Compositor security

Contents

The *compositor* is the component of Apertis that is responsible for drawing application windows and other graphical elements on the screen.

In Apertis the compositor runs as the agl-compositor[1] executable from a **systemd user unit** and launches **maynard** as desktop manager. This is a thin executable wrapper around the library **libweston**, which provides the majority of its functionality.

In Wayland, the compositor *is* the display server. Graphical programs arrange for their graphics to be displayed by creating a buffer (a *surface*) in GPU memory, drawing their text, images etc. into that buffer, then sending requests to the Wayland compositor which ask the compositor to include that surface in the final 2D scene. Unprivileged programs cannot display graphics until the compositor is ready, so we can be sure that the compositor's policies are applied to every surface.

We aim to provide the usual security properties described in the Security design document[2]:

- confidentiality
- integrity
- availability

for the two mechanisms provided by the compositor:

- output (placing application windows on the screen)

[1]https://docs.automotivelinux.org/en/master/#5_Component_Documentation/1_agl-compositor/
[2]https://www.apertis.org/concepts/security/

- input (dispatching input events such as touchscreen touches and gestures to applications)

[Wayland Compositors - Why and How to Handle Privileged Clients] provides a good overview of how those security properties apply to compositors.

Use-cases

“The platform”refers to the overall Apertis platform, including the compositor, application manager and so on.

Because we anticipate that the desired graphical presentation and user experience (UX) will be a point of differentiation for OEMs, each of these requirements should be interpreted as a requirement that it is possible for the platform to behave as specified, and a recommendation that OEMs’platform variants should do so unless it conflicts with their desired UX. For example, for brevity, we will use “the compositor must …”as shorthand for “it must be possible for the compositor to …, and we recommend that OEMs’compositors should have that behaviour unless it conflicts with their desired UX”.

Home screen

In some circumstances, such as when the Apertis device is switched on for the first time, it must go into a default state.

- The platform must draw a “home screen”or launcher from which further programs can be launched.
- The home screen may either be part of the compositor, or a separate graphical program.
- Pressing a button or menu entry representing an application entry point[3] results in the relevant graphical program being started.

(These are aspects of input and output availability).

On Apertis, this job is accomplish by **maynard** which is the default desktop manager.

Platform UI elements

In addition to the home screen, there might be UI elements which are outside the scope of any particular application window, such as a status bar, notifications, system-modal dialogs, or the UI controls used for application-switching.

- The OEM-specific visual design might reserve regions of the screen for these visual elements. We recommend that this is done.
  - For example, the equivalent features in Android are the small region at the top of the screen that is normally reserved for the status

[3]https://www.apertis.org/concepts/application-entry-points/

    bar, and the larger region at the bottom or side of the screen that is normally reserved for the navigation bar (Back, Home and Apps buttons).
- The compositor may either draw each of those UI elements itself, or arrange for separate programs to provide them.
- Some of these UI elements must remain visible at all times (they must be displayed on top of ordinary program windows), unless the compositor's UX calls for them to be hidden under certain specific circumstances.
  - For example, Android allows applications to request that the status bar and navigation bar are hidden, but the gestures to reinstate them are always available, and the operating system displays a reminder of those gestures when they become hidden.
- If separate programs provide some or all of these UI elements, then normal platform startup must arrange for them to be launched.

(These are aspects of input and output availability).

The approach used in **agl-compositor** is to support this by providing protocol extensions. Thanks to them, surfaces can have roles, such as popup, fullscreen, split_vertical or split_horizontal and it is also possible to configure them as panels to be always visible and anchored to one of the edges. On Apertis, **maynard** implements these protocol extensions to display the UI elements.

Trusted output

- The compositor must not allow unprivileged programs to display their content in the regions of the screen that are reserved for these UI elements, unless the compositor's UX design specifically allows it. This is a *trusted path* with which the platform can display information to the user. *(Output integrity)*
  - Ideally, the APIs provided to programs should be designed so that it is impossible to request display in a forbidden area.
  - If the APIs provided to programs are such that the program can attempt to display in these regions, and an unprivileged program attempts to do so, this must be detected and prevented.

