., the top 50 lines of the screen. The presentation order (z-order), location, and size of each window is encoded into pixel values and sent in-band within the digital display data to the CDDC by drawing directly to the desktop canvas in the reserved space. Other windows are prevented from being located in this area, stopping the in-band information being obscured, and also preventing windows from residing underneath the trusted banner. For similar reasons, the domain-side software also hides the cursor. An example of the domain software and a close-up of the in-band information is shown in Figure 4.

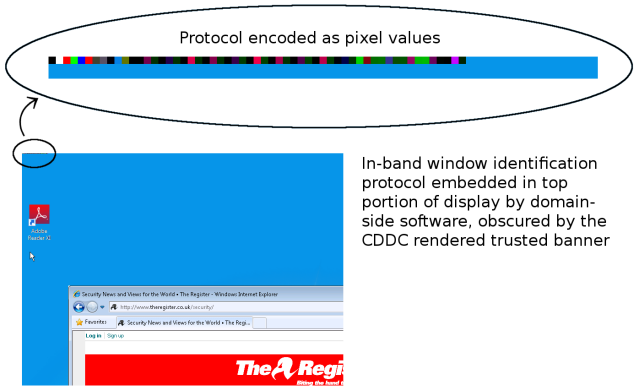

Figure 4: In-band encoded window information

A defined format including a header and checksum provides for reliable in-band communications.

The CDDC extracts the in-band information for each domain, allowing it to operate on the identified window regions. The CDDC also renders the top portion of the output display with the trusted banner indicating the currently active domain, obscuring the in-band communications from the user's perspective.

Having all domains running the same operating environment (e.g., Windows 7) at the same resolution, aids the integrated look and feel of the CDDC. If the task bars and desktop icons are in the same positions for the different domains, then only the elements for the active domain are displayed and decorated, with the other elements obscured in the composition; providing a level of cognitive integration and uniformity when interacting with the different domains. This circumstance is not uncommon with many large enterprises deploying a standard operating environment.

5.2 Hardware

We developed a hardware prototype, Figure 5, that accepts three displayport inputs, operating up to 1920x1200 resolution and outputs a composited display at 1920x1200 resolution. The hardware also accepts a single USB keyboard and mouse input and switches the output to any of the three domains. The composition, control, and switching is performed in a Xilinx Kintex 7 FPGA.

Figure 5: Displayport hardware prototype

5.2.1 *Architecture*

The natural decoupling between the keyboard and mouse data, and the display data allows separate processing chains for each. Figure 6 shows the very simple hardware architecture. Three displayport input streams flow separately into an FPGA for processing, composition, and output on a single displayport output. The keyboard and mouse are input through USB host interfaces, which connect to the FPGA via a Serial Peripheral Interface (SPI) link. The keyboard and mouse are output to each domain through a USB client interface, which also connects to the FPGA via an SPI link.

Figure 7 shows the firmware architecture of the modules contained in the FPGA. The display inputs and keyboard and mouse outputs are unidirectional and the hardware enforces this unidirectionality within the FPGA logic.

Isolating the input video paths and output USB paths allows the TCB to be kept simple. Any malicious action in these paths could also occur on the single-level secure domains, hence they are not considered part of the TCB. The TCB consists of the *compositor*, HID switch, video transmitter, mouse SPI interface, and the unidirectional links. The keyboard interface is not trusted and cannot affect the operation of the compositor.

5.2.2 *Cursor Control*

The CDDC rendered desktop cursor is the only element that crosses between the display composition and the input devices. It is the users actions through the cursor that unifies the trusted composition and the trusted switching to

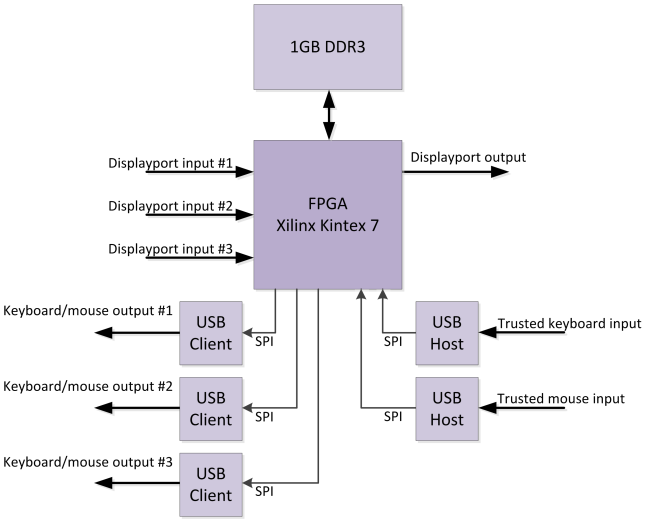

Figure 6: Hardware block diagram

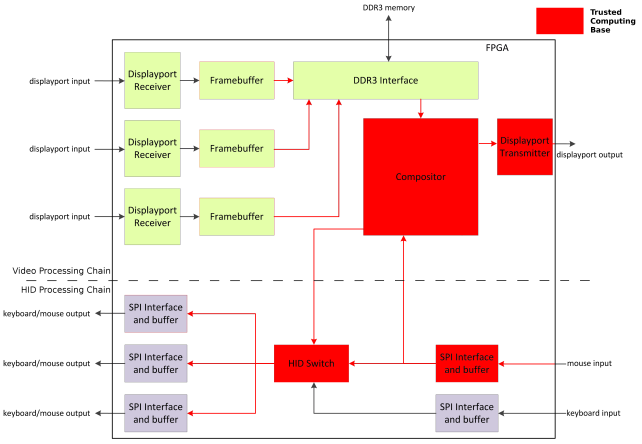

Figure 7: Firmware block diagram

create the converged desktop experience – the cursor crosses the trust boundary between the domains. The cursor also provides user control over the MLS actions of the CDDC. Given the importance of the cursor, its position is controlled and rendered by the CDDC, informed by the physically connected pointing device.

