



Security

Contents

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

46	Further discussion	42
47	This document discusses and details solutions for the security requirements of	
48	the Apertis system.	
49	Security boundaries and threat model describes the various aspects of the secu-	
50	rity model, and the threat model for each.	
51	Local attacks to obtain private data or damage the system, including those	
52	performed by malicious applications that get installed in the device somehow	
53	or through exploiting a vulnerable application are covered in Mandatory access	
54	control (MAC). It is also the main line of defense against malicious email attach-	
55	ments and web content, and for minimizing the damage that root is able to do	
56	are also mainly covered by the MAC infrastructure. This is the main security	
57	infrastructure of the system, and the depth of the discussion is proportional to	
58	its importance.	
59	Denial of Service attacks through abuse of system resources such as CPU and	
60	memory are covered by Resource usage control . Attacks coming in through	
61	the device's network connections and possible strategies for firewall setup are	
62	covered in Network filtering	
63	Attacks to the driver assistance system coming from the infotainment system are	
64	handled by many of these security components, so it is discussed in a separate	
65	section: Protecting the driver assistance system from attacks . Internet threats	
66	are the main subject of 10, Protecting the system from internet threats .	
67	Secure software distribution discusses how to provide ways to make installing	
68	and upgrade software secure, by guaranteeing packages are unchanged, undam-	
69	aged and coming from a trusted repository.	
70	Secure boot for protecting the system against attacks done by having physical	
71	access to the device is discussed in Secure boot . Data encryption and removal ,	
72	is concerned with features whose main focus is to protect the privacy of the	
73	user.	
74	Stack protection , discusses simple but effective techniques that can be used	
75	to harden applications and prevent exploitation of vulnerabilities. Confining	
76	applications in containers , discusses the pros and cons of using the lightweight	
77	Linux Containers infrastructure for a system like Apertis.	
78	The IMA Linux integrity subsystem , wraps up this document by discussing how	
79	the Integrity Measurement Architecture works and what features it brings to	
80	the table, and at what cost.	

81 Terminology

82 Privilege

83 A component that is able to access data that other components cannot is said
84 to be *privileged*. If two components have different privileges –that is, at least

85 one of them can do something that the other cannot –then there is said to be a
86 *privilege boundary* between them.

87 **Trust**

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

96 One subtlety of [Apertis’ app-centric design](https://apertis-website-0b3586.pages.apertis.org/concepts/archive/application/applications/)¹ is that there is a privilege boundary
97 between *application bundles* even within the context of one user. As a result, a
98 multi-user design has two main layers in its security model: system-level security
99 that protects users from each other, and user-level security that protects a user’s
100 apps from each other. Where we need to distinguish between those layers, we
101 will refer to the *TCB for security between users* or the *TCB for security*
102 *between app bundles* respectively.

103 **Integrity, confidentiality and availability**

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

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

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

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

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

¹<https://apertis-website-0b3586.pages.apertis.org/concepts/archive/application/applications/>

122 Security boundaries and threat model

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

124 Security between applications

125 The Apertis platform provides for installation of Flatpak application bundles,
126 which may come from the platform developer or third parties. These are de-
127 scribed in the [Application Framework](https://apertis-website-0b3586.pages.apertis.org/concepts/archive/application_framework/application-framework/)² design document.

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

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

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

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

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

²https://apertis-website-0b3586.pages.apertis.org/concepts/archive/application_framework/application-framework/

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

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

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

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

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

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

186 **Communication between applications**

187 In a system that supports capabilities such as data handover between applica-
188 tions, it is likely that pairs of application bundles can communicate with each
189 other, either mediated by platform services or directly. The [Interface Discovery](#)³
190 and [Data Sharing](#)⁴ pages have more information on this topic.

191 The mechanisms for communicating between application bundles, or between
192 application bundle and the platform, are to be classified into *public* and *non-*
193 *public* interfaces. Application bundles may enumerate all of the providers of
194 *public* interfaces and may communicate with those providers, but it is not accept-
195 able for application bundles to enumerate or communicate with the providers
196 of *non-public* interfaces. The platform is considered to be trusted, and may
197 communicate with any *public* or *non-public* interface.

³https://apertis-website-0b3586.pages.apertis.org/concepts/archive/application_framework/interface_discovery/

⁴https://apertis-website-0b3586.pages.apertis.org/architecture/application/data_sharing/

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

203 **Security between users**

204 The Apertis platform is potentially a multi-user environment; see the [Multiuser](#)⁵
205 design document for full details. This results in a two-level hierarchy: users are
206 protected from each other, and within the context of a user, apps are protected
207 from other apps.

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

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

217 **Security between platform services**

218 Within the platform, not all services and components require the same access
219 to platform data.

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

227 The remaining platform components have considerations similar to those ap-
228 plied to applications: they should have “least privilege”. Because platform com-
229 ponents are part of the operating system image, they can be assumed not to be
230 malicious; however, it is desirable to have “defence in depth” against design or
231 implementation flaws that might allow an attacker to gain control of them. As
232 such, the threat model for these components is that we assume an attacker gains
233 control over the component (arbitrary code execution), and the desired property
234 is that the integrity, confidentiality and availability impact is minimized, given

⁵https://apertis-website-0b3586.pages.apertis.org/concepts/archive/application_security/multiuser/

235 the constraint that the component’s privileges must be sufficient for it to carry
236 out its normal operation.

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

244 **Security between the device and the network**

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

254 **Physical security**

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

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

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

267 **Solutions adopted by popular platforms**

268 As background for the discussions of this document, the following sections pro-
269 vide an overview of the approaches other mobile platforms have chosen for secu-
270 rity, including an explanation of the trade-offs or assumptions where necessary.

271 Android

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

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

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

285 Early versions of Android worked under the assumption that the system will
286 be used by a single user, and no attempt was made towards supporting any
287 kind of multi-user use case. Based on this assumption, Android re-purposed the
288 concept of UNIX user ID (uid), making each application run as a different user
289 ID. This allows for very tight control over what files each application is able to
290 access by simply using user-based permissions; this provides isolation between
291 applications (**Security between applications**). In later Android versions, which
292 do have multi-user support, user IDs are used to provide two separate security
293 boundaries –isolating applications from each other, and isolating users from each
294 other (**Security between users**) –with one user ID per (user, app) pair. This is
295 discussed in more detail in the [Multiuser design document](#)⁶.

296 The system’s main file system is mounted read-only to protect against unau-
297 thorized tampering with system files (integrity for platform data, **Security be-**
298 **tween platform services**). In addition, by default, the bootloader will check the
299 integrity of the root filesystem built on hardware-backed key pairs, thus any
300 tampering attempts would fail the boot process. The only way to skip these
301 checks is by unlocking the bootloader, which requires a full device wipe, thus
302 it is not possible to access the encrypted user data on a device with a locked
303 bootloader, even with physical access (**Physical security**).

304 Verified Boot in Android, [https://source.android.com/security/veri](https://source.android.com/security/verifiedboot)
305 [fiedboot](https://source.android.com/security/verifiedboot)

306 Encryption of the user data partition through the standard *dm-crypt* kernel
307 facility (confidentiality despite physical access, **Physical security**) is supported
308 if the user configures a password for their device. Users using gesture-based or
309 other unlock mechanisms are unable to use this feature.

⁶https://apertis-website-0b3586.pages.apertis.org/concepts/archive/application_security/multiuser/

310 Older versions of Android had an unrestricted root user, but recent versions
311 have heavily utilized SELinux to restrict the access levels of system processes.
312 This ensures that a compromised process is unable to access resources that it
313 should not be able to interact with, even if the process is running as root. Even
314 for services that have direct access to the block devices, verified boot would
315 ensure that a compromised system partition would not be booted.

