



Inter-domain communication

1	Contents	
2	Terminology and concepts	2
3	Automotive domain	2
4	Consumer-electronics domain	2
5	Connectivity domain	3
6	Trusted path	3
7	Control stream	3
8	Data stream	4
9	Traffic control	4
10	Use cases	4
11	Standalone setup	4
12	Basic virtualised setup	4
13	Linux container setup	5
14	Separate CPUs setup	5
15	Separate boards setup	5
16	Separate boards setup with other devices	6
17	Multiple CE domains setup	6
18	Touchscreen events	6
19	Wi-Fi access	7
20	Bluetooth access	7
21	Audio transfer	8
22	Video decoding	9
23	Streaming media	10
24	Downloads of firmware updates	10
25	Offline and online map data	10
26	Phonebook integration	10
27	Tinkering vehicle owner on the network	11
28	Tinkering vehicle owner on the boards	11
29	Support multiple AD operating systems	11
30	Before-market upgrades	11
31	After-market upgrades	12
32	Testability	12
33	Malicious CE	12
34	Malicious CD	12
35	After-market upgrade of a domain	12
36	Power cycle independence of domains (CE down)	13
37	Power cycle independence of domains (AD down, single screen)	13
38	Power cycle independence of domains (AD down, multiple screens)	13
39	Temporary communications problem	14
40	New version of AD software	14
41	New version of AD interfaces	14
42	Unsupported AD interfaces	15
43	Contacts sharing	15
44	Protocol compatibility	15
45	Navigation system	16

46	Marshalling resource usage	16
47	Feedback for malicious applications	16
48	Compromised CE with delayed fix	16
49	Denial of service through flooding	17
50	Malicious CE UI	17
51	Plug-and-play CE device	17
52	Connecting an SDK to a development vehicle	17
53	Security model	18
54	Attackers	18
55	Security domains	20
56	Security model	20
57	Non-use-cases	21
58	Production CE domain used in multiple configurations	21
59	Requirements	22
60	Separated transport layer	22
61	Message integrity and confidentiality in transport layer	23
62	Reliability and error checking in transport layer	23
63	Mutual authentication between domains	23
64	Separate authentication for developer and production mode devices	23
65	Individually addressed domains	23
66	Traffic control for latency	24
67	Traffic control for bandwidth	24
68	Traffic control for frequency	24
69	Separation of control and data streams	24
70	No untrusted access to AD hardware	24
71	Trusted path for users to update the CE operating system	25
72	Safety limits on AD APIs	25
73	Rate limiting on control messages	25
74	Ignore unrecognised messages	26
75	Portable transport layer	26
76	Support push mode and pull mode communications	26
77	OEM AD integration API	26
78	Flexibility in OEM AD integration API	26
79	Inflexibility in OEM AD integration API	27
80	Service discovery	27
81	Stability in inter-domain communications protocol	27
82	Testability of protocols	27
83	Testability of protocol parsers and writers	28
84	Testability of processes	28
85	CE system services separated from transport layer	28
86	No dependency on CE specific hardware	28
87	Immediate error response if service on peer is unavailable	29
88	Immediate error response if peer is unavailable	29
89	Timeout error response if peer does not respond	29
90	All inter-domain communications APIs are asynchronous	29
91	Reconnect to peer as soon as it is available	30

92	External domain watchdog	30
93	Reporting system for malicious applications	30
94	Ability to disable the consumer–electronics domain	30
95	Tamper evidence	31
96	No global keys in vehicles	31
97	Existing inter-domain communication systems	31
98	Approach	31
99	Overall architecture	32
100	Security domains	34
101	Protocol design	35
102	Traffic control	52
103	Protocol library and inter-domain services	53
104	Non Linux-based domains	54
105	Service discovery	54
106	Automotive domain export layer	56
107	Consumer-electronics domain adapter layer	57
108	Interaction of the export and adapter layers	58
109	Flow for a given SDK API call	59
110	Trusted path to the AD	60
111	Developer mode	60
112	Mock SDK implementation	60
113	Debuggability	61
114	External watchdog	62
115	Tamper evidence and hardware encryption	63
116	Disabling the CE domain	64
117	Reporting malicious applications	65
118	Suggested roadmap	66
119	Requirements	66
120	Open questions	66
121	Summary of recommendations	67
122	Appendix: D-Bus components and licensing	67
123	Licensing	68
124	Appendix: D-Bus performance	68
125	Appendix: Software versus hardware encryption	69
126	Software encryption (without encryption acceleration instructions)	70
127	Software encryption (with encryption acceleration instructions)	70
128	Secure cryptoprocessor	71
129	Hardware security module	71
130	Conclusion	72
131	Appendix: Audio and video streaming standards	72
132	Appendix: Multiplexing RTP and RTCP	73
133	Appendix: Audio and video decoding	74
134	Memory bandwidth usage on the i.MX6 Sabrelite	75
135	Security Vulnerabilities in GStreamer	75
136	This documents a suggested design for an inter-domain communication sys-	

137 tem, which exports services between different domains. Some domains can be
138 trusted such as the automotive domain. Some domains are untrusted such as
139 the consumer-electronics domain. Those domains can execute on a variety of
140 possible configurations.

141 The major considerations with an inter-domain communication system are:

- 142 • Security. The purpose of having separate domains is for security, so that
143 untrusted code (application bundles) can be run in one domain while min-
144 imizing the attack surface of the safety-critical systems which drive the
145 car.
- 146 • Flexibility for different hardware configurations. The domains may be
147 running in one of many configurations: virtualised under a hypervisor;
148 on separate CPUs on the same board; on separate boards connected by
149 a private in-vehicle network; as separate boards connected to a larger in-
150 vehicle network with unrelated peers on it; in separate containers.
- 151 • Flexibility for services exposed. The services exposed by the automo-
152 tive domain are dependent on the vendor which implemented the automo-
153 tive domain. The consumer-electronics domain depends on third-parties.
154 Their update and enhancement cycle and security rules may differ.
- 155 • Asynchronism and race conditions. This is a distributed system, and hence
156 is subject to all of the [challenges](#)¹ typical of distributed systems.

157 Terminology and concepts

158 Automotive domain

159 The *automotive domain* (AD) is a security domain which runs automotive pro-
160 cesses, with direct access to hardware such as audio output or the in-vehicle bus
161 (for example, a CAN bus or similar).

162 In some literature this domain is known as the ‘blue world’. This document will
163 consistently use the term *automotive domain* or *AD*.

164 Consumer-electronics domain

165 The *consumer-electronics domain* (CE domain; CE) is a security domain which
166 runs the user’s infotainment processes, including downloaded applications and
167 processing of untrusted content such as downloaded media. Apertis is one im-
168 plementation of the CE domain.

169 In some literature this domain is known as the ‘red world’, ‘infotainment do-
170 main’ or ‘IVI domain’. This document will consistently use the term *consumer-*
171 *electronics domain* or *CE domain* or *CE*.

¹<https://www.cl.cam.ac.uk/teaching/1516/ConcDisSys/materials.html>

172 **Connectivity domain**

173 In some setups the *AD* and *CE* are not directly exposed to external networks and
174 hardware. In those cases a *connectivity domain* hosts agents which can directly
175 access the Internet or plug-and-play hardware devices such as USB keys, SD
176 cards or Bluetooth devices and provide their services to applications running in
177 the more isolated domains. This domain can be referred to as *CD*.

178 **Trusted path**

179 A [trusted path](#)² is an end-to-end communications channel from the user to a
180 specific software component, which the user can be confident has integrity, and
181 is addressing the component they expect. This encompasses technical security
182 measures, plus unforgeable UI indications of the trusted path.

183 An example of a trusted path is the old Windows login screen, which required
184 the user to press Ctrl+Alt+Delete to open the login dialogue. If a malicious ap-
185 plication was impersonating the login dialogue, pressing Ctrl+Alt+Delete would
186 open the task manager instead of the login dialogue, exposing the subversion.

187 In the context of Apertis, an example situation calling for a trusted path is
188 when the user needs to interact with a UI provided by the AD. They must be
189 sure that this UI is not being forged by a malicious application running in the
190 CE.

191 **Control stream**

192 A *control stream* is a network connection which transmits low bandwidth, la-
193 tency insensitive messages which typically contain metadata about data being
194 transferred in a data stream. In networking, it is sometimes known as the *control*
195 *plane*.

196 A control stream for one protocol may be treated as a data stream if it is being
197 carried by a higher layer (or wrapper) protocol, as the control data in the stream
198 is meaningless to the higher layer protocol.

199 If a designer is concerned about whether a particular stream's performance
200 requirements make it suitable for running as a control stream, it almost certainly
201 is not a control stream, and should be treated as a data stream. A new control
202 protocol should be built to carry more limited metadata about it.

203 A control stream can operate without a data stream (for example, if there is no
204 performance-sensitive data to transmit).

205 **Data stream**

206 A *data stream* is a network connection which transmits the data referred to by
207 a control stream. This data may be high bandwidth or latency sensitive, or it

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

208 may be neither. In networking, it is sometimes known as the *data plane*.

209 A data stream cannot operate without an associated control stream (which
210 carries its metadata).

211 **Traffic control**

212 Traffic control (or [bandwidth management](#)³) is the term for a variety of tech-
213 niques for measuring and controlling the connections on a network link, to try
214 and meet the quality of service requirements for each connection, in terms of
215 bandwidth and latency.

