



Security

1	Contents	
2	Terminology	3
3	Privilege	3
4	Trust	3
5	Integrity, confidentiality and availability	3
6	Security boundaries and threat model	4
7	Security between applications	4
8	Communication between applications	5
9	Security between users	6
10	Security between platform services	6
11	Security between the device and the network	7
12	Physical security	7
13	Solutions adopted by popular platforms	7
14	Android	8
15	Bada	10
16	iOS	10
17	Mandatory Access Control	11
18	Linux Security Modules (LSM)	11
19	Comparison	15
20	Performance impact	17
21	Conclusion	18
22	polkit (PolicyKit)	24
23	Motivation for polkit	24
24	polkit's solution	26
25	Recommendation	27
26	Resource Usage Control	27
27	Imposing limits on I/O for block devices	28
28	Network filtering	29
29	Protecting the driver assistance system from attacks	29
30	Protecting devices whose usage is restricted	30
31	Protecting the system from Internet threats	30
32	Other sources of potential exploitation	32
33	Secure Software Distribution	32
34	Secure Boot	33
35	Data encryption and removal	34
36	Data encryption	34
37	Data removal	35
38	Stack Protection	35
39	Confining applications in containers	35
40	LXC Containment	35
41	The Flatpak framework	37
42	The IMA Linux Integrity Subsystem	37
43	Conclusion regarding IMA and EVM	38
44	Seccomp	39
45	The role of the app store process for security	40

46	How does security affect developer usage of a device?	41
47	Further discussion	41
48	This document discusses and details solutions for the security requirements of	
49	the Apertis system.	
50	Security boundaries and threat model describes the various aspects of the secu-	
51	rity model, and the threat model for each.	
52	Local attacks to obtain private data or damage the system, including those	
53	performed by malicious applications that get installed in the device somehow	
54	or through exploiting a vulnerable application are covered in Mandatory access	
55	control (MAC). It is also the main line of defense against malicious email attach-	
56	ments and web content, and for minimizing the damage that root is able to do	
57	are also mainly covered by the MAC infrastructure. This is the main security	
58	infrastructure of the system, and the depth of the discussion is proportional to	
59	its importance.	
60	Denial of Service attacks through abuse of system resources such as CPU and	
61	memory are covered by Resource usage control . Attacks coming in through	
62	the device's network connections and possible strategies for firewall setup are	
63	covered in Network filtering	
64	Attacks to the driver assistance system coming from the infotainment system are	
65	handled by many of these security components, so it is discussed in a separate	
66	section: Protecting the driver assistance system from attacks . Internet threats	
67	are the main subject of 10, Protecting the system from internet threats .	
68	Secure software distribution discusses how to provide ways to make installing	
69	and upgrade software secure, by guaranteeing packages are unchanged, undam-	
70	aged and coming from a trusted repository.	
71	Secure boot for protecting the system against attacks done by having physical	
72	access to the device is discussed in Secure boot . Data encryption and removal ,	
73	is concerned with features whose main focus is to protect the privacy of the	
74	user.	
75	Stack protection , discusses simple but effective techniques that can be used	
76	to harden applications and prevent exploitation of vulnerabilities. Confining	
77	applications in containers , discusses the pros and cons of using the lightweight	
78	Linux Containers infrastructure for a system like Apertis.	
79	The IMA Linux integrity subsystem , wraps up this document by discussing how	
80	the Integrity Measurement Architecture works and what features it brings to	
81	the table, and at what cost.	

82 Terminology

83 Privilege

84 A component that is able to access data that other components cannot is said
85 to be *privileged*. If two components have different privileges – that is, at least
86 one of them can do something that the other cannot – then there is said to be
87 a *privilege boundary* between them.

88 Trust

89 A *trusted* component is a component that is technically able to violate the secu-
90 rity model (i.e. it is relied on to enforce a privilege boundary), such that errors
91 or malicious actions in that component could undermine the security model.
92 The *trusted computing base (TCB)* is the set of trusted components. This
93 is independent of its quality of implementation – it is a property of whether the
94 component is relied on in practice, and not a property of whether the component
95 is *trustworthy*, i.e. safe to rely on. For a system to be secure, it is necessary
96 that all of its trusted components be trustworthy.

97 One subtlety of Apertis’ [app-centric design](#)¹ is that there is a privilege boundary
98 between *application bundles* even within the context of one user. As a result, a
99 multi-user design has two main layers in its security model: system-level security
100 that protects users from each other, and user-level security that protects a user’s
101 apps from each other. Where we need to distinguish between those layers, we
102 will refer to the *TCB for security between users* or the *TCB for security*
103 *between app bundles* respectively.

104 Integrity, confidentiality and availability

105 Many documents discussing security policies divide the desired security proper-
106 ties into integrity, confidentiality and availability. The definitions used here are
107 taken from the USA National Information Assurance Glossary.

108 Committee on National Security Systems, CNSS Instruction No.
109 4009 National Information Assurance (IA) Glossary, April 2010.
110 http://www.ncsc.gov/publications/policy/docs/CNSSI_4009.pdf

111 *Integrity* is the property that data has not been changed, destroyed, or lost in
112 an unauthorized or accidental manner. For example, if a malicious application
113 altered the user’s contact list, that would be an integrity failure.

114 *Confidentiality* is the property that information is not disclosed to system
115 entities (users, processes, devices) unless they have been authorized to access
116 the information. For example, if a malicious application sent the user’s contact
117 list to the Internet, that would be a confidentiality failure.

¹<https://em.pages.apertis.org/apertis-website/concepts/applications/>

118 **Availability** is the property of being accessible and usable upon demand by
119 an authorized entity. For example, if an application used so much CPU time,
120 memory or disk space that the system became unusable (a denial of service
121 attack), or if a security mechanism incorrectly denied access to an authorized
122 entity, that would be an availability failure.

123 **Security boundaries and threat model**

124 This section discusses the security properties that we aim to provide.

125 **Security between applications**

126 The Apertis platform provides for installation of *application bundles*, which may
127 come from the platform developer or third parties. These are described in the
128 Applications design document.

129 Our model is that there is a trust boundary between these application bun-
130 dles, providing confidentiality, integrity and availability. In other words, an
131 application bundle should not normally be able to read data stored by another
132 application bundle, alter or delete data stored by the other application bundle,
133 or interfere with the operation of the other application bundle. As a necessary
134 prerequisite for those properties, processes from an application bundle must not
135 be able to gain the effective privileges of processes or programs from another
136 application bundle (privilege escalation).

137 In addition to the application bundles, the Apertis *platform* (defined in the Ap-
138 plications design document, and including libraries, system services, and any
139 user-level services that are independent of application bundles) has higher priv-
140 ilege than any particular application bundle. Similarly, an application bundle
141 should not in general be able to read, alter or delete non-application data stored
142 by the platform, except for where the application bundle has been granted per-
143 mission to do so, such as a navigation application reading location data (a
144 “least-privilege” approach); and the application bundle must not be able to gain
145 the effective privileges of processes or programs from the platform.

146 The threat model here is to assume that a user installs a malicious application,
147 or an application that has a security flaw leading to an attacker being able to
148 gain control over it. The attacker is presumed to be able to execute arbitrary
149 code in the context of the application.

150 Our requirement is that the damage that can be done by such applications is
151 limited to: reading files that are non-sensitive (such as read-only OS resources)
152 or are specifically shared between applications; editing or deleting files that
153 are specifically shared between applications; reducing system performance, but
154 to a sufficiently limited extent that the user is able to recover by terminating
155 or uninstalling the malicious or flawed application; or taking actions that the
156 application requires for its normal operation.

157 Some files, particularly large media files such as music, might be specif-
158 ically shared between applications; such files do not have any integrity,
159 confidentiality or availability guarantees against a malicious or subverted
160 application. This is a trade-off for usability, similar to Android’s Environ-
161 ment.getExternalStorageDirectory().

162 To apply this security model to new platform services, it is necessary for those
163 platform services to have a coherent security model, which can be obtained by
164 classifying any data stored by those platform services using questions similar to
165 these:

- 166 • Can it be read by all applications, applications with a specific privilege
167 flag, specific applications (for example the application that created it), or
168 by some combination of those?
- 169 • Can it be written by all applications, applications with a specific privilege
170 flag, specific applications, or some combination of those?

171 It is also necessary to consider whether data stored by different users using the
172 same application must be separated (see [Security between users](#)).

173 For example, a platform service for downloads might have the policy that each
174 application’s download history can be read by the matching application, or by
175 applications with a “Manage Downloads” privilege (which might for instance be
176 granted to a platform Settings application).

177 As another example, a platform service for app-bundle installation might have
178 a policy stating that the trusted “Application Installer” HMI is the only com-
179 ponent permitted to install or remove app-bundles. Depending on the desired
180 trade-off between privacy and flexibility, the policy might be that any appli-
181 cation may read the list of installed app-bundles, that only trusted platform
182 services may read the list of installed app-bundles, or that any application may
183 obtain a subset of the list (bundles that are considered non-sensitive) but only
184 trusted platform services may read the full list.

185 A service can be considered to be secure if it implements its security policy as
186 designed, and that security policy is appropriate to the platform’s requirements.

187 **Communication between applications**

188 In a system that supports capabilities such as data handover between applica-
189 tions, it is likely that pairs of application bundles can communicate with each
190 other, either mediated by platform services or directly. The [Interface Discov-
191 erty](#)² and [Data Sharing](#)³ designs on the Apertis wiki have more information on
192 this topic.

193 The mechanisms for communicating between application bundles, or between
194 application bundle and the platform, are to be classified into *public* and *non-*

²https://em.pages.apertis.org/apertis-website/concepts/interface_discovery/

³https://em.pages.apertis.org/apertis-website/architecture/data_sharing/

195 *public* interfaces. Application bundles may enumerate all of the providers of
196 *public* interfaces and may communicate with those providers, but it is not accept-
197 able for application bundles to enumerate or communicate with the providers
198 of *non-public* interfaces. The platform is considered to be trusted, and may
199 communicate with any *public* or *non-public* interface.

200 The security policy described here is one of many possible policies that can be
201 implemented via the same mechanisms, and could be replaced or extended with
202 a finer-grained security policy at a later date, for example one where applications
203 can be granted the capability to communicate with some but not all non-public
204 interfaces.

205 **Security between users**

206 The Apertis platform is potentially a multi-user environment; see the [Multiuser](#)⁴
207 design document for full details. This results in a two-level hierarchy: users are
208 protected from each other, and within the context of a user, apps are protected
209 from other apps.

210 In at least some of the possible multi-user models described in the Multiuser
211 design document, there is a trust boundary between users, again providing confi-
212 dentiality, integrity and availability (see above). Once again, privilege escalation
213 must be avoided.

214 As with security between applications, some files (perhaps the same files that are
215 shared between applications) might be specifically shared between users. Such
216 files do not have any integrity, confidentiality or availability guarantees against
217 a malicious user. Android’s `Environment.getExternalStorageDirectory()` is one
218 example of a storage area shared by both applications and users.

219 **Security between platform services**

220 Within the platform, not all services and components require the same access
221 to platform data.

222 Some platform components, notably the Linux kernel, are sufficiently highly-
223 privileged that it does not make sense to attempt to restrict them, because
224 carrying out their normal functionality requires sufficiently broad access that
225 they can violate one of the layers of the security model. As noted in [Terminology](#),
226 these components are said to be part of the *trusted computing base* for that layer;
227 the number and size of these components should be minimized, to reduce the
228 exposure of the system as a whole.