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

318 The idea of restricting the services an application can use to those specified in
319 the application's manifest also exists in Android. Before installation, Android
320 shows a list of system services the application intends to access and installation
321 only initiates if the user agrees. This differs slightly from the [Applications](#)
322 [design in Apertis](#)⁷, in which some permissions are subject to prompting similar
323 to Android's, while other permissions are checked by the app store curator and
324 unconditionally granted on installation.

325 Android provides APIs to verify a process has a given permission, but no central
326 control is built into the API layer or the IPC mechanism as planned for Apertis
327 –checking whether a caller has the required permissions to make that call is left
328 to the service or application that provides the IPC interface or API, similar to
329 how most GNOME services work by using [polkit](#)⁸ (see section 6 for more on this
330 topic).

331 See, for instance, how the A2DP service verifies the caller has the
332 required permission: [https://cs.android.com/android/platform/su](https://cs.android.com/android/platform/superproject/+ /master:packages/apps/Bluetooth/src/com/android/bluetooth/a2dp/A2dpService.java;l=629-632;drc=7ff83d0d573c518a219ff2f3d5b803dade7d44b1)
333 [perproject/+ /master:packages/apps/Bluetooth/src/com/android/](https://cs.android.com/android/platform/superproject/+ /master:packages/apps/Bluetooth/src/com/android/bluetooth/a2dp/A2dpService.java;l=629-632;drc=7ff83d0d573c518a219ff2f3d5b803dade7d44b1)
334 [bluetooth/a2dp/A2dpService.java;l=629-632;drc=7ff83d0d573c51](https://cs.android.com/android/platform/superproject/+ /master:packages/apps/Bluetooth/src/com/android/bluetooth/a2dp/A2dpService.java;l=629-632;drc=7ff83d0d573c518a219ff2f3d5b803dade7d44b1)
335 [8a219ff2f3d5b803dade7d44b1](https://cs.android.com/android/platform/superproject/+ /master:packages/apps/Bluetooth/src/com/android/bluetooth/a2dp/A2dpService.java;l=629-632;drc=7ff83d0d573c518a219ff2f3d5b803dade7d44b1)

336 No effort is made specifically towards thwarting applications misbehaving and
337 causing a Denial of Service on system services or the IPC mechanism. Android
338 uses two very simple strategies to forcibly stop an application: 1) it kills appli-
339 cations when the device is out of memory; 2) it notifies the user of [unresponsive](#)
340 [applications](#)⁹ and allows them to force the application to close, similar to how
341 GNOME does it.

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

⁷<https://apertis-website-0b3586.pages.apertis.org/concepts/archive/application/applications/>

⁸<https://www.freedesktop.org/software/polkit/docs/latest/polkit.8.html>

⁹<http://developer.android.com/guide/practices/design/responsiveness.html>

349 from the application with a timeout; if the timeout is hit, then the application
350 is considered not responding.

351 iOS

352 iOS is a closed platform, so [details are sometimes difficult to obtain](#)¹⁰, but Apple
353 does use some Open Source components (at the lower levels, in particular). iOS
354 has an [application sandbox](#)¹¹ that is very similar in functionality to AppArmor,
355 discussed below. The technology is based on Mandatory Access Control pro-
356 vided by the [TrustedBSD](#)¹² project and has been marketed under the *Seatbelt*
357 name.

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

364 Seatbelt is considered to be the single canonical solution to sandboxing applica-
365 tions on iOS; this is in contrast with Linux, in which AppArmor is one option
366 among many (system calls can be mediated by seccomp, the [Secure Computing](#)
367 [API](#)¹³ described in section 17 of this document, in addition to up to one MAC
368 layer such as AppArmor, SELinux or Smack).

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

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

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

377 For instance, an application can query the GPS for location; when that happens,
378 the system will take over and present the user with a request for permission. If
379 the user accepts the request will be successful and the application will be white-
380 listed for future queries. The same goes for interacting with the camera: the
381 application can request a picture be taken, but the UI that is presented for
382 taking the picture is controlled by the system as is actual interaction with the
383 camera.

¹⁰https://github.com/V3RL4223N3/Programming/blob/master/iOS/iOS_Security_May12.pdf

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

¹²<http://www.trustedbsd.org/mac.html>

¹³<http://lwn.net/Articles/475043/>

384 This is analogous to the way in which Linux services can use [polkit](#)¹⁴ to mediate
385 privileged actions (see section 6), although on iOS the authorization step is
386 specifically considered to be an implementation detail of the API used, whereas
387 some Linux services do make the calling application aware of whether there was
388 an interactive authorization step.

389 Mandatory Access Control

390 The goal of the Linux Discretionary Access Control (DAC) is a separation of
391 multiple users and their data ([Security between users](#), [Security between plat-](#)
392 [form services](#)). The policies are based on the identity of a subject or their groups.
393 Since in Apertis applications from the same user should not trust each other ([Security between applications](#)), the utilization of a Mandatory Access Control
394 (MAC) system is recommended. MAC is implemented in Linux by one of the
395 available Linux Security Modules (LSM).
396

397 Linux Security Modules (LSM)

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

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

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

410 **SELinux** [SELinux](#)¹⁵ is one of the most well-known LSMs. It is supported by
411 default in Red Hat Enterprise Linux and Fedora. It is infamous for how difficult
412 it is to maintain the security policies; however, being the most flexible and not
413 having any limitation regarding what it can label, it is the reference in terms of
414 features. For every user or process, SELinux assigns a context which consists of
415 a role, user name and domain/type. The circumstances under which the user is
416 allowed to enter into a certain domain must be configured into the policies.

417 SELinux works by applying rules defined by a policy when kernel-mediated
418 actions are taken. Any file-like object in the system, including files, directories,
419 and network sockets can be labeled. Those labels are set on file system objects
420 using extended file system attributes. That can be problematic if the file system

¹⁴<https://www.freedesktop.org/software/polkit/docs/latest/polkit.8.html>

¹⁵http://selinuxproject.org/page/Main_Page

421 that is being used in a given product or situation lacks support for extended
422 attributes. While support has been built for storing labels in frequently used
423 networking file systems like NFS, usage in newer file systems may be challenging.
424 Note that BTRFS does support extended attributes.

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

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

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

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

444 **TOMOYO Linux** TOMOYO Linux¹⁶ focuses on the behavior of a system
445 where every process is created with a certain purpose and allows each process
446 to declare behaviors and resources needed to achieve their purposes. TOMOYO
447 Linux is not officially supported by any popular Linux distribution.

448 **SMACK** Simplicity is the primary design goal of SMACK¹⁷. It was used
449 by MeeGo before that project was cancelled; Tizen¹⁸ appears to be the only
450 general-purpose Linux distribution using SMACK as of 2015.

451 SMACK works by assigning labels to the same system objects and to processes as
452 SELinux does; similar capabilities were proposed by Intel for D-Bus integration,
453 but their originators did not follow up on reviews¹⁹, and the changes were not
454 merged. SMACK also relies on extended file system attributes for the labels,
455 which means it suffers from the same shortcomings that come from that as
456 SELinux.

¹⁶<https://tomoyo.sourceforge.net/>

¹⁷<http://schaufler-ca.com/>

¹⁸<https://developer.tizen.org/sdk.html>

¹⁹https://bugs.freedesktop.org/show_bug.cgi?id=47581

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

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

467 **AppArmor** Of all LSM modules that were reviewed, Application Armor (Ap-
468 pArmor²⁰) can be seen as the most focused on application containment.