216 **Use cases**

217 A variety of use cases which must be satisfied by an inter-domain communication
218 system are given below. Particularly important discussion points are highlighted
219 at the bottom of each use case.

220 All of these use cases are relevant to an inter-domain communication system,
221 but some of them (for example, [Video or audio decoder bugs](#)) may equally well
222 be solved by other components in the system.

223 **Standalone setup**

224 An app-centric consumer electronics domain (CE) is running in a virtual ma-
225 chine on a developer's laptop, and they are using it to develop an application for
226 Apertis. There is no automotive domain (AD) for this CE to run against, but it
227 must provide all the same services via its SDK APIs as the CE running in a ve-
228 hicle which has an Apertis device. The CE must run without an accompanying
229 AD in this configuration.

230 **Basic virtualised setup**

231 An embedded automotive domain (AD) and an app-centric consumer electronics
232 domain (CE) are running as separate virtualised operating systems under a
233 hypervisor, in order to save costs on the bill of materials by only having one
234 board and CPU. The AD has access to the underlying physical hardware; the
235 CE does not. The two domains have a high bandwidth connection to each other
236 (for example, Ethernet, USB, PCI Express or virtio). The two domains need to
237 communicate so that the CE can access the hardware controlled by the AD.

238 **Linux container setup**

239 Containers are based on Linux kernel containment features, including, but not
240 limited to, Linux kernel namespaces, control groups, chroots (`pivot_root`), ca-
241 pabilities.

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

242 Both AD and CE are dedicated Linux containers on a host directly running on
243 the hardware or in a virtual machine. AD is allowed to access safety-sensitive
244 devices. CE is not allowed any access to safety-sensitive devices but may be able
245 to access external devices like smartphones over Bluetooth, USB mass storage
246 or security keys.

247 Communication is based on the Unix Domain Sockets (UDS) mechanism pro-
248 vided by the Linux kernel.

249 This setup can be used both for production setups on hardware board and on
250 a developer's system for Apertis application development. It can be possible to
251 provide a fake AD container for emulation and testing purposes.

252 Isolation between containers is unavoidably limited when compared to the isola-
253 tion between virtual machines, just like separate boards provide more isolation
254 than VMs. This is due to the fact that a single kernel is shared by all contain-
255 ers. However in this document we assume processes are not able to escape from
256 the isolated environment or get access to resources on the host system or other
257 containers for which they haven't been explicitly granted access.

258 **Multiple CE domains** are allowed with the above setup. In this setup, a **Con-**
259 **nectivity Domain** can also coexist with AD and CE. It is responsible for any
260 interaction with external networks and provides isolation in the case a network
261 stack is compromised when that stack is not implemented in the shared kernel.

262 **Separate CPUs setup**

263 The AD is running on one CPU, and the CE is running on another CPU on the
264 same board. The two CPUs have separate memory hierarchies. They maybe
265 using separate architectures or endianness. The AD has access to all of the
266 underlying physical hardware; the CE only has access to a limited number of
267 devices, such as its own memory and some kind of high bandwidth connection
268 to the AD (for example, Ethernet, USB, or PCI Express). The two domains
269 need to communicate so that the CE can access the hardware controlled by the
270 AD.

271 **Separate boards setup**

272 The AD is running on one mainboard, and the CE is running on another main-
273 board, which is physically separate from the first. They may be using separate
274 architectures or endianness. The two boards are connected by some kind of
275 vehicle network (for example, Ethernet; but other technologies could be used).
276 There are no other devices on this network. The vehicle owner (and any other
277 attacker) might have physical access to this network. The AD has access to
278 various devices which are connected to its board and not to the CE's board.
279 The two domains need to communicate so that the CE can access the hardware
280 controlled by the AD.

281 **Separate boards setup with other devices**

282 The AD is running on one mainboard, and the CE is running on another main-
283 board, which is physically separate from the first. They may be using separate
284 architectures or endianness. The two boards are connected by some kind of
285 vehicle network (for example, Ethernet; but other technologies could be used).
286 There are many other devices on this network, which are addressable but whose
287 traffic is irrelevant to the CE-AD connection (for example, a telematics modem,
288 or a high-end amplifier). The vehicle owner (and any other attacker) might have
289 physical access to this network. The AD has access to various devices which are
290 connected to its board and not to the CE's board. The two domains need to
291 communicate so that the CE can access the hardware controlled by the AD.

292 *(Note: This is a much lower priority than other setups, but should still be*
293 *considered as part of the overall design, even if the code for it will be implemented*
294 *as a later phase.)*

295 **Multiple CE domains setup**

296 The AD is running on one mainboard. Multiple CE domains are running, each
297 on a separate mainboard, each physically separate from each other and from the
298 AD. The boards are connected by some kind of vehicle network (for example,
299 Ethernet; but other technologies could be used). There are many other devices
300 on this network, which are addressable but whose traffic is irrelevant to the CE-
301 AD connections (for example, a telematics modem, or a high-end amplifier).
302 The vehicle owner (and any other attacker) might have physical access to this
303 network. The AD has access to various devices which are connected to its board
304 and not to the CEs' boards. Each CE domain needs to communicate with the
305 AD so that it can access the hardware controlled by the AD.

306 *(Note: This is a much lower priority than other setups, but should still be*
307 *considered as part of the overall design, even if the code for it will be implemented*
308 *as a later phase.)*

309 **Touchscreen events**

310 The touchscreen hardware is controlled by the AD, but content from the CE is
311 displayed on it. In order to interact with this, touch events which are relevant to
312 content from the CE must be forwarded from the AD to the CE. Users expect
313 a minimal latency for touch screen event handling. Touchscreen events must
314 continue to be delivered reliably and on time even if there is a large amount
315 of bandwidth being consumed by other inter-domain communications between
316 AD and CE.

317 **Wi-Fi access**

318 The Wi-Fi hardware is controlled by the AD or CD. The CE needs to use it
319 for internet access, including connecting to a network. The Wi-Fi device can

320 return data at high bandwidth, but also has a separate control channel. The
 321 control channel always needs to be available, even if traffic is being dropped due
 322 to bandwidth limitations in the inter-domain communication channel.

323 As the Wi-Fi is used for general internet access, sensitive information might
 324 be transferred between domains (for example, authentication credentials for a
 325 website the user is logging in to). Attackers who are snooping the inter-domain
 326 connection must not be able to extract such sensitive data from the inter-domain
 327 communications link.

328 *(Note that they may still be able to extract sensitive data from insecure con-*
 329 *nections over the wireless connection itself, or elsewhere in transit outside the*
 330 *vehicle; so any solution here is the best mitigation we can manage for the problem*
 331 *of a website being insecure.)*

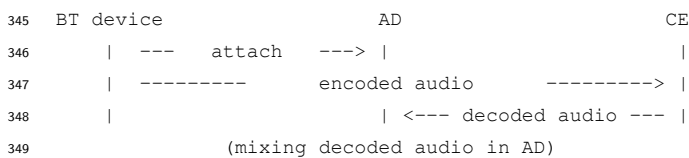
332 Bluetooth access

333 The Bluetooth hardware might be attached to the AD or CD. The CE needs
 334 to be able to send data bi-directionally to other Bluetooth devices and also
 335 needs to be able to control the Bluetooth device, controlling pairing and other
 336 functions of the Bluetooth hardware.

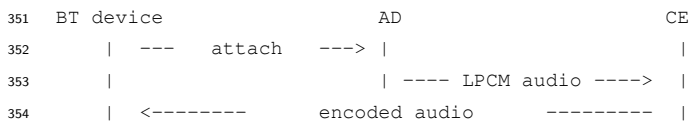
337 To support the A2DP and HSP/HFP audio profiles it may be desirable to keep
 338 the CE in charge of decoding and encoding the audio streams coming from
 339 and directed to the Bluetooth devices. The AD will be responsible for mixing
 340 the output streams directed to the car speakers and capturing input streams
 341 (possibly with noise cancellation) from the car microphones.

342 The following diagrams depict the data and control flow when the Bluetooth
 343 device is attached to the AD.

344 Sending audio stream from BT to AD

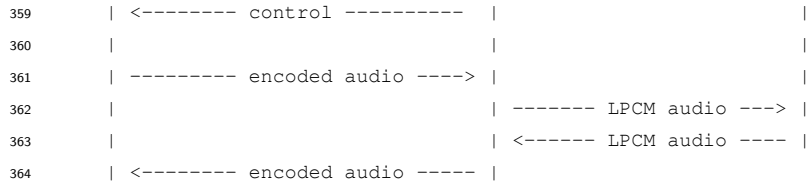


350 Sending audio stream from AD to BT

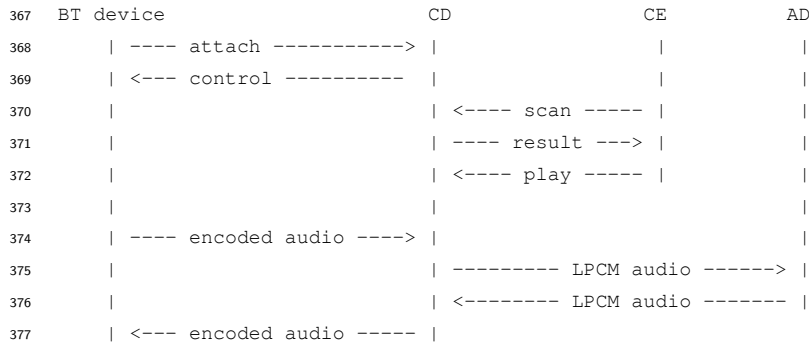


355 The following diagram depicts the data and control flow when the Bluetooth
 356 device is directly attached to the CE instead.