5.2.3 *Composition*

Xilinx displayport IP cores handle the display input and output from the FPGA. The input streams are buffered in independent, 3-frame cyclic buffers to compensate for video frequency variations, prevent I/O contention, and allow synchronous processing of the input display streams.

Frames are then synchronously processed in real-time in a raster fashion. Whilst processing the top portion of a frame (the bit corresponding to the position of the trusted banner), the identified windows regions are extracted from the pixel data and stored separately for each domain. In our prototype, this region of the display was 50 pixels high, corresponding to 50 lines of video.

The compositor module also creates and stores separate decoration regions representing the extents to be decorated; trimming the original identified window regions if necessary. In our prototype the decoration was 4 pixels wide. Pixel processing then follows the pipeline shown in Figure 8.

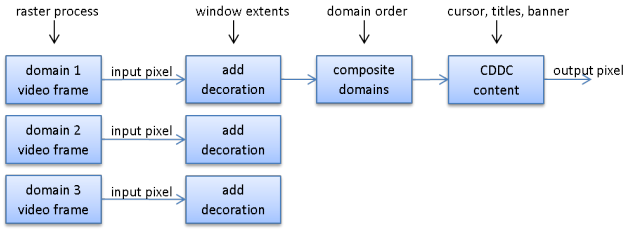

Figure 8: Pixel processing pipeline

Firstly, for every pixel in each domain frame, hardware comparators compare the current pixel raster location with the stored window and decoration regions for that domain, creating an intermediate decorated pixel output for each domain. This decorated pixel contains either: original windowed content; domain decoration colour; or no content. The z-order of the windows within a domain is important in correctly applying decorations, the algorithm used is included in Appendix A for reference.

The intermediate pixel values are then combined together to form the composited output. The output is generated based on the domain ordering and the existence of content at a raster pixel location, giving priority to the domains based on their ordering:

In the final pipeline step, the CDDC adds its generated content, including the active domain banner, virtual buttons, and the cursor.

For regions of the composited output that contain no pixel information, i.e., no windowed domain content, no decoration content, and no CDDC generated content, the compositor renders the background of the currently active domain greyed out by pixel modification. Greying out the background can prevent certain spoofing attacks where a domain attempts to render its own decorated content without reporting its window location.

5.2.4 *Domain Switching*

Seamless domain switching underpins the intuitive interface provided by the CDDC. On startup the CDDC defines and composites the domains in a specified order. The domain order can be changed by clicking on content or decoration from another domain, clicking on a CDDC generated virtual button, or pressing a physical button corresponding to a different domain. When a domain switch occurs, the new domain becomes the active domain and the ordering of all other domains remains the same. Composition ordering is updated at the start of the next video frame.

5.2.5 *HID Switching*

USB keyboard and mouse events generated by the trusted input devices are converted into unidirectional serial streams by FTDI VNC2 USB integrated circuits.

To facilitate responsive domain switching, a separate USB proxy device maintains a continual USB keyboard and USB mouse connection with each domain, similar to [20], using a Cypress PSoC acting as a serial to USB converter. Unmodified keyboard packets, and updated mouse packets are exclusively directed to the proxy for the currently ac-

tive domain. In this manner domains are unaware of being connected or disconnected to the real keyboard and mouse. When a domain switch occurs, the CDDC ensures any existing keystrokes are flushed to the currently active domain before switching to the newly active domain.

The USB mouse proxy is reported as an absolute positioning device, this ensures the cursor is positioned correctly in the active domain when a domain switch occurs.

5.3 Software Emulator

A software emulator of the CDDC has been developed. It connects to multiple desktop computers via the VNC protocol and composites the received video output. The keyboard and mouse are logically switched between the domains as required. The software emulator provides a base for further experimentation and refinement of composition algorithms.

6. FORMAL ANALYSIS

Formal modelling in Isabelle/HOL [12] was used to exercise the design early in the development, identifying data structures, exposing assumptions and exploring weaknesses. The emphasis for this work was to provide timely input, and thus the system's design was modelled at a relatively high level of abstraction and a global confidentiality property in the style of noninterference [7] was proved.

Even at this relatively coarse level of abstraction, however, the resulting security property captures a number of potential channels in the system. These include, for instance, buffered keyboard data from the current domain that needs to be flushed (i.e. cleared) when switching to a different domain to prevent information flows due to *residual* data, as well as potential information flows arising from *when* the user chooses to switch between domains.

In this section, we give just a flavour of the overall structure and intuition of the formal model and the security property proved for it. The model is formally defined as an *event system* comprising a number of concurrently executing *components*, and the security property comes from Murray et al.'s value-dependent noninterference formulations [11].

6.1 Formal Model

Let each of the *external* domains (i.e. the domain-side computers, each of whose video output is plugged into the CDDC and each of which receives its keyboard and mouse input from the CDDC) be denoted by a unique natural number between 1 and the number N of such external domains, and let $\mathsf{extdom} = \{1 \ldots N\}$. Then the formal model comprises the components: Keyboard, Mouse, CDDC and WS_d for each $d \in \mathsf{extdom}$.