Since **agl-compositor** is a Wayland compositor, applications cannot request a specific region to display their content. It is the responsibility of the compositor to choose the appropriate place while enforcing its policies.

Launching a program

When a graphical program is launched, after carrying some non-graphical initialization, it will create a surface, fill it with the first frame that it wants to be displayed, and submit that surface to the compositor for display.

- The compositor must be able to identify that surface as having come from that graphical program. In particular, it must be able to determine the

app-bundle[4] and user account[5] that originated the surface. *(Input and output integrity)*
    - *Non-requirement:* If an app-bundle is allowed to contain multiple graphical programs, the ability to distinguish between those graphical programs is optional. We treat the app-bundle as a security boundary, but we do not place a security boundary between individual graphical programs within an app-bundle.
- This identification must be securely authenticated. If a different user account or app-bundle asks to display a surface, one of these options must be true:
    1. (Preferred) The compositor obtains the originating program's user account and app-bundle directly from the Linux kernel or some other trusted platform component, and there is no opportunity for the originating program to give false information.
    2. The originating program tells the compositor which user account and app-bundle it claims to be, and the compositor verifies in a secure way that this claim is true.
    - *Non-requirement:* If an app-bundle is allowed to contain multiple graphical programs and the compositor distinguishes between them, it is acceptable for it to be possible for a graphical program to be able to impersonate a different graphical program in the same bundle.
- The compositor must perform whatever appropriate smooth graphical transition is desired (for example a cross-fade, animated movement, or a simple atomic change between one frame and the next) between the home screen and the graphical program's surface as the main contents of the screen.
- If the compositor's UX involves multiple tiled content areas, the graphical program must be displayed in the desired content area.
- If the compositor's UX involves floating or cascading windows (as seen in GNOME, Windows, etc.), the graphical program must be displayed in the location chosen by the compositor. It may influence that location by setting "hints"in its requests, but the compositor must be free to ignore those hints.
- The compositor must arrange for any platform UI elements that should remain visible at all times to remain visible and interactive during this process *(input and output availability)*:
    - if they are provided by the compositor itself, they must be layered above the graphical program's surfaces in the compositor's scene-graph;
    - if they are provided by a separate "shell"program, the surfaces representing them must be layered above the surfaces from the graphical program.
- The compositor must deliver location-specific input events such as touch-

[4] https://www.apertis.org/glossary/#app-bundle
[5] https://www.apertis.org/glossary/#user-account

screen touches to the application at the relevant location, and to no other application. *(Input availability, input confidentiality)*

- In particular, if application windows can overlap (for example stacking or cascading), and application A is in front of application B, then application A must not be able to trick the user into entering confidential input that was intended for application B by making itself transparent or almost-transparent, so that the user interface of application B shows through (clickjacking[6]). *(Input confidentiality)*
- The compositor must deliver non-location-specific input events such as touchscreen edge-swipe gestures to the current application, using a definition of "current"that is part of its UX, and to no other application. *(Input availability, input confidentiality)*

Thanks to the fact that **agl-compositor** is based on Wayland, an application only controls the contents of its surfaces and the compositor chooses where applications are displayed. That makes input and output availability, as well as input confidentiality easy to ensure.

The Wayland protocol operates via an AF_UNIX socket[7], just like D-Bus, so applications can be identified by their AppArmor profile and uid using the same credentials-passing mechanisms that are already available in D-Bus.

Also, since user applications are meant to be deployed and launched using the Apertis application framework[8], applying the principles described in security boundaries and thread model[9] minimises the risks.

Last-used mode

In some circumstances, such as when the Apertis device is switched off with a particular app active, UX designers may wish to return to a previous saved state, for example one that was saved during device shutdown ("last-used mode").

- The platform must arrange for each of the graphical programs that was previously active and visible (in the foreground) to be restarted.
- When one of those graphical programs asks the compositor to display a surface, the compositor must place it in the same location where it was previously visible.
- The platform may launch other graphical programs that were running but not visible when the state was saved. They must not become visible until the user makes a request to switch to them. Alternatively, the platform may delay starting those graphical programs until the user makes a request to switch to them.