365 The following diagram depicts the data and control flow when the Bluetooth
366 device is directly attached to the CD.



378 Multiple variations are possible on this model.

379 Audio transfer

380 The audio amplifier hardware might be attached to the AD hardware, or might
381 be set up as a separate hardware amplifier attached to the in-vehicle network.
382 The CE needs to be able to send multiple streams of decoded audio output
383 to the AD, to be mixed with audio output from the AD according to some
384 prioritisation logic.

385 The decoded audio streams should be in LPCM format, but other formats may
386 be negotiated by the domains using application specific APIs.

387 Metadata can be sent alongside the audio, such as track names or timing infor-
388 mation.

389 Audio output needs predictable latency output, and for video conferencing it
390 needs low latency as well; conversely, some level of packet loss is acceptable for
391 audio traffic. However, the latency should not exceed a certain amount of time
392 in some specific cases:

- 393 • Voice recognition systems provided through phone integration require that
394 the maximum latency of the audio buffer from the time it gets captured
395 by the microphone controlled by the AD to the time it gets delivered to
396 the phone attached to the CE domain must not exceed 35ms.
- 397 • Text-to-speech systems provided through phone integration require that
398 the maximum latency of the audio buffer from the time it is received by

399 the CE domain from the attached phone to the time it gets played back
400 on the speakers attached to the AD must not exceed 35ms.

- 401 • The total round-trip time must not exceed 275ms when the phone is at-
402 tached to the CE domain through a wired transports (for instance, USB
403 CDC-NCM as used by CarPlay or the Android Open Accessory Protocol)
404 and 415ms on wireless transports (WiFi in particular, Bluetooth A2DP is
405 not recommended in this case).
- 406 • Bluetooth SCO can be used when there is a latency constraint. It will
407 be lower quality, but the transfer time over the air is guaranteed. The
408 whole audio chain needs to satisfy the latency condition though. This
409 is why in some setup, the Bluetooth audio is routed directly to the AD
410 amplifier. When this is the case, an API to enable this link is provided by
411 the domain that owns the Bluetooth hardware. It can be the AD, or the
412 CD embedding a Bluetooth stack. The API calls would be issued by the
413 CE domain.

414 **Video decoding**

415 There might be a specific hardware video decoder attached to the AD hardware,
416 which the CE operating system wishes to use for offloading decoding of trusted
417 or untrusted video content. This is high bandwidth, but means that the output
418 from the video decoder could potentially be directed straight onto a surface on
419 the screen.

420 (See the appendix on [Audio and video decoding](#) for a discussion of options for
421 video and audio decoding.)

422 **Video or audio decoder bugs**

423 The CE has a software video or audio decoder for a particular video or audio
424 codec, and a security critical bug is found in this decoder, which could allow
425 malicious video or audio content to gain arbitrary code execution privileges when
426 it's decoded. An update for the Apertis operating system is released which fixes
427 this bug, and users need to apply it to their vehicles. To reduce the window of
428 opportunity for exploitation, this update has to be applied by the vehicle owner,
429 rather than taking the vehicle into a garage (which could take weeks).

430 For example, like the series of exploitable bugs which [affected the 'secure' media
431 decoding library on Android⁴](#) in 2015.

432 This means we cannot securely support decoding untrusted video or audio con-
433 tent in the AD, due to its slow software update cycle, unless we use a *hardware*
434 video decoder which is specifically designed to cope with malicious inputs.

⁴[https://en.wikipedia.org/wiki/Stagefright_\(bug\)](https://en.wikipedia.org/wiki/Stagefright_(bug))

435 **Streaming media**

436 The media player backend on the CE accesses local files or internet streams and
437 sends the streams to the Media Player HMI running in the AD. The CE might
438 be able to perform demuxing, decoding or at least partly verifying the streams.

439 The AD might accept fully decoded streams, but the media file or stream is usu-
440 ally encoded and multiplexed. In some cases, the multiplexed stream can have
441 synchronization sensitive metadata like subtitles. Therefore, if demuxing and
442 decoding are performed in different domains, the AD should support multiple
443 channels and mix the streams with time synchronization information.

444 It is also possible that the AD sends the stream to the CE. For example, in
445 the case of Internet phone applications, the CE provides the HMI and needs to
446 be able to capture video and audio streams from the AD, before encoding and
447 multiplexing them on the CE.

448 When handling data streams that don't need strict synchronization, the bulk
449 data transfer mechanism is recommended. For example, sharing still pictures
450 does not require real time processing so it is not suited for the streaming media
451 mechanism.

452 **Downloads of firmware updates**

453 An OTA update agent in the Connectivity domain downloads or retrieves from
454 an attached USB stick firmware images as large as 20GB each and needs to
455 share them with the Automotive domain where the FOTA backend can flash
456 the attached devices.

457 Since firmware are very large, storing them twice should be avoided as the
458 available space may not be sufficient to do so.

459 **Offline and online map data**

460 An offline map agent in the Connectivity domain downloads map data for offline
461 usage by the navigation system running in the Automotive domain.

462 Conversely, an online map agent in the Connectivity domain handles requests
463 from the Automotive domain for map tiles to download.

464 **Phonebook integration**

465 A phonebook agent in the Connectivity domain retrieves approximately 500
466 256×256px profile pictures, validates and re-encodes them to PNG and makes
467 them available to the Automotive domain, possibly using an uncompressed zip
468 file instead of sharing 500 files.

469 **Tinkering vehicle owner on the network**

470 The owner of a vehicle containing an Apertis device likes to tinker with it,
471 and is probing and injecting signals on the connection between the AD and
472 CE, or even replacing the CE completely with a device under their control.
473 They should not be able to make the automotive domain do anything outside
474 its normal operating range; for example, uncontrolled acceleration, or causing
475 services in the domain to crash or shut down.

476 The tampering must be detectable by the vendor when the vehicle is serviced
477 or investigated after an accident.

478 **Tinkering vehicle owner on the boards**

479 The owner of a vehicle containing an Apertis device likes to tinker with it,
480 and has gained access to the bootloaders and storage for both the AD and CE
481 boards. They have managed to add some custom software to the CE image,
482 which is now sending messages to the AD which it does not expect. Or vice-
483 versa. The domain receiving the messages must not crash, must ignore invalid
484 messages, and must not cause unsafe vehicle behaviour.

485 The tampering must be detectable by the vendor when the vehicle is serviced
486 or investigated after an accident.

487 [Secure bootloading](#)⁵ itself is a separate topic.

488 **Support multiple AD operating systems**

489 The OEM for a vehicle wants to choose the operating system used in the AD
490 — for example, it might be GENIVI Linux, or QNX, or something else. There
491 is limited opportunity to modify this operating system to implement Apertis-
492 specific features. Whichever CE or CD system is installed needs to interface to
493 it. Each AD operating system may expose its underlying hardware and services
494 with a variety of different non-standardised APIs which use push- and pull-style
495 APIs for transferring data. The OEM wishes to be provided with an inter-
496 domain communication library to integrate into their choice of AD operating
497 system, which will provide all the functionality necessary to communicate with
498 Apertis as the CE or CD operating system.

499 **Before-market upgrades**

500 The OEM for a vehicle has chosen a specific version of an operating system for
501 their AD, and has initially released their vehicle with Apertis 17.09 on another
502 domain, such as CE and/or CD. For the latest incremental version of this vehicle,
503 they want to upgrade the other domain to use Apertis 18.06. The OS in the
504 AD cannot be changed, due to having stricter stability and testing requirements
505 than the other domains.

⁵<https://sjoerd.pages.apertis.org/apertis-website/architecture/secure-boot/>

506 **After-market upgrades**

507 A user has bought a vehicle which runs Apertis 17.09 in its CE. Apertis 18.06
508 is released by their car vendor, and their garage offers it as an upgrade to
509 the user as part of their next car service. The garage performs this software
510 upgrade to the CE, without having to touch the AD. It verifies that the system
511 is operational, and returns the car to the user, who now has access to all the
512 new features in Apertis 18.06 which are supported by their vehicle's hardware.

513 **Testability**

514 When developing a new vehicle, an OEM wants to iterate quickly on changes
515 to the CE, but also wants to test them thoroughly for compatibility against a
516 specific AD version, to ensure that the two domains will work together. They
517 want this testing to include a number of valid and invalid conversations between
518 the CE and AD, to ensure that the two domains implement error handling (and
519 hence a large part of their security) correctly.

520 **Malicious CE**

521 Somehow, a third party application installed onto the CE manages to compro-
522 mise a system service and gain arbitrary code execution privileges in the CE.
523 It uses these privileges to send malicious messages to the AD. From the user's
524 point of view, this could result in a loss of IVI functionality, and unexpected
525 behaviour from vehicle actuators, but must not result in loss of control of the
526 vehicle.

527 **Malicious CD**

528 Recent protocol failures have been discovered that allowed an attacker to take
529 control of a device remotely. To mitigate this, the network management stack
530 has been moved to a Connectivity Domain. The impact of those attacks must
531 be minimised. While the CD functionality can be degraded, it must not result
532 in loss of control of the vehicle.