For the sake of brevity, in this presentation we restrict our attention to those parts of the model that deal with keyboard and mouse input processing, including domain switching. The full formal model also captures the processing of video frames, including their composition and rendering of CDDC-generated content, like the mouse cursor, and the banner across the top of the screen indicating the currently active domain and the trusted window decorations. However, these operations pose little security threat, given the unidirectional nature of the information streams involved. Hence, they are modelled abstractly and the confidentiality property says less about them.

Figure 9 depicts the component decomposition of the formal model. The CDDC component models the behaviour of

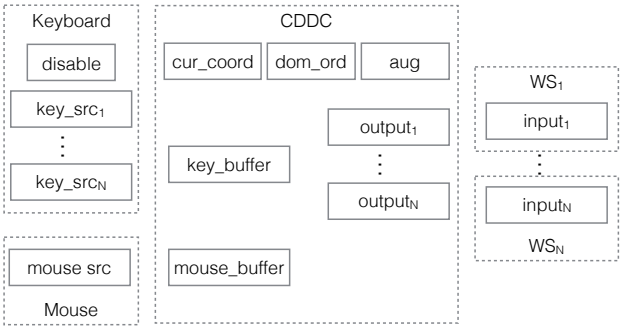

Figure 9: Relevant components of the CDDC formal model.

the internal keyboard/mouse input processing loop of the CDDC and its interaction with the other components. Its internal state includes dom_ord the current external domain ordering (the topmost domain of which is the currently *active* domain): when a domain switch occurs, this ordering is updated by putting the newly active domain on top. Its state also includes a number of internal buffers, including key_buffer, a keyboard buffer, and mouse_buffer, a mouse buffer, as well as output buffers output_d for each of the external domains $d \in \mathsf{extdom}$. Finally, in this simplified presentation, its state includes the framebuffer aug, which holds the augmented display data generated by the CDDC, including the top banner that indicates to the user the currently active domain. A system-wide invariant that we proved, and is required for security to be proved, is that the topmost domain in dom_ord agrees with the currently active domain as indicated by aug. We return to the CDDC component shortly.

Each WS_d component models the untrusted domain-side computers connected to the CDDC. Its internal state includes an input buffer input_d into which the CDDC component places keyboard and mouse input data. A single execution step for a WS_d component has it read the next item of data in the CDDC's output_d buffer and copy it to the input_d buffer, modelling the receipt of user input data by the domain-side computer connected to the CDDC.

The Keyboard component models a trustworthy user typing on a trustworthy keyboard. Its internal state includes, for each external domain $d \in \mathsf{extdom}$, an infinite stream $\mathsf{key\_src}_d$ of characters representing user input typed at level d. Its state also includes a disable flag (not depicted) that it shares with the Mouse component, modelling a mechanism by which the CDDC component temporarily disables the keyboard and mouse while it is analysing a mouse-click (see below, and Section 7.2). For each computation step, if the disable flag is not set, the Keyboard component reads the next input character from $\mathsf{key\_src}_{active}$, where *active* is the currently active domain as indicated by the CDDC-generated augmented display data aug. It places this character in key_buffer, the keyboard input buffer of the CDDC component. This models a faithful user who always types input at the level indicated by the CDDC-generated banner across the top of the display.

The Mouse component models the trustworthy user providing mouse input to the CDDC. It contains a single infinite

```
GetInput:    Read datum dt from mouse_buffer or,
               if none available, key_buffer;
             if dt is a mouse click then
               set disable flag and goto TestClick;
             else
               update cur_coord as needed;
               goto Ferry(dt);
TestClick:   let d = domainOf(cur_coord);
             update dom_ord and aug as needed;
             if d is not the old active domain then
               goto Flush(0);
             else
               unset disable flag;
               goto Ferry(click(cur_coord));
Ferry(dt):   let activedom = topmost(dom_ord);
             put dt into output_activedom;
             goto GetInput;
Flush(n):    if n < BUFLEN then
               clear(key_buffer [n]);
               goto Flush(n + 1);
             else
               unset disable flag;
               goto Ferry(click(cur_coord))
```

Figure 10: Behaviour of the CDDC component. Given coordinates c, the function $domainOf(c)$ returns the topmost domain in dom_ord whose windows occupy position c, while $click(c)$ creates a mouse click event at location c.

stream mouse_src of mouse input data. At each computation step, if the disable flag is not set, this component reads the next item from mouse_src and places it in mouse_buffer, the mouse input buffer of the CDDC component.

The CDDC component has the most complex behaviour. Its internal state includes a structured *program counter* variable and each computation step atomically executes an entire labelled block (i.e. everything up to execution of the next **goto** statement) of the input processing loop described in Figure 10, namely the block identified by the current value of the CDDC's program counter. The **goto** statements in Figure 10 show how the program counter variable is updated in each execution step.

The entire system executes the above components in parallel, interleaving their individual execution steps to form execution traces. No component ever blocks, nor synchronises with any other, so issues such as enabledness and termination-sensitivity are irrelevant.

6.2 The Security Property

The security property is a variant of noninterference, and essentially forbids information at the level of one external domain d being observed at another d'.

To phrase this requirement we group the data in the system into (sometimes overlapping) collections, called *security labels*, and define a global information flow policy that says how information is allowed to flow between these labels. For the CDDC formal model, the set of security labels includes just the following. Each external domain $d \in$ extdom has its own label, $\mathsf{External}_d$. However, we also include an extra label Internal, for data that at no instant in time belong to any of the external domains. The Internal label includes, for instance, the mouse input data which, because it defines *when* domain switches occur, cannot be labelled at the level of any of the external domains, as otherwise it would create a trivial information channel between all such domains [11]. Specifically, when a domain switch occurs, the domain being switched to necessarily learns that the mouse cursor was (clicked) over one of its windows, and so learns something about the current mouse position. Thus the mouse data cannot be labelled with any external label.