229 The remaining platform components have considerations similar to those ap-
230 plied to applications: they should have “least privilege”. Because platform com-
231 ponents are part of the operating system image, they can be assumed not to be
232 malicious; however, it is desirable to have “defence in depth” against design or

⁴<https://em.pages.apertis.org/apertis-website/concepts/multiuser/>

233 implementation flaws that might allow an attacker to gain control of them. As
234 such, the threat model for these components is that we assume an attacker gains
235 control over the component (arbitrary code execution), and the desired property
236 is that the integrity, confidentiality and availability impact is minimized, given
237 the constraint that the component's privileges must be sufficient for it to carry
238 out its normal operation.

239 Note that the concept of the trusted computing base applies to each of the two
240 layers of the security policy. A system service that communicates with all users
241 might be part of the TCB for isolation between users, but not part of the TCB
242 for isolation between platform components or between applications. Conversely,
243 a per-user service such as dconf might be part of the TCB for isolation between
244 applications, but not part of the TCB for isolation between users. The Linux
245 kernel is one example of a component that is part of the TCB for both layers.

246 **Security between the device and the network**

247 Apertis devices may be connected to the Internet, and should protect confiden-
248 tiality and integrity of data stored on the Apertis device. The threat model
249 here is that an attacker controls the network between the Apertis device and
250 any Internet service of interest, and may eavesdrop on network traffic (passive
251 attack) and/or substitute spoofed network traffic (active attack); we assume
252 that the attacker does not initially control platform or application code running
253 on the Apertis device. Our requirement is that normal operation of the Apertis
254 device does not result in the attacker gaining the ability to read or change data
255 on that device.

256 **Physical security**

257 An attack that could be considered is one where the attacker gains physical
258 access to the Apertis system, for example by stealing the car in which it is
259 installed. It is obviously impossible to guarantee availability in this particular
260 threat model (the attacker could steal or destroy the Apertis system), but it is
261 possible to provide confidentiality, via encryption "at rest".

262 A variation on this attack is to assume that the attacker has physical access
263 to the system and then returns it to the user, perhaps repeatedly. This raises
264 the question of whether integrity is provided (whether the user can be sure that
265 they are not subsequently entering confidential data into an operating system
266 that has been modified by the attacker).

267 This type of physical security can come with a significant performance and
268 complexity overhead; as a trade-off, it could be declared to be out-of-scope.

269 **Solutions adopted by popular platforms**

270 As background for the discussions of this document, the following sections pro-
271 vide an overview of the approaches other mobile platforms have chosen for secu-

272 rity, including an explanation of the trade-offs or assumptions where necessary.

273 **Android**

274 Android uses the Linux kernel, and as such relies on it being secure when it
275 comes to the most basic security features of modern operating systems, such
276 as process isolation and an access permissions model. On top of that, Android
277 has a Java-based virtual machine environment which runs regular applications
278 and provides them with APIs that have been designed specifically for Android.
279 Regular applications can execute arbitrary native code within their application
280 sandbox, for example by using the NDK interfaces.

281 [https://developer.android.com/training/articles/security-tips.
282 html#Dalvik](https://developer.android.com/training/articles/security-tips.html#Dalvik) notes that “On Android, the Dalvik VM is not a
283 security boundary”.

284 However, some system functionality is not directly available within the appli-
285 cation sandbox, but can be accessed by communicating with more-privileged
286 components, typically using Android’s Java APIs.

287 Early versions of Android worked under the assumption that the system will
288 be used by a single user, and no attempt was made towards supporting any
289 kind of multi-user use case. Based on this assumption, Android re-purposed the
290 concept of UNIX user ID (uid), making each application run as a different user
291 ID. This allows for very tight control over what files each application is able to
292 access by simply using user-based permissions; this provides isolation between
293 applications (**Security between applications**). In later Android versions, which
294 do have multi-user support, user IDs are used to provide two separate security
295 boundaries – isolating applications from each other, and isolating users from
296 each other (**Security between users**) – with one user ID per (user, app) pair.
297 This is discussed in more detail in the [Multiuser design document](#)⁵.

298 The system’s main file system is mounted read-only to protect against unautho-
299 rized tampering with system files (integrity for platform data, **Security between
300 platform services**); however, this does not protect integrity against an attacker
301 with physical access (**Physical security**). Encryption of the user data partition
302 through the standard *dm-crypt* kernel facility (confidentiality despite physical
303 access, **Physical security**) is supported if the user configures a password for their
304 device. Users using gesture-based or other unlock mechanisms are unable to use
305 this feature.

306 The root user on Android is all-powerful, and can do anything to the system.
307 Android makes no attempt to limit the power of processes running as UID 0 (the
308 root user ID); in other words, they are part of the TCB. All security of system
309 services, and the core system and applications rely on the separation of users
310 already discussed and in assuming nothing other than the essential (the kernel
311 itself and a very small number of system services) runs with root privileges.

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

312 Older versions of Android did not use Mandatory Access Control, discussed in
313 this document's chapter 5. More recent versions use SELinux to augment the
314 uid-based sandbox.

315 Security-Enhanced Linux in Android, [https://source.android.com/
316 devices/tech/security/selinux/](https://source.android.com/devices/tech/security/selinux/)

317 The idea of restricting the services an application can use to those specified in
318 the application's manifest also exists in Android. Before installation, Android
319 shows a list of system services the application intends to access and installation
320 only initiates if the user agrees. This differs slightly from the [Applications
321 design in Apertis⁶](#), in which some permissions are subject to prompting similar
322 to Android's, while other permissions are checked by the app store curator and
323 unconditionally granted on installation.

324 Android provides APIs to verify a process has a given permission, but no central
325 control is built into the API layer or the IPC mechanism as planned for Apertis
326 – checking whether a caller has the required permissions to make that call is left
327 to the service or application that provides the IPC interface or API, similar to
328 how most GNOME services work by using [PolicyKit⁷](#) (see section 6 for more on
329 this topic).

330 See, for instance, how the A2DP service verifies the caller has the
331 required permission: [https://android.googlesource.com/platform/
332 frameworks/base/+22f7dfd23490a3de2f21ff96949ba47003aac8f8/
333 core/java/android/server/BluetoothA2dpService.java#251](https://android.googlesource.com/platform/frameworks/base/+22f7dfd23490a3de2f21ff96949ba47003aac8f8/core/java/android/server/BluetoothA2dpService.java#251)

334 No effort is made specifically towards thwarting applications misbehaving and
335 causing a Denial of Service on system services or the IPC mechanism. Android
336 uses two very simple strategies to forcibly stop an application: 1) it kills appli-
337 cations when the device is out of memory; 2) it notifies the user of [unresponsive
338 applications⁸](#) and allows them to force the application to close, similar to how
339 GNOME does it.

340 An application is deemed to not be responding after about 5 seconds of not being
341 able to handle user input. This feature is implemented by the Android window
342 manager service, which is responsible for dispatching events read from the ker-
343 nel input events interface (the files under `/dev/input`) to the application, in
344 cooperation with the activity manager service, which shows the application not
345 responding dialog and kills the application if the user decides to close it. After
346 dispatching an event, the window manager service waits for an acknowledgement
347 from the application with a timeout; if the timeout is hit, then the application
348 is considered not responding.

⁶<https://em.pages.apertis.org/apertis-website/concepts/applications/>

⁷<https://wiki.gnome.org/Projects/PolicyKit>

⁸<http://developer.android.com/guide/practices/design/responsiveness.html>

349 **Bada**

350 Bada is not an Open Source platform, so closer inspection of the inner work-
351 ings is not feasible. However, the documentation indicates that Bada also kills
352 applications when under memory pressure.

353 It also uses a simple *API privilege level* framework as the base of its security
354 and reliability architecture. Applications running with the *Normal* API privilege
355 level need to specify which *API privilege groups* it needs to be able to access in
356 their manifest file.

357 Some APIs are restricted under the *System* API level and can be used only
358 by Samsung or its authorized partners. It's not possible to say whether those
359 restrictions are applied in a general way or by having the modules that provide
360 the APIs perform validation checks, but the latter seems more likely given these
361 are C++ APIs that do not go through any kind of central service.

362 **iOS**

363 iOS is, like Bada, a closed platform, so [details are sometimes difficult to obtain](#)⁹,
364 but Apple does use some Open Source components (at the lower levels, in par-
365 ticular). iOS has an [application sandbox](#)¹⁰ that is very similar in functionality
366 to AppArmor, discussed below. The technology is based on Mandatory Access
367 Control provided by the [TrustedBSD](#)¹¹ project and has been marketed under
368 the *Seatbelt* name.

369 Like AppArmor, it uses configuration files that specify profiles, using path-based
370 rules for file system access control. Also like AppArmor, other functionality such
371 as network access can be controlled. The actual confinement is applied when the
372 application uses system calls to request that the kernel carries out an action on
373 the application's behalf (in other words, when the privilege boundary between
374 user-space and the kernel is crossed).

375 Seatbelt is considered to be the single canonical solution to sandboxing applica-
376 tions on iOS; this is in contrast with Linux, in which AppArmor is one option
377 among many (system calls can be mediated by seccomp, the [Secure Computing](#)
378 [API](#)¹² described in section 17 of this document, in addition to up to one MAC
379 layer such as AppArmor, SELinux or Smack).

380 None of this complexity is exposed to apps developed for iOS, though; they are
381 merely implementation details.

382 Apparently, there are no central controls whatsoever protecting the system from
383 applications that hang or try to DoS system services. The only real limitation
384 imposed is the available system memory.

⁹https://github.com/V3RL4223N3/Programming/blob/master/iOS/iOS_Security_May12.pdf

¹⁰<http://www.usefulsecurity.com/2007/11/apple-sandboxes-part-1/>

¹¹<http://www.trustedbsd.org/mac.html>

¹²<http://lwn.net/Articles/475043/>

385 Applications are free to use any APIs available, there are no explicit declarative
386 permissions system like the one used in Android. However, some functionality
387 are always mediated by the system, including through system-controlled UI.

388 For instance, an application can query the GPS for location; when that happens,
389 the system will take over and present the user with a request for permission.
390 If the user accepts the request will be successful and the application will be
391 white-listed for future queries. The same goes for interacting with the camera:
392 the application can request a picture be taken, but the UI that is presented for
393 taking the picture is controlled by the system as is actual interaction with the
394 camera.

395 This is analogous to the way in which Linux services can use PolicyKit to mediate
396 privileged actions (see section 6), although on iOS the authorization step is
397 specifically considered to be an implementation detail of the API used, whereas
398 some Linux services do make the calling application aware of whether there was
399 an interactive authorization step.

400 **Mandatory Access Control**

401 The goal of the Linux Discretionary Access Control (DAC) is a separation of
402 multiple users and their data ([Security between users](#), [Security between plat-](#)
403 [form services](#)). The policies are based on the identity of a subject or their
404 groups. Since in Apertis applications from the same user should not trust each
405 other ([Security between applications](#)), the utilization of a Mandatory Access
406 Control (MAC) system is recommended. MAC is implemented in Linux by one
407 of the available Linux Security Modules (LSM).

408 **Linux Security Modules (LSM)**

409 Due to the different nature and objectives of various security models there is no
410 real consensus about which security model is the best, thus support for loading
411 different security models and solutions became available in Linux in 2001. This
412 mechanism is called Linux Security Modules (LSM).

413 Although it is in theory possible to provide generic support for any LSM, in
414 practice most distributions pick one and stick to it, since both policies and
415 threat models are very specific to any particular LSM module.

416 The first implementation on top of LSM was SELinux developed by the US
417 National Security Agency (NSA). In 2009 the TOMOYO Linux module was
418 also included in the kernel followed by AppArmor in the same year. The sub-
419 sections below gives a short introduction on the security models that are officially
420 supported by the Linux Kernel.