533 **After-market upgrade of a domain**

534 A user has bought a vehicle containing a low-end Apertis device. They wish to
535 upgrade to a more fully-featured Apertis device, and this hardware upgrade is
536 offered by their garage. The garage performs the upgrade, which replaces the
537 existing CE hardware with a new separate CE board. If the existing hardware
538 combined the AD and CE on a single board or virtualised processor, the entire
539 board is replaced with two new, separate boards, one for each domain (though
540 as this is a complex operation, some garages or vendors might not offer it). If
541 the existing hardware already had separate boards for the two domains, only
542 the CE board is upgraded — this may be a service offered by all garages.

543 **Power cycle independence of domains (CE down)**

544 Due to a bug, the CE crashes. The AD must not crash, and must continue
545 to function safely. It may display an error message to the user, and the user
546 may lose unsaved data. Once the CE restarts, the AD should reconnect to it
547 and reestablish a normal user interface. The CE should reboot quickly and the
548 cross-domain state be restored as much as reasonable once restarted.

549 Any partially-complete inter-domain communications must error out rather than
550 remaining unanswered indefinitely.

551 The same situation applies if both domains are booting simultaneously, but the
552 CE is slower to boot than the AD, for example — the AD will be up before the
553 CE, and hence must deal with not being able to communicate with it. See also
554 [Plug-and-play CE device](#).

555 **Power cycle independence of domains (AD down, single screen)**

556 On a system where the AD and CE are sharing a single screen, if the AD crashes,
557 the CE must not crash, and may gracefully shut down, and only restart once the
558 AD has finished rebooting. The AD should reboot quickly and the cross-domain
559 state be restored as much as reasonable once restarted

560 Any partially-complete inter-domain communications must error out rather than
561 remaining unanswered indefinitely.

562 The same situation applies if both domains are booting simultaneously, but the
563 AD is slower to boot than the CE, for example — the CE will be up before the
564 AD, and hence must deal with not being able to communicate with it. See also
565 [Plug-and-play CE device](#).

566 **Power cycle independence of domains (AD down, multiple screens)**

567 On a system with multiple output screens, if the AD crashes, the CE must not
568 crash, and should continue to run on all its screens, as another user may be
569 using the CE (without requiring any functionality from the AD) on one of the
570 screens. Once the AD restarts, the CE should reconnect to it and reestablish
571 a normal user interface on all screens. The AD should reboot quickly and the
572 cross-domain state be restored as much as reasonable once restarted.

573 Any partially-complete inter-domain communications must error out rather than
574 remaining unanswered indefinitely.

575 The same situation applies if both domains are booting simultaneously, but the
576 AD is slower to boot than the CE, for example — the CE will be up before the
577 AD, and hence must deal with not being able to communicate with it. See also
578 [Plug-and-play CE device](#).

579 **Temporary communications problem**

580 There is a temporary communications problem between a service on the AD
581 and its counterpart on the CE. Either:

- 582 • The service (on the AD or CE) has crashed.
- 583 • There is a problem with the physical connection between the domains,
584 such as dropped packets due to congestion; but both domains are still
585 running fine.
- 586 • The entire domain or its inter-domain communications service has crashed.

587 The different situations can be detected by the parts of the stack which are still
588 working

589 If a service has crashed, the inter-domain communication service should return
590 an appropriate error code to the other domain, which could propagate the error
591 to a calling application, or wait for the other domain to restart that service and
592 try again.

593 If there is packet loss, the reliability in the inter-domain communication protocol
594 should cause the lost packets to be re-sent. Services should wait for that to
595 happen. If the communications problem continues longer than a timeout, the
596 domains must assume that each other have crashed and behave accordingly.

597 If a domain has crashed, the other domain must wait for it to be restarted via
598 its watchdog, as in [Power cycle independence of domains \(CE down\)](#).

599 In all cases, the domain which is still running must not shut down or enter a
600 'paused' state, as that would allow denial of service attacks.

601 **New version of AD software**

602 An OEM has released a vehicle with version A of their AD operating system,
603 and version 15.06 of Apertis running in the CE. For the next minor update to
604 their vehicle, the OEM has made a number of changes to the underlying AD
605 software, but not to its external interfaces. They wish to keep the same version
606 of Apertis running in the CE and release the vehicle using this version B of their
607 AD operating system, and version 15.06 of Apertis.

608 **New version of AD interfaces**

609 An OEM has released a vehicle with version A of their AD operating system,
610 and version 15.06 of Apertis running in the CE. For the next minor update to
611 their vehicle, the OEM has made a number of changes to the underlying AD
612 software, and has changed a few of its external interfaces and exposed a few
613 more vehicle-specific features in new interfaces. They want to make appropriate
614 modifications to Apertis to align it with these changed interfaces, but do not
615 wish to make major modifications to Apertis, and wish to (broadly) stick with

616 version 15.06. They will release the vehicle using this version B of their AD
617 operating system, and a tweaked version 15.06 of Apertis.

618 In other words, this scenario applies only when the OEM has updated the AD,
619 and wants to make a corresponding update to the CE. For the reverse scenario
620 where the CE has been upgraded, it is required that the AD does not need to
621 be updated: see [Plug-and-play CE device](#) and [After market CE upgrades](#).

622 **Unsupported AD interfaces**

623 An OEM uses an AD operating system which exposes a large number of inter-
624 faces to various esoteric automotive components. Only a few of these com-
625 ponents are currently supported by Apertis version A, which they are running
626 in their CE. Apertis version B supports some more of these components, and
627 exposes them in its SDK APIs. The OEM wishes to release a new version of the
628 same vehicle, keeping the same version of the AD operating system, but using
629 version B of Apertis and exposing the now-supported components in the SDK
630 APIs.

631 However, some of the other components which are exposed by the AD operating
632 system in its inter-domain interface cannot be securely supported by Apertis (for
633 example, they may allow unrestricted write access to the in-vehicle network).
634 These should not be accessible by the SDK APIs at any time.

635 **Contacts sharing**

636 A vehicle maintains an address book in its AD operating system, which stores
637 some of the user's contacts on a removable SD card. The user interface, run by
638 the CE, needs to be able to display and modify these contacts in the Apertis
639 address book application.

640 **Protocol compatibility**

641 An older vehicle, using an old version A of some AD operating system was
642 using a corresponding version A of Apertis in its CE. The CE operating system
643 is upgraded to a recent version of Apertis, version B, by the garage when the
644 vehicle is taken in for a service. This version of Apertis uses a much more recent
645 version of the underlying software for the inter-domain communication protocol.
646 It needs to continue to work with the old version A of the AD operating system,
647 which is running a much older version of the protocol software.

648 **kdbus protocol compatibility**

649 If, for example, the inter-domain communication protocol is implemented using
650 dbus-daemon in version A of the AD operating system, and in the corresponding
651 version A of Apertis; and version B of Apertis uses kdbus instead of dbus-
652 daemon, the two OSs must still communicate successfully.

653 **Navigation system**

654 A proprietary navigation system is running on the AD, with full access to the
655 vehicle's navigation hardware, including inertial sensors and a GPS receiver. A
656 tour application on the CE wishes to use location-based services, reading the
657 vehicle's location from the navigation system on the AD, then requesting to the
658 navigation service to set its destination to a new location for the next place
659 in the tour. It sends a stream of points of interest to the navigation system
660 to display on the map while the driver is navigating. This stream is not high
661 bandwidth; neither are the location updates from the GPS.

662 **Marshalling resource usage**

663 The 'proxy' software on either side of the inter-domain connection which handles
664 the low-level communication link is the first software in a domain to handle
665 malicious input. If malicious input is sent to a domain with the intent of causing
666 a denial of service in that software, the rest of the software in the domain should
667 be unaffected, and should treat the connection as timing out or compromised.
668 The behaviour of the proxy software should be confined so that it cannot use
669 excess resources in the domain and hence extend the denial of service attack to
670 the whole domain.

671 **Feedback for malicious applications**

672 If an application uses SDK APIs incorrectly (for example, by providing param-
673 eters which are outside valid ranges), it may be reported to the Apertis store as
674 a 'misbehaving application' and scheduled for further investigation and possible
675 removal from the Apertis store. Similarly if the inter-domain communication
676 APIs are used incorrectly (for example, if the AD returns an error stating that
677 input validation checks have failed for an API call).

678 This could also result in an application being blacklisted by the CE's application
679 manager, disallowing it from running in future until it is updated from the
680 Apertis store.

681 **Compromised CE with delayed fix**

682 An attacker has somehow completely compromised the CE operating system,
683 and has root access to it. It will take the OEM a few weeks to produce, test
684 and distribute a fix for the exploit used by the attacker, but vehicle owners
685 would like to continue to use their vehicles, with reduced functionality (no CE
686 domain) in the meantime, because the attack has not compromised the AD.
687 The OEM has provided them with an authenticated method of informing the
688 AD to shut down the CE and keep it shut down until an authenticated update
689 has been applied and has fixed the exploit and removed the attacker from the
690 CE (probably by overwriting the entire OS with a fresh copy). This update can
691 only be applied at a garage, but in order to allow speedy deployment, the user

692 can switch the AD to this stand-alone mode themselves, using a trusted input
693 path to the AD.

694 **Denial of service through flooding**

695 A speedometer application bundle constantly requests vehicle speed information
696 from the AD. Hundreds of requests are made per second. The AD ensures
697 this does not affect overall system performance, potentially at the cost of its
698 responsiveness to the speedometer application's requests.

