



Compositor security

1 Contents

2	Background	2
3	Other platforms	3
4	Use-cases	3
5	Home screen	3
6	Platform UI elements	4
7	Launching a program	5
8	Last-used mode	6
9	Main window selection	7
10	Child windows	8
11	Notifications	9
12	Focus-stealing	10
13	Non-graphical programs	11
14	Screenshots	11
15	Synthesized input	12
16	Trusted input paths	12
17	Interaction with the automotive domain	13
18	Design notes	14
19	Recommendations	14
20	Further reading	14

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

23 Background

24 The *compositor* is a process responsible for combining surfaces (texture buffers)
25 representing application windows into the single 2D image displayed on the
26 screen. In an X11 environment, it combines the roles of a [window manager](#)¹
27 and a [compositing manager](#)². In a Wayland environment, it also takes on the
28 role of the [display server](#)³ from X11.

29 In Apertis 15.12 with either X11 or Wayland, the compositor runs as the `mutter`
30 executable. This is a thin executable wrapper around the library `libmutter`,
31 which provides the majority of its functionality; both of these components are
32 part of GNOME’s Mutter project. Additionally, the `mildenhall-mutter-plugin`
33 component is loaded by the `mutter` executable as a plugin, and provides an
34 Apertis-specific reference user experience (UX). Near-future versions of Apertis
35 might move to a model more like GNOME Shell, where the component respon-
36 sible for the compositor’s UX is a standalone executable linked to `libmutter`,
37 with the equivalent of the code from `mildenhall-mutter-plugin` included in that
38 executable; this would have little effect on the compositor’s behaviour.

39 If Apertis moves from Mutter to Weston as its compositor in a future release,

¹https://en.wikipedia.org/wiki/Window_manager

²https://en.wikipedia.org/wiki/Compositing_window_manager

³https://en.wikipedia.org/wiki/Display_server

40 we anticipate that the UX layer equivalent to mildenhall-mutter-plugin would
41 become a Weston plugin.

42 In X11, unprivileged [graphical programs](#)⁴ cannot display their graphics before
43 the display server has started. Programs arrange for their graphics to be dis-
44 played by connecting to the X11 display server and sending a request to create
45 a window. Such requests are always granted: if the compositor has not yet been
46 started, the X11 display server itself carries out fallback window management
47 behaviour in which the window is displayed with the size and position that the
48 program requested. If the compositor has already been started, the window
49 is not immediately displayed, but is instead made available to the compositor,
50 which may choose whether to composite the window into the final 2D scene (and
51 if so, where to place it).

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

59 We aim to provide the usual security properties described in the [Security design](#)
60 [document](#)⁵:

- 61 • confidentiality
- 62 • integrity
- 63 • availability

64 for the two mechanisms provided by the compositor:

- 65 • output (placing application windows on the screen)
- 66 • input (dispatching input events such as touchscreen touches and gestures
67 to applications)

68 [Wayland Compositors - Why and How to Handle Privileged Clients](#)⁶ provides
69 a good overview of how those security properties apply to compositors.

70 **Other platforms**

71 In GNOME 3 on either Wayland or X11, GNOME Shell is a standalone exe-
72 cutable linked to the libmutter library, similar to the design proposed above.

73 Android's SurfaceFlinger and Windows' Desktop Window Manager also fulfil
74 essentially the same role as our compositor.

⁴<https://em.pages.apertis.org/apertis-website/glossary/#graphical-program>

⁵<https://em.pages.apertis.org/apertis-website/concepts/security/>

⁶<http://www.mupuf.org/blog/2014/02/19/wayland-compositors-why-and-how-to-handle/>

75 **Use-cases**

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

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

86 **Home screen**

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

- 89 • The platform must draw a “home screen” or launcher from which further
90 programs can be launched.
- 91 • The home screen may either be part of the compositor, or a separate
92 graphical program.
- 93 • Pressing a button or menu entry representing an application [entry point](#)⁷
94 results in the relevant graphical program being started.

95 *(These are aspects of input and output availability).*

96 **Platform UI elements**

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

- 100 • The OEM-specific visual design might reserve regions of the screen for
101 these visual elements. We recommend that this is done.
 - 102 – For example, the equivalent features in Android are the small re-
103 gion at the top of the screen that is normally reserved for the status
104 bar, and the larger region at the bottom or side of the screen that
105 is normally reserved for the navigation bar (Back, Home and Apps
106 buttons).
- 107 • The compositor may either draw each of those UI elements itself, or ar-
108 range for separate programs to provide them.
- 109 • Some of these UI elements must remain visible at all times (they must be
110 displayed on top of ordinary program windows), unless the compositor’s
111 UX calls for them to be hidden under certain specific circumstances.