421 **SELinux**

422 SELinux¹³ is one of the most well-known LSMs. It is supported by default
423 in Red Hat Enterprise Linux and Fedora. It is infamous for how difficult it
424 is to maintain the security policies; however, being the most flexible and not
425 having any limitation regarding what it can label, it is the reference in terms of
426 features. For every user or process, SELinux assigns a context which consists of
427 a role, user name and domain/type. The circumstances under which the user is
428 allowed to enter into a certain domain must be configured into the policies.

429 SELinux works by applying rules defined by a policy when kernel-mediated
430 actions are taken. Any file-like object in the system, including files, directories,
431 and network sockets can be labeled. Those labels are set on file system objects
432 using extended file system attributes. That can be problematic if the file system
433 that is being used in a given product or situation lacks support for extended
434 attributes. While support has been built for storing labels in frequently used
435 networking file systems like NFS, usage in newer file systems may be challenging.
436 Note that BTRFS does support extended attributes.

437 Users and processes also have labels assigned to them. Labels can be of a more
438 general kind like, for instance, the `sysadm_t` label, which is used to determine
439 that a given resource should be accessible to system administrators, or of a more
440 specific kind.

441 Locking down a specific application, for instance, may involve creating new
442 labels specifically for its own usage. A label “`browser_cache_t`” may be created,
443 for instance, to protect the browser cache storage. Only applications and users
444 which have their label assigned to them will be able to access and manage those
445 files. The policy will specify that any files created by the browser on that specific
446 directory are assigned that label automatically.

447 Labels are automatically applied to any resources created by a process, based
448 on the labels the process itself has, including sockets, files, devices represented
449 as files and so on. SELinux, as other MAC systems, is not designed to impose
450 performance-related limitations, such as specifying how much CPU time a pro-
451 cess may consume, or how many times a process duplicates itself, but supports
452 pretty much everything in the area it was designed to target.

453 The SELinux support built into D-Bus allows enhancement of the existing D-
454 Bus security rules by associating names, methods and signals with SELinux
455 labels, thus bringing similar policy-making capabilities to D-Bus.

456 TOMOYO Linux

457 TOMOYO Linux¹⁴ focuses on the behavior of a system where every process is
458 created with a certain purpose and allows each process to declare behaviors and
459 resources needed to achieve their purposes. TOMOYO Linux is not officially
460 supported by any popular Linux distribution.

¹³http://selinuxproject.org/page/Main_Page

¹⁴<http://tomoyo.sourceforge.jp/>

461 **SMACK**

462 Simplicity is the primary design goal of [SMACK](#)¹⁵. It was used by MeeGo before
463 that project was cancelled; [Tizen](#)¹⁶ appears to be the only general-purpose Linux
464 distribution using SMACK as of 2015.

465 SMACK works by assigning labels to the same system objects and to processes as
466 SELinux does; similar capabilities were proposed by Intel for D-Bus integration,
467 but their originators did not follow up on [reviews](#)¹⁷, and the changes were not
468 merged. SMACK also relies on extended file system attributes for the labels,
469 which means it suffers from the same shortcomings that come from that as
470 SELinux.

471 There are a few special predefined labels, but the administrator can create and
472 assign as many different labels as desired. The rules regarding what a process
473 with a given label is able to perform on an object with another given label are
474 specified in the system-wide policy file `/etc/smack/accesses`, or can be set in
475 run-time using the `smackfs` virtual file system.

476 MeeGo used SMACK by assigning a separate label to each service in the system,
477 such as “Cellular” and “Location”. Every application would get their own labels
478 and on installation the packaging system would read a manifest that listed the
479 systems the application would require, and SMACK rules would then be created
480 to allow those accesses.

481 **AppArmor**

482 Of all LSM modules that were reviewed, Application Armor ([AppArmor](#)¹⁸) can
483 be seen as the most focused on application containment.

484 AppArmor allows the system administrator to associate an executable with a
485 given profile in order to limit access to resources. These resource limitations can
486 be applied to network and file system access and other system objects. Unlike
487 SMACK and SELinux, AppArmor does not use extended file system attributes
488 for storing labels, making it file system agnostic.

489 Also in contrast with SELinux and SMACK, AppArmor does not have a system-
490 wide policy, but application profiles, associated with the application binaries.
491 This makes it possible to disable enforcement for a single application, for in-
492 stance. In the event of shipping a policy with an error that leads to users not
493 being able to use an application it is possible to quickly restore functionality for
494 that application without disabling the security for the system as a whole, while
495 the incorrect profile is fixed.

496 Since AppArmor uses the path of the binary for profile selection, changing the
497 path through manipulation of the file system name space (i.e. through links

¹⁵<http://schaufler-ca.com/>

¹⁶<https://developer.tizen.org/sdk.html>

¹⁷https://bugs.freedesktop.org/show_bug.cgi?id=47581

¹⁸<https://gitlab.com/apparmor/apparmor/-/wikis/home>

498 or mount points) is a potential way of working-around the limits that are put
499 in place; while this is cited as a weakness, in practice it is not an issue, since
500 restrictions exist to block anyone trying to do this. Creation of symbolic links
501 is only allowed if the process doing so is allowed to access the original file, and
502 links are followed to enforce any policy assigned to the binary they link to.
503 Confined processes are also not allowed to mount file systems unless they are
504 given explicit permission.

505 Here's an example of how restricting ping's ability to create raw sockets cannot
506 be worked around through linking – lines beginning with \$ represent commands
507 executed by a normal user, and those starting with # have been executed by
508 the root user:

```
1  $ ping debian.org
2  ping: icmp open socket: Operation not permitted
3  $ ln -s /bin/ping
4  $ ./ping debian.org
5  ping: icmp open socket: Operation not permitted
6  $ ln /bin/ping ping2
7  ln: failed to create hard link `ping2' => `/bin/ping': Operation not permitted
8  # ping debian.org
9  ping: icmp open socket: Operation not permitted
10 # ln -s /bin/ping /bin/ping2
11 # ping2 debian.org
12 ping: icmp open socket: Operation not permitted
13 #
```

509 AppArmor restriction applying to file system links

510 Copying the file would make it not trigger the containment. However, even if
511 the user was able to symlink the binary or use mount points to work-around
512 the path-based restrictions that should not mean privilege escalation, given the
513 white-list approach that is being adopted. That approach means that any binary
514 escaping its containment profile would in actuality be dropping privileges, not
515 escalating them, since the restrictions imposed on binaries that do not have
516 their own profile can be quite extensive.

517 Note that Collabora is proposing mounting partitions that should only contain
518 data with the option that disallows execution of code contained in them, so even
519 if the user manages to escape the strict containment of the user session and
520 copied a binary to one of the directories they have write access to, they would
521 not be able to run it. Refer to the System updates & rollback and Application
522 designs for more details on file system and partition configuration.

523 Integration with D-Bus was developed by Canonical and shipped in Ubuntu for
524 several years, before being merged upstream in dbus-daemon 1.9 and AppArmor

525 2.9. The implementation includes patches to AppArmor's user-space tools, to
526 make the new D-Bus rules known to the profile parser, and to dbus-daemon, so
527 that it will check with AppArmor before allowing a request.

528 AppArmor will be used by shipping profiles for all components of the platform,
529 and by requiring that third-party applications ship with their own profiles that
530 specify exactly what requests the application should be allowed.

531 Creating a new profile for AppArmor is a reasonably simple process: a new pro-
532 file is generated automatically running the program under AppArmor's profile
533 generator, [aa-genprof](#)¹⁹, and exercising its features so that the profile generator
534 can capture all of the accesses the application is expected to make. After the
535 initial profile has been generated it must be reviewed and fine-tuned by manual
536 editing to make sure the permissions that are granted are not beyond what is
537 expected.

538 In AppArmor there is no default profile applied to all processes, but a process
539 always inherits limitations imposed to its parent. Setting up a proper profile
540 for components such as the session manager is a practical and effective way of
541 implementing this requirement.

542 Comparison

543 Since all those Linux Security Modules rely on the same kernel API and have
544 the same overall goals, the features and resources they are able to protect are
545 very similar, thus not much time will be spent covering those. The policy
546 format and how control over the system and its components is exerted varies
547 from framework to framework, though, which leads to different limitations. The
548 table below has a summary of features, simplicity and limitations:

	SELinux	AppArmor	SMACK
Maintainability	Complex	Simple	Simple
Profile creation	Manual/Tools	Manual/Tools	Manual
D-Bus integration	Yes	Yes	Not proposed upstream
File system agnostic	No	Yes	No
Enforcement scope	System-wide	Per application	System-wide

549 Comparison of LSM features

550 Historically LSM modules have focused on kernel-mediated accesses, such as
551 access to file system objects and network resources. Modern systems, though,
552 have several important features being managed by user-space daemons. D-Bus is
553 one such daemon and is specially important since it is the IPC mechanism used
554 by those daemons and applications for communication. There is clear benefit

¹⁹https://gitlab.com/apparmor/apparmor/-/wikis/Profiling_with_tools

555 in allowing D-Bus to cooperate with the LSM to restrict what applications can
556 talk to which services and how.

557 In that regard SELinux and AppArmor are in advantage since D-Bus is able to
558 let these frameworks decide whether a given communication should be allowed
559 or not, and whether a given process is allowed to acquire a particular name on
560 the bus. Support for SMACK mediation was worked on by Intel for use in Tizen,
561 but has not been proposed for upstream inclusion in D-Bus, and is believed to
562 add considerable complexity to dbus-daemon. There is no work in progress to
563 add TOMOYO support.

564 Like D-Bus' built-in support for applying “policy” to message delivery, AppAr-
565 mor mediation of D-Bus messages has separate checks for whether the sender
566 may send a message to the recipient, and whether the recipient may receive a
567 message from the sender. Either or both of these can be used, and the mes-
568 sage will only succeed if both sending and receiving were allowed. The sender's
569 AppArmor profile determines whether it can send (usually conditional on the
570 profile name of the recipient), and the recipient's AppArmor profile determines
571 whether it can receive (either conditional on the profile name of the sender, or
572 unconditionally), so some coordination between profiles is needed to express a
573 particular high-level security policy.

574 The main difference between the SELinux and SMACK label-based mediation in
575 terms of features is how granular you can get. With the [D-Bus additions to the
576 AppArmor profile language²⁰](#), for instance, in addition to specifying which ser-
577 vices can be called upon by the constrained process it is also possible to specify
578 which interfaces and paths are allowed or denied. This is unlike [SELinux media-
579 tion²¹](#), which only checks whether a given client can talk to a given service. One
580 caveat regarding fine-grained (interface- and path-based) D-Bus access control
581 is that it is often not directly useful, since the interface and path is not nec-
582 essarily sufficient to determine whether an action should be allowed or denied
583 (for example, [Motivation for polkit](#) describes why this is the case for the udisks
584 service). As a result of considerations like this, the developers of kdbus oppose
585 the addition of fine-grained access control within kdbus, and have indicated
586 that kdbus' access-control will never go beyond allowing or rejecting a client
587 communicating with a service.

588 kdbus is a kernel module that has been proposed to take over the role
589 of the user-space dbus-daemon in D-Bus on Linux systems. [https:
590 //github.com/gregkh/kdbus](https://github.com/gregkh/kdbus)

591 Software that is being used by large distributions is often more tested and tested
592 in more diverse scenarios. For this reason Collabora believes that being used by
593 one of the main distributions is a very important feature to look for in a LSM.

²⁰[https://gitlab.com/apparmor/apparmor/-/wikis/AppArmor_Core_Policy_Reference#
dbus-rules](https://gitlab.com/apparmor/apparmor/-/wikis/AppArmor_Core_Policy_Reference#dbus-rules)

²¹<http://dbus.freedesktop.org/doc/dbus-daemon.1.html#lbAg>

594 Flexibility is also good to have, since more complex requirements can be modeled
595 more precisely. However, there is a trade-off between complexity and flexibility
596 that should be taken into consideration.