699 *(Note: This assumes that the corresponding denial of service rate limiting which
700 is implemented in the SDK API used by the speedometer application has some-
701 how failed or been bypassed. In reality, all SDK APIs are also responsible for
702 implementing their own rate limiting as a first level of protection against denial
703 of service attacks.)*

704 **Malicious CE UI**

705 An attacker has somehow completely compromised the CE operating system,
706 and has root access to it. They can display whatever they like on the graphics
707 output from the CE, which is shared with that from the AD on a single screen.
708 The attacker tries to replicate the AD UI on the CE's output and trick the user
709 into entering personal data or security credentials in this faked UI, believing
710 it to be the actual AD UI. There should be a way for the user to determine
711 whether they are inputting details via a trusted path to the AD.

712 **Plug-and-play CE device**

713 In a particular vehicle, the CE device can be unplugged from the dashboard by
714 the user, and passed around the car so that, for example, a rear seat passenger
715 could play a game. This disconnects it from the AD, but it should continue
716 to function with some features (such as Wi-Fi or Bluetooth) disabled until
717 it is reconnected. Once reconnected to the dashboard it should reestablish
718 its connections. See also, [Power cycle independence of domains \(CE down\)](#),
719 [Power cycle independence of domains \(AD down, single screen\)](#), [Power cycle
720 independence of domains \(AD down, multiple screens\)](#)

721 *(Note: This is a much lower priority than other setups, but should still be
722 considered as part of the overall design, even if the code for it will be implemented
723 as a later phase.)*

724 **Connecting an SDK to a development vehicle**

725 A developer is running the SDK as a standalone CE system in a virtual envi-
726 ronment on a laptop. They connect the laptop to the AD physically installed
727 in a development car using an Ethernet cable, and expect to receive sensor data
728 from the car, using the sensors and actuators SDK API, which was previously
729 returning mock results from the standalone system.

730 **Connecting an SDK to a production vehicle**

731 The developer wonders what would happen if they tried connecting their SDK
732 laptop to the AD in a production vehicle. They try this, and nothing happens
733 — they cannot get sensor data out of the vehicle, nor use any of its other APIs.

734 **Security model**

735 See the [Security concept design](#)⁶ for general terminology including the defini-
736 tions used for *integrity*, *availability*, *confidentiality* and *trust*.

737 **Attackers**

738 **Vehicle’s owner**

739 The vehicle’s owner may be an attacker. They have physical access to the vehi-
740 cle, including its in-vehicle network, the physical inter-domain communications
741 link, and the board or boards which the automotive domain (AD) and consumer-
742 electronics domain (CE) are on. We assume they do not have the capabilities
743 to perform invasive attacks on silicon on the boards. Specifically, this means
744 that in a virtualised setup where the AD and CE are run as separate virtual
745 machines on the same CPU, we assume the attacker cannot read or modify the
746 inter-domain communications link between them.

747 However, we do assume that they can perform semi-invasive or non-invasive
748 [attacks](#)⁷ on silicon on the boards. This means that they could (with difficulty)
749 extract encryption keys from a secure key store on the board. A secure key
750 store may be provided by the Secure Boot design, but may not be present due
751 to hardware limitations — if so, the vehicle’s owner will be able to extract
752 encryption keys from the device more easily.

753 As of February 2016, the Secure Boot design is still forthcoming

754 The vehicle’s owner may wish to attack their vehicle in order to get access to
755 licenced content which they would otherwise have to pay for.

756 See the [Conditional Access design](#)⁸

757 We assume they do not want to take control of the vehicle, or to gain arbitrary
758 code execution privileges — they can drive the vehicle normally, or develop and
759 choose to install their own application bundle for this.

760 **Passenger**

761 The passenger is a special kind of third party attacker (**Third parties**), who
762 additionally has access to the in-vehicle network. This may be possible if, for

⁶<https://sjoerd.pages.apertis.org/apertis-website/designs/security/>

⁷<http://www.cl.cam.ac.uk/techreports/UCAM-CL-TR-630.html>

⁸https://sjoerd.pages.apertis.org/apertis-website/designs/conditional_access/

763 example, the Apertis device in the vehicle is removable so it can be passed to a
764 passenger, exposing a connector behind it.

765 The passenger may be trying to access confidential information belonging to the
766 vehicle owner (if a multi-user system is in use).

767 **Third parties**

768 Any third party may be an attacker. We assume they have physical access to the
769 exterior of the vehicle, but not to anything under the bonnet, including the in-
770 vehicle network, the physical inter-domain communications link, and the board
771 or boards which the domains are on. This means that all garage mechanics
772 must be trusted. They do, however, have access to all communications into and
773 out of the vehicle, including Bluetooth, 4G, GPS and Wi-Fi.

774 We assume any third party attacker can develop and deploy applications, and
775 convince the owner of a vehicle to install them. These applications are subject
776 to the normal sandboxing applied to any application installed on an Apertis sys-
777 tem. These applications are also subject to the normal Apertis store validation
778 procedures, but we assume that a certain proportion of malicious applications
779 may get past these procedures temporarily, before being discovered and removed
780 from the store.

781 We assume that a third party attacker does not have access to the Apertis store
782 servers. This means that all staff who have access to them must be trusted.

783 A third party attacker may be trying to:

- 784 • Access confidential information belonging to the vehicle owner.
- 785 • Compromise the integrity of the vehicle's control system (the automotive
786 domain). For example, to trigger unintended acceleration or to change
787 the radio channel to spook the driver.
- 788 • Compromise the integrity of the CE domain to, for example, make it part
789 of a botnet, or cause it to call premium rate numbers owned by the attacker
790 to generate money.
- 791 • Compromise the availability of the vehicle's control system (the automo-
792 tive domain) to bring the vehicle to a halt.
- 793 • Compromise the availability of the vehicle's infotainment system (the CE
794 domain) to cause a nuisance to the driver or passengers.
- 795 • Compromise the confidentiality of the device key (see the [Conditional
796 Access design](#)⁹) in order to extract licenced content (for example, music)
797 from application bundles.

⁹https://sjoerd.pages.apertis.org/apertis-website/designs/conditional_access/

798 **Trusted dealer**

799 As above, all authorized vehicle dealers, garages or other sale/repair locations
800 have to be trusted, as they have more unsupervised access to the vehicle's hard-
801 ware, and more capabilities, than the vehicle owner, passenger or a third party.

802 **Security domains**

- 803 • Automotive domain
 - 804 – There may be security sub-domains within the automotive domain,
805 but for the purposes of this design it is treated as a black box
- 806 • Consumer-electronics domain:
 - 807 – Each application sandbox in the consumer-electronics domain
 - 808 – CE domain operating system (this includes all the daemons for the
809 SDK APIs — these are technically separate security domains, but
810 since they communicate only with sandboxes and the CE domain
811 proxy, this makes the model more complex for no analytical advan-
812 tage)
 - 813 – CE domain proxy for the inter-domain communication
- 814 • Connectivity domain:
 - 815 – Connectivity domain handles the communication between AD and
816 the outer world.
 - 817 – Different protocol stacks.
 - 818 – CD domain proxy for communicating with AD
- 819 • Other devices on the in-vehicle network, and the outside world
- 820 • Hypervisor (if running as virtualised domains)

821 **Security model**

- 822 • Domains must assume that the inter-domain communication link has no
823 confidentiality or integrity, and is controlled by an attacker (a man in the
824 middle with the ability to modify traffic)
 - 825 – This means they must not trust any traffic from other devices on the
826 network
- 827 • The AD, CD and CE operating systems must assume all input from ex-
828 ternal sources (Wi-Fi, Bluetooth, GPS, 4G, etc.) is malicious
- 829 • The CE operating system may assume all API calls from the AD (as
830 proxied by the CE proxy) are *not* controlled by an attacker, assuming
831 they have come over an authenticated channel which guarantees integrity

- 832 between the AD and CE proxy; in other words, the AD must not deny
833 confidentiality or integrity to the CE
- 834 • The AD may deny availability to the CE operating system, by closing the
835 inter-domain link in response to the user disabling the CE while waiting
836 for a critical security update
 - 837 • The AD must assume all API calls from the CE are malicious, in case the
838 CE has been compromised
 - 839 • The CE must assume that all input and output from third party applica-
840 tions in sandboxes is malicious, including all their API calls
 - 841 • If a hypervisor is present:
 - 842 – The AD and CE operating systems may assume all control calls from
843 the hypervisor are *not* controlled by an attacker
 - 844 – The hypervisor must assume all input from the CE is malicious
 - 845 – The hypervisor may assume that all input from the AD is *not* mali-
846 cious
 - 847 * Note that, when combined with the fact that the AD cannot be
848 updated easily, this makes security bugs in the AD extremely
849 critical and extremely hard to fix
 - 850 • Tampering with any domain software must be detectable even if it is not
851 preventable (tamper evidence)
 - 852 • If one vehicle is attacked and compromised, the same effort must be re-
853 quired to compromise other vehicles

854 **Non-use-cases**

855 **Production CE domain used in multiple configurations**

856 A production CE domain operating system cannot be used in multiple config-
857 urations, for example as both an operating system running on one CPU of a
858 two-CPU board shared with the automotive domain OS; and then as an im-
859 age running on a separate board connected to an in-vehicle network with other
860 devices connected.