⁷<https://em.pages.apertis.org/apertis-website/concepts/application-entry-points/>

- 112 – For example, Android allows applications to request that the status
113 bar and navigation bar are hidden, but the gestures to reinstate them
114 are always available, and the operating system displays a reminder
115 of those gestures when they become hidden.
- 116 • If separate programs provide some or all of these UI elements, then normal
117 platform startup must arrange for them to be launched.

118 *(These are aspects of input and output availability).*

119 **Trusted output**

- 120 • The compositor must not allow unprivileged programs to display their
121 content in the regions of the screen that are reserved for these UI elements,
122 unless the compositor’s UX design specifically allows it. This is a *trusted*
123 *path* with which the platform can display information to the user. (*Output*
124 *integrity*)
 - 125 – Ideally, the APIs provided to programs should be designed so that it
126 is impossible to request display in a forbidden area.
 - 127 – If the APIs provided to programs are such that the program can
128 attempt to display in these regions, and an unprivileged program
129 attempts to do so, this must be detected and prevented.
- 130 • Trusted paths are discussed in academic security literature, for example
131 [References](#).

132 **Launching a program**

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

- 136 • The compositor must be able to identify that surface as having come from
137 that graphical program. In particular, it must be able to determine the
138 [app-bundle](#)⁸ and [user account](#)⁹ that originated the surface. (*Input and*
139 *output integrity*)
 - 140 – *Non-requirement:* If an app-bundle is allowed to contain multiple
141 graphical programs, the ability to distinguish between those graph-
142 ical programs is optional. We treat the app-bundle as a security
143 boundary, but we do not place a security boundary between individ-
144 ual graphical programs within an app-bundle.
- 145 • This identification must be securely authenticated. If a different user
146 account or app-bundle asks to display a surface, one of these options must
147 be true:
 - 148 1. (Preferred) The compositor obtains the originating program’s user
149 account and app-bundle directly from the Linux kernel or some other

⁸<https://em.pages.apertis.org/apertis-website/glossary/#app-bundle>

⁹<https://em.pages.apertis.org/apertis-website/glossary/#user-account>

- 150 trusted platform component, and there is no opportunity for the
 151 originating program to give false information.
- 152 2. The originating program tells the compositor which user account and
 153 app-bundle it claims to be, and the compositor verifies in a secure
 154 way that this claim is true.
- 155 – *Non-requirement:* If an app-bundle is allowed to contain multiple
 156 graphical programs and the compositor distinguishes between them,
 157 it is acceptable for it to be possible for a graphical program to be able
 158 to impersonate a different graphical program in the same bundle.
 - 159 • The compositor must perform whatever appropriate smooth graphical
 160 transition is desired (for example a cross-fade, animated movement, or
 161 a simple atomic change between one frame and the next) between the
 162 home screen and the graphical program’s surface as the main contents of
 163 the screen.
 - 164 • If the compositor’s UX involves multiple tiled content areas, the graphical
 165 program must be displayed in the desired content area.
 - 166 – *In Wayland, the application only controls the content of its surfaces,*
 167 *and the compositor chooses where they are displayed, so this is easy*
 168 *to ensure.*
 - 169 • If the compositor’s UX involves floating or cascading windows (as seen
 170 in GNOME, Windows, etc.), the graphical program must be displayed in
 171 the location chosen by the compositor. It may influence that location by
 172 setting “hints” in its requests, but the compositor must be free to ignore
 173 those hints.
 - 174 – *Again, this is how Wayland always works in any case.*
 - 175 • The compositor must arrange for any **UI elements that should remain**
 176 **visible at all times** to remain visible and interactive during this process
 177 (*input and output availability*):
 - 178 – if they are provided by the compositor itself, they must be layered
 179 above the graphical program’s surfaces in the compositor’s scene-
 180 graph;
 - 181 – if they are provided by a separate “shell” program, the surfaces repre-
 182 senting them must be layered above the surfaces from the graphical
 183 program.
 - 184 • The compositor must deliver location-specific input events such as touch-
 185 screen touches to the application at the relevant location, and to no other
 186 application. (*Input availability, input confidentiality*)
 - 187 • In particular, if application windows can overlap (for example stacking or
 188 cascading), and application A is in front of application B, then application
 189 A must not be able to trick the user into entering confidential input that
 190 was intended for application B by making itself transparent or almost-
 191 transparent, so that the user interface of application B shows through
 192 (**clickjacking**¹⁰). (*Input confidentiality*)
 - 193 • The compositor must deliver non-location-specific input events such as

