Exploring Usability and Security of CDDC

Xiaoyu Sun, xsun4@student.unimelb.edu.au
University of Melbourne

Related Work

Multi-Level Secure system promises unprecedented usability and security, especially in high assurance user environment like Defence. This conclusion depends on hard assumptions about the secure user behaviour, which means users would be motivated to protect confidential information due to the typical users like Defence and Intelligence personnel. However, according to the research about the user behaviour [2], this hard assumption cannot be used for normal users with less security conscious. This vulnerability can result in a compromise of the CDDC's security.

The Cross Domain Desktop Compositor (CDDC) is a hardware-based multi-level secure (MLS) user interface which is composited with multiple physical-isolated single-level secure domains. The architecture of system is using mainly commercial-off-the-shelf components complemented by small reliable hardware elements to stay simplicity fulfil high-assurance requirements and achieve a low-cost of accreditation [1].

Problem statement

The extant CDDC interface is hardware-based MLS user interface running on multiple PCs, which cannot be customised to satisfy users requirements. This research seeks to develop an emulator based on the software which has the similar functions to solve this problem.

Project aims

The Cross Domain Desktop Compositor (CDDC) is a multi-level secure user interface through composition of digital display data from multiple physically-isolated single-level secure domains.

This project aims to exploring the usability of CDDC and develop a software emulator which is running on the software-based but with similar functions as CDDC.

Compared with the extant CDDC system, the emulator is software-based running on single computer instead of multiple PCs [2]. The emulator allows users to access different applications within different colours of borders to emulate the operational functions in the MLS system provided by CDDC.

Requirements for the Emulator:

1. Monitoring application windows locations & running status
   The emulator should monitor the applications running on the PC. When the emulator works, it can capture a list of running applications as the windows task management. After that, locations of the application window and the running status (e.g. active window/ inactive window) will be collected for the second step. (The application window will be activated only when the user clicks on it.)
2. Draw decorations/borders around the application windows
   After capturing the application windows positions, the emulator can draw borders for each of the application (e.g. coloured borders around application windows). The colour

of the window border depends on the application running status. For instance, red border for the active application windows, while blue border for the inactive ones.

Overall, the function of the emulator is supposed to be the same with the extant CDDC, which can be leveraged for the further research on investigating users' behaviour or training purposes.

The working process of emulator:

1. Capture a list of the applications/tasks
2. Return process ID & window positions (bounds left/right/bottom/top)
3. Attach the border to the extant windows (Note: Instead of drawing new borders, borders should be attached to application windows. When windows move, borders will move at the same time.)
4. Identify different colour for multiple applications with different status (active/inactive)
5. Customizing the emulator for training/investigating purpose

Trusted composition:

- Decoration of the windows (different colours for different types of applications)
- Active banner decoration should be different from inactive ones
- Switch banner alert

Research Methodology

- Windows APIs: Low-level Microsoft Windows (graphics) APIs
- Development Language: C / C#
- Task Manager

Research contributions

- Provide an experimental platform for users to practice on for further experimentation on investigating the users' behaviour.
- Provide a similar platform for training users before they use CDDC
- Customise the user interface for different purposes
- Provide an alternative user interface of CDDC

Cognitive Threats & Possible Attacks (For future improvements)

- Domain switch alert
- Window switch alert
- Spoofing attacks
- Wrong window position
- Multiple cursors

Reference

1. Beaumont, M., J. McCarthy, and T. Murray. *The Cross Domain Desktop Compositor: Using hardware-based video compositing for a multi-level secure user interface*. in *Proceedings of the 32nd Annual Conference on Computer Security Applications*. 2016. ACM.
2. Issa, A., T. Murray, and G. Ernst. *In search of perfect users: towards understanding the usability of converged multi-level secure user interfaces*. in *Proceedings of the 30th Australian Conference on Computer-Human Interaction*. 2018. ACM.

Code Functions/Methods

1. EnumDelegate (IntPtr hWnd, int lParam)

- Parameters :

  hwnd [in]

  A handle to a top-level window.

  lParam [in]

  The application-defined value given in **EnumWindows** or **EnumDesktopWindows**.

- Return value:
  To continue enumeration, the callback function must return TRUE; to stop enumeration, it must return FALSE.

2. EnumDesktopWindows (IntPtrh Desktop, EnumDelegate lpEnumCallbackFunction, IntPtr lParam)

- Function:

Enumerates all top-level windows associated with the specified desktop. It passes the handle to each window, in turn, to an application-defined callback function.

- Parameters:

  Lpfn

  A pointer to an application-defined EnumWindowsProc callback function.

  lParam

  An application-defined value to be passed to the callback function.

3. GetWindowThreadProcessId

- The return value is the identifier of the thread that created the window.

4. GetWindowRect

- Retrieves the dimensions of the bounding rectangle of the specified window. The dimensions are given in screen coordinates that are relative to the upper-left corner of the screen.

Development Process

PROBLEM: Drawing Borders Attached to Application Windows

SOLUTION 1. Add a bright border around an application window (Currently used)

The Chrome Window: The chrome window is a standard Windows Form whose border is modified to show a specified bright colour and width. The content of the form is removed from inside of the border, leaving the form transparent except for the border itself.