597 The recommendation on the selection of the framework is a combination of the
598 adoption of the framework by existing distributions, features, maintainability,
599 cost of deployment and experience of the developers involved. The table below
600 contains a comparison of the adoption of the existing security models. Only
601 major distributions that ship and enable the module by default are listed.

Name	Distributions	Merged to mainline	Maintainer
SELinux	Fedora, Red Hat Enterprise	08 Aug 2003	NSA, Network Associates, Secure Computing
AppArmor	SUSE, OpenSUSE, Ubuntu	20 Oct 2010	SUSE, Canonical
SMACK	Tizen	11 Aug 2007	Intel, Samsung2
TOMOYO		10 Jun 2009	NTT Data Corp.

602 Comparison of LSM adoption and maturity

603 Performance impact

604 The performance impact of MAC solutions depends heavily on the workload
605 of the application, so it's hard to rely upon a single metric. It seems major
606 adopters of these technologies are not too concerned about their real-world
607 impact, even though they may be expressive in benchmarks, since there are no
608 recent measurements of performance impact for the major MAC solutions.

609 That said, early tests indicate that SELinux has a performance impact [floating](#)
610 [around 7% to 10%](#)²², with tasks that are more CPU intensive having *less* impact,
611 since they are not making many system calls that are checked. SELinux performs
612 checks on every operation that touches a labeled resource, so when reading or
613 writing a file all read/write operations would cause a check. That means making
614 larger operations instead of several smaller ones would also make the overhead
615 go down.

616 AppArmor generally does fewer checks than SELinux since only operations that
617 open, map or execute a file are checked: the individual read/write operations
618 that follow are not checked independently. Novell's documentation and FAQs
619 state a 0.2% overhead is expected on best-case scenarios – writing a big file, for
620 instance, with a 2% overhead in worst-case scenarios (an application touching
621 lots of files once). Collabora's own testing on a 2012 x86-64 system puts the
622 worst case scenario leaning towards the 5% range. The test measured reading
623 3000 small files with a hot disk cache, and ranged from ~89ms to ~94ms average
624 duration.

625 SMACK's performance characteristics should be similar to that of SELinux,
626 given their similar approach to the problem. SMACK has been tested for a TV

²²<http://blog.larsstrand.no/2007/11/rhel5-selinux-benchmark.html>

627 embedded scenario which has shown performance degradation from 0% all the
628 way to 30% on a worst-case scenario of deleting a 0-length file. Degradation
629 varied greatly depending on the benchmark used.

630 The only conclusion Collabora believes can be drawn from these numbers is
631 that an approach which checks less often (as is the case for AppArmor) can
632 be expected to have less impact on performance, in general. That said, these
633 numbers should be taken with a grain of salt, since they haven't been measured
634 in the exact same hardware and with the exact same methodology. They may
635 also suffer from bias caused by benchmark tests which may not represent real-
636 world usage scenarios.

637 No numbers exist measuring the impact on performance of the existing D-Bus
638 SELinux and AppArmor mediation, nor with the in-development SMACK me-
639 diation. The overhead caused to each D-Bus call should be similar to that of
640 opening a file, since the same procedure is involved: a check needs to be done
641 each time a message is received from a client that is contained. It should be
642 noted that D-Bus is not designed to be used for high-frequency communica-
643 tion due to its per-message overhead, so the additional overhead for AppArmor
644 should not be problematic unless D-Bus is already being misused.

645 Where higher-frequency communication is required, D-Bus' file descriptor pass-
646 ing feature can be used to negotiate a private channel (a pipe or socket) between
647 two processes. This negotiation can be as simple as a single D-Bus method call,
648 and only incurs the cost of AppArmor checks once (when it is first set up).
649 Subsequent messages through the private channel bypass D-Bus and are not
650 checked individually by AppArmor, avoiding any per-message overhead in this
651 case.

652 A more realistic and reliable assessment of the overhead imposed on a real-world
653 system would only be feasible on the target hardware, with actual applications,
654 where variables like storage device and file system would also be better con-
655 trolled.

656 **Conclusion**

657 Collabora recommends the adoption of a MAC solution, specifically AppArmor.
658 It solves the problem of restricting applications to the privileges they require to
659 work, and is an effective solution to the problem of protecting applications from
660 other applications running for the same user, which a DAC model is not able
661 to provide.

662 SMACK and TOMOYO have essentially no adoption and support when com-
663 pared to solutions like SELinux and AppArmor, without providing any clear
664 advantages. MeeGo would have been a good testing ground for SMACK, but
665 the fact that it was never really deployed in enforcing mode means that the
666 potential was never realized.

667 SELinux offers the most flexible configuration of security policies, but it intro-

668 duces a lot of complexity on the setup and maintenance of the policies, not only
669 for distribution maintainers but also for application developers and packagers,
670 which impacts on the costs of the solution. It is quite common to see Fedora
671 users running into problems caused by SELinux configuration issues.

672 AppArmor stands out as a good middle-ground between flexibility and main-
673 tainability while at the same time having significant adoption: by the biggest
674 end-user desktop distribution (Ubuntu) and by one of the two biggest enterprise
675 distributors (SUSE). The fact that it is the security solution already supported
676 and included in the Ubuntu distribution, which is the base of the Apertis plat-
677 form, minimizes the initial effort to create a secure baseline and reduces the
678 effort needed to maintain it. Since Ubuntu ships with AppArmor, some of the
679 services and applications will already be covered by the profiles shipped with
680 Ubuntu. Creation of additional profiles is made easy by the profile generator
681 tool that comes with AppArmor. it records everything the application needs to
682 do during normal operation, and allows for further refining after the recording
683 session is done.

684 Collabora will integrate and validate the existing Ubuntu profiles that are rele-
685 vant to the Apertis platform as well as modify or write any additional profiles
686 required by the base platform. Collabora will also assist in the creation of pro-
687 files for higher level applications that ship with the final product and on the
688 strategy for profile management for third party applications.

689 **AppArmor Policy and management examples**

690 Looking at a few examples might help better visualize how AppArmor works,
691 and what creating new policies entails. Let's look at a simple policy file:

```
1 $ cat /etc/apparmor.d/bin.ping
2 ...
3 /bin/ping {
4     #include <abstractions/base>
5     #include <abstractions/consoles>
6     #include <abstractions/nameservice>
7
8     capability net_raw,
9     capability setuid,
10    network inet raw,
11    /bin/ping mixr,
12    /etc/modules.conf r,
13    ## Site-specific additions and overrides. See local/README for details.
14    #include \<local/bin.ping\>
15 }
16 $
```

692 AppArmor policy shipped for ping in Ubuntu

693 This is the policy for the ping command. The binary is specified, then a few
694 includes that have common rules for the kind of binary ping (console), and ser-
695 vices it consumes (nameservice). Then we have two rules specifying capabilities
696 that the program is allowed to use, and we state the fact that it is allowed to
697 do perform raw network operations. Then it's specified that the process should
698 be able to memory map (m) /bin/ping, inherit confinement from the parent (i),
699 execute the binary /bin/ping (x) and read it (r). It's also specified that ping
700 should be able to read /etc/modules.conf.

701 If an attack was able to execute arbitrary code by hijacking the ping process,
702 then that is all it would be able to do. No reading of /etc/password would be
703 allowed, for instance. If ping was a very core feature of the device and starts
704 failing because of a bad policy, it is possible to disable security enforcement just
705 for ping, leaving the rest of the system secured (something that would not be
706 easily done with SMACK or SELinux), by running *aa-disable* with ping's path
707 as the parameter, or by installing a symbolic link in /etc/apparmor.d/disable:

```
1 $ aa-disable /bin/ping
2 Disabling /bin/ping.
3 $ ls -l /etc/apparmor.d/disable/
4 total 0
5 lrwxrwxrwx 1 root root 24 Feb 20 19:38 bin.ping ->
6 /etc/apparmor.d/bin.ping
```

708 A symbolic link to disable the ping AppArmor policy

709 Note that *aa-disable* is only a convenience tool to unload a profile and link it
710 to the **/etc/apparmor.d/disable** directory. Note that the convenience script
711 is not currently shipped in the image intended for the target hardware. It is
712 available in the repository though, and is available in the development and SDK
713 images since it makes it more convenient to test and debug issues.

714 Note, also, that writing to the **/etc/apparmor.d/disable** directory is required
715 for creating the symlink there, and the UNIX DAC permissions system already
716 protects that directory for writing - only root is able to write to this directory.
717 As discussed in [A note about root](#), if an attacker becomes root the system is
718 already compromised.

719 Also, as discussed in the System update & rollback, the system partition will
720 be mounted read-only, so that is an additional protection layer already. And in
721 addition to that, the white-list approach discussed in [Implementing a white list](#)
722 [approach](#) will already deny writing to anywhere in the file system, so anything
723 running under the application manager will have an additional layer of security
724 imposed on them.

725 For these reasons, Collabora doesn't see any reason to add additional security
726 such as AppArmor profiles specifically for protecting the system against unau-
727 thorized disabling of profiles.

728 Profiles for libraries

729 AppArmor profiles are always attached to a binary. That means there is no way
730 to attach a profile to every program that uses a given library. However, devel-
731 opers can write files called *abstractions* with rules that can be included through
732 the *#include* directive, similar to how libraries work for programming. Using
733 this feature Collabora has written rules for the WebKit library, for instance,
734 that can be included by the browser application as well as by any application
735 that uses the library.

736 There is also concern with protecting internal, proprietary libraries, so that
737 they cannot be used by applications. In the profiles and abstractions shipped
738 with Apertis right now, all applications are allowed to use all libraries that are
739 installed in the public library paths (such as `/usr/lib`).

740 The rationale for this is libraries are only pieces of code that could be included
741 by the applications themselves, and it would be very time-consuming and error
742 prone having to specify each and every library and module the application may
743 need to use directly or that would be used indirectly by a library used by the
744 application.

745 Collabora recommends that proprietary libraries that are used only by one or a
746 few services should be installed in a private location, such as the application's
747 directory. That would put those libraries outside of the paths covered by the
748 existing rules, and they would thus be out of reach for any other application
749 already, given the white-list approach to session lockdown, as discussed in [Im-
750 plementing a white list approach](#).

751 If that is not possible, because the library hardcodes paths or some other issue,
752 an explicit deny rule could be added to the **chaiwala-base** abstraction that
753 implements the general rules that apply to most applications, including the one
754 that allows access to all libraries. Collabora can help deciding what to do with
755 specific libraries through support tickets opened in the bug tracking system.

756 Chaiwala was a development codename for parts of the Apertis sys-
757 tem. The name is retained here for compatibility reasons.

758 Application installation and upgrades

759 For installations and upgrades to be performed, no changes to the running sys-
760 tem's security are necessary, since the processes that manage upgrade, including
761 the creation of the required snapshots will have enough power given to them

762 An application's profile is read at startup time. That means an application that
763 has been upgraded will only be contained with the new rules after it has been

764 restarted. The D-Bus integration works by querying the kernel interface for the
765 PID it is communicating with, not its own, so D-Bus itself does not need to be
766 restarted when new profiles are installed.

767 When a *.deb* package is installed its AppArmor profile will be installed to the
768 system AppArmor profile location (*/etc/apparmor.d/*), but in the new snapshot
769 created for the upgrade rather than on the running system.

770 The new version of the upgraded package and its new profile will only take effect
771 after the system has been rebooted. For details about how *.deb* packages will
772 be handled when the system is upgraded please see the *System Updates and*
773 *Rollback* document.