¹⁰<https://en.wikipedia.org/wiki/Clickjacking>

194 touchscreen edge-swipe gestures to the current application, using a defi-
195 nition of “current” that is part of its UX, and to no other application.
196 (*Input availability, input confidentiality*)

197 **Last-used mode**

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

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

211 (*Input and output availability*)

212 **Main window selection**

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

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

- 219 • The compositor must have the opportunity to intercept input events
220 (touchscreen touches, touchscreen gestures, hardware button presses)
221 regardless of the actions of the program. (*Input availability*)
- 222 • The compositor should always provide a way to return to a home screen
223 or application switcher, from which an unresponsive program can be ter-
224 minated. (*Input and output availability*)
- 225 • The way to return to a home screen or application switcher should be
226 consistent and predictable. For example, Android reserves a small area of
227 the screen for Back, Home and Applications buttons. In older Android
228 versions, applications such as the camera may request that these buttons
229 are displayed unobtrusively, but are not able to hide them altogether; in
230 newer versions, these buttons can be hidden, but the swipe gesture to make
231 them available cannot be disabled, and the user is given a reminder of that
232 gesture which cannot be hidden by the application. (*Input availability,*
233 *output integrity*)

- 234 – Optionally, specially privileged app-bundles might be given the
235 opportunity to hide these UI elements, or arrange for one of the
236 app-bundle’s surfaces to be displayed as an overlay “above” them.
237 However, this should be a “red flag” in app-store review, to be
238 granted only to trusted applications. For example, Android requires
239 the `SYSTEM_ALERT_WINDOW` permission¹¹ for applications
240 that use overlays, and additionally requires that the user has been
241 specifically prompted by the platform to grant this permission to
242 this app. (*Output integrity*)
- 243 • If the compositor receives an input event that it interprets as a request to
244 switch away from the graphical program, for example pressing a “home”
245 or “application switcher” button, then this switch must occur within a
246 reasonable time, even if the current graphical program does not cooperate
247 with that operation. This must have a smooth graphical transition (cross-
248 fade or animation) if that is the desired UX. (*Input and output availability*)
 - 249 – For example, if a bug in the current graphical program results in
250 it ceasing to respond to messages from the compositor (for example
251 a deadlock or live-lock situation) and the window switching opera-
252 tion involves communicating with it, the compositor must not wait
253 indefinitely for a response. If it gets a response, it may switch imme-
254 diately; if it does not, it may wait a short time, but after that time
255 it must continue switching anyway. The maximum wait time should
256 be chosen so that switching still appears responsive.
 - 257 – Similarly, if the current graphical program is deliberately/maliciously
258 written with the intention of delaying task-switching as much as pos-
259 sible, the compositor must still switch within a reasonable time.
 - 260 • Each window offered for switching must be associated with the relevant
261 app-bundle, for example with a title and/or icon, so that when the user
262 believes they are switching to a particular window, they can know that
263 they are in fact switching to a window from the correct trust domain.
264 (*Input and output integrity*)
 - 265 – The ability to distinguish between windows from different graphical
266 programs in the same app-bundle is optional, because graphical pro-
267 grams in an app-bundle share a trust domain.
 - 268 • A UX designer might require a limit on the number of simultaneous win-
269 dows per app-bundle. For example, an app-bundle might be limited to
270 having up to 5 entry points in the same or different processes, each with
271 up to 2 main windows open at any given time.