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

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

481 Since AppArmor uses the path of the binary for profile selection, changing the
482 path through manipulation of the file system name space (i.e. through links
483 or mount points) is a potential way of working-around the limits that are put
484 in place; while this is cited as a weakness, in practice it is not an issue, since
485 restrictions exist to block anyone trying to do this. Creation of symbolic links
486 is only allowed if the process doing so is allowed to access the original file, and
487 links are followed to enforce any policy assigned to the binary they link to.
488 Confined processes are also not allowed to mount file systems unless they are
489 given explicit permission.

490 Here’s an example of how restricting ping’s ability to create raw sockets cannot
491 be worked around through linking -lines beginning with \$ represent commands
492 executed by a normal user, and those starting with # have been executed by
493 the root user:

²⁰<https://gitlab.com/apparmor/apparmor/-/wikis/home>

```
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 #
```

AppArmor restriction applying to file system links

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

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

Integration with D-Bus was developed by Canonical and shipped in Ubuntu for several years, before being merged upstream in dbus-daemon 1.9 and AppArmor 2.9. The implementation includes patches to AppArmor's user-space tools, to make the new D-Bus rules known to the profile parser, and to dbus-daemon, so that it will check with AppArmor before allowing a request.

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

Creating a new profile for AppArmor is a reasonably simple process: a new profile is generated automatically running the program under AppArmor's profile generator, [aa-genprof](https://gitlab.com/apparmor/apparmor/-/wikis/Profiling_with_tools)²¹, and exercising its features so that the profile generator can capture all of the accesses the application is expected to make. After the initial profile has been generated it must be reviewed and fine-tuned by manual

²¹https://gitlab.com/apparmor/apparmor/-/wikis/Profiling_with_tools

521 editing to make sure the permissions that are granted are not beyond what is
 522 expected.

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

527 Comparison

528 Since all those Linux Security Modules rely on the same kernel API and have the
 529 same overall goals, the features and resources they are able to protect are very
 530 similar, thus not much time will be spent covering those. The policy format and
 531 how control over the system and its components is exerted varies from framework
 532 to framework, though, which leads to different limitations. The table below has
 533 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

534 Comparison of LSM features

535 Historically LSM modules have focused on kernel-mediated accesses, such as
 536 access to file system objects and network resources. Modern systems, though,
 537 have several important features being managed by user-space daemons. D-Bus is
 538 one such daemon and is specially important since it is the IPC mechanism used
 539 by those daemons and applications for communication. There is clear benefit
 540 in allowing D-Bus to cooperate with the LSM to restrict what applications can
 541 talk to which services and how.

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

549 Like D-Bus's built-in support for applying "policy" to message delivery, AppArmor
 550 mediation of D-Bus messages has separate checks for whether the sender may
 551 send a message to the recipient, and whether the recipient may receive a message
 552 from the sender. Either or both of these can be used, and the message will only
 553 succeed if both sending and receiving were allowed. The sender's AppArmor

554 profile determines whether it can send (usually conditional on the profile name
 555 of the recipient), and the recipient's AppArmor profile determines whether it can
 556 receive (either conditional on the profile name of the sender, or unconditionally),
 557 so some coordination between profiles is needed to express a particular high-level
 558 security policy.

559 The main difference between the SELinux and SMACK label-based mediation in
 560 terms of features is how granular you can get. With the [D-Bus additions to the](#)
 561 [AppArmor profile language](#)²², for instance, in addition to specifying which ser-
 562 vices can be called upon by the constrained process it is also possible to specify
 563 which interfaces and paths are allowed or denied. This is unlike [SELinux media-](#)
 564 [tion](#)²³, which only checks whether a given client can talk to a given service. One
 565 caveat regarding fine-grained (interface- and path-based) D-Bus access control
 566 is that it is often not directly useful, since the interface and path is not nec-
 567 essarily sufficient to determine whether an action should be allowed or denied
 568 (for example, [Motivation for polkit](#) describes why this is the case for the udisks
 569 service).

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

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

576 The recommendation on the selection of the framework is a combination of the
 577 adoption of the framework by existing distributions, features, maintainability,
 578 cost of deployment and experience of the developers involved. The table below
 579 contains a comparison of the adoption of the existing security models. Only
 580 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 Computin
AppArmor	SUSE, OpenSUSE, Ubuntu	20 Oct 2010	SUSE, Canonical
SMACK	Tizen	11 Aug 2007	Intel, Samsung2
TOMOYO		10 Jun 2009	NTT Data Corp.

581 Comparison of LSM adoption and maturity

582 **Performance impact**

583 The performance impact of MAC solutions depends heavily on the workload
 584 of the application, so it's hard to rely upon a single metric. It seems major

²²[https://gitlab.com/apparmor/apparmor/-/wikis/AppArmor_Core_Policy_Reference#](https://gitlab.com/apparmor/apparmor/-/wikis/AppArmor_Core_Policy_Reference#dbus-rules)
[dbus-rules](#)

²³<http://dbus.freedesktop.org/doc/dbus-daemon.1.html#lbAg>

585 adopters of these technologies are not too concerned about their real-world
586 impact, even though they may be expressive in benchmarks, since there are no
587 recent measurements of performance impact for the major MAC solutions.

588 That said, early tests indicate that SELinux has a performance impact [floating](#)
589 [around 7% to 10%](#)²⁴, with tasks that are more CPU intensive having *less* im-
590 pact, since they are not making many system calls that are checked. SELinux
591 performs checks on every operation that touches a labeled resource, so when
592 reading or writing a file all read/write operations would cause a check. That
593 means making larger operations instead of several smaller ones would also make
594 the overhead go down.

595 AppArmor generally does fewer checks than SELinux since only operations that
596 open, map or execute a file are checked: the individual read/write operations
597 that follow are not checked independently. Novell's documentation and FAQs
598 state a 0.2% overhead is expected on best-case scenarios –writing a big file, for
599 instance, with a 2% overhead in worst-case scenarios (an application touching
600 lots of files once). Collabora's own testing on a 2012 x86-64 system puts the
601 worst case scenario leaning towards the 5% range. The test measured reading
602 3000 small files with a hot disk cache, and ranged from ~89ms to ~94ms average
603 duration.

604 SMACK's performance characteristics should be similar to that of SELinux,
605 given their similar approach to the problem. SMACK has been tested for a TV
606 embedded scenario which has shown performance degradation from 0% all the
607 way to 30% on a worst-case scenario of deleting a 0-length file. Degradation
608 varied greatly depending on the benchmark used.

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

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

624 Where higher-frequency communication is required, D-Bus's file descriptor pass-
625 ing feature can be used to negotiate a private channel (a pipe or socket) between

²⁴<http://blog.larsstrand.no/2007/11/rhel5-selinux-benchmark.html>

two processes. This negotiation can be as simple as a single D-Bus method call, and only incurs the cost of AppArmor checks once (when it is first set up). Subsequent messages through the private channel bypass D-Bus and are not checked individually by AppArmor, avoiding any per-message overhead in this case.

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

Conclusion

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

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

SELinux offers the most flexible configuration of security policies, but it introduces a lot of complexity on the setup and maintenance of the policies, not only for distribution maintainers but also for application developers and packagers, which impacts on the costs of the solution. It is quite common to see Fedora users running into problems caused by SELinux configuration issues.

AppArmor stands out as a good middle-ground between flexibility and maintainability while at the same time having significant adoption: by the biggest end-user desktop distribution (Ubuntu) and by one of the two biggest enterprise distributors (SUSE). The fact that it is the security solution already supported and included in the Ubuntu distribution, which is closely related to the base of the Apertis platform (Debian), minimizes the initial effort to create a secure baseline and reduces the effort needed to maintain it. Since Ubuntu ships with AppArmor, some of the services and applications will already be covered by the profiles that can be pulled from Ubuntu. Creation of additional profiles is made easy by the profile generator tool that comes with AppArmor. It records everything the application needs to do during normal operation, and allows for further refining after the recording session is done.