The information flow policy $\rightsquigarrow$, then, says that information is allowed to flow from the Internal label to all others, but that no other information flows are permitted between labels.

$$\mathsf{Internal} \rightsquigarrow d \qquad d \rightsquigarrow d' \longrightarrow d = d'$$

This policy ensures that, while *when* domain switches occur can (and will) be controlled by Internal state, this is the *only* state that can do so. In practice, the CDDC formal model adheres to this requirement because it allows domain switches to occur only in response to mouse clicks.

To state the security property, we then need to define the labelling of the data in the system. This labelling is defined by, for each label l, defining an equivalence relation $\overset{l}{\sim}$ on states of the system so that $s \overset{l}{\sim} s'$ holds for any two states s and s' precisely when the values of all data labelled by l are identical between s and s'. We say that the equivalence relation $\overset{l}{\sim}$ *includes* all data labelled by l.

Before describing the labelling and the equivalence relations, we first define the confidentiality property that we prove for the system, to show that it adheres to the information flow policy $\rightsquigarrow$ defined above. This property is equivalent to the following.

Definition 6.1. Confidentiality *holds when for all labels l, for all reachable states s and t, and all states s' and t' reachable from these respectively after a single execution step performed by the same component in each,*

$$s \overset{l}{\sim} t \ \wedge \ s \overset{\mathsf{Internal}}{\sim} t \longrightarrow s' \overset{l}{\sim} t'$$

Confidentiality simply requires that each execution step reveals to an arbitrary label l only information that l already knew, or Internal information (which, recall, $\rightsquigarrow$ allows anyone to learn).

The essence of what this security property means and enforces, then, is captured by the labelling defined in terms of the equivalence relations $\overset{\cdot}{\sim}$.

Note that by phrasing the labelling via equivalence relations on states s and t, it may depend on the contents of s and t themselves. This allows the labelling to depend on fixed parts of the state, namely the Internal-labelled state that the policy $\rightsquigarrow$ allows all other labels to observe. We make use of this to have the labelling for labels $\mathsf{External}_d$ depend on whether d is the currently active domain.

Specifically, the state labelled $\mathsf{External}_d$ naturally includes all (containers holding) d-classified data in the system: the Keyboard's input stream $\mathsf{key\_src}_d$, the CDDC's output buffer output_d, and the WS_d's input buffer input_d. When d is the active domain, it also includes the CDDC's key_buffer. An implication is that the security property then requires (i.e. it enforces) that the key_buffer only ever contain the data of the currently active domain. This is why it must be cleared when switching to a new domain, and why the keyboard must be disabled during this process, by the CDDC (see Figure 10).

Finally, when d is the active domain, the CDDC's program counter (mentioned above) is also labelled by $\mathsf{External}_d$. As we will see below, the behaviour of the CDDC component is largely independent of the keyboard data it receives from the current domain, except when the program counter is e.g. $\texttt{Ferry}(dt)$, in which case the program counter's value directly encodes potential keyboard data dt from the current domain. Clearly values of this form for the program counter contain information of the current domain. However, all other information contained in the program counter (besides dt) contains only publicly observable information (i.e. that derived from Internal data).

This allows the remainder of the information in the CDDC's program counter to be labelled Internal. Specifically, $\overset{\mathsf{Internal}}{\sim}$ relates two program counter values that are equal, as well as any two values of the form $\texttt{Ferry}(dt)$ and $\texttt{Ferry}(dt')$. The remainder of the state not labelled by any $\mathsf{External}_d$ label (i.e. the Keyboard's mouse_src and disable flag, plus the CDDC's mouse_buffer, cur_coord, dom_ord and aug) is all labelled Internal. It is precisely this state that controls when domain switches occur.

The confidentiality security property is sound and complete to one over entire execution traces, for which we refer the reader to [11].

6.3 Discussion

The formal analysis was extremely helpful internally to the development, as a careful attempt to write down what was meant by the technical ideas in the design. In particular, it suggested several covert channels – discussed in Section 7.2, and a closer analysis of the keyboard state. Moreover, the implementations above for policy and the label equivalence relations were just one choice of many, which led to explorations of the alternatives.

The security property above rules out information flows from the user's keyboard input to any but the currently active domain. It is predicated on the assumption of a faithful user who never suffers confusion about which domain is currently active (i.e. always heeds the banner rendered by the CDDC at the top of their screen).

However, importantly, it permits information flows from the user's mouse inputs to all domains. This means that a CDDC implementation that *broadcast* all mouse data to all domains would satisfy the property.

A more nuanced model and security definition might apply a more fine-grained (state-dependent) labelling to the mouse input stream, to allow only the mouse clicks that will cause a domain switch to be labelled Internal. One could then specify that all other mouse input should be directed only to the currently active domain. We leave this exploration for future work.

7. SECURITY ANALYSIS

Achieving perfect security in an MLS user interface is a difficult proposition, in such systems, security is often in conflict with the usability of the user interface. Increased usability is usually obtained through complexity, making it more difficult to evaluate and accredit a solution. The CDDC gives a user most of the functionality they desire, whilst minimising the trusted computing elements. Custom hardware, small trusted modules for composition and switching, and enforced unidirectional flows provide a realistic target for formal analysis and subsequent evaluation as a high assurance device.

The authors believe that usable security, even in the presence of small covert channels is better than perfect, unusable security that drives a user to bypass security mechanisms.