272 Child windows

273 A graphical program might include `dialogs`¹² in its UX.

¹¹<https://developer.android.com/reference/android/provider/Settings.html#canDrawOverlays%28android.content.Context%29>

¹²https://en.wikipedia.org/wiki/Dialog_box

- 274 • We recommend that dialogs should normally appear as a direct result of
- 275 user activity, but they may also appear as a result of an external event.
- 276 • If the graphical program’s corresponding main window is currently dis-
- 277 played in a particular location, the dialog should overlay that location.
- 278 If the API to open dialogs makes it possible to attempt to place dialogs
- 279 elsewhere, and the program does so, the compositor must prevent this.
- 280 (*Output integrity*)
- 281 • If surfaces (windows) are tiled, stacked or floating, the dialog may ext-
- 282 end outside the boundaries of the graphical program’s main window if
- 283 desired, but we recommend that this pattern is discouraged. If this is
- 284 done, it should always be made obvious which surface the dialog belongs
- 285 to. (*Output integrity*)
- 286 • The dialog must not prevent the user from **switching away from the pro-**
- 287 **gram**, even if it extends outside the main window; in other words, it may
- 288 be app-modal or document-modal, but must not be system-modal. (*Input*
- 289 *and output availability*)
- 290 • We suggest encouraging the use of **document-modal dialogs**¹³ similar to
- 291 those in **OS X**¹⁴ and **GNOME**¹⁵.

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

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

297 Notifications

298 External events might result in a *notification*, typically implemented as a “pop-

299 up” window.

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

304 These notifications should be displayed by the platform user interface (HMI),

305 either as part of the compositor (like in GNOME Shell) or a separate process.

- 306 • If there is a current notification, the platform should draw a visual repre-
- 307 sentation of it, displaying it “above” any current window. (*Output avail-*
- 308 *ability for the notification*)
- 309 • If there is no current notification, any program (including non-graphical
- 310 programs such as agents) may trigger a new notification. (*Output avail-*
- 311 *ability for the notification*)

¹³https://en.wikipedia.org/wiki/Dialog_box#Document%20modal

¹⁴https://developer.apple.com/library/mac/documentation/UserExperience/Conceptual/OSXHIGuidelines/WindowDialogs.html#//apple_ref/doc/uid/20000957-CH43-SW2

¹⁵<https://wiki.gnome.org/Design/OS/ModalDialogs>

- 312 • Each notification should be visually associated with the appropriate app-
313 bundle, perhaps via an icon and title. (*Output integrity*)
- 314 • Notifications should be drawn in such a way that only the compositor (or
315 the trusted notification service, if separate) can produce the same visual
316 result, for example by displaying it over the top of **Platform UI elements** in
317 a way that would not be possible or would not be allowed for an ordinary
318 application window. (*Output integrity*)
 - 319 – As with **Platform UI elements**, this is an example of trusted path
320 graphics; see **References**.
- 321 • There should be a straightforward mechanism by which the driver can close
322 any notification, minimizing distraction. (*Input and output availability for
323 other UI components*)
- 324 • High-priority platform components such as navigation must be able to
325 force their notifications to be displayed instead of, or “above”, other com-
326 ponents’ notifications. (*Output availability for the higher-priority noti-
327 fication*)
- 328 • Excessive notifications by an application might distract the driver. The
329 compositor must have the opportunity to limit the number of notifications
330 per app-bundle or deny notification display altogether, with an optional
331 user-configurable limit per application so that the user could selectively
332 silence an app-bundle that they found distracting.
- 333 • The precise handling of notifications (for example topics such as how mul-
334 tiple simultaneous notifications are handled) is outside the scope of this
335 document.
- 336 • If the notification has “actions”, for example a button to go to the relevant
337 app-bundle, these actions must be able to bring that app-bundle to the
338 foreground.

339 GNOME’s design page for notifications, in addition to **GNOME’s own designs**¹⁶,
340 has **some useful references to other platforms in the “See Also” section**¹⁷.

341 **Focus-stealing**

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

- 344 • These windows must not be displayed or given input focus, to avoid user
345 distraction and **focus-stealing**¹⁸.
- 346 • We recommend encouraging application developers to use **Notifications**
347 instead.
- 348 • Some programs ported from non-Apertis environments might rely on the
349 ability to create a window at any time as a way to get the user’s attention.
350 If a program does this, the compositor must not display it or give it input
351 focus until the user requests **main window switching**.

¹⁶<https://wiki.gnome.org/Design/OS/Notifications>

¹⁷https://wiki.gnome.org/Design/OS/Notifications#See_Also

¹⁸https://en.wikipedia.org/wiki/Focus_stealing

- 352 – The compositor could handle this with no user distraction at all, by
353 making the window available in the **Main window selection** list, but
354 not showing it. However, this would not have the desired effect of
355 informing the user that something has happened.
- 356 – Additionally, the compositor could optionally provide a visual cue to
357 the user while minimizing distraction, by behaving as though that
358 program had requested a notification, with content based on the pro-
359 gram and/or window title, and one action button which would bring
360 the new window to the foreground.
- 361 • If the window would exceed a limit on the number of simultaneous windows
362 or graphical programs in an app-bundle, as described in **Main window**
363 **selection**, the compositor must not display those excessive windows, and
364 may terminate the graphical program.