[6]https://en.wikipedia.org/wiki/Clickjacking

[7]http://wayland.freedesktop.org/docs/html/ch04.html

[8]https://www.apertis.org/concepts/application-framework/#the-next-generation-apertis-application-framework

[9]https://www.apertis.org/concepts/security/#security-boundaries-and-threat-model

(Input and output availability)

Main window selection

The user should have the opportunity to switch between the main (top-level) windows presented by various programs.

A graphical program might make it difficult for the user to leave, either accidentally (because the program has become unresponsive) or deliberately as a denial of service (because the program is maliciously written or has been compromised by an attacker).

- The compositor must have the opportunity to intercept input events (touchscreen touches, touchscreen gestures, hardware button presses) regardless of the actions of the program. *(Input availability)*
- The compositor should always provide a way to return to a home screen or application switcher, from which an unresponsive program can be terminated. *(Input and output availability)*
- The way to return to a home screen or application switcher should be consistent and predictable. For example, Android reserves a small area of the screen for Back, Home and Applications buttons. In older Android versions, applications such as the camera may request that these buttons are displayed unobtrusively, but are not able to hide them altogether; in newer versions, these buttons can be hidden, but the swipe gesture to make them available cannot be disabled, and the user is given a reminder of that gesture which cannot be hidden by the application. *(Input availability, output integrity)*
    - Optionally, specially privileged app-bundles might be given the opportunity to hide these UI elements, or arrange for one of the app-bundle's surfaces to be displayed as an overlay "above"them. However, this should be a "red flag"in app-store review, to be granted only to trusted applications.
        * For example, Android requires the SYSTEM_ALERT_WINDOW permission[10] for applications that use overlays, and additionally requires that the user has been specifically prompted by the platform to grant this permission to this app.
- If the compositor receives an input event that it interprets as a request to switch away from the graphical program, for example pressing a "home"or "application switcher"button, then this switch must occur within a reasonable time, even if the current graphical program does not cooperate with that operation. This must have a smooth graphical transition (cross-fade or animation) if that is the desired UX. *(Input and output availability)*
    - For example, if a bug in the current graphical program results in it ceasing to respond to messages from the compositor (for example

[10]https://developer.android.com/reference/android/provider/Settings.html#canDrawOverlays%28android.content.Context%29

    a deadlock or live-lock situation) and the window switching operation involves communicating with it, the compositor must not wait indefinitely for a response. If it gets a response, it may switch immediately; if it does not, it may wait a short time, but after that time it must continue switching anyway. The maximum wait time should be chosen so that switching still appears responsive.
  - Similarly, if the current graphical program is deliberately/maliciously written with the intention of delaying task-switching as much as possible, the compositor must still switch within a reasonable time.
- Each window offered for switching must be associated with the relevant app-bundle, for example with a title and/or icon, so that when the user believes they are switching to a particular window, they can know that they are in fact switching to a window from the correct trust domain. *(Input and output integrity)*
  - The ability to distinguish between windows from different graphical programs in the same app-bundle is optional, because graphical programs in an app-bundle share a trust domain.
- A UX designer might require a limit on the number of simultaneous windows per app-bundle. For example, an app-bundle might be limited to having up to 5 entry points in the same or different processes, each with up to 2 main windows open at any given time.

On Apertis **maynard** displays a panel that cannot be hidden which allows to show the list of available applications and to switch to any of them. It supports `.desktop` files as source for applications metadata.

The compositor enforce timeouts when interacting with surfaces in order to prevent not responding application to compromise user experience.

Child windows

A graphical program might include dialogs[11] in its UX.

- We recommend that dialogs should normally appear as a direct result of user activity, but they may also appear as a result of an external event.
- If the graphical program's corresponding main window is currently displayed in a particular location, the dialog should overlay that location. If the API to open dialogs makes it possible to attempt to place dialogs elsewhere, and the program does so, the compositor must prevent this. *(Output integrity)*
- If surfaces (windows) are tiled, stacked or floating, the dialog may extend outside the boundaries of the graphical program's main window if desired, but we recommend that this pattern is discouraged. If this is done, it should always be made obvious which surface the dialog belongs to. *(Output integrity)*

[11] https://en.wikipedia.org/wiki/Dialog_box

- The dialog must not prevent the user from [leaving the program], even if it extends outside the main window; in other words, it may be app-modal or document-modal, but must not be system-modal. *(Input and output availability)*
- We suggest encouraging the use of document-modal dialogs[12] similar to those in OS X[13] and GNOME[14]

A graphical program might include pop-up or drop-down menus in its UX.