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

667 **AppArmor Policy and management examples** Looking at a few exam-
668 ples might help better visualize how AppArmor works, and what creating new
669 policies entices. 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 $
```

670 AppArmor policy shipped for ping in Ubuntu

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

679 If an attack was able to execute arbitrary code by hijacking the ping process,
680 then that is all it would be able to do. No reading of /etc/passwd would be
681 allowed, for instance. If ping was a very core feature of the device and starts
682 failing because of a bad policy, it is possible to disable security enforcement just
683 for ping, leaving the rest of the system secured (something that would not be
684 easily done with SMACK or SELinux), by running *aa-disable* with ping's path
685 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
```

686 A symbolic link to disable the ping AppArmor policy

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

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

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

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

706 **Profiles for libraries** AppArmor profiles are always attached to a binary.
707 That means there is no way to attach a profile to every program that uses a
708 given library. However, developers can write files called *abstractions* with rules
709 that can be included through the *#include* directive, similar to how libraries
710 work for programming. Using this feature Collabora has written rules for the
711 WebKit library, for instance, that can be included by the browser application
712 as well as by any application that uses the library.

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

717 The rationale for this is libraries are only pieces of code that could be included
718 by the applications themselves, and it would be very time-consuming and error
719 prone having to specify each and every library and module the application may

720 need to use directly or that would be used indirectly by a library used by the
721 application.

722 Collabora recommends that proprietary libraries that are used only by one or
723 a few services should be installed in a private location, such as the application's
724 directory. That would put those libraries outside of the paths covered by
725 the existing rules, and they would thus be out of reach for any other applica-
726 tion already, given the white-list approach to session lockdown, as discussed in
727 **Implementing a white list approach**.

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

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

735 **Application installation and upgrades** For installations and upgrades to
736 be performed, no changes to the running system's security are necessary, since
737 the processes that manage upgrade, including the creation of the required snap-
738 shots will have enough power given to them

739 An application's profile is read at startup time. That means an application that
740 has been upgraded will only be contained with the new rules after it has been
741 restarted. The D-Bus integration works by querying the kernel interface for the
742 PID it is communicating with, not its own, so D-Bus itself does not need to be
743 restarted when new profiles are installed.

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

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

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

755 **A note about root** As has been demonstrated in listing *AppArmor restric-*
756 *tion applying to file system links*, AppArmor can restrict even the powers of the
757 root user. Most platforms do not try to limit that power in any way, since if an
758 attacker has breached the system to get root privileges it's likely that all bets
759 are already off. That said, it should be possible to limit the root user's ability to

760 modify the AppArmor profiles, leaving that task solely for the package manager
761 (see the Applications design for details).

762 **Implementing a white-list approach** Collabora recommends the use of
763 a white-list approach in which the app-launcher will be confined to a policy
764 that denies almost everything, and specific permissions will be granted by the
765 application profiles. This means all applications will only be able to access what
766 is expressively allowed by their specific policies, providing Apertis with a very
767 tight least-privilege implementation.

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

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

779 Then permission is given for the shell to run the *dconf* command. *dconf* is
780 GNOME's settings storage. Notice that we have *p* as the prefix for *x* this time.
781 This means we want this application to use its own rules; if no rules had been
782 specified, then AppArmor would have fallen back to using the shell's confinement
783 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 }

```

784 Sample profiles for implementing white-listing

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