365 **Non-graphical programs**

366 A previously non-graphical program could connect to the display server and
367 create a new main window, becoming a graphical program.

- 368 • **Unresolved:** what happens?
 - 369 – The simplest resolution would be to treat it as though it had always
370 been graphical and was previously in the background, and apply the
371 **Focus-stealing** requirements to it. Is this sufficient?
 - 372 – If there is a requirement that we are able to classify programs into
373 (potentially) graphical and non-graphical in the manifest, with only
374 graphical programs allowed to open windows, this would somewhat
375 undermine the idea that there is no security boundary within an
376 app-bundle.
- 377 • If the window creation is allowed, it must be treated as though a graphical
378 program in the background had opened that window, for the purposes of
379 **Focus-stealing** prevention.

380 **Screenshots**

- 381 • A program from one app-bundle must not be able to copy the texture data
382 of a window from a different app-bundle, which might contain confidential
383 information. (*Output confidentiality*)
 - 384 – In particular, this forbids taking screenshots of a program from a
385 different app-bundle.
 - 386 – The ability for programs in the same app-bundle to take screenshots
387 of each other is optional. For “least-privilege”, we suggest that the
388 platform should not allow app-bundles to request that the platform
389 takes a screenshot of that app-bundle. The programs can communi-
390 cate directly with each other to share their texture data, if desired,
391 so the platform’s involvement is not needed.
- 392 • A program from an app-bundle must not be able to copy the texture data
393 of platform UI elements, which might contain confidential information.

394 (*Output confidentiality*)

395 – In particular, this forbids screenshots again.

- 396 • **Unresolved:** Is there a requirement that specially privileged app-bundles
397 must be able to take screenshots, bypassing these restrictions?
 - 398 – If this is required, we suggest an interface similar to GNOME Shell’s
399 org.gnome.Shell.Screenshot D-Bus API, with which these privileged
400 app-bundles can submit a request to the compositor, which the com-
401 positor can accept or reject according to the permissions flags in that
402 app-bundle’s manifest.
- 403 • Screencasting or video recording is essentially equivalent to an ongoing
404 stream of screenshots, and has equivalent requirements.

405 **Synthesized input**

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

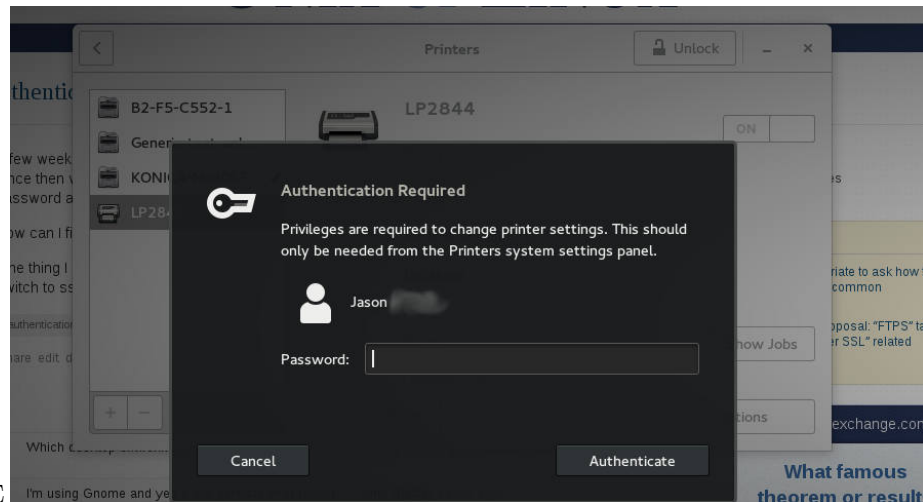
414 **Trusted input paths**

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

423 GNOME uses *system-modal dialogs* for a similar purpose when carrying out
424 platform-related actions like [asking for confirmation of a potentially dangerous](#)
425 [system-wide action](#)¹⁹ or [when unlocking access to stored passwords](#)²⁰: to make
426 it more difficult for an ordinary application to present the same visual effect,