774 For more details on how applications from the store will be handled, the *Appli-*
775 *cations* document produced by Collabora goes into details about how the per-
776 missions specified in the manifest will be transformed into AppArmor profiles
777 and on how they will be installed and loaded.

778 **A note about root**

779 As has been demonstrated in listing *AppArmor restriction applying to file system*
780 *links*, AppArmor can restrict even the powers of the root user. Most platforms
781 do not try to limit that power in any way, since if an attacker has breached the
782 system to get root privileges it's likely that all bets are already off. That said,
783 it should be possible to limit the root user's ability to modify the AppArmor
784 profiles, leaving that task solely for the package manager (see the Applications
785 design for details).

786 **Implementing a white-list approach**

787 Collabora recommends the use of a white-list approach in which the app-
788 launcher will be confined to a policy that denies almost everything, and specific
789 permissions will be granted by the application profiles. This means all applica-
790 tions will only be able to access what is expressively allowed by their specific
791 policies, providing Apertis with a very tight least-privilege implementation.

792 A simple example of how that can be achieved using AppArmor is provided in the
793 following examples. The examples will emulate the proposed solution by locking
794 down a shell, which represents the Apertis application launcher, and granting
795 specific privileges to a couple applications so that they are able to access the
796 files they require.

797 Listing *Sample profiles for implementing white-listing* shows a profile for the
798 shell, essentially denying it access to everything by not allowing access to any
799 files. It gives the shell permission to run both *ls* and *cat*. Note that flags *rix*
800 are used for this, meaning the shell can read the binaries (*r*), and execute them
801 (*x*); the *i* preceding the *x* tells AppArmor that these binaries should inherit the
802 shell's confinement rules, even if they have rules of their own.

803 Then permission is given for the shell to run the *dconf* command. *dconf* is
804 GNOME's settings storage. Notice that we have *p* as the prefix for *x* this time.
805 This means we want this application to use its own rules; if no rules had been
806 specified, then AppArmor would have fallen back to using the shell's confinement
807 rules.

```
1 $ cat /etc/apparmor.d/bin.zsh4
2 ## Last Modified: Fri May 11 11:43:44 2012
3
4 #include <tunables/global>
5 /bin/zsh4 {
6     #include <abstractions/base>
7     #include <abstractions/consoles>
8     #include <abstractions/nameservice>
9     /bin/ls rix,
10    /bin/cat rix,
11    /usr/bin/dconf rpx,
12    /bin/zsh4 mr,
13    /usr/lib/zsh/*/zsh/* mr,
14 }
15
16 $ cat /etc/apparmor.d/usr.bin.dconf
17 ## Last Modified: Fri May 11 11:59:09 2012
18
19 #include <tunables/global>
20 /usr/bin/dconf {
21     #include <abstractions/base>
22     #include <abstractions/nameservice>
23     @{HOME}/.cache/dconf/user rw,
24     @{HOME}/.config/dconf/user r,
25     /usr/bin/dconf mr,
26 }
```

808 Sample profiles for implementing white-listing

809 The profile for *dconf* allows reading (and only reading) the user configuration
810 for *dconf* itself, and allows reading and writing to the cache. By using these
811 rules we have both guaranteed that no application executed from this shell will
812 be able to look at or interfere with *dconf*'s files, and that *dconf* itself is able to
813 function when used. Here's the result:

```
814 % cat .config/dconf/user
815 cat: .config/dconf/user: Permission denied
816 % dconf read /apps/empathy/ui/show-offline
817 true
```


818 %

819 Effects of white-list approach profiles

820 As shown by this example, the application launcher itself and any applications
821 which do not possess profiles can be restricted to the bare minimum permissions,
822 and applications can be given the more specific privileges they require to do
823 their job, using the *p* prefix to let AppArmor know that's what is desired.

824 **polkit (PolicyKit)**

825 polkit (formerly PolicyKit) is a service used by various upstream components
826 in Apertis, as a way to centralize security policy for actions delegated by one
827 process to another. The central problems addressed by polkit are that the
828 desired security policies for various privileged actions are system-dependent and
829 non-trivial to evaluate, and that generic components such as the kernel's DAC
830 and MAC subsystems do not have enough context to understand whether a
831 privileged action is acceptable.

832 **Motivation for polkit**

833 Broadly, there are two ways a process can carry out a desired action: it can
834 do it directly, or it can use inter-process communication to ask a service to do
835 that operation on its behalf. If the action is done directly, the components that
836 say whether it can succeed are the Linux kernel's normal discretionary access
837 control (DAC) permissions checks, and if configured, a mandatory access control
838 module (MAC, section 5).

839 However, the kernel's relatively coarse-grained checks are not sufficient to ex-
840 press the desired policies for consumer-focused systems. A frequent example is
841 mounting file systems on removable devices: if a user plugs in a USB stick with
842 a FAT filesystem, it is reasonable to expect the user interface layer to either
843 mount it automatically, or let the user choose to mount it. Similarly, to avoid
844 data loss, the user should be able to unmount the removable device when they
845 have finished with it.

846 Applying the desired policy using the kernel's permission checks is not possi-
847 ble, because mounting and unmounting a USB stick is fundamentally the same
848 system call as mounting and unmounting any other file system, which is not de-
849 sired: if ordinary users can make arbitrary mount system calls, they can mount
850 a file system that contains setuid executables and achieve privilege escalation.
851 As a result, the kernel disallows direct mount and unmount actions by unpriv-
852 ileged processes; instead, user processes may request that a privileged system
853 process carries out the desired action. In the case of device mounting, Apertis
854 uses the privileged udisks2 service to mount and unmount devices.

855 In environments that use a MAC framework like AppArmor, actions that would
856 normally be allowed can also become privileged: for instance, in a framework for
857 sandboxed applications, most apps should not be allowed to record audio. The

858 resulting AppArmor adjustments prevent carrying out these actions directly.
859 The result is that, again, the only way to achieve them is that a service with a
860 suitable privilege carries out the action (perhaps with a mandatory user interface
861 prompt first, as in certain iOS features).

862 These privileged requests are commonly sent via the D-Bus interprocess com-
863 munication (IPC) system; indeed, this is one of the purposes for which D-Bus
864 was designed. D-Bus has facilities for allowing or forbidding messages between
865 particular processes in a somewhat fine-grained way, either directly or mediated
866 by MAC frameworks. However, this has the same issue as the kernel's checks for
867 direct mount operations: the generic D-Bus IPC framework does not understand
868 the context of the messages. For example, it can allow or forbid messages that
869 ask to mount a device, but cannot discriminate based on whether the device in
870 question is a removable device or a system partition, because it does not have
871 that domain-specific information.

872 This means that the security decision – having received this request, should the
873 service obey it? – must be at least partly made by the service itself (for example
874 `udisks2`), which does have the necessary domain-specific context to do so.

875 The `kdbus` subsystem proposed for inclusion in the Linux kernel, which aims to
876 supersede the user-space implementation of D-Bus, has an additional restriction:
877 to minimize the amount of code in the TCB, it only parses the parts of a
878 message that are necessary for normal message-routing. As a result, it does not
879 discriminate between messages by their interface, member name or object-path,
880 only by attributes of the source and destination processes. This is another
881 reason why permissions checking for services such as disk-mounting must be
882 done at least partly by the domain-specific service such as `udisks2`.

883 The desired security policies for certain actions are also relatively complex. For
884 example, `udisks2` as deployed in a modern Linux desktop system such as Debian
885 8 would normally allow mounting devices if and only if:

- 886 • the requesting user is `root`, or
- 887 • the requesting user is in group `sudo`, or
- 888 • all of
 - 889 – the device is removable or external, and
 - 890 – the mount point is in `/media`, and
 - 891 – the mount options are reasonable, and
 - 892 – the device's `seat` (in multi-seat computing) matches one of the seats
 - 893 at which the user is logged-in, and
 - 894 – either
 - 895 * the user is in group `plugdev`, or
 - 896 * all of

- 897 · the user is logged-in locally, and
- 898 · the user is logged-in on the foreground virtual console

899 This is already complex, but it is merely a default, and is likely to be ad-
900 justed further for special purposes (such as a single-user development laptop, a
901 locked-down corporate desktop, or an embedded system like Apertis). It is not
902 reasonable to embed these rules, or a sufficiently powerful parser to read them
903 from configuration, into every system service that must impose such a policy.

904 **polkit’s solution**

905 polkit addresses this by dividing the authorization for actions into two phases.

906 In the first phase, the domain-specific service (such as `udisks2` for disk-
907 mounting) interprets the request and classifies it into one of several *actions*
908 which encapsulate the type of request. The principle is that the *action*
909 combines the verb and the object for the desired operation: if a security policy
910 would commonly produce different results when performing the same verb on
911 different objects, then they are represented by different actions. For example,
912 `udisks2` divides the high-level operation “mount a disk” into the actions
913 `org.freedesktop.udisks2.filesystem-mount`, `org.freedesktop.udisks2.filesystem-`
914 `mount-system`, `org.freedesktop.udisks2.filesystem-mount-other-seat` and
915 `org.freedesktop.udisks2.filesystem-fstab` depending on attributes of the disk. It
916 also gathers information about the process making the request, such as the
917 user ID and process ID. `polkit` clients do not currently record the LSM context
918 (AppArmor profile, etc.) used by MAC frameworks, but could be enhanced to
919 do so.

920 In the second phase, the service sends a D-Bus request to `polkit` with the desired
921 action, and the attributes of the process making the request. `polkit` processes
922 this request according to its configuration, and returns whether the request
923 should be obeyed.

924 In addition to “yes” or “no”, `polkit` security policies can request that a user, or a
925 user with administrative (root-equivalent) privileges, authenticates themselves
926 interactively; if this is done, `polkit` will not respond to the request until the user
927 has responded to the *polkit agent*, either by authenticating or by cancelling the
928 operation.

929 We recommend that this facility is not used with a password prompt in Apertis,
930 since that user experience would be highly distracting. For operations that
931 are deemed to be allowed or rejected by the platform designer, either the policy
932 should return “yes” or “no” instead of requesting authorization, or the platform-
933 provided `polkit` agent should return that result in response to authorization
934 requests without any visible prompting. However, a prompt for authorization,
935 without requiring authentication, might be a desired UX in some cases.

936 Recommendation

937 We recommend that Apertis should continue to provide polkit as a system ser-
938 vice. If this is not done, many system components will need to be modified to
939 refrain from carrying out the polkit check.

940 If the desired security policy is merely that a subset of user-level components
941 may carry out privileged actions via a given system service, and that all of
942 those user-level components have equal access, we recommend that Apertis’
943 polkit configuration should allow and forbid actions appropriately.

944 If it is required that certain user-level components can communicate with a given
945 system service with different access levels, we recommend enhancing polkit so
946 that it can query AppArmor, giving the *action* as a parameter, before carrying
947 out its own checks; this parallels what dbus-daemon currently does for SELinux
948 and AppArmor.

949 Alternative design: rely entirely on AppArmor checks

950 The majority of services that communicate with polkit do so through the
951 libpolkit-gobject library. This suggests an alternative design: the polkit service
952 and its D-Bus API could be removed entirely, and the AppArmor check
953 described above could be carried out in-process by each service, by providing
954 a “drop-in” compatible replacement for libpolkit-gobject that performed an
955 AppArmor query itself instead of querying polkit.

956 We do not recommend this approach: it would be problematic for services such
957 as systemd that do not use libpolkit-gobject, it would remove the ability for
958 the policy to be influenced by facts that are not known to AppArmor (such
959 as whether a user is logged-in and active), and it would be a large point of
960 incompatibility with upstream software.

961 Resource Usage Control

962 Resource usage here refers to the limitation and prioritization of hardware re-
963 sources usage. Common resources to limit usage of are CPU, memory, network,
964 disk I/O and IPC.