```

790 % cat .config/dconf/user
791 cat: .config/dconf/user: Permission denied
792 % dconf read /apps/empathy/ui/show-offline
793 true
794 %

```

795 Effects of white-list approach profiles

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

800 **polkit (PolicyKit)**

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

808 **Motivation for polkit**

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

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

822 Applying the desired policy using the kernel's permission checks is not possible,
823 because mounting and unmounting a USB stick is fundamentally the same sys-
824 tem call as mounting and unmounting any other file system, which is not desired:
825 if ordinary users can make arbitrary mount system calls, they can mount a file
826 system that contains setuid executables and achieve privilege escalation. As a
827 result, the kernel disallows direct mount and unmount actions by unprivileged
828 processes; instead, user processes may request that a privileged system process
829 carries out the desired action. In the case of device mounting, Apertis uses the
830 privileged udisks2 service to mount and unmount devices.

831 In environments that use a MAC framework like AppArmor, actions that would
832 normally be allowed can also become privileged: for instance, in a framework
833 for sandboxed applications, most apps should not be allowed to record audio.
834 The resulting AppArmor adjustments prevent carrying out these actions directly.
835 The result is that, again, the only way to achieve them is that a service with a
836 suitable privilege carries out the action (perhaps with a mandatory user interface
837 prompt first, as in certain iOS features).

838 These privileged requests are commonly sent via the D-Bus interprocess com-
839 munication (IPC) system; indeed, this is one of the purposes for which D-Bus
840 was designed. D-Bus has facilities for allowing or forbidding messages between

particular processes in a somewhat fine-grained way, either directly or mediated by MAC frameworks. However, this has the same issue as the kernel's checks for direct mount operations: the generic D-Bus IPC framework does not understand the context of the messages. For example, it can allow or forbid messages that ask to mount a device, but cannot discriminate based on whether the device in question is a removable device or a system partition, because it does not have that domain-specific information.

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

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

- the requesting user is *root*, or
- the requesting user is in group *sudo*, or
- all of
 - the device is removable or external, and
 - the mount point is in `/media`, and
 - the mount options are reasonable, and
 - the device's *seat* (in multi-seat computing) matches one of the seats at which the user is logged-in, and
 - either
 - * the user is in group *plugdev*, or
 - * all of
 - the user is logged-in locally, and
 - the user is logged-in on the foreground virtual console

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

polkit's solution

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

In the first phase, the domain-specific service (such as `udisks2` for disk-mounting) interprets the request and classifies it into one of several *actions* which encapsulate the type of request. The principle is that the *action*

877 combines the verb and the object for the desired operation: if a security
878 policy would commonly produce different results when performing the same
879 verb on different objects, then they are represented by different actions. For
880 example, `udisks2` divides the high-level operation “mount a disk” into the actions
881 `org.freedesktop.udisks2.filesystem-mount`, `org.freedesktop.udisks2.filesystem-`
882 `mount-system`, `org.freedesktop.udisks2.filesystem-mount-other-seat` and
883 `org.freedesktop.udisks2.filesystem-fstab` depending on attributes of the disk. It
884 also gathers information about the process making the request, such as the
885 user ID and process ID. `polkit` clients do not currently record the LSM context
886 (AppArmor profile, etc.) used by MAC frameworks, but could be enhanced to
887 do so.

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

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

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

904 Recommendation

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

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

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

917 **Alternative design: rely entirely on AppArmor checks** The majority
918 of services that communicate with polkit do so through the libpolkit-gobject
919 library. This suggests an alternative design: the polkit service and its D-Bus
920 API could be removed entirely, and the AppArmor check described above could
921 be carried out in-process by each service, by providing a “drop-in”compatible
922 replacement for libpolkit-gobject that performed an AppArmor query itself in-
923 stead of querying polkit.

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

929 Resource Usage Control

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

933 The proposed solution is Control Groups ([cgroup-v1](https://www.kernel.org/doc/Documentation/cgroup-v1/cgroups.txt)²⁵, [cgroup-v2](https://www.kernel.org/doc/Documentation/cgroup-v2.txt)²⁶), which is
934 a Linux kernel feature to limit, account, isolate and prioritize resource usage
935 of process groups. It protects the platform from resource exhaustion and DoS
936 attacks. The groups of processes can be dynamically created and modified. The
937 groups are divided by certain criteria and each group inherits limits from its
938 parent group.

939 The interface to configure a new group is via a pseudo file system that contains
940 directories to label the groups and each directory can have sub-directories (sub-
941 groups). All those directories contain files that are used to set the parameters
942 or provide information about the groups.

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

949 Imposing limits on I/O for block devices

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

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

²⁶<https://www.kernel.org/doc/Documentation/cgroup-v2.txt>

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

959 For example, suppose a process A is on a cgroup with weight **10** (the minimum
960 value possible) is working on mass-processing of photos, and process B is on a
961 cgroup with weight **1000** (the maximum). If process A is the only one making
962 I/O requests, it has the full available I/O bandwidth available for itself. As
963 soon as process B starts doing its own I/O requests, however, it will get around
964 **99%** of all the requests that get through, while process A will have only **1%** for
965 its requests.

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

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

978 As is usual for tunables such as these, more specific details on what settings
979 should be specified for which applications is something that needs to be devel-
980 oped in an empirical, iterative way, throughout the development of the platform,
981 and with actual target hardware. More details on the *blkio* subsystem support
982 for cgroups can be obtained from [Linux documentation](https://www.kernel.org/doc/Documentation/cgroup-v1/blkio-controller.txt)²⁷.

983 Network filtering

984 Collabora recommends the use of the Netfilter framework to filter network traffic.
985 Netfilter provides a set of hooks inside the Linux kernel that allow kernel modules
986 to register callback functions with the network stack. A registered callback
987 function is then called back for every packet that traverses the respective hook
988 within the network stack. Iptables is a generic table structure for the definition
989 of rule sets. Each rule within an iptable consists of a number of classifiers
990 (iptables matches) and one connected action (iptables target).

991 Netfilter, when used with iptables, creates a powerful network packet filtering
992 system which can be used to apply policies to both IPv4 and IPv6 network
993 traffic. A base rule set that blocks all incoming connections will be added to the

²⁷<https://www.kernel.org/doc/Documentation/cgroup-v1/blkio-controller.txt>

994 platform by default, but port 80 access will be provided for devices connected
995 to the Apertis hotspot, so they can access the web server hosted on the system.
996 See the Connectivity document for more information on how this will work.

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

1000 Collabora will offer support in refining the rules for the final product. Some
1001 network interactions may be handled by means of an AppArmor profile instead.

1002 **Protecting the driver assistance system from attacks**

1003 All communication with the driver assistance system will be done through a
1004 single service that can be talked to over D-Bus. This service will be the only
1005 process allowed to communicate with the driver assistance system. This means
1006 this service can belong to a separate user that will be the only one capable of
1007 executing the binary, which is Collabora's first recommendation.

1008 The daemon will use an IP connection to the driver assistance system, through
1009 a simple serial connection. This means that the character device entry for
1010 this serial connection shall be protected both by an `udev`²⁸ rule that assigns
1011 permissions for only this particular user. Access to the device entry should also
1012 be denied by the AppArmor profile which covers all other applications, making
1013 sure the daemon's profile allows it.

1014 Additionally, process namespace functionality can be used to make sure the
1015 driver assistance network interface is only seen and usable by the daemon that
1016 acts as gatekeeper. This is done by using a Linux-specific flag to the `clone`²⁹
1017 system call, `CLONE_NEWNET`, which creates a new process with its network
1018 namespace limited to viewing the loopback interface.

1019 Having the process in its own cgroup also helps making it more robust, since
1020 Linux tries to be fair among cgroups, so is a good idea in general. Systemd
1021 already puts each service it starts in a separate cgroup, so making the daemon
1022 a system service is enough to take advantage of that fairness.

1023 The driver assistance communication daemon shall be started with this flag on,
1024 and have the network interface for talking to the driver assistance system be
1025 assigned to its namespace. When a network interface is assigned to a namespace