7.1 Architectural Security

Keeping most of the functionality untrusted, including the graphics subsystems, operating environments, and existing commodity computing infrastructure can reduce the total cost of the solution. The most contentious decision is allowing the domain-side software and hence the windowing information to remain untrusted – this also allows the video processing blocks to remain untrusted. Malicious software could attempt to thwart security by sending incorrect information, such as locations of non-windows, or by rapidly changing the size, number, and location of the windows, or by sending no information. The CDDC hardware is trusted to correctly decorate each region and always display the correct active domain banner. Whilst the display might look confusing, the banner can still be trusted and incorrectly drawn windows should alert users to potential issues.

7.2 Covert Channel Analysis

The tension between usability and security also influences the covert channels present in the CDDC. Our goal was to ensure no information flowed between domains, and from a hardware perspective this has been achieved – under the implicit assumption that we trust the external keyboard, mouse and display. A malicious display cannot leak information back to another domain through the CDDC, the threat is equivalent to a malicious display attached to a standalone system. The mouse only provides input to the CDDC, which always displays the correct location of the cursor. A malicious mouse could annoy a user and at worse, provide an increased channel for some automated attacks discussed below. A malicious keyboard could act as a storage channel – recording keystrokes from one domain and replaying them to another domain. The CDDC can mitigate against this by powering down the keyboard between domain switches.

The cursor is the only element that crosses the trust boundary between all the domains – the movement and switching activities are potentially visible to more than one domain. We believe the only timing covert channels present in the system are through the user and the use of this *domain crossing* cursor. Two types of covert channel are explored here, incidental channels and malicious attacks. As the cursor moves across the screen, the mouse position is sent to the *active desktop*. When a domain switch occurs the cursor position is then directed to the newly active domain. Although a domain does not know when it has been switched to, or from, monitoring cursor activity may give some indication.

By examining the cursor position when an incidental domain switch occurs, the previously active domain may gain some knowledge about the positions of windows in the newly active domain. Information can then be transferred from one domain to another encoded within the window positions.

In general the window positions are under user control, and given the likely fidelity of window positions discernible through a domain switch, orchestrating data leakage through this channel would prove difficult.

It is possible to mitigate against this timing channel, for example extra mouse movements could be constantly injected into all non-active domains, this may have unintended

side effects, such as tool tips and other mouse hover functionalities occurring. Other policy-based methods can also prevent or limit these timing covert channels at the expense of usability. For example: the CDDC could only pass through cursor clicks and not cursor movement to the domains; cursor movement could be passed through only when above an identified window; the CDDC could limit the number of domain switches, could enforce switching using physical or virtual buttons, or provide a keyboard shortcut to switch domains. Ideally switching using mouse clicks on windows is the most natural interaction with the CDDC interface and the policy enforced would be a matter for accreditation.

In a more malicious scenario, if a user can be enticed to click somewhere specific in one domain through actions of another domain, then information may be transferred. One way to entice a user to click somewhere specific is to have a malicious non-active domain quickly display and hide a dialog box, hoping it would be composited and visible. A user then attempts to click on the dialog box but it disappears and instead clicks at a specific location in the currently active domain, transferring information between domains.

This activity would be quite orchestrated and user training and awareness should highlight the impact of unusual behaviours in the system.

7.3 Cognitive Threats

Another threat to the CDDC is cognitive, whereby mode confusion may cause a user to act in a way that leaks information either directly, for example by typing in the incorrect domain, or indirectly, for example by incorrectly believing certain information viewed to originate from a specific domain. The ability for a user to successfully operate within such an MLS user interface is part of our ongoing research.

One important cognitive requirement is that a user knows when a domain switch has occurred. At the moment, the CDDC alerts a user to this change by changing the trusted banner. A domain switch may require a more distinctive action, for example, an audible alert, or a flash of the screen.

Specific cognitive threats to the CDDC include spoofing attacks, whereby a domain draws a window within one of its own windows, does not report the sub-window and draws its own decoration the colour of another domain. While a user might click on this content, the trusted banner will always correctly show where the keyboard and mouse are directed.

Malicious software can try and confuse a user by sending wrong window positions, it is hoped this would be obvious to a user, and the trusted banner would still indicate the current context. Domains can also have very small or thinly shaped windows where no content at all can be visible, yet the active domain banner will still be shown.

The domains can also render their own cursors, which may confuse a user if multiple cursors are displayed on the screen. Hardware mitigations might include being able to uniquely identify the CDDC rendered cursor, for example making it flash a specific colour when requested.

These covert channels and cognitive issues are not unique to the CDDC, but need to be carefully considered in any implementation and follow-on deployment. Mitigations include correct user training and awareness, along with careful consideration of any policy-based mechanisms.

8. INTEGRATED APPLICATIONS

The coarse grained window-level composition of the CDDC provides a useful converged MLS desktop interface. Through finer grained composition we can create converged MLS-like applications. In this scenario, sub-elements of an application are provided by each domain, the composition that occurs creates a single MLS looking application on the desktop. Whilst the individual elements of the application are still single-level secure and respond at level to keyboard and mouse interaction, the user cognitively fuses the individual domain elements and can interact as if it were a true MLS application, without mixing the underlying data.

A simple example is shown in Figure 11, where each domain renders part of an email client, when the CDDC composites the regions the user is presented with an MLS-like email client that allows a user to check email from all their isolated domains in a "single" application.