¹⁹<https://wiki.gnome.org/Design/OS/AuthorizationDialog>

²⁰<https://wiki.gnome.org/Design/OS/KeyringDialog>



427 GNOME

- 428 • The compositor must be able to request input from the user regardless of
- 429 any other factors, for example application windows or notifications. For
- 430 example, if this is done via system-modal dialogs like the ones in GNOME,
- 431 then the system-modal dialog must replace or be displayed “above” all
- 432 application or notification windows. (*Availability, integrity*)
- 433 • Other platform components might need to request input from the user in
- 434 a similar way.
 - 435 – We anticipate that this would be implemented by providing a privileged
 - 436 API on the compositor that is only accessible by those components.
- 437 • Unprivileged app-bundles must not be able to make equivalent requests.
- 438 (*Output integrity; output availability for everything else*)
- 439 • The trusted input path must be displayed in such a way that only the
- 440 compositor or another trusted service can produce the same visual result,
- 441 for example by displaying it over the top of Platform UI elements in a
- 442 way that would not be possible or would not be allowed for an ordinary
- 443 application window. (*Output integrity, input integrity*)
- 444 – This is an example of a trusted output path; see References.

445 Interaction with the automotive domain

446 If the Apertis device (infotainment domain, CE domain) shares its input and out-
 447 put device with a separate automotive domain²¹, graphics from the automotive
 448 domain must in general be displayed “above” anything from the infotainment
 449 domain. As an exception, if the relevant surfaces in the automotive domain
 450 are associated with something for which input and output availability and in-
 451 tegrity does not need to be preserved against a potentially hostile infotainment
 452 domain, they may be displayed differently. For example, if the main navigation
 453 view in a navigation app is to be displayed by the automotive domain, it could

²¹<https://em.pages.apertis.org/apertis-website/glossary/#automotive-domain>

454 be displayed in the same way as an ordinary app window originating from the
455 infotainment domain.

456 The requirements in this document can be re-stated for the compositor in the
457 automotive domain, with the infotainment domain taking on the role of an ordi-
458 nary application from the automotive compositor’s point of view. For example,
459 **Synthesized input** requires that ordinary applications cannot send input events
460 to the infotainment compositor or to each other. The corresponding require-
461 ment for the automotive compositor is that the infotainment domain must not
462 be able to send input to the automotive compositor, or to another client of the
463 automotive compositor (if there are others).

464 The UX of the automotive domain might reserve particular areas of the screen for
465 platform UI and/or a trusted path. If it does, the compositor in the infotainment
466 domain must avoid relying those areas for its own UX (either for application
467 windows or its own platform UI), because they would never be visible in practice:
468 the automotive domain would draw its UI elements “above” the output of the
469 compositor.

470 **Unresolved:** Are trusted input and output paths to the automotive domain
471 within the scope of this document?

472 Design notes

473 Some of these requirements are [known to be impossible to meet in X11](#)²², so we
474 do not aim to solve them there.

475 Platform features which are likely to be useful in implementing this:

- 476 • Wayland surfaces are not displayed unless the compositor chooses to do
477 so. If we can write down whatever policy is required for a particular UX,
478 then the compositor can be programmed to have exactly that policy.
- 479 • The Wayland protocol operates [via an AF_UNIX socket](#)²³, just like D-
480 Bus, so we can identify peer applications by their AppArmor profile and
481 uid using the same credentials-passing mechanisms that we already use in
482 D-Bus.
 - 483 – Wayland already has API for the uid/gid/pid. Similar API for the
484 LSM context should be straightforward to add.

485 Recommendations

486 *not yet written*

²²<http://www.mupuf.org/blog/2014/02/19/wayland-compositors-why-and-how-to-handle/>

²³<http://wayland.freedesktop.org/docs/html/ch04.html>

487 **Further reading**

- 488 • [http://www.mupuf.org/blog/2014/02/19/wayland-compositors-why-](http://www.mupuf.org/blog/2014/02/19/wayland-compositors-why-and-how-to-handle/)
489 [and-how-to-handle/](http://www.mupuf.org/blog/2014/02/19/wayland-compositors-why-and-how-to-handle/)
- 490 • Trusted paths: for example User Interaction Design for Secure Systems
491 (Ka-ping Yee, 2002)²⁴

²⁴<http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.65.5837>