965 The proposed solution is Control Groups ([cgroup-v1](https://www.kernel.org/doc/Documentation/cgroup-v1/cgroups.txt)²³, [cgroup-v2](https://www.kernel.org/doc/Documentation/cgroup-v2.txt)²⁴), which is
966 a Linux kernel feature to limit, account, isolate and prioritize resource usage
967 of process groups. It protects the platform from resource exhaustion and DoS
968 attacks. The groups of processes can be dynamically created and modified. The
969 groups are divided by certain criteria and each group inherits limits from its
970 parent group.

971 The interface to configure a new group is via a pseudo file system that contains
972 directories to label the groups and each directory can have sub-directories (sub-

²³<https://www.kernel.org/doc/Documentation/cgroup-v1/cgroups.txt>

²⁴<https://www.kernel.org/doc/Documentation/cgroup-v2.txt>

973 groups). All those directories contain files that are used to set the parameters
974 or provide information about the groups.

975 By default, when the system is booted, the init system Collabora recommends
976 for this project, systemd, will assign separate control groups to each of the sys-
977 tem services. Collabora will further customize the cgroups of the base platform
978 to clearly separate system services, built-in applications and third-party applica-
979 tions. Support will be provided by Collabora for fine-tuning the cgroup profiles
980 for the final product.

981 **Imposing limits on I/O for block devices**

982 The *blkio* subsystem is responsible for dealing with I/O operations concerning
983 storage devices. It exports a number of controls that can be tuned by the
984 *cgroups* subsystem. Those controls fall into one of two possible strategies: setting
985 proportional weights for different cgroups or absolute upper bounds.

986 The main advantage of using proportional weights is that it allows the I/O
987 bandwidth to be saturated – if nothing else is running, an application always
988 gets all of the available I/O bandwidth. If, however, two or more processes in
989 different cgroups are competing for access to the I/O bandwidth, then they will
990 get a share that is proportional to the weights of their cgroups.

991 For example, suppose a process A is on a cgroup with weight **10** (the minimum
992 value possible) is working on mass-processing of photos, and process B is on a
993 cgroup with weight **1000** (the maximum). If process A is the only one making
994 I/O requests, it has the full available I/O bandwidth available for itself. As
995 soon as process B starts doing its own I/O requests, however, it will get around
996 **99%** of all the requests that get through, while process A will have only **1%**
997 for its requests.

998 The second strategy is setting an absolute limit on the I/O bandwidth,
999 often called *throttling*. This is done by writing how many bytes per
1000 second a cgroup should be able to transfer into a virtual file called
1001 **blkio.throttle.read_bps_device**, that lives inside the cgroup. This
1002 allows a great deal of control, but also means applications belonging to that
1003 cgroup are not able to take advantage of the full I/O bandwidth even if they
1004 are the only ones running at a given point in time.

1005 Specifying a default weight to all applications, lower weights for mass-processing
1006 jobs, and higher weights for time-critical applications is a good first step in not
1007 only securing the system, but also improving the user experience. The hard-
1008 limit of an upper bound on I/O operations can also serve as a way to make sure
1009 no application monopolizes the system's I/O.

1010 As is usual for tunables such as these, more specific details on what settings
1011 should be specified for which applications is something that needs to be devel-
1012 oped in an empirical, iterative way, throughout the development of the platform,

1013 and with actual target hardware. More details on the *blkio* subsystem support
1014 for cgroups can be obtained from [Linux documentation](#)²⁵.

1015 **Network filtering**

1016 Collabora recommends the use of the Netfilter framework to filter network traf-
1017 fic. Netfilter provides a set of hooks inside the Linux kernel that allow kernel
1018 modules to register callback functions with the network stack. A registered call-
1019 back function is then called back for every packet that traverses the respective
1020 hook within the network stack. Iptables is a generic table structure for the defi-
1021 nition of rule sets. Each rule within an iptable consists of a number of classifiers
1022 (iptables matches) and one connected action (iptables target).

1023 Netfilter, when used with iptables, creates a powerful network packet filtering
1024 system which can be used to apply policies to both IPv4 and IPv6 network
1025 traffic. A base rule set that blocks all incoming connections will be added to the
1026 platform by default, but port 80 access will be provided for devices connected
1027 to the Apertis hotspot, so they can access the web server hosted on the system.
1028 See the Connectivity document for more information on how this will work.

1029 The best way to do that seems to be to add acceptance rules for the prede-
1030 fined private network address space the DHCP server will use for clients of the
1031 hotspot.

1032 Collabora will offer support in refining the rules for the final product. Some
1033 network interactions may be handled by means of an AppArmor profile instead.

1034 **Protecting the driver assistance system from attacks**

1035 All communication with the driver assistance system will be done through a
1036 single service that can be talked to over D-Bus. This service will be the only
1037 process allowed to communicate with the driver assistance system. This means
1038 this service can belong to a separate user that will be the only one capable of
1039 executing the binary, which is Collabora's first recommendation.

1040 The daemon will use an IP connection to the driver assistance system, through
1041 a simple serial connection. This means that the character device entry for
1042 this serial connection shall be protected both by an [udev](#)²⁶ rule that assigns
1043 permissions for only this particular user. Access to the device entry should also
1044 be denied by the AppArmor profile which covers all other applications, making
1045 sure the daemon's profile allows it.

1046 Additionally, process namespace functionality can be used to make sure the
1047 driver assistance network interface is only seen and usable by the daemon that
1048 acts as gatekeeper. This is done by using a Linux-specific flag to the [clone](#)²⁷

²⁵<https://www.kernel.org/doc/Documentation/cgroup-v1/blkio-controller.txt>

²⁶<http://en.wikipedia.org/wiki/Udev>

²⁷<https://man7.org/linux/man-pages/man2/clone.2.html>

1049 system call, CLONE_NEWNET, which creates a new process with its network
1050 namespace limited to viewing the loopback interface.

1051 Having the process in its own cgroup also helps making it more robust, since
1052 Linux tries to be fair among cgroups, so is a good idea in general. Systemd
1053 already puts each service it starts in a separate cgroup, so making the daemon
1054 a system service is enough to take advantage of that fairness.

1055 The driver assistance communication daemon shall be started with this flag on,
1056 and have the network interface for talking to the driver assistance system be
1057 assigned to its namespace. When a network interface is assigned to a namespace
1058 only processes in that namespace can see and interact with it. This approach
1059 has the advantage of both protecting the interface from processes other than the
1060 proxy daemon, and protecting the daemon from the other network interfaces.

1061 **Protecting devices whose usage is restricted**

1062 One or more cameras will be available for Apertis to control, but they should
1063 not be accessed by any applications other than the ones required to implement
1064 the driver assistance use cases. Cameras are made available as device files in
1065 the /dev file system and can thus be controlled by both DAC permissions and
1066 by making the default AppArmor policy deny access to it as well.

1067 **Protecting the system from Internet threats**

1068 The Internet is riddled with malicious or buggy code that present threats other
1069 than those that come from direct attacks to the device's IP connection. The
1070 user of a system such as the Apertis may face attacks such as emails that link
1071 to viruses, trojan horses and other kinds of malware, web sites that mislead the
1072 user or that try to cause the system to misbehave or become unresponsive.

1073 There is no single answer to such threats, but care should be exercised to make
1074 each of the subsystems and applications involved in dealing with content from
1075 the Internet robust to such malicious and buggy content. The solutions that
1076 have been presented in the previous sections are essential for that.

1077 The first line of defence is, of course, a good firewall setup that disallows incom-
1078 ing connections, protecting the IP interfaces of the device. The second line of
1079 defence is making sure that the applications that deal with those threats are
1080 well-written. Web browsers have also grown many techniques to protect the
1081 user from both direct attacks such as denial of service or private information
1082 disclosure and indirect forms of attack such as social engineering.

1083 The basic rule of protecting the user from web content in a browser is essentially
1084 assuming all content is untrusted. There are fewer APIs that allow a web
1085 application to interact with local resources such as local files than there are
1086 for native applications. The ones that do exist are usually made possible only
1087 through express user interaction, such as when the user selects a file to upload.

1088 Newer API that allows access to device capabilities such as the geolocation
1089 facilities only work after the user has granted permission.

1090 Browsers also try to make sure users are not fooled into believing they are in
1091 a different site than the one they are really at, known as “phishing”, which
1092 is one of the main social engineering attacks used on the web. The basic SSL
1093 certificate checks, along with proper UI to warn the user about possible problems
1094 can help prevent [man-in-the-middle](#)²⁸ attacks. The HTTP library used by the
1095 clutter port of WebKit is able to verify certificates using the system’s trusted
1096 Certificate Authorities.

1097 The *ca-certificates* package in Debian and Ubuntu carry those

1098 In addition to those basic checks, WebKit includes a feature called *XSS Auditor*
1099 which implements a number of rules and checks to prevent [cross-site scripting](#)²⁹
1100 attacks, sometimes used to mix elements from both a fake and a legitimate site.

1101 The web browser can be locked down, like any other application, to limit the
1102 resources it can use up or get access to, and Collabora will be helping build an
1103 AppArmor profile for it. This is what protects the system from the browser in
1104 case it is exploited. By limiting the amount of damage the browser can do to
1105 the system itself, any exploits are also hindered from reaching the rest of the
1106 system.

1107 It is also important that the UI of the browser behaves well in general. For
1108 instance, user interfaces that make it easy to run executables downloaded from
1109 the web make the system more vulnerable to attacks. A user interface that
1110 makes it easier to distinguish the domain from the rest of the URI is [sometimes](#)³⁰
1111 employed to help careful users be sure they are where they wanted to go.

1112 Automatically loading pages that were loaded or loading when the browser had
1113 to be terminated or crashed would make it hard for the user to regain control of
1114 the browser too. Existing browsers usually load an alternate page with a button
1115 the user can click to load the page, which is probably also a good idea for the
1116 Apertis browser.

1117 Collabora evaluated taking the WebKit Clutter port to the new WebKit2 archi-
1118 tecture as part of the Apertis project; as of 2012 it was deemed risky given the
1119 time and budget constraints.

1120 As of 2015, it has been decided that Apertis will switch away from WebKit
1121 Clutter and onto the GTK+ port, which is already built upon the WebKit2
1122 architecture. The main feature of that architecture is that it has several dif-
1123 ferent classes of processes: the UI process deals with user interaction, the Web
1124 processes render page contents, the Network process mediates access to remote
1125 data, and the Plugin processes are responsible for running plugins.

²⁸https://en.wikipedia.org/wiki/Man-in-the-middle_attack

²⁹https://en.wikipedia.org/wiki/Cross-site_scripting

³⁰<https://chrome.googleblog.com/2010/10/understanding-omnibox-for-better.html>

1126 The fact that the processes are separate provides a great way of locking them
1127 down properly. The Web processes, which are the most likely to be exploited in
1128 case of successful attack are also the one that needs the least privileges when it
1129 comes to interfacing with the system, so the AppArmor policies that apply to
1130 it can be very strict. If a limited set of plugins is supported, the same can be
1131 applied to the Plugin processes. In fact, the WebKit codebase contains support
1132 for using seccomp filters (see [Seccomp](#)) to sandbox the WebKit2 processes. It
1133 may be a useful addition in the future.

1134 **Other sources of potential exploitation**

1135 Historically, document viewers and image loaders have had vulnerabilities ex-
1136 ploited in various ways to execute arbitrary code. PDF and spreadsheet files, for
1137 instance, feature domain-specific scripting languages. These scripting facilities
1138 are often sandboxed and limited in what they can do, but have been a source of
1139 security issues nevertheless. Images do not usually feature scripting, but their
1140 loaders have historically been the source of many security issues, caused by pro-
1141 gramming errors, such as buffer overflows. These issues have been exploited to
1142 cause denial of service or run arbitrary code.