- Menus typically behave like a document-modal window immediately above their "parent"window.
- The requirements are essentially the same as for dialogs, although the visual presentation is likely to be different.

To enforce these rules, **agl-compositor** only supports top-level surfaces and only allows one main surface per application. Under these restrictions, applications requiring additional surfaces need to create them as child objects of the main one.

The support of system popups is implemented through custom protocol extensions which can be only used by privileged applications which implement them, like **maynard** does.

Notifications

External events might result in a *notification*, typically implemented as a "pop-up"window.

- A calendar might trigger notifications as time passes, for example when an appointment will occur soon.
- A messaging application (for example email or Twitter) might trigger a notification when new messages are available.

These notifications should be displayed by the platform user interface (HMI), either as part of the compositor (like in GNOME Shell) or a separate process.

- If there is a current notification, the platform should draw a visual representation of it, displaying it "above"any current window. *(Output availability for the notification)*
- If there is no current notification, any program (including non-graphical programs) may trigger a new notification. *(Output availability for the notification)*
- Each notification should be visually associated with the appropriate app-bundle, perhaps via an icon and title. *(Output integrity)*

[12]https://en.wikipedia.org/wiki/Dialog_box#Document%20modal

[13]https://developer.apple.com/library/mac/documentation/UserExperience/Conceptual/OSXHIGuidelines/WindowDialogs.html#//apple_ref/doc/uid/20000957-CH43-SW2

[14]https://wiki.gnome.org/Design/OS/ModalDialogs

- Notifications should be drawn in such a way that only the compositor (or the trusted notification service, if separate) can produce the same visual result, for example by displaying it over the top of platform UI elements in a way that would not be possible or would not be allowed for an ordinary application window. *(Output integrity)*
- There should be a straightforward mechanism by which the driver can close any notification, minimizing distraction. *(Input and output availability for other UI components)*
- High-priority platform components such as navigation must be able to force their notifications to be displayed instead of, or "above", other components'notifications. *(Output availability for the higher-priority notification)*
- Excessive notifications by an application might be distracting. The compositor must have the opportunity to limit the number of notifications per app-bundle or deny notification display altogether, with an optional user-configurable limit per application so that the user could selectively silence an app-bundle that they found distracting.
- The precise handling of notifications (for example topics such as how multiple simultaneous notifications are handled) is outside the scope of this document.
- If the notification has "actions", for example a button to go to the relevant app-bundle, these actions must be able to bring that app-bundle to the foreground.

In Apertis, the custom `agl-protocol` protocol extensions are supported, providing the basis to implement notifications and display them as part of the system panel. Currently this feature is not implemented.

Focus-stealing

A graphical program might attempt to get the user's attention by creating new main windows while it is in the background.

- These windows must not be displayed or given input focus, to avoid user distraction and focus-stealing[15].
- We recommend encouraging application developers to use notifications instead.
- Some programs ported from non-Apertis environments might rely on the ability to create a window at any time as a way to get the user's attention. If a program does this, the compositor must not display it or give it input focus until the user requests main window switching.
  - The compositor could handle this with no user distraction at all, by making the window available in the main window selection list, but not showing it. However, this would not have the desired effect of informing the user that something has happened.

[15]https://en.wikipedia.org/wiki/Focus_stealing

    - Additionally, the compositor could optionally provide a visual cue to the user while minimizing distraction, by behaving as though that program had requested a notification, with content based on the program and/or window title, and one action button which would bring the new window to the foreground.
- If the window would exceed a limit on the number of simultaneous windows or graphical programs in an app-bundle, as described in main window selection, the compositor must not display those excessive windows, and may terminate the graphical program.

As part of the restrictions imposed by **agl-compositor**, only one main window is allowed per application, so the focus-stealing is not impossible. In order to request user attention, applications should use notifications.