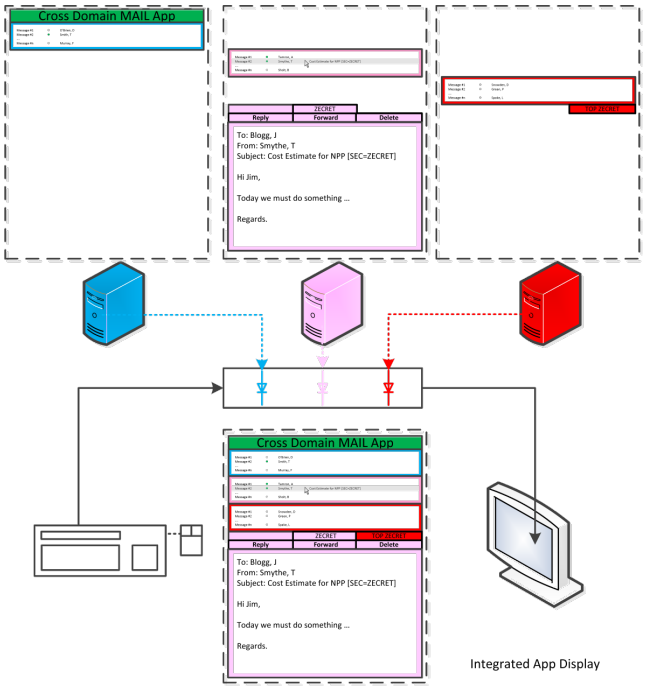

Figure 11: Simple MLS-like email composition

The benefit of this approach is that we do not need to construct, evaluate and accredit a true MLS application. Instead we can use untrusted email clients at-level on each domain. We only need to trust the CDDC to correctly composite the windows identified. Following the mantra of usable security, here we achieve a high degree of the desired functionality of an MLS application, without the burden of verifying the security of a true MLS application.

This approach to MLS application construction is similar in philosophy to the Annex PRISM architecture introduced by Owen et al. in [14]. Here the authors replicated single-level secure applications and then in conjunction with a small TCB created integrated MLS-like applications. These applications provided a consistent user interface, for example through Microsoft Word to some underlying MLS document. The authors termed this Multiple integrated Levels of Security (MiLS), which also seems an apt description of the MLS-like applications that are achieved through our fine-grained composition. One could imagine MiLS applications being constructed for multi-level: database interfaces, logis-

tics clients, service catalogues, and web browsers.

We prototyped a MiLS RSS Reader application on the hardware-based CDDC, where news feeds from multiple security domains are integrated. The user interface and composited construction is shown in Figure 12.

Figure 12: MiLS RSS Reader

The security of the composition remains the same as the coarse grained converged desktop. The user is alerted to the context via the trusted banner and the keyboard and mouse are switched appropriately between domains. The composition is just a carefully contrived geometric arrangement of application fragments. Getting this arrangement correct is key in providing a convincing MLS-like interface for a user. The news feeds on the left composite together for an integrated view, whilst the main story is composited over the top of each other, with the active domain content displayed.

In these simple examples the composition keeps all elements from the untrusted domain software in the same position. A more complex example is presented in Appendix B.

8.1 Managed Information Flows

Visually, the converged desktop provides a convenient user interface. In practice it would be advantageous to overlay some desktop functionality across the domains, albeit in a limited manner. The CDDC has scope and we have demonstrated the ability to provide managed (deliberate) information flows between domains. To copy and paste information from one domain to another, data can be sent in-band from one domain within the video stream, and pasted to another domain in-band in the USB stream, i.e., injecting characters into the USB keyboard stream.

Combining this managed information flow channel with the MiLS applications can provide a very powerful user interface. Further, taking advantage of existing accredited information flow devices (e.g., data diodes, and security gateways) we can provide increased functionality and cross domain integration of our MiLS applications. An extended example is described in Appendix B.

9. FUTURE RESEARCH

Current and future research around the CDDC includes: increasing the flexibility of the CDDC hardware and software ecosystem; increasing the usability and examining how that may be controlled through sensible policy; and experimenting with MiLS applications and managed information flows to construct more tightly integrated applications.

We are investigating a supervisory processor to control aspects of the composition and aid with the orchestration of managed information flows and MiLS applications.

By embedding zero-client functionality into the CDDC and using remote desktop connections, the CDDC could provide, a converged endpoint, ready for connection to extant network infrastructure. The domain-side software already operates over remote desktop links via thin-client endpoints.

CDDC policy could include: the shape of the cursor over a specific domain; operation without decoration (e.g., SCADA environments); view-only domains (e.g., Financial environments); keyboard broadcasting (e.g., lock screen), mouse broadcasting (e.g., multi-domain application launching); and mouse hover over non-active domain (e.g., switch mouse to another domain and not keyboard).

The policy and hardware mechanisms surrounding the managed information flows and the implementation of the copy and paste are ongoing research. MiLS applications with controlled information flows e.g., MiLS wikis with cross-domain hyper-linking are also promising research areas.

Human factors analysis plays an important role in the use and adoption of an MLS user interface technology. We have started investigating the work flow and cognitive threats associated with using the CDDC.

10. CONCLUDING REMARKS

True MLS user interfaces and even constrained implementations rely heavily on trusting large amounts of underlying infrastructure, be it software or hardware, making it difficult to evaluate and accredit these solutions for use in high assurance environments.

The CDDC provides a very small trusted computing base, and functionality built upon the simple premise of hardware-based composition and trustworthy keyboard and mouse switching. The simplicity makes the system amenable to formal analysis, allowing us to reason about security properties of the device, specifically the non-interference properties. The approach removes most of the semantic information before the multi-level data is handled, yet still provides most of the benefits of more integrated solutions.