861 This requirement would mean that the inter-domain communications system
862 would have to support runtime reconfiguration, which would be a vector for
863 protocol-downgrade attacks while bringing no major benefits. An attacker could
864 try to trick the CE domain into believing it was in (for example) a virtualised
865 configuration when it wasn't, which could potentially disable its encryption, due
866 to the assumption the domain could make about its inter-domain communica-
867 tions link having inbuilt confidentiality.

868 **Requirements**

869 **Separated transport layer**

870 The transport layer for transmitting inter-domain communications between the
871 domains must be separated from the APIs being transported, in order to allow
872 for different physical links between the domains, with different security proper-
873 ties.

874 **Transport to SDK APIs**

875 Support a configuration where the CE is running in a virtual machine with the
876 Apertis SDK, so the peer (which would normally be the AD) is a mock AD
877 daemon running against the SDK.

878 See [Standalone setup](#).

879 **Transport over virtio**

880 Support a configuration where the CE and AD communicate over a virtio link
881 between two virtual machines under a hypervisor.

882 See [Basic virtualised setup](#).

883 **Transport over a private Ethernet link**

884 Support a configuration where the CE and AD are on separate CPUs and com-
885 municate over a point-to-point Ethernet link.

886 See [Separate CPUs setup](#), [Separate boards setup](#).

887 **Transport over a private Ethernet link to a development vehicle**

888 Support a configuration where the CE is running in an SDK on a laptop, and
889 the AD is running in a developer-mode Apertis device in a vehicle, and the two
890 communicate over a wider shared Ethernet.

891 See [Connecting an SDK to a development vehicle](#).

892 **Transport over a shared Ethernet link**

893 Support a configuration where the CE and AD are on separate CPUs are are
894 both connected to some wider shared Ethernet.

895 See [Separate boards setup with other devices](#), [Multiple CE domains setup](#).

896 **Transport over Unix Domain Socket**

897 Support a configuration where AD and CE are on the same host running as
898 Linux containers and connected via UDS. The same transport can be used on
899 OEM deployments and on SDK environments.

900 See [Linux container setup](#), [Multiple CE domains setup](#).

901 **Message integrity and confidentiality in transport layer**

902 Some of the possible physical links between domains do not guarantee integrity
903 or confidentiality of messages, so these must be implemented in the software
904 transport layer.

905 See [Separate CPUs setup](#), [Separate boards setup](#), [Separate boards setup with
906 other devices](#), [Multiple CE domains setup](#), [Wi-Fi access](#).

907 **Reliability and error checking in transport layer**

908 Some of the possible physical links between domains do not guarantee reliable
909 or error-free transfer of messages, so these must be implemented in the software
910 transport layer.

911 See [Separate boards setup](#), [Separate boards setup with other devices](#), [Multiple
912 CE domains setup](#).

913 **Mutual authentication between domains**

914 An attacker may interpose on the inter-domain communications link and at-
915 tempt to impersonate the AD to the CE, or the CE to the AD. The domains
916 must mutually authenticate before accepting any messages from each other.

917 See [Tinkering vehicle owner on the network](#).

918 **Separate authentication for developer and production mode devices**

919 A CE running in an SDK must be able to connect to and authenticate with
920 an AD running in a vehicle which is in a special ‘developer mode’. If the same
921 CE is connected to a production vehicle, it must not be able to connect and
922 authenticate.

923 See [Connecting an SDK to a development vehicle](#), [Connecting an SDK to a
924 production vehicle](#).

925 **Individually addressed domains**

926 In order to support multiple CE domains using the same automotive domain,
927 each domain (consumer–electronics and automotive) must be individually ad-
928 dressable. The system must not assume that there are only two domains in the
929 network.

930 See [Multiple CE domains setup](#).

931 **Traffic control for latency**

932 In order to support delivery of touchscreen events with low latency (so that UI
933 responsiveness is not perceptibly slow for the user), the system must guarantee
934 a low latency for all communications, or provide a traffic control system to
935 allow certain messages (for example, touchscreen messages) to have a guaranteed
936 latency.

937 See [Touchscreen events](#).

938 **Traffic control for bandwidth**

939 In order to prevent some kinds of high bandwidth message from using all the
940 bandwidth provided by the physical link, the system must provide a traffic
941 control system to ensure all types of message have fair access to bandwidth
942 (where ‘fairness’ is measured according to some rigorous definition).

943 This may be implemented by separating ‘control’ and ‘data’ streams (see sections
944 2.4 and 2.5), or by applying traffic control algorithms.

945 See [Wi-Fi access](#), [Bluetooth access](#).

946 **Traffic control for frequency**

947 In order to prevent denial of service due to a service sending too many messages
948 at once (so the communication overheads of those messages start to dominate
949 bandwidth usage), the system must guarantee fair access to enqueue messages.
950 This is subtly different from fair access to bandwidth: service A sending 100000
951 messages of 1KB per second and service B sending 1 message of 100000KB
952 per second have the same bandwidth requirements; but if the inter-domain link
953 saturates at 100000KB per second, some of the messages from service A must
954 be delayed or dropped as the messaging overheads exceed the bandwidth limit.

955 See [Denial of service through flooding](#).

956 **Separation of control and data streams**

957 Certain APIs will need to provide data and control streams separately, with dif-
958 ferent latency and bandwidth requirements for both. The system must support
959 multiple streams; this may be via an explicit separation between ‘control’ and
960 ‘data’ streams, or by applying traffic control algorithms.

961 See [Wi-Fi access](#), [Bluetooth access](#), [Audio transfer](#), [Video decoding](#).

962 **No untrusted access to AD hardware**

963 The entire point of an inter-domain communication system is to isolate the CE
964 from direct access to sensitive hardware, such as vehicle actuators or hardware
965 with direct memory access (DMA) rights to the AD CPU’s memory. This must
966 apply equally to decoder hardware — decoders or other hardware handling

967 untrusted data from users must not be trusted by the AD if the CE can send
968 untrusted user data to it, unless it is certified as a security boundary, able to
969 handle malicious user input without being exploited.

970 Specifically, this means that hardware decoders must only access memory which
971 is accessible by the AD CPU via an input/output memory management unit
972 (IOMMU), which provides memory protection between the two, so that the
973 hardware decoder cannot access arbitrary parts of memory and proxy that access
974 to a malicious or compromised application in the CE.

975 Note that it is not possible to check audio or video content for ‘badness’ before
976 sending it to a decoder, as that entails doing the full decoding process anyway.

977 See [Audio transfer](#), [Video decoding](#), [Video or audio decoder bugs](#), [Connecting
978 an SDK to a production vehicle](#).

979 **Trusted path for users to update the CE operating system**

980 There must exist a trusted path from the user to the system updater in the CE,
981 or to a component in the AD which will update the CE. The user must always
982 have access to this update system (it must always be *available*).

983 This trusted path may also be used by garages to upgrade the CE when servicing
984 a vehicle; or a different path may be used.

985 See [Video or audio decoder bugs](#), [After market CE upgrades](#), [Malicious CE UI](#).

986 **Safety limits on AD APIs**

987 The automotive domain must apply suitable safety limits to all of its APIs,
988 which are enforced within the AD, so that even if a properly authenticated and
989 trusted CE makes an API call, it is ignored if the call would make the AD do
990 something unsafe.

991 In this case, ‘safety’ is defined differently for each actuator or combination of
992 actuator settings, and will vary between AD implementations. It might not be
993 possible to detect all unsafe situations (in the sense of an unsafe situation which
994 could lead to an accident).

995 See [Tinkering vehicle owner on the boards](#), [Malicious CE](#).

996 **Rate limiting on control messages**

997 The inter-domain service in the CE and AD should impose rate limiting on
998 control messages coming from the CE, to avoid a compromised service in the CE
999 from using a denial of service attack to prevent other messages being transmitted
1000 successfully.

1001 This should be in addition to rate limiting implemented in the SDK APIs in the
1002 CE themselves, which are expected to be the first line of defence against denial
1003 of service attacks.

1004 See [Denial of service through flooding](#).

1005 **Ignore unrecognised messages**

1006 Both the CE and AD must ignore (and log warnings about) inter-domain com-
1007 munication messages which they do not recognise. If the message expects a
1008 reply, an error reply must be sent. The domains must not, for example, shut
1009 down or crash when receiving an unrecognised message, as that would lead to
1010 a denial of service vulnerability.

1011 See [Tinkering vehicle owner on the boards](#), [Malicious CE](#).

1012 **Portable transport layer**

1013 The transport layer must be portable to a variety of operating systems and
1014 architectures, in order that it may be used on different AD operating systems.
1015 This means, for example, that it must not depend on features added to very
1016 recent versions of the Linux kernel, or must have fallback implementations for
1017 them.

1018 See [Support multiple AD operating systems](#).

1019 **Support push mode and pull mode communications**

1020 The CE must be able to use pull mode communications with the AD, where
1021 it makes a method call and receives a reply; and push mode communications,
1022 where the AD emits a signal for an event, and the CE receives this.

1023 See [Support multiple AD operating systems](#).

1024 **OEM AD integration API**

1025 In order to allow any OEM to connect their AD to the system, there must
1026 be a well defined API which they connect their OEM-specific APIs for vehicle
1027 functionality to, in order for that functionality to be exposed over the inter-
1028 domain communication link.