Non-graphical programs

A previously non-graphical program could connect to the display server and create a new main window, becoming a graphical program. This situation leads to similar issues as described in focus-stealing[16].

In order to simplify the design and make it more coherent, applications should be either non-graphical or graphical across all their lifetime. Applications requiring special behaviour should solve this by decoupling their graphical part from the non-graphical by creating a graphical application and a service. In order to use the graphical interface a service can request user attention by using notifications

Screenshots

Screenshots impose a security risk that should be considered.

- A program from one app-bundle must not be able to copy the texture data of a window from a different app-bundle, which might contain confidential information. *(Output confidentiality)*
    - In particular, this forbids taking screenshots of a program from a different app-bundle.
    - The ability for programs in the same app-bundle to take screenshots of each other is optional. For "least-privilege", we suggest that the platform should not allow app-bundles to request that the platform takes a screenshot of that app-bundle. The programs can communicate directly with each other to share their texture data, if desired, so the platform's involvement is not needed.
- A program from an app-bundle must not be able to copy the texture data of platform UI elements, which might contain confidential information. *(Output confidentiality)*
    - In particular, this forbids screenshots again.

[16]https://en.wikipedia.org/wiki/Focus_stealing

- In some scenarios specially privileged app-bundles must be able to take screenshots, bypassing the restrictions mentioned.
- Screencasting or video recording is essentially equivalent to an ongoing stream of screenshots, and has equivalent requirements.

On **agl-compositor** this functionality is provided through protocol extensions that can be implemented by a privileged application.

Synthesized input

- A program from one app-bundle must not be able to synthesize input events for delivery to a window in a different app-bundle, which could be used to force the target program to carry out undesired actions. *(Input integrity)*
- A program from one app-bundle must not be able to synthesize input events for delivery to the compositor, which could be used to force the compositor or other programs to carry out undesired actions. *(Input integrity)*

Trusted input paths

In some situations the platform may need to ask the user for input, in such a way that the user can be confident that their input will in fact go to the platform and not to a potentially malicious app-bundle. One prominent example of a trusted input path is the "Ctrl+Alt+Del to log in"mechanism in Windows operating systems: Windows does not allow ordinary applications to intercept this key sequence, which means that the user can be confident that the resulting login dialog actually belong to Windows, and not an ordinary application that is mimicking it.

GNOME uses *system-modal dialogs* for a similar purpose when carrying out platform-related actions like asking for confirmation[17] of a potentially dangerous system-wide action or when unlocking access to stored passwords[18].

- The compositor must be able to request input from the user regardless of any other factors, for example application windows or notifications. *(Availability, integrity)*
- Other platform components might need to request input from the user in a similar way.
- Unprivileged app-bundles must not be able to make equivalent requests. *(Output integrity; output availability for everything else)*
- The trusted input path must be displayed in such a way that only the compositor or another trusted service can produce the same visual result, for example by displaying it over the top of Platform UI elements in a

[17] https://wiki.gnome.org/Design/OS/AuthorizationDialog
[18] https://wiki.gnome.org/Design/OS/KeyringDialog

way that would not be possible or would not be allowed for an ordinary application window. *(Output integrity, input integrity)*

As previously commented, **agl-compositor** enforces that applications only create one main window and standard popup surfaces are not supported. However, using its protocol extensions, a privileged application can display a popup surface to provide a trusted input path.

Further reading

- Wayland Protocol security and authentication[19]
- Security in Wayland-Based DEs[20]
- Wayland Compositors - Why and How to Handle Privileged Clients[21]
- User Interaction Design for Secure Systems (Ka-ping Yee, 2002)[22]
- The status of Wayland security[23]

[19]https://wayland.freedesktop.org/docs/html/ch04.html#sect-Protocol-Security-and-Authentication

[20]https://www.x.org/wiki/Events/XDC2014/XDC2014DodierPeresSecurity/xorg-talk.pdf

[21]http://www.mupuf.org/blog/2014/02/19/wayland-compositors-why-and-how-to-handle/

[22]http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.65.5837

[23]https://lwn.net/Articles/589147/