1026 only processes in that namespace can see and interact with it. This approach
1027 has the advantage of both protecting the interface from processes other than the
1028 proxy daemon, and protecting the daemon from the other network interfaces.

²⁸<http://en.wikipedia.org/wiki/Udev>

²⁹<https://man7.org/linux/man-pages/man2/clone.2.html>

1029 **Protecting devices whose usage is restricted**

1030 One or more cameras will be available for Apertis to control, but they should
1031 not be accessed by any applications other than the ones required to implement
1032 the driver assistance use cases. Cameras are made available as device files in
1033 the /dev file system and can thus be controlled by both DAC permissions and
1034 by making the default AppArmor policy deny access to it as well.

1035 **Protecting the system from Internet threats**

1036 The Internet is riddled with malicious or buggy code that present threats other
1037 than those that come from direct attacks to the device's IP connection. The
1038 user of a system such as the Apertis may face attacks such as emails that link
1039 to viruses, trojan horses and other kinds of malware, web sites that mislead the
1040 user or that try to cause the system to misbehave or become unresponsive.

1041 There is no single answer to such threats, but care should be exercised to make
1042 each of the subsystems and applications involved in dealing with content from
1043 the Internet robust to such malicious and buggy content. The solutions that
1044 have been presented in the previous sections are essential for that.

1045 The first line of defence is, of course, a good firewall setup that disallows incom-
1046 ing connections, protecting the IP interfaces of the device. The second line of
1047 defence is making sure that the applications that deal with those threats are
1048 well-written. Web browsers have also grown many techniques to protect the
1049 user from both direct attacks such as denial of service or private information
1050 disclosure and indirect forms of attack such as social engineering.

1051 The basic rule of protecting the user from web content in a browser is essentially
1052 assuming all content is untrusted. There are fewer APIs that allow a web
1053 application to interact with local resources such as local files than there are
1054 for native applications. The ones that do exist are usually made possible only
1055 through express user interaction, such as when the user selects a file to upload.
1056 Newer API that allows access to device capabilities such as the geolocation
1057 facilities only work after the user has granted permission.

1058 Browsers also try to make sure users are not fooled into believing they are in
1059 a different site than the one they are really at, known as “phishing”, which
1060 is one of the main social engineering attacks used on the web. The basic SSL
1061 certificate checks, along with proper UI to warn the user about possible problems
1062 can help prevent [man-in-the-middle](#)³⁰ attacks. The HTTP library used by the
1063 clutter port of WebKit is able to verify certificates using the system's trusted
1064 Certificate Authorities.

1065 The *ca-certificates* package in Debian and Ubuntu carry those

1066 In addition to those basic checks, WebKit includes a feature called *XSS Auditor*

³⁰https://en.wikipedia.org/wiki/Man-in-the-middle_attack

1067 which implements a number of rules and checks to prevent [cross-site scripting](#)³¹
1068 attacks, sometimes used to mix elements from both a fake and a legitimate site.

1069 The web browser can be locked down, like any other application, to limit the
1070 resources it can use up or get access to, and Collabora will be helping build an
1071 AppArmor profile for it. This is what protects the system from the browser in
1072 case it is exploited. By limiting the amount of damage the browser can do to
1073 the system itself, any exploits are also hindered from reaching the rest of the
1074 system.

1075 It is also important that the UI of the browser behaves well in general. For
1076 instance, user interfaces that make it easy to run executables downloaded from
1077 the web make the system more vulnerable to attacks. A user interface that
1078 makes it easier to distinguish the domain from the rest of the URI is [sometimes](#)³²
1079 employed to help careful users be sure they are where they wanted to go.

1080 Automatically loading pages that were loaded or loading when the browser had
1081 to be terminated or crashed would make it hard for the user to regain control of
1082 the browser too. Existing browsers usually load an alternate page with a button
1083 the user can click to load the page, which is probably also a good idea for the
1084 Apertis browser.

1085 Collabora evaluated taking the WebKit Clutter port to the new WebKit2 archi-
1086 tecture as part of the Apertis project; as of 2012 it was deemed risky given the
1087 time and budget constraints.

1088 As of 2015, it has been decided that Apertis will switch away from WebKit
1089 Clutter and onto the GTK+ port, which is already built upon the WebKit2
1090 architecture. The main feature of that architecture is that it has several dif-
1091 ferent classes of processes: the UI process deals with user interaction, the Web
1092 processes render page contents, the Network process mediates access to remote
1093 data, and the Plugin processes are responsible for running plugins.

1094 The fact that the processes are separate provides a great way of locking them
1095 down properly. The Web processes, which are the most likely to be exploited in
1096 case of successful attack are also the one that needs the least privileges when it
1097 comes to interfacing with the system, so the AppArmor policies that apply to
1098 it can be very strict. If a limited set of plugins is supported, the same can be
1099 applied to the Plugin processes. In fact, the WebKit codebase contains support
1100 for using seccomp filters (see [Seccomp](#)) to sandbox the WebKit2 processes. It
1101 may be a useful addition in the future.

1102 Other sources of potential exploitation

1103 Historically, document viewers and image loaders have had vulnerabilities ex-
1104 ploited in various ways to execute arbitrary code. PDF and spreadsheet files, for
1105 instance, feature domain-specific scripting languages. These scripting facilities

³¹https://en.wikipedia.org/wiki/Cross-site_scripting

³²<https://chrome.googleblog.com/2010/10/understanding-omnibox-for-better.html>

are often sandboxed and limited in what they can do, but have been a source of security issues nevertheless. Images do not usually feature scripting, but their loaders have historically been the source of many security issues, caused by programming errors, such as buffer overflows. These issues have been exploited to cause denial of service or run arbitrary code.

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

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

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

Secure Software Distribution

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

This section is only related with security aspects, not the whole software distribution update mechanism, which will be covered in a separate document. The technology used for this is the same one used by Debian since 2005: [Secure APT](#)³³.

Every Debian package that is made available through an APT repository is hashed and the hash is stored on the file that lists what packages are available, called the “Packages” file. That file is then hashed and the hash is stored in the [Release file](#)³⁴, which is signed using a PGP private key.

The public PGP key is shipped along with the product. When the package manager obtains updates or new packages it checks that the signature on the Release file is valid, and that all hashes match. The security of this approach relies on the fact that any tampering with the package or with the Packages

³³<https://wiki.debian.org/SecureApt>

³⁴https://wiki.debian.org/SecureApt#Secure_apr_groundwork:_checksums

1143 file would make the hashes not match, and any changes done to the Release file
1144 would render the signature invalid.

1145 Additional public keys can be distributed through upgrades to a package that
1146 ships installed; this is how Debian and Ubuntu distribute their public keys.
1147 This mechanism can be used to add new third-party providers, or to replace the
1148 keys used by the app store. Collabora will provide documentation and provide
1149 assistance on setting up the package repositories and signing infrastructure.

1150 Secure Boot

1151 The objective of [secure boot](#)³⁵ is to ensure that the system is booted using
1152 sanctioned components. The extent to which this is ultimately taken will vary
1153 between implementations, some may use secure boot avoid system kernel re-
1154 placement, whilst others may also use it to ensure a [Trusted Execution Envi-](#)
1155 [ronment](#)³⁶ is loaded without interference.

1156 The steps required to implement secure boot are vendor specific and thus the
1157 full specification for the solution depends on a definition from the specific silicon
1158 vendor, such as Freescale.

1159 A solution that has been adopted by Freescale in the past is the High Assurance
1160 Boot (HAB), which ensures two basic attributes: authenticity and integrity.
1161 This is done by validating that the code image originated from a trusted source
1162 (authenticity), and verify that the code is in its original form (integrity). HAB
1163 uses digital signatures to validate the code images and thereby establishes the
1164 security level of the system.

1165 To verify the signature the device uses the Super Root Key (SRK) which is
1166 stored on-chip in non-volatile memory. To enhance the robustness of HAB
1167 security, multiple Super Root keys (RSA public keys) are stored in internal
1168 ROM. Collabora recommends the utilization of SRK with 2048-bit RSA keys.

1169 In case a signature check fails because of incomplete or broken upgrade it should
1170 be possible to fall back to an earlier kernel automatically. Details of how that
1171 would be achieved are only possible after details about the hardware support for
1172 such a feature are provided by Freescale, and are probably best handled in the
1173 document about safely upgrading, system snapshots and rolling back updates.

1174 More discussion of system integrity checking, its limitations and alternatives
1175 can be found later on, when the IMA system is investigated. See [Conclusion](#)
1176 [regarding IMA and EVM](#) in particular.

1177 The signature and verification processes are described in the Freescale white
1178 paper “Security Features of the i.MX31 and i.MX31L”.