The CDDC does not provide perfect security, we would contend that most useful systems will not have perfect security. What it does provide is usable security with strong guarantees for certain security properties and known information channels that can be mitigated by policy. The CDDC can be configured to provide a balance between security and usability suitable for the deployed environment.

The CDDC can be built upon to provide unique converged MLS-like applications without the need to trust the construction of the applications to anywhere near the level of traditional MLS software. The notion of MiLS applications is powerful from both a user and a security perspective, allowing us to keep data at-level and have a user operate on that data at the correct level – adding carefully managed information flows when it really is required.

11. REFERENCES

[1] Air Force Research Laboratory AFRL/RIEB. SecureView overview. *http://www.ainfosec.com/wp-content/uploads/2013/10/SecureView_Overview_Master_PA_Cleared_7Oct13.pdf*, October 2013.

[2] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield. Xen and the art of virtualization. *ACM SIGOPS Operating Systems Review*, 37(5):164–177, 2003.

[3] Common Criteria Sponsoring Organisations. *Common Criteria for Information Technology Security Evaluation, Part 3: Security assurance components, Version 3.1 Rev 4*. http://www.commoncriteriaportal.org/cc/, Sept. 2012.

[4] J. Epstein, J. McHugh, R. Pascale, H. Orman, G. Benson, C. Martin, A. Marmor-Squires, B. Danner, and M. Branstad. A prototype B3 trusted X Window System. In *Computer Security Applications Conference, 1991. Proceedings., Seventh Annual*, pages 44–55. IEEE, 1991.

[5] N. Feske and C. Helmuth. A Nitpicker's guide to a minimal-complexity secure GUI. In *Computer Security Applications Conference, 21st Annual*, pages 85–94. IEEE, 2005.

[6] General Dynamics, C4 Systems. Secure virtualisation: Today's reality, 2009. WP-TVE-1-0409.

[7] J. Goguen and J. Meseguer. Security policies and security models. In *Security and Privacy (SP), 1982 IEEE Symposium on*, pages 11–20, Oakland, California, USA, 1982.

[8] D. Hardin, R. Richards, and M. Wilding. High assurance guard for security applications utilizing authentication and authorization services for sources of network data, Nov. 4 2014. US Patent 8,881,260.

[9] M. Kang, A. Moore, and I. Moskowitz. Design and assurance strategy for the NRL Pump. In *High-Assurance Systems Engineering Workshop, 1997., Proceedings*, pages 64–71, Aug 1997.

[10] R. Kerber and B. Globe. Cost of data breach at TJX soars to $256 m. *Boston Globe, http://www.boston.com/business/globe/articles/2007/08/15/cost_of_data_breach_at_tjx_soars_to_256m*, 2007.

[11] T. Murray, D. Matichuk, M. Brassil, P. Gammie, and G. Klein. Noninterference for operating system kernels. In Chris Hawblitzel and Dale Miller, editor, *The Second International Conference on Certified Programs and Proofs*, pages 126–142, Kyoto, Dec. 2012.

[12] T. Nipkow, L. Paulson, and M. Wenzel. Isabelle/HOL: A Proof Assistant for Higher-Order Logic. In *Lecture Notes in Computer Science*, volume 2283. Springer-Verlag, Germany, 2002.

[13] H. Okhravi and D. Nicol. TrustGraph: Trusted graphics subsystem for high assurance systems. In *Computer Security Applications Conference, 2009. ACSAC '09. Annual*, pages 254–265, Dec 2009.

[14] C. Owen, D. Grove, T. Newby, A. Murray, C. North, and M. Pope. PRISM: Program replication and integration for seamless MILS. In *Security and Privacy (SP), 2011 IEEE Symposium on*, pages 281–296. IEEE, 2011.

[15] R. Quinn and B. Kerrigan. Faciltiating user interaction with multiple domains while preventing cross-domain transfer of data, Mar. 13 2013. US Patent App. 13/800,262.

[16] Raytheon Company. Raytheon Trusted Thin Client (rtn_216411.pdf), 2014. http://www.raytheoncyber.com.

[17] J. Rutkowska and R. Wojtczuk. Qubes OS architecture. *Invisible Things Lab Tech Rep*, page 54, 2010.

[18] R. H. Sherman, G. W. Dinolt, and F. Hubbard. Multilevel secure workstation, Dec. 24 1991. US Patent 5,075,884.

[19] Smart Security Labs. K424F-SH Brochure. *http://www.smartavi.com/assets/files/b_K424F_Brochure.pdf*.

[20] A. Soffer and O. Vaisband. Secure KVM device ensuring isolation of host computers, July 1 2014. US Patent 8,769,172.

[21] G. Stoneburner. Developer-focused assurance requirements [Evaluation Assurance Level and Common Criteria for IT system evaluation]. *Computer*, 38(7):91–93, July 2005.

[22] M. Tehranipoor and F. Koushanfar. A survey of hardware trojan taxonomy and detection. *Design Test of Computers, IEEE*, 27(1):10–25, Jan 2010.

[23] J. M. Wing. A symbiotic relationship between formal methods and security. In *Computer Security, Dependability and Assurance: From Needs to Solutions, 1998. Proceedings*, pages 26–38. IEEE, 1998.

[24] Y. Yeh. Triple-triple redundant 777 primary flight computer. In *Aerospace Applications Conference, 1996. Proceedings., 1996 IEEE*, volume 1, pages 293–307 vol.1, Feb 1996.