1143 Although these cases do deserve mention specifically for the inherent risk they
1144 bring, there is no silver bullet for this problem. Keeping applications up-to-
1145 date with security fixes, using hardening techniques such as stack protection,
1146 discussed in [Stack protection](#), and locking the application down to its minimum
1147 access requirements are the tools that can be employed to reduce the risks.

1148 **Launching applications based on MIME type**

1149 It is common in the desktop world to allow launching an application through
1150 the files that they are able to read. For instance, while reading email the user
1151 may want to view an attachment; by “opening” the attachment an application
1152 that is able to display that kind of file would be launched with the attachment
1153 as an argument.

1154 Collabora is recommending that all kinds of application launching always go
1155 through the application manager. By doing that, there will be a centralized
1156 way of controlling and limiting the launching of applications through MIME or
1157 other types of content association, including being able to blacklist applications
1158 with known security issues, for instance.

1159 **Secure Software Distribution**

1160 Secure software updates are a very important topic in the security of the plat-
1161 form. Checking integrity and authenticity of the software packages installed in
1162 the system is crucial; an altered package might compromise the security of the
1163 whole platform.

1164 This section is only related with security aspects, not the whole software distri-
1165 bution update mechanism, which will be covered in a separate document. The
1166 technology used for this is the same one used by Ubuntu. It's called [Secure](#)
1167 [APT](#)³¹ and was introduced in Debian in 2005.

1168 Every Debian or Ubuntu package that is made available through an APT repos-
1169 itory is hashed and the hash is stored on the file that lists what packages are
1170 available, called the "Packages" file. That file is then hashed and the hash is
1171 stored in the [Release file](#)³², which is signed using a PGP private key.

1172 The public PGP key is shipped along with the product. When the package
1173 manager obtains updates or new packages it checks that the signature on the
1174 Release file is valid, and that all hashes match. The security of this approach
1175 relies on the fact that any tampering with the package or with the Packages
1176 file would make the hashes not match, and any changes done to the Release file
1177 would render the signature invalid.

1178 Additional public keys can be distributed through upgrades to a package that
1179 ships installed; this is how Debian and Ubuntu distribute their public keys.
1180 This mechanism can be used to add new third-party providers, or to replace the
1181 keys used by the app store. Collabora will provide documentation and provide
1182 assistance on setting up the package repositories and signing infrastructure.

1183 **Secure Boot**

1184 The objective of [secure boot](#)³³ is to ensure that the system is booted using
1185 sanctioned components. The extent to which this is ultimately taken will vary
1186 between implementations, some may use secure boot avoid system kernel re-
1187 placement, whilst others may also use it to ensure a [Trusted Execution Envi-](#)
1188 [ronment](#)³⁴ is loaded without interference.

1189 The steps required to implement secure boot are vendor specific and thus the
1190 full specification for the solution depends on a definition from the specific silicon
1191 vendor, such as Freescale.

1192 A solution that has been adopted by Freescale in the past is the High Assurance
1193 Boot (HAB), which ensures two basic attributes: authenticity and integrity.
1194 This is done by validating that the code image originated from a trusted source
1195 (authenticity), and verify that the code is in its original form (integrity). HAB
1196 uses digital signatures to validate the code images and thereby establishes the
1197 security level of the system.

1198 To verify the signature the device uses the Super Root Key (SRK) which is
1199 stored on-chip in non-volatile memory. To enhance the robustness of HAB

³¹<https://wiki.debian.org/SecureApt>

³²https://wiki.debian.org/SecureApt#Secure_apt_groundwork:_checksums

³³<https://em.pages.apertis.org/apertis-website/architecture/secure-boot/>

³⁴<https://em.pages.apertis.org/apertis-website/concepts/op-tee/>

1200 security, multiple Super Root keys (RSA public keys) are stored in internal
1201 ROM. Collabora recommends the utilization of SRK with 2048-bit RSA keys.

1202 In case a signature check fails because of incomplete or broken upgrade it should
1203 be possible to fall back to an earlier kernel automatically. Details of how that
1204 would be achieved are only possible after details about the hardware support for
1205 such a feature are provided by Freescale, and are probably best handled in the
1206 document about safely upgrading, system snapshots and rolling back updates.

1207 More discussion of system integrity checking, its limitations and alternatives
1208 can be found later on, when the IMA system is investigated. See [Conclusion
1209 regarding IMA and EVM](#) in particular.

1210 The signature and verification processes are described in the Freescale white
1211 paper “Security Features of the i.MX31 and i.MX31L”.

1212 **Data encryption and removal**

1213 **Data encryption**

1214 The objective of data encryption is to protect the user data for security and
1215 privacy reasons. In the event of the car being stolen, for instance, important
1216 user data such as passwords should not be easily readable. While providing full
1217 disk encryption is both not practical and harmful to overall system performance,
1218 encryption of a limited set of the data such as saved passwords is possible.

1219 The [Secrets D-Bus service](#)³⁵ is a very practical way of storing passwords for
1220 applications. Its [GNOME implementation](#)³⁶ provides an easy to use API, uses
1221 [locked down memory](#)³⁷ when handling the passwords and encrypted storage for
1222 the passwords on disk. Collabora will provide these tools in the base platform
1223 and will support the implementation of secure password storage in the applica-
1224 tions that will be developed.

1225 One unresolved issue for data encryption, whether via the Secrets service, a
1226 full-disk encryption system (as optionally used in Android) or some other im-
1227 plementation, is that a secret token must be provided in order to decrypt the
1228 encrypted data. This is normally a password, but prompting for a password is
1229 likely to be undesired in an automotive environment. One possible implementa-
1230 tion is to encode an unpredictable token in each car key, and use those tokens
1231 to decrypt stored secrets, with any of the keys for a particular car equally able
1232 to decrypt its data. In the simplest version of that implementation, loss of all
1233 of the car keys would result in loss of access to the encrypted data, but the car
1234 vendor could retain copies of the keys’ tokens (and a record of which car is the
1235 relevant one) if desired

³⁵<https://specifications.freedesktop.org/secret-service/latest/re01.html>

³⁶<https://wiki.gnome.org/Projects/GnomeKeyring>

³⁷<https://wiki.gnome.org/Projects/GnomeKeyring/Memory>

1236 **Data removal**

1237 A data removal feature is important to guarantee that personal user data that
1238 resides on the device can be removed before the car changes hands, for instance.
1239 Returning the device configuration to factory is also important because it allows
1240 resetting of any customization and preferences.

1241 Collabora recommends these features be implemented by making sure user data
1242 and settings are stored in a separate storage area. By removing this area both
1243 user data and configuration are removed.

1244 Proper data wiping is only necessary to defeat forensic analysis of the hardware
1245 and would not pose a privacy risk for the simpler cases of the car changing
1246 hands. Such procedures rely on hardware support, so would only be possible
1247 if that is in place, and even in that case they may be very time consuming.
1248 It's also worth noting that flash storage will usually perform wear levelling,
1249 which defeats software techniques such as writing over a block multiple times.
1250 Collabora recommends not supporting this feature.

1251 **Stack Protection**

1252 It is recommended to enable stack protection, which provides protection against
1253 stack-based attacks such as a stack buffer overflow. Ubuntu, the distribution
1254 used as a base for Apertis has enabled a stack protection mechanism offered by
1255 GCC called [SSP](#)³⁸. Modern processors have the capability to mark memory seg-
1256 ments (like stack) executable or not, which can be used by applications to make
1257 themselves safer. Some initial tests with the Freescale kernel 2.6.38 provided on
1258 imx6 board shows correct enforcement behaviour.

1259 Memory protection techniques like disabling execution of stack or heap memory
1260 are not possible with some applications, in particular execution engines such as
1261 programming language interpreters that include a just in time compiler, includ-
1262 ing the ones for JavaScript currently present in most web engines. Cases such
1263 as this and also cases in which the limitations should apply but are not being
1264 respected will be documented.

1265 Collabora will also document best practices for building software with this fea-
1266 ture so that others can take advantage of stack protection for higher level li-
1267 braries and applications.

1268 **Confining applications in containers**

1269 **LXC Containment**

1270 [LXC](#)³⁹ is a solution that was developed to be a lightweight alternative to virtu-
1271 alization, built on top of cgroups and namespaces, mainly. Its main focus is on

³⁸<https://wiki.ubuntu.com/GccSsp>

³⁹<https://linuxcontainers.org/>

1272 servers, though. The goal is to separate processes completely, including using
1273 a different file system and a different network. This means the applications
1274 running inside an LXC container are effectively running in a different system,
1275 for all practical purposes. While this does have the potential of helping protect
1276 the main system, it also brings with it huge problems with the integration of
1277 the application with the system.

1278 For graphical applications the X server will have to run with a TCP port open, so
1279 that applications running in a container are able to connect, 3D acceleration will
1280 be impossible or very difficult to achieve for applications running in a container.
1281 D-Bus setup will be significantly more complex.

1282 Besides increasing the complexity of the system, LXC essentially duplicates
1283 functionality offered by cgroups, AppArmor, and the Netfilter firewall. When
1284 LXC was originally suggested it was to be used only for system services. By
1285 using systemd the Apertis system will already have every service on the system
1286 running on their own cgroup, and properly locked down by AppArmor profiles.
1287 This means adding LXC would only add redundancy and no additional value.

1288 Protection for the driver assistance and limiting the damage root can do to the
1289 system can both be achieved by AppArmor policies, which can be applied to
1290 both system services and applications, as opposed to LXC, which would only
1291 be safely applicable to services. There are no advantages at all in using LXC
1292 for these cases. Limiting resources can also be easily done through cgroups,
1293 which will not be limited to system services, too. For these reasons Collabora
1294 recommends against using LXC.

1295 **Making X11, D-Bus and 3D work with LXC**

1296 For the sake of completeness, this section provides a description of possible
1297 solutions for LXC shortcomings.

1298 LXC creates what, for all practical purposes, is a separate system. X supports
1299 TCP socket connections, so it could be made to work, but that would require
1300 opening the TCP port and that would be another interface that needs protec-
1301 tion.

1302 D-Bus has the same pros and cons of X11 – it can be connected to over a [TCP](#)
1303 [port](#)⁴⁰, but that again increases the surface area that needs to be protected, and
1304 adds complexity for managing the connection. It is also not a popular use case
1305 so it does not get a lot of testing.

1306 3D over network has not yet been made to work on networked X. All solutions
1307 available, such as [Virtual GL](#)⁴¹ involve a lot of copying back and forth, which
1308 would make performance suffer substantially, which is something that needs to
1309 be avoided given the high importance of performance on Apertis requirements.

⁴⁰<https://www.freedesktop.org/wiki/Software/DBusRemote/>

⁴¹<https://virtualgl.org/>

1310 Collabora’s perspective is that using LXC for applications running on the user
1311 session adds nothing that cannot be achieved with the means described in this
1312 document, while at the same time adding complexity and indirection.

1313 **The Flatpak framework**

1314 [Flatpak](https://flatpak.org/)⁴² is a framework for “sandboxed” desktop applications, under develop-
1315 ment by several GNOME developers. Like LXC, it makes use of existing Linux
1316 infrastructure such as cgroups (see [Resource usage control](#)) and namespaces.

1317 Unlike LXC, Flatpak’s design goals are focused on confining individual applica-
1318 tions within a system, which makes it an interesting technology for Apertis. We
1319 recommend researching Flatpak further, and evaluating its adoption as a way
1320 to reduce the development effort for our sandboxed applications.