Problems/Concerns:

- Not attached
- The border is separate from the application window

After checked the code and description in the Microsoft documents of this link below:
https://docs.microsoft.com/en-us/lync/desktop/how-to-show-a-bright-border-around-a-locally-shared-resource

The sample code here can solve the problem we have and handle the dragging move and resizing of the application window.
https://code.msdn.microsoft.com/displaying-a-highlighted-30039fd2

Description of the Sample Code: This is a windows application that shows how to draw a border around a process window or desktop. It also handles the situation where, a monitor is added or removed, a process window is moved, minimized, maximized, resized, or closed. This is done by creating a window via windows forms that contains an invisible inner rectangle, and the visible part of the window is a colour of your choosing. This window is then placed on top of the desktop, or outside of a process window.

If the user drags the shared application on the desktop, your bright chrome border needs to move along with the shared application. The same is true with application resizing.

The downloadable sample that **ViewChrome** is taken from has event handling code for these move and resize events. If you want to use a bright border as a UI hint, you should download the sample to see how these event handlers are implemented.

SOLUTION 2. Custom Window Frame Using DWM

A window manager is system software that controls the placement and appearance of windows within a windowing system in a graphical user interface. [1] Most window managers are designed to help provide a desktop environment. They work in conjunction with the **underlying graphical system** that provides required functionality—support for graphics hardware, pointing devices, and a keyboard, and are often written and created using a widget toolkit.

Desktop Window Manager (DWM) APIs: In Windows Vista and later, the appearance of the non-client areas of application windows (the title bar, icon, window border, and caption

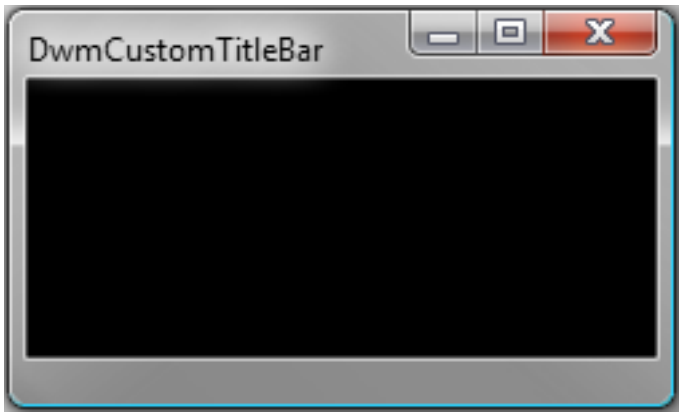
buttons) is controlled by the DWM. Using the DWM APIs, you can change the way the DWM renders a window's frame.

Link: https://docs.microsoft.com/en-us/windows/desktop/dwm/customframe#extending-the-client-frame

- **Steps:**
  - Extending the Client Frame
  - Removing the Standard Frame
  - Drawing in the Extended Frame Window

With the removal of the standard frame, your client area now consists of the entire window, including the extended frame. This includes the region where the caption buttons are drawn. In the following side-by-side comparison, the client area for both the standard frame and the custom extended frame is highlighted in red. The client area for the standard frame window (left) is the black region. On the extended frame window (right), the client area is the entire window.

Because the entire window is your client area, you can simply draw what you want in the extended frame. To add a title to your application, just draw text in the appropriate region. The following image shows themed text drawn on the custom caption frame. The title is drawn using the **DrawThemeTextEx** function. To view the code that paints the title, see Appendix B: Painting the Caption Title.



- **Note:**

When drawing in your custom frame, be careful when placing **UI controls**. Because the entire window is your client region, you must adjust your UI control placement for each frame width if you do not want them to appear on or in the extended frame.

- **Problems/Concerns:**

- Whole window is the client region
- UI controls will be affected (basic functions cannot be implemented)

SOLUTION 3: Using Theme APIs to draw the border of a control

Link: https://blogs.msdn.microsoft.com/dsui_team/2013/06/26/using-theme-apis-to-draw-the-border-of-a-control/

Each window class is responsible for defining its appearance, which includes the client and non-client portions of the window. A window's border is typically part of its non-client area. Windows will handle drawing the border of a window when non-client window messages such as WM_NCPAINT are passed to the DefWindowProc function.

However, **DefWindowProc** will use the "classic" border appearance for child windows. Controls themselves are responsible for drawing their borders if the appearance defined for the current visual style is desired.

SOLUTION 4. Drawing Custom Borders in Windows Forms & Constructing Custom Borders from Bitmap Parts

Part1 Link: http://geekswithblogs.net/kobush/articles/CustomBorderForms.aspx
Part2 Link: http://www.geekswithblogs.net/kobush/articles/CustomBorderForms2.aspx

The code has two primary classes.

1. **FormWithNonClientArea**, extends the standard Form class by adding support for non-client area messages (more on this shortly) and can be uses for various scenarios.
2. **CustomBorderForm**, utilizes these messages and represents a base class for drawing graphical borders composed of several bitmap parts. It also draws a form header including icon, text and form buttons (more on this later). This way I can separate dirty plumbing required for enabling non-client drawing and actual drawing of the graphical elements

(Window ghosting is a Windows Manager feature that lets the user minimize, move, or close the main window of an application that is not responding)

OTHER SOLUTIONS:

- Dialog box
- GtkWindow

A GtkWindow is a toplevel window which can contain other widgets. Windows normally have decorations that are under the control of the windowing system and allow the user to manipulate the window (resize it, move it, close it,...).
Problem**:** Linux Based-consider only Windows system for now
https://developer.gnome.org/gtk3/stable/GtkWindow.html#gtk-window-set-decorated