APPENDIX

A. DECORATION ALGORITHM

The decoration algorithm is important. It allows the hardware, an FPGA in our prototype, to quickly determine if a specific pixel should be decorated with a border, left undecorated, or have no content at all. This needs to be performed in real-time for each pixel location and be based solely on the windowing information provided in-band from each domain.

window region, $w = (x, y, w, h)$
input pixel, $p = (x_1, y_1)$
list of windows, $w_l = [w_1, w_2, .., w_n]$
where: w_k is in front of w_{k-1}

apply function *extend* to create a decoration region:
list of decoration regions, $d_l = [d_1, d_2, .., d_n]$

apply function $include(p, w)$ returning 0 or 1
to check if a pixel is within the window or decoration:
$in\_windows_l = [b_1, b_2, .., b_n]$
$in\_decorations_l = [b_1, b_2, .., b_n]$

the output pixel p_o is then calculated as:
if $in\_decorations_l = in\_windows_l = 0$ **then**
  $p_{decorated} = no\_content$
else if $in\_decorations_l > in\_windows_l$ **then**
  $p_{decorated} = decoration$
else
  $p_{decorated} = window\_content$
end if

B. EXTENDED MILS APPLICATION

Incorporating a frame-buffer in the CDDC would allow input windows to be rearranged, allowing for the construction of more intertwined, and cognitively integrated applications.

We have prototyped a fine-grained composition email client on our software emulator. The emulator was used as it allowed for quick implementations of the buffering and rearranging functions. A screen shot is shown in Figure 13. In this instance the window decoration has been replaced with a decoration colour blob next to the inbox items.

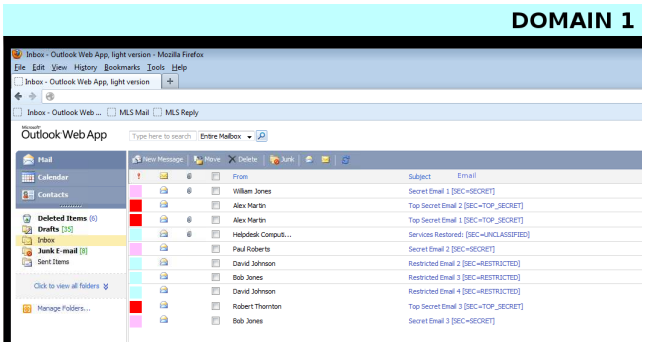

Figure 13: More complex MiLS email composition

Using the CDDC to help orchestrate the MiLS applications can increase their complexity. Examples include: keyboard broadcasting that allows a user to search across all domains at once, or mouse broadcasting to concurrently launch multiple applications across different domains.

The notion of fine-grained MiLS composition is not limited to the CDDC and could be usefully applied to existing secure user interfaces, for examples Qubes OS [17], or Nitpicker [5].

Introducing managed information flows between domains can increase the integration of these applications. We prototyped *replying to* and *forwarding* emails on different domains. This functionality was implemented through CDDC rendered MLS buttons (Figure 14) and external data diodes (Figure 15), allowing for example, an email on DOMAIN 1 to be replied to on DOMAIN 2 by first sending the email from DOMAIN 1 to DOMAIN 2 over the data diode, and then displaying the email on DOMAIN 2. Having all user interaction occur within a single MiLS composited application, provides the integrated feel of a true MLS email application.

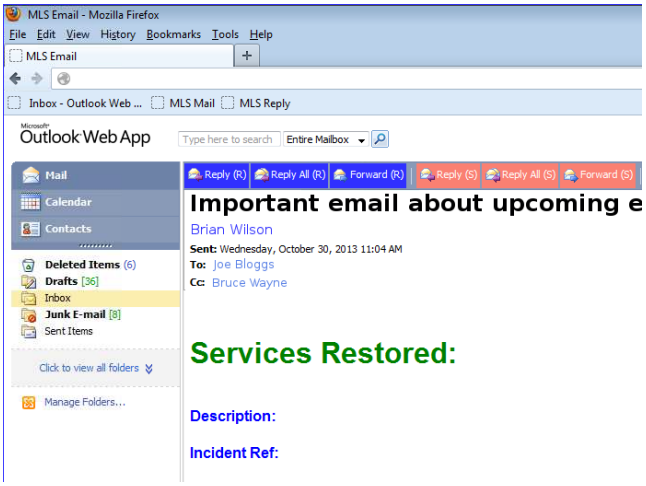

Figure 14: MiLS email composition - *Reply* buttons

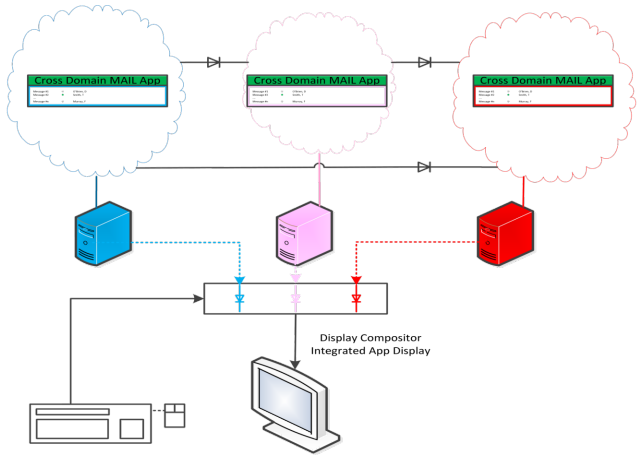

Figure 15: MiLS email composition - external data diodes

Untrusted software is relied upon to send, receive and marshal the data required on each separate security domain. If this software acts maliciously we are presented with the same cognitive threats discussed in Section 7, as well as any at-level attacks the software could normally perform against a single-level secure system.