³⁵<https://apertis-website-0b3586.pages.apertis.org/architecture/platform/secure-boot/>

³⁶<https://apertis-website-0b3586.pages.apertis.org/concepts/distribution/op-tee/>

1179 Data encryption and removal

1180 Data encryption

1181 The objective of data encryption is to protect the user data for security and
1182 privacy reasons. In the event of the car being stolen, for instance, important
1183 user data such as passwords should not be easily readable. While providing full
1184 disk encryption is both not practical and harmful to overall system performance,
1185 encryption of a limited set of the data such as saved passwords is possible.

1186 The [Secrets D-Bus service](#)³⁷ is a very practical way of storing passwords for
1187 applications. Its [GNOME implementation](#)³⁸ provides an easy to use API, uses
1188 [locked down memory](#)³⁹ when handling the passwords and encrypted storage for
1189 the passwords on disk. Collabora will provide these tools in the base platform
1190 and will support the implementation of secure password storage in the applica-
1191 tions that will be developed.

1192 One unresolved issue for data encryption, whether via the Secrets service, a
1193 full-disk encryption system (as optionally used in Android) or some other im-
1194 plementation, is that a secret token must be provided in order to decrypt the
1195 encrypted data. This is normally a password, but prompting for a password is
1196 likely to be undesired in an automotive environment. One possible implementa-
1197 tion is to encode an unpredictable token in each car key, and use those tokens
1198 to decrypt stored secrets, with any of the keys for a particular car equally able
1199 to decrypt its data. In the simplest version of that implementation, loss of all
1200 of the car keys would result in loss of access to the encrypted data, but the car
1201 vendor could retain copies of the keys/tokens (and a record of which car is the
1202 relevant one) if desired

1203 Data removal

1204 A data removal feature is important to guarantee that personal user data that
1205 resides on the device can be removed before the car changes hands, for instance.
1206 Returning the device configuration to factory is also important because it allows
1207 resetting of any customization and preferences.

1208 Collabora recommends these features be implemented by making sure user data
1209 and settings are stored in a separate storage area. By removing this area both
1210 user data and configuration are removed.

1211 Proper data wiping is only necessary to defeat forensic analysis of the hardware
1212 and would not pose a privacy risk for the simpler cases of the car changing
1213 hands. Such procedures rely on hardware support, so would only be possible
1214 if that is in place, and even in that case they may be very time consuming.
1215 It's also worth noting that flash storage will usually perform wear levelling,

³⁷<https://specifications.freedesktop.org/secret-service/latest/re01.html>

³⁸<https://wiki.gnome.org/Projects/GnomeKeyring>

³⁹<https://wiki.gnome.org/Projects/GnomeKeyring/Memory>

1216 which defeats software techniques such as writing over a block multiple times.
1217 Collabora recommends not supporting this feature.

1218 **Stack Protection**

1219 It is recommended to enable stack protection, which provides protection against
1220 stack-based attacks such as a stack buffer overflow. Debian, the distribution
1221 used as a base for Apertis has enabled a stack protection mechanism offered by
1222 GCC called [SSP](https://wiki.ubuntu.com/GccSsp)⁴⁰. Modern processors have the capability to mark memory seg-
1223 ments (like stack) executable or not, which can be used by applications to make
1224 themselves safer. Some initial tests with the Freescale kernel 2.6.38 provided on
1225 imx6 board shows correct enforcement behaviour.

1226 Memory protection techniques like disabling execution of stack or heap memory
1227 are not possible with some applications, in particular execution engines such as
1228 programming language interpreters that include a just in time compiler, includ-
1229 ing the ones for JavaScript currently present in most web engines. Cases such
1230 as this and also cases in which the limitations should apply but are not being
1231 respected will be documented.

1232 Collabora will also document best practices for building software with this fea-
1233 ture so that others can take advantage of stack protection for higher level li-
1234 braries and applications.

1235 **Confining applications in containers**

1236 **LXC Containment**

1237 [LXC](https://wiki.ubuntu.com/GccSsp)⁴¹ is a solution that was developed to be a lightweight alternative to virtu-
1238 alization, built on top of cgroups and namespaces, mainly. Its main focus is on
1239 servers, though. The goal is to separate processes completely, including using
1240 a different file system and a different network. This means the applications
1241 running inside an LXC container are effectively running in a different system,
1242 for all practical purposes. While this does have the potential of helping protect
1243 the main system, it also brings with it huge problems with the integration of
1244 the application with the system.

1245 For graphical applications the X server will have to run with a TCP port open, so
1246 that applications running in a container are able to connect, 3D acceleration will
1247 be impossible or very difficult to achieve for applications running in a container.
1248 D-Bus setup will be significantly more complex.

1249 Besides increasing the complexity of the system, LXC essentially duplicates
1250 functionality offered by cgroups, AppArmor, and the Netfilter firewall. When
1251 LXC was originally suggested it was to be used only for system services. By
1252 using systemd the Apertis system will already have every service on the system

⁴⁰<https://wiki.ubuntu.com/GccSsp>

⁴¹<https://linuxcontainers.org/>

1253 running on their own cgroup, and properly locked down by AppArmor profiles.
1254 This means adding LXC would only add redundancy and no additional value.

1255 Protection for the driver assistance and limiting the damage root can do to the
1256 system can both be achieved by AppArmor policies, which can be applied to
1257 both system services and applications, as opposed to LXC, which would only
1258 be safely applicable to services. There are no advantages at all in using LXC
1259 for these cases. Limiting resources can also be easily done through cgroups,
1260 which will not be limited to system services, too. For these reasons Collabora
1261 recommends against using LXC.

1262 **Making X11, D-Bus and 3D work with LXC** For the sake of complete-
1263 ness, this section provides a description of possible solutions for LXC shortcom-
1264 ings.

1265 LXC creates what, for all practical purposes, is a separate system. X supports
1266 TCP socket connections, so it could be made to work, but that would require
1267 opening the TCP port and that would be another interface that needs protec-
1268 tion.

1269 D-Bus has the same pros and cons of X11 –it can be connected to over a [TCP](#)
1270 [port](#)⁴², but that again increases the surface area that needs to be protected, and
1271 adds complexity for managing the connection. It is also not a popular use case
1272 so it does not get a lot of testing.

1273 3D over network has not yet been made to work on networked X. All solutions
1274 available, such as [Virtual GL](#)⁴³ involve a lot of copying back and forth, which
1275 would make performance suffer substantially, which is something that needs to
1276 be avoided given the high importance of performance on Apertis requirements.

1277 Collabora’s perspective is that using LXC for applications running on the user
1278 session adds nothing that cannot be achieved with the means described in this
1279 document, while at the same time adding complexity and indirection.

1280 **The Flatpak framework**

1281 [Flatpak](#)⁴⁴ is a framework for “sandboxed” desktop applications, under develop-
1282 ment by several GNOME developers. Like LXC, it makes use of existing Linux
1283 infrastructure such as cgroups (see [Resource usage control](#)) and namespaces.

1284 Unlike LXC, Flatpak’s design goals are focused on confining individual applica-
1285 tions within a system, which makes it an interesting technology for Apertis. We
1286 recommend researching Flatpak further, and evaluating its adoption as a way
1287 to reduce the development effort for our sandboxed applications.

⁴²<https://www.freedesktop.org/wiki/Software/DBusRemote/>

⁴³<https://virtualgl.org/>

⁴⁴<https://flatpak.org/>

1288 One secondary benefit of Flatpak is that by altering the application bundle's
1289 view of the filesystem, it can provide a way to manage major-version upgrades
1290 without app-visible compatibility breaks, by continuing to run app bundles that
1291 were designed for the old “runtime” in an environment more closely resembling
1292 that old version, while using the new “runtime” for app bundles that have been
1293 tested in that environment.

1294 The IMA Linux Integrity Subsystem

1295 The goal of the Integrity Measurement Architecture (IMA⁴⁵) subsystem is to
1296 make sure that a given set of files have not been altered and are authentic –
1297 in other words, provided by a trusted source. The mechanism used to provide
1298 these two features are essentially keeping a database of file hashes and RSA
1299 signatures. IMA does not protect the system from changes, it is simply a way
1300 of knowing that changes have been made so that measures to fix the problem
1301 can be taken as quickly as possible. The authenticity module of IMA is still not
1302 available, so we won't be discussing it.

1303 In its simpler mode of operation, with the default policy IMA will intercept
1304 calls that cause memory mapping and execution of a file or any access done by
1305 root and perform a hash of the file before the access goes through. This means
1306 execution of all binaries and loading of all libraries are intercepted. To hash a
1307 file, IMA needs to read the whole file and calculate a cryptographic sum of its
1308 contents. That hash is then kept in kernel memory and extended attributes of
1309 the file system, for further verification after system reboots.

1310 This means that running any program will cause its file and any libraries it uses
1311 to be fully read and cryptographically processed before anything can be done
1312 with it, which causes a significant impact in the performance of the system. A
1313 10% impact has been reported⁴⁶ by the IMA authors in boot time on a default
1314 Fedora. There are no detailed information on how the test was performed, but
1315 the performance impact of IMA is mainly caused by increased I/O required to