1029 This API must support an implementation which uses the services in the Apertis
1030 SDK.

1031 See [Support multiple AD operating systems](#), [Standalone setup](#).

1032 **Flexibility in OEM AD integration API**

1033 As the functionality exported by different ADs differs, the integration API for
1034 connecting it to the inter-domain communication system must be a general one
1035 — it must not require certain functionality or data types, and must support
1036 functionality which was not initially expected, or which is not currently sup-
1037 ported by any CE. This functionality should be exposed on the inter-domain
1038 communications link, in case future versions of the CE can take advantage of it.

1039 See [Support multiple AD operating systems](#), [Before market CE upgrades](#), [After](#)
1040 [market CE upgrades](#), [New version of AD software](#), [New version of AD interfaces](#).

1041 **Inflexibility in OEM AD integration API**

1042 The OEM AD integration API must not allow access to arbitrary services or
1043 APIs on the AD. It must only allow access to the services and APIs explicitly
1044 exposed by the OEM in their use of the integration API.

1045 See [Unsupported AD interfaces](#).

1046 **Service discovery**

1047 Domains should be able to detect where specific services are hosted in case of
1048 multiple CE domains. If a service is moved from one CE domain to another
1049 CE domain, other domains should not require any reconfiguration. CE domains
1050 should not be able to spoof services that are meant to be provided by the AD.

1051 **Stability in inter-domain communications protocol**

1052 As the versions of the AD and CE change at different rates, the inter-domain
1053 communications protocol must be well defined and stable — it must not change
1054 incompatibly between one version of the CE and the next, for example.

1055 If the protocol uses versioning to add new features, both domains must support
1056 protocol version negotiation to find a version which is supported if the latest
1057 one is not.

1058 See [Before market CE upgrades](#), [After market CE upgrades](#), [New version of AD](#)
1059 [software](#), [Unsupported AD interfaces](#), [Protocol compatibility](#).

1060 **Testability of protocols**

1061 All IPC links in the inter-domain communications system must be testable in-
1062 dividually, without requiring the other parts of the system to be running. For
1063 example, the link between applications and SDK API services must be testable
1064 without running an automotive domain; the link between SDK API services and
1065 the inter-domain interface at the boundary of the CE domain must be testable
1066 without running an automotive domain; etc.

1067 See [Testability](#), [New version of AD software](#), [Unsupported AD interfaces](#).

1068 **Testability of protocol parsers and writers**

1069 All protocol parsers and writers in the inter-domain communications system
1070 must be testable individually, using unit tests and test vectors which cover all
1071 facets of the protocol. These tests must include negative tests — checks that
1072 invalid input is correctly rejected. For example, if a protocol requires a certificate

1073 to authenticate a peer, a test must be included which attempts a connection
1074 with different types of invalid certificate.

1075 See [Testability, New version of AD software, Unsupported AD interfaces](#).

1076 **Testability of processes**

1077 The code implementing all processes in the inter-domain communications system
1078 must be testable individually, without having to run each process as a subprocess
1079 in a test harness (because this makes testing slower and error prone). This means
1080 implementing each process as a library, with a well defined and documented API,
1081 and then using that library in a trivial wrapper program which hooks it up to
1082 input and output streams and accepts command line arguments.

1083 See [Testability, New version of AD software, Unsupported AD interfaces](#).

1084 **CE system services separated from transport layer**

1085 There must be a trust boundary between each service on the CE which has access
1086 to the inter-domain communication link, and the service which provides access
1087 to the inter-domain communications link itself. The inter-domain service should
1088 validate that messages from a service are related to that service (for example,
1089 by having a whitelist of types of message which each service can send).

1090 This limits the potential for escalation if service A is exploited — then the
1091 attacker can only use the inter-domain service to impersonate A, rather than
1092 to impersonate all services in the CE. It also allows the resource usage of the
1093 inter-domain service to be limited, to limit the impact of a denial of service
1094 attack on it.

1095 See [Malicious CE, Marshalling resource usage](#).

1096 **No dependency on CE specific hardware**

1097 As the CE hardware may be upgraded by a garage at some point, the inter-
1098 domain communications should not depend on specific identifiers in this hard-
1099 ware, such as an embedded cryptographic key. Such keys may be used, but the
1100 AD should accept multiple keys (for example, all keys signed by some overall
1101 key provided by Apertis to all OEMs), rather than only accepting the specific
1102 key from the hardware it was originally run against.

1103 This requirement may also be satisfied by including provisions for updating the
1104 copy of a key in the AD if such a dependency on a specific CE key is a sensible
1105 implementation choice.

1106 See [After market upgrade of a domain](#).

1107 **Immediate error response if service on peer is unavailable**

1108 If a service on the peer has crashed or is unresponsive, but the peer itself (including its inter-domain communications link) is still responsive, that peer should
1109 return an error to the other domain, which should propagate it to any caller of
1110 SDK APIs which use the failing service. An error response must be returned,
1111 otherwise the caller will time out.
1112

1113 See [Power cycle independence of domains \(CE down\)](#), [Power cycle independence of domains \(AD down, single screen\)](#), [Power cycle independence of domains \(AD down, multiple screens\)](#), [Plug-and-play CE device](#)

1116 **Immediate error response if peer is unavailable**

1117 If the peer has crashed, or is not currently connected to the physical inter-domain communications link (either because it has been unplugged or due to a
1118 fault), the other peer must generate a local error response in the inter-domain
1119 service and return that to any caller of SDK APIs which require inter-domain
1120 communications. An error response must be returned, otherwise the caller will
1121 time out.
1122

1123 See [Power cycle independence of domains \(CE down\)](#), [Power cycle independence of domains \(AD down, single screen\)](#), [Power cycle independence of domains \(AD down, multiple screens\)](#), [Plug-and-play CE device](#)

1126 **Timeout error response if peer does not respond**

1127 If the peer is unresponsive to a particular inter-domain message, the other peer
1128 must generate a local error response in the inter-domain service and return that
1129 to the caller of the SDK API which required inter-domain communications. An
1130 error response must be returned, otherwise the caller will wait for a response
1131 indefinitely (or have to implement its own timeout logic, which would be redundant).
1132

1133 See [Power cycle independence of domains \(CE down\)](#), [Power cycle independence of domains \(AD down, single screen\)](#), [Power cycle independence of domains \(AD down, multiple screens\)](#), [Plug-and-play CE device](#)

1136 **All inter-domain communications APIs are asynchronous**

1137 As inter-domain communications may have some latency, or may time out after
1138 a number of seconds, all SDK APIs which require inter-domain communications
1139 must be asynchronous, in the [GLib sense](#)¹⁰: the call must be started, a handler
1140 for its response added to the caller's main loop, and the caller must continue
1141 with other tasks until the response arrives from the other domain.

¹⁰<https://developer.gnome.org/gio/stable/GAsyncResult.html>

1142 This encourages UIs to be written to not block on SDK API calls which might
1143 take multiple seconds to complete, as during that time, the UI would not be
1144 redrawn at all, and hence would appear to ‘freeze’.

1145 See [Temporary communications problem](#).

1146 **Reconnect to peer as soon as it is available**

1147 If a domain has crashed and restarted, or was disconnected from the inter-
1148 domain communications link and then reconnected, the domain must reconnect
1149 to its peer as soon as the peer can be found on the network. If, for example,
1150 both domains had crashed, this may involve waiting for the peer to connect to
1151 the network itself.

1152 See [Plug-and-play CE device](#).

1153 **External domain watchdog**

1154 Both domains must be connected to an external watchdog device which will
1155 restart them if they crash and fail to restart themselves.

1156 The watchdog must be external, rather than being the other domain, in case
1157 both domains crash at the same time.

1158 See [Power cycle independence of domains \(CE down\)](#), [Power cycle independence](#)
1159 [of domains \(AD down, single screen\)](#), [Power cycle independence of domains \(AD](#)
1160 [down, multiple screens\)](#).

1161 **Reporting system for malicious applications**

1162 There should exist a trusted path from the application launcher in the CE to
1163 the Apertis store to allow the launcher to provide feedback about applications
1164 which are detected to have done ‘malicious’ things, such as called an SDK API
1165 with parameters which are obviously out of range.

1166 If such a path exists, the inter-domain service in the CE must be able to detect
1167 error responses from the AD which indicate that malicious behaviour has been
1168 detected and rejected, and must be able to forward those notifications to the
1169 reporting system.

1170 See [Feedback for malicious applications](#).

1171 **Ability to disable the consumer–electronics domain**

1172 There must exist a trusted path to a setting in the AD to allow the vehicle
1173 owner to disable the CE because it has been compromised, pending taking the
1174 vehicle to a trusted dealer to install an update.

1175 As well as preventing booting the CE, this must disable all inter-domain com-
1176 munications from within the inter-domain service in the AD.

1177 See [Compromised CE with delayed fix](#).

1178 **Tamper evidence**

1179 If the CE or AD, or communications between them are tampered with by an
1180 attacker, it must be possible for an investigator (who is trusted by and has access
1181 to tools provided by the OEM) to determine that the software or hardware was
1182 modified — although it might not be possible for them to determine *how* it was
1183 modified. This will allow for liability to be attributed in the event of an accident
1184 or warranty claim.

1185 See [Tinkering vehicle owner on the network](#), [Tinkering vehicle owner on the](#)
1186 [boards](#).

1187 **No global keys in vehicles**

1188 The security which protects the inter-domain communication system (including
1189 any trusted boot security) must use unique keys for each vehicle, and must not
1190 have a global key (one which is the same in all vehicles) as a single point of
1191 failure.