1321 One secondary benefit of Flatpak is that by altering the application bundle’s
1322 view of the filesystem, it can provide a way to manage major-version upgrades
1323 without app-visible compatibility breaks, by continuing to run app bundles that
1324 were designed for the old “runtime” in an environment more closely resembling
1325 that old version, while using the new “runtime” for app bundles that have been
1326 tested in that environment.

1327 **The IMA Linux Integrity Subsystem**

1328 The goal of the Integrity Measurement Architecture ([IMA](#)⁴³) subsystem is to
1329 make sure that a given set of files have not been altered and are authentic –
1330 in other words, provided by a trusted source. The mechanism used to provide
1331 these two features are essentially keeping a database of file hashes and RSA
1332 signatures. IMA does not protect the system from changes, it is simply a way
1333 of knowing that changes have been made so that measures to fix the problem
1334 can be taken as quickly as possible. The authenticity module of IMA is still not
1335 available, so we won’t be discussing it.

1336 In its simpler mode of operation, with the default policy IMA will intercept
1337 calls that cause memory mapping and execution of a file or any access done by
1338 root and perform a hash of the file before the access goes through. This means
1339 execution of all binaries and loading of all libraries are intercepted. To hash a
1340 file, IMA needs to read the whole file and calculate a cryptographic sum of its
1341 contents. That hash is then kept in kernel memory and extended attributes of
1342 the file system, for further verification after system reboots.

1343 This means that running any program will cause its file and any libraries it uses
1344 to be fully read and cryptographically processed before anything can be done
1345 with it, which causes a significant impact in the performance of the system. A
1346 10% impact has been [reported](#)⁴⁴ by the IMA authors in boot time on a default

⁴²<https://flatpak.org/>

⁴³<https://sourceforge.net/p/linux-ima/wiki/Home/>

⁴⁴https://blog.linuxplumbersconf.org/2009/slides/David-Stafford-IMA_LPC.pdf

1347 Fedora. There are no detailed information on how the test was performed, but
1348 the performance impact of IMA is mainly caused by increased I/O required to
1349 read the whole of all executable and library files used during the boot for hash
1350 verification. All executables will take longer to start up after a system boot
1351 up because they need to be fully read and hashed to verify they match what's
1352 recorded (if any recording exists).

1353 The fact that the hashes are maintained in the file system extended attributes,
1354 and are otherwise created from scratch when the file is first mapped or executed
1355 means that in this mode IMA does not protect the system from modification
1356 while offline: an attacker with physical access to the device can boot using a
1357 different operating system modify files and reset the extended attributes. Those
1358 changes will not be seen by IMA.

1359 To overcome this problem IMA is able to work with the hardware's trusted
1360 platform module through the extended verification module (EVM⁴⁵), added⁴⁶ to
1361 Linux in version 3.2: hashes of the extended attributes are signed by the trusted
1362 platform module (TPM) hardware, and written to the file system as another
1363 extended attribute. For this to work, though, TPM hardware is required. The
1364 fact that TPM modules are currently only widely available and supported for
1365 Intel-based platforms is also a problem.

1366 **Conclusion regarding IMA and EVM**

1367 IMA and EVM both are only useful for detecting that the system has been
1368 modified. They do so using a method that incurs significant impact on the per-
1369 formance, particularly application startup and system boot up. Considering the
1370 strict boot up requirements for the Apertis system, this fact alone indicates that
1371 IMA and EVM are suboptimal solutions. However, EVM and IMA also suffer
1372 from being very new technologies as far as Linux mainline is concerned, and
1373 have not been integrated and used by any major distributions. This means im-
1374 plementing them in Apertis means incurring into significant development costs.

1375 In addition to that, Collabora believes that the goals of detecting breaches,
1376 protecting the base system and validating the authenticity of system files are
1377 attained in much better ways through other means, such as keeping the system
1378 files separate and read-only during normal operation, and using secure methods
1379 for installing and updating software, such as those described in [Protecting the
1380 driver assistance system from attacks](#).

1381 For these reasons Collabora advises against the usage of IMA and EVM for this
1382 project. An option to provide some security for the system in this case is making
1383 it hard to disconnect and remove the actual storage device from the system, to
1384 minimize the risk of tampering.

⁴⁵<https://sourceforge.net/p/linux-ima/wiki/Home/#linux-extended-verification-module-evm>

⁴⁶https://kernelnewbies.org/Linux_3.2#head-03576b924303bb0fad19cabb35efcbd33ceed084

1385 **Seccomp**

1386 **Seccomp**⁴⁷ is a sandboxing mechanism in the Linux kernel. In essence, it is a
1387 way of specifying which system calls a process or thread should be able to make.
1388 As such, it is very useful to isolate processes that have strict responsibilities.
1389 For instance, a process that should not be able to write or read from the disk
1390 should not be able to make an *open* system call.

1391 Most security tools that were discussed in this document provide a system-
1392 wide infrastructure and protect the system in a general way from outside the
1393 application's process. As opposed to those, **seccomp** is something that is very
1394 granular and very application-specific: it needs to be built into the application
1395 source code.

1396 In other words, applications need to be written with an architecture which allows
1397 a separation of concerns, isolating the work that deals with untrusted processes
1398 or data to a separate process or thread that will then use **seccomp** filters to limit
1399 the amount of damage it is able to do through system calls.

1400 For use by applications, **seccomp** needs to be enabled in the kernel that is
1401 shipped with the middleware. There is a library called **libseccomp**⁴⁸, which
1402 provides a more convenient way of specifying filters. Should feature be used
1403 and made it available through the SDK, the **seccomp** support can be enabled
1404 in the kernel and **libseccomp** can be shipped in the middleware image provided
1405 by Collabora.

1406 The **seccomp** filter should be used on system services designed for Apertis whose
1407 architecture and intended functionality allow dropping privileges. Suppose, for
1408 instance, that Apertis has a health management daemon which needs to be able
1409 to kill applications that misbehave but has no need whatsoever of writing data
1410 to a file descriptor. It might be possible to design that daemon to use **seccomp**
1411 to filter out system calls such as **open** and **write**. The **open** system call might
1412 need to be allowed to go through for opening files for reading, depending on how
1413 the health daemon monitors processes – it might need to read information from
1414 files in the **/proc** file system, for instance. For that reason, filtering for **open**
1415 would need to be more granular, just disallowing it being called with certain
1416 arguments.

1417 Depending on how the health management daemon works it would also not
1418 need to fork new processes itself, so filtering out system calls such as **fork**,
1419 and **clone** is a possibility. As explained before, to take advantage of these
1420 opportunities, the architecture of such a daemon needs to be thought through
1421 from the onset with these limitations in mind. Opportunities, such as the ones
1422 discussed here, should be evaluated on a case-by-case basis, for each service
1423 intended for deployment on Apertis.

⁴⁷https://www.kernel.org/doc/Documentation/prctl/seccomp_filter.txt

⁴⁸<https://lwn.net/Articles/494252/>

1424 AppArmor and seccomp are complementary technologies, and can be used to-
1425 gether. Some of their purposes overlap (for example, denying filesystem write
1426 access altogether could be achieved equally well with either technology), and
1427 they are both part of the kernel and hence in the TCB.

1428 The main advantage of seccomp over AppArmor is that it inhibits all system
1429 calls, however obscure: all system calls that were not considered when writ-
1430 ing a policy are normally denied. Its in-kernel implementation is also simpler,
1431 and hence potentially more robust, than AppArmor. This makes it suitable
1432 for containing a module whose functionality has been designed to be strongly
1433 focused on computation with minimal I/O requirements, for example the render-
1434 ing modules of browser engines such as WebKit2. However, its applicability to
1435 code that was not designed to be suitable for seccomp is limited. For example,
1436 if the confined module has a legitimate need to open files, then its seccomp filter
1437 will need to allow broad categories of file to be opened.

1438 The main advantage of AppArmor over seccomp is that it can perform finer-
1439 grained checking on the arguments and context of a system call, for example
1440 allowing filesystem reads from files owned by the process's uid, but denying
1441 reads from other uids' files. This makes it possible to confine existing general-
1442 purpose components using AppArmor, with little or no change to the confined
1443 component. Conversely, it groups together closely-related system calls with sim-
1444 ilar security implications into an abstract operation such as "read" or "write",
1445 making it considerably easier to write correct profiles.

1446 **The role of the app store process for security**

1447 The model which is used for the application stores should precludes automated
1448 publishing of software to the store by developers. All software, including new
1449 versions of existing applications will have to go through an audit before publish-
1450 ing.

1451 The app store vetting process will generate the final package that will reach
1452 the store front. That means only signatures made by the app store curator's
1453 cryptographic keys will be valid, for instance. Another consequence of this
1454 approach is that the curator will have not only the final say on what goes in,
1455 but will also be able to change pieces of the package to, say, disallow a given
1456 permission the application's author specified in the application's manifest.

1457 This also presents a good opportunity to convert high level descriptions such
1458 as the permissions in the manifest and an overall description of files used into
1459 concrete configuration files such as AppArmor profiles in a centralized fashion,
1460 and provides the curator with the ability to fine tune said configurations for
1461 specific devices or even to rework how a given resource is protected itself, with
1462 no need for intervention from third-parties.

1463 Most importantly, from the perspective of this document, is the fact that the app
1464 store vetting process provides an opportunity for final screening of submissions

1465 for security issues or bad practices both in terms of code and user interface, so
1466 that should be taken into consideration.

1467 **How does security affect developer usage of a device?**

1468 How security impacts a developer mode depends heavily on how that developer
1469 mode of work is specified. This chapter considers that the two main use cases
1470 for such a mode would be installing an application directly to the target through
1471 the Eclipse *install to target* plugin and running a remote debugging session for
1472 the application, both of which are topics discussed in the SDK design.

1473 The *install to target* functionality that was made available through an Eclipse
1474 plugin uses an **sftp** connection with an arbitrary user and password pair to
1475 connect to the device. This means that putting the device in developer mode
1476 should ensure the **ssh** server is running and add an exception to the firewall
1477 rules discussed in [Network filtering](#), to allow an inbound connection to port 22.

1478 Upon login, the SSH server will start user sessions that are not constrained by
1479 the AppArmor infrastructure. In particular the white-list policy discussed in
1480 section [Implementing a white list approach](#), will not apply to ssh user sessions.
1481 This means the user the IDE will connect with needs file system access to the
1482 directory where the application needs to be installed or be able to tell the
1483 application installer to install it.

1484 The procedure for installing an application using an **sftp** connection is not
1485 too different from the *install app from USB stick* use case described in the
1486 Applications document, that similarity could be exploited to share code for
1487 these features.

1488 The main difference is the developer mode would need to either ignore signature
1489 checking or accept a special “developer” signature for the packages. Decision on
1490 how to implement this piece of the feature needs a more complete assessment
1491 of proposed solutions on how the app store and system DRM could work, and
1492 how open (or openable) the end user devices will be.

1493 Running the application for remote debugging also requires that the **gdb-**
1494 **server**’s default port, 2345, be open. Other than that, the main security
1495 constraint that will need to be tweaked when the system is put in developer
1496 mode is AppArmor. While under developer mode AppArmor should probably
1497 be put in complain mode, since the application’s own profile will not yet exist.

1498 **Further discussion**

1499 This chapter lists topics that require further thinking and/or discussion, or a
1500 more detailed design. These may be better written as Wiki pages rather than
1501 formal designs, given they require and benefit from iterating on an implementa-
1502 tion.

- 1503 • Define which cgroups ([Resource usage control](#)) to have, how they will be
1504 created and managed
- 1505 • Define exactly what Netfilter rules ([Network filtering](#)) should be installed
1506 and how they will be made effective at boot time
- 1507 • Evaluate Flatpak ([The Flatpak framework](#))