1316 read the whole of all executable and library files used during the boot for hash
1317 verification. All executables will take longer to start up after a system boot
1318 up because they need to be fully read and hashed to verify they match what's
1319 recorded (if any recording exists).

1320 The fact that the hashes are maintained in the file system extended attributes,
1321 and are otherwise created from scratch when the file is first mapped or executed
1322 means that in this mode IMA does not protect the system from modification
1323 while offline: an attacker with physical access to the device can boot using a
1324 different operating system modify files and reset the extended attributes. Those
1325 changes will not be seen by IMA.

1326 To overcome this problem IMA is able to work with the hardware's trusted

⁴⁵<https://sourceforge.net/p/linux-ima/wiki/Home/>

⁴⁶https://blog.linuxplumbersconf.org/2009/slides/David-Stafford-IMA_LPC.pdf

platform module through the extended verification module (EVM⁴⁷), added⁴⁸ to Linux in version 3.2: hashes of the extended attributes are signed by the trusted platform module (TPM) hardware, and written to the file system as another extended attribute. For this to work, though, TPM hardware is required. The fact that TPM modules are currently only widely available and supported for Intel-based platforms is also a problem.

Conclusion regarding IMA and EVM

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

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

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

Seccomp

[Seccomp](#)⁴⁹ is a sandboxing mechanism in the Linux kernel. In essence, it is a way of specifying which system calls a process or thread should be able to make. As such, it is very useful to isolate processes that have strict responsibilities. For instance, a process that should not be able to write or read from the disk should not be able to make an *open* system call.

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

⁴⁷<https://sourceforge.net/p/linux-ima/wiki/Home/#linux-extended-verification-module-evm>

⁴⁸https://kernelnewbies.org/Linux_3.2#head-03576b924303bb0fad19cabb35efcbd33eed0

84

⁴⁹https://www.kernel.org/doc/Documentation/prctl/seccomp_filter.txt

1363 In other words, applications need to be written with an architecture which allows
1364 a separation of concerns, isolating the work that deals with untrusted processes
1365 or data to a separate process or thread that will then use seccomp filters to limit
1366 the amount of damage it is able to do through system calls.

1367 For use by applications, seccomp needs to be enabled in the kernel that is
1368 shipped with the middleware. There is a library called **libseccomp**⁵⁰, which
1369 provides a more convenient way of specifying filters. Should feature be used
1370 and made it available through the SDK, the seccomp support can be enabled
1371 in the kernel and libseccomp can be shipped in the middleware image provided
1372 by Collabora.

1373 The seccomp filter should be used on system services designed for Apertis whose
1374 architecture and intended functionality allow dropping privileges. Suppose, for
1375 instance, that Apertis has a health management daemon which needs to be able
1376 to kill applications that misbehave but has no need whatsoever of writing data
1377 to a file descriptor. It might be possible to design that daemon to use seccomp
1378 to filter out system calls such as **open** and **write**. The **open** system call might
1379 need to be allowed to go through for opening files for reading, depending on how
1380 the health daemon monitors processes –it might need to read information from
1381 files in the **/proc** file system, for instance. For that reason, filtering for **open**
1382 would need to be more granular, just disallowing it being called with certain
1383 arguments.

1384 Depending on how the health management daemon works it would also not
1385 need to fork new processes itself, so filtering out system calls such as **fork**,
1386 and **clone** is a possibility. As explained before, to take advantage of these
1387 opportunities, the architecture of such a daemon needs to be thought through
1388 from the onset with these limitations in mind. Opportunities, such as the ones
1389 discussed here, should be evaluated on a case-by-case basis, for each service
1390 intended for deployment on Apertis.

1391 AppArmor and seccomp are complementary technologies, and can be used to-
1392 gether. Some of their purposes overlap (for example, denying filesystem write
1393 access altogether could be achieved equally well with either technology), and
1394 they are both part of the kernel and hence in the TCB.

1395 The main advantage of seccomp over AppArmor is that it inhibits all system
1396 calls, however obscure: all system calls that were not considered when writ-
1397 ing a policy are normally denied. Its in-kernel implementation is also simpler,
1398 and hence potentially more robust, than AppArmor. This makes it suitable
1399 for containing a module whose functionality has been designed to be strongly
1400 focused on computation with minimal I/O requirements, for example the render-
1401 ing modules of browser engines such as WebKit2. However, its applicability to
1402 code that was not designed to be suitable for seccomp is limited. For example,
1403 if the confined module has a legitimate need to open files, then its seccomp filter
1404 will need to allow broad categories of file to be opened.

⁵⁰<https://lwn.net/Articles/494252/>

1405 The main advantage of AppArmor over seccomp is that it can perform finer-
1406 grained checking on the arguments and context of a system call, for example
1407 allowing filesystem reads from files owned by the process's uid, but denying
1408 reads from other uids'files. This makes it possible to confine existing general-
1409 purpose components using AppArmor, with little or no change to the confined
1410 component. Conversely, it groups together closely-related system calls with
1411 similar security implications into an abstract operation such as "read"or "write"
1412 , making it considerably easier to write correct profiles.

1413 The role of the app store process for security

1414 The model which is used for the application stores should precludes automated
1415 publishing of software to the store by developers. All software, including new
1416 versions of existing applications will have to go through an audit before publish-
1417 ing.

1418 The app store vetting process will generate the final package that will reach
1419 the store front. That means only signatures made by the app store curator's
1420 s cryptographic keys will be valid, for instance. Another consequence of this
1421 approach is that the curator will have not only the final say on what goes in,
1422 but will also be able to change pieces of the package to, say, disallow a given
1423 permission the application's author specified in the application's manifest.

1424 This also presents a good opportunity to convert high level descriptions such
1425 as the permissions in the manifest and an overall description of files used into
1426 concrete configuration files such as AppArmor profiles in a centralized fashion,
1427 and provides the curator with the ability to fine tune said configurations for
1428 specific devices or even to rework how a given resource is protected itself, with
1429 no need for intervention from third-parties.

1430 Most importantly, from the perspective of this document, is the fact that the app
1431 store vetting process provides an opportunity for final screening of submissions
1432 for security issues or bad practices both in terms of code and user interface, so
1433 that should be taken into consideration.

1434 How does security affect developer usage of a device?

1435 How security impacts a developer mode depends heavily on how that developer
1436 mode of work is specified. This chapter considers that the two main use cases
1437 for such a mode would be installing an application directly to the target through
1438 the Eclipse *install to target* plugin and running a remote debugging session for
1439 the application, both of which are topics discussed in the SDK design.

1440 The *install to target* functionality that was made available through an Eclipse
1441 plugin uses an **sftp** connection with an arbitrary user and password pair to
1442 connect to the device. This means that putting the device in developer mode
1443 should ensure the **ssh** server is running and add an exception to the firewall
1444 rules discussed in [Network filtering](#), to allow an inbound connection to port 22.

1445 Upon login, the SSH server will start user sessions that are not constrained by
1446 the AppArmor infrastructure. In particular the white-list policy discussed in
1447 section [Implementing a white list approach](#), will not apply to ssh user sessions.
1448 This means the user the IDE will connect with needs file system access to the
1449 directory where the application needs to be installed or be able to tell the
1450 application installer to install it.

1451 The procedure for installing an application using an **sftp** connection is not
1452 too different from the *install app from USB stick* use case described in the
1453 Applications document, that similarity could be exploited to share code for
1454 these features.

1455 The main difference is the developer mode would need to either ignore signature
1456 checking or accept a special “developer”signature for the packages. Decision on
1457 how to implement this piece of the feature needs a more complete assessment
1458 of proposed solutions on how the app store and system DRM could work, and
1459 how open (or openable) the end user devices will be.

1460 Running the application for remote debugging also requires that the **gdbserver**
1461 s default port, 2345, be open. Other than that, the main security constraint that
1462 will need to be tweaked when the system is put in developer mode is AppArmor.
1463 While under developer mode AppArmor should probably be put in complain
1464 mode, since the application’s own profile will not yet exist.

1465 Further discussion

1466 This chapter lists topics that require further thinking and/or discussion, or a
1467 more detailed design. These may be better written as Wiki pages rather than
1468 formal designs, given they require and benefit from iterating on an implementa-
1469 tion.

- 1470 • Define which cgroups ([Resource usage control](#)) to have, how they will be
1471 created and managed
- 1472 • Define exactly what Netfilter rules ([Network filtering](#)) should be installed
1473 and how they will be made effective at boot time
- 1474 • Evaluate Flatpak ([The Flatpak framework](#))