1192 This means that if an attacker manages to compromise one vehicle, they must
1193 not be able to learn anything (any keys) which would allow them to compromise
1194 another vehicle with less effort.

1195 See [Tinkering vehicle owner on the network](#), [Tinkering vehicle owner on the](#)
1196 [boards](#).

1197 **Existing inter-domain communication systems**

1198 As this is quite a unique problem, we know of no directly comparable systems.
1199 More generally, this is an instance of a distributed system, and hence similar
1200 in some respects to a number of existing remote procedure call systems or dis-
1201 tributed middleware systems.

1202 If comparisons with specific systems would be beneficial, they can be included
1203 in a future revision of this document.

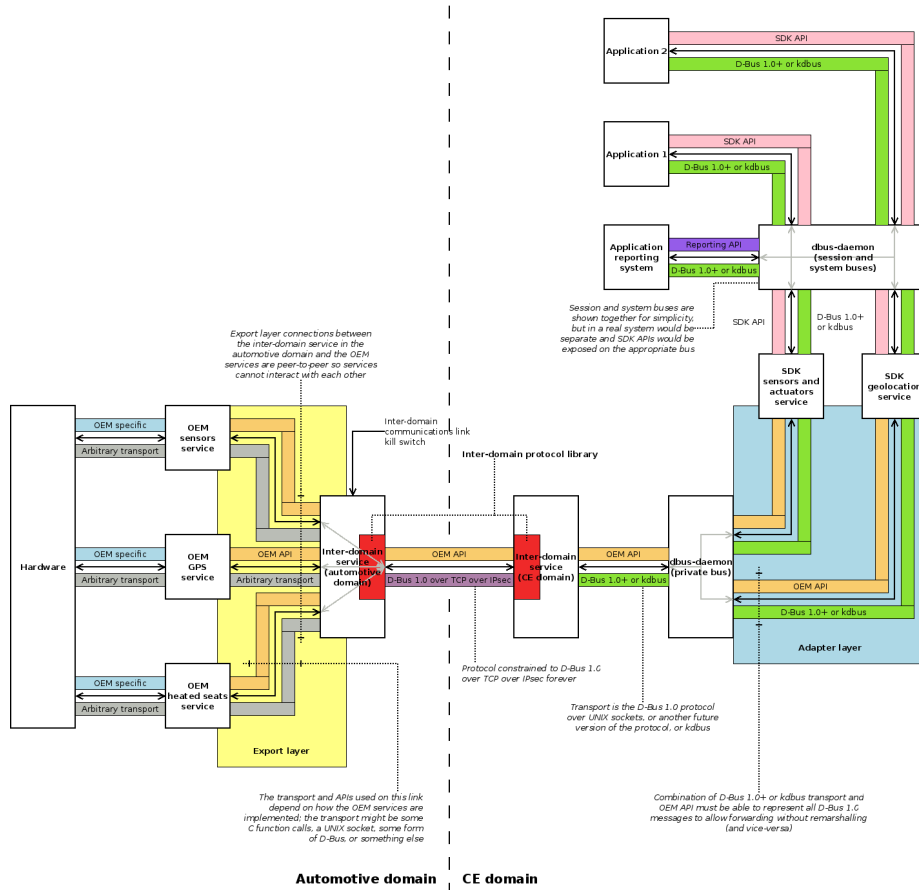
1204 **Open question:** Are there any relevant existing systems to compare against?

1205 **Approach**

1206 Based on the [above research][Existing domain communications system] and
1207 [Requirements](#), we recommend the following approach as an initial sketch of an
1208 inter-domain communication system.

1209 **Overall architecture**

1210 In the following figure, each box represents a process, and hence each connection
 1211 between them is a trust boundary.



1212

1213 Apertis IDC architecture. The ‘OEM specific’ APIs are also known
 1214 as ‘native OEM APIs’; and the ‘OEM API’ is also known as the
 1215 ‘Apertis automotive API’. For more information on the export and
 1216 adapter layer, see [Automotive domain export layer](#) and [Consumer-
 1217 electronics domain adapter layer](#).

1218 APIs from the automotive domain are exported by an *export layer* ([Automotive domain export layer](#)) as D-Bus objects on the inter-domain communications link. This link runs a known version of the D-Bus protocol (and requires backwards compatibility indefinitely) between an *inter-domain service* process in each domain ([Protocol library and inter-domain services](#)). The inter-domain service in the CE domain sends and receives D-Bus messages for the objects exported by the automotive domain, and proxies them to a private bus in the

1225 CE domain. SDK services in the CE domain connect to this bus, and an *adapter*
1226 *layer* **Consumer-electronics domain adapter layer** in each service converts the
1227 APIs from the automotive domain to the SDK APIs used in the version of Aper-
1228 tis in use in the CE domain. These SDK APIs are exported onto the normal
1229 D-Bus session bus, to be used by applications (**Flow for a given SDK API call**).

1230 The export layer and adapter layer provide abstraction of the APIs from the
1231 automotive domain: the export layer converts them from C APIs, QNX message
1232 passing, or however they are implemented in the automotive OS, to a D-Bus API
1233 which is specific to that OEM, but which has stability guarantees through use
1234 of API versioning (**Interaction of the export and adapter layers**). The adapter
1235 layer converts from this D-Bus API to the current version of the Apertis SDK
1236 APIs. Both layers are OEM-specific.

1237 The use of the D-Bus protocol throughout the system means that between the
1238 export layer and the adapter layer, message contents do not need to be re-
1239 marshalled — messages only need their headers to be changed before they are
1240 forwarded. This should eliminate a common cause of poor performance (re-mar-
1241 shalling).

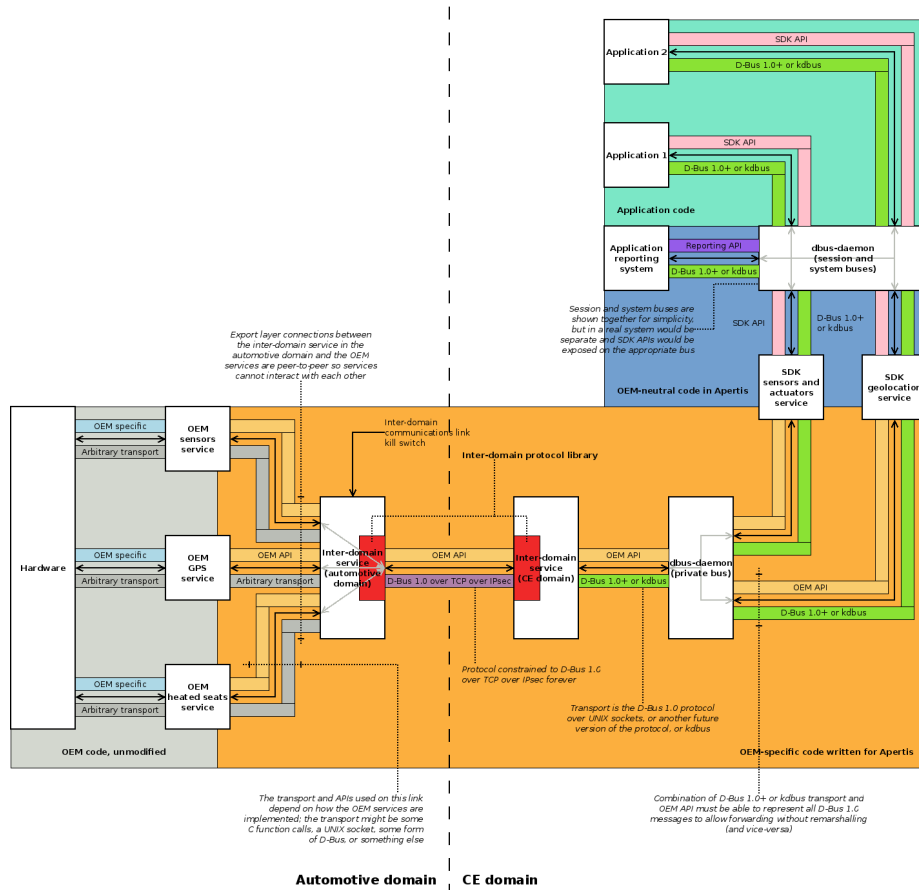
1242 High-bandwidth **Data connections** are provided in parallel with the *control con-*
1243 *nection* which runs this D-Bus protocol (**Control protocol**). They use TCP,
1244 UDP or Unix sockets, and are opened between the two inter-domain services on
1245 request. Applications and services must define their own protocols for commu-
1246 nicating over these links, which are appropriate to the data being transferred
1247 (for example, audio data or a Bluetooth file transfer).

1248 Authentication, confidentiality and integrity of all inter-domain communications
1249 (the control connection and data connections) are provided by using IPsec
1250 as the bottom layer of the protocol stack (**Encryption**). The same protocol stack
1251 is used for all configurations of the two domains (from a standalone CE domain
1252 through to multiple CE domains on a shared network with an automotive do-
1253 main), to ensure that the same code path is used for all configurations and hence
1254 is widely tested (**Configuration designs**).

1255 Addressing and discovery of domains, before the initial connection between
1256 them, is provided by IPv6 neighbour discovery (**Traffic control**).

1257 Traffic control is implemented in the CE domain using standard Linux kernel
1258 traffic control mechanisms, with the policy specified by the inter-domain ser-
1259 vice (section 8.4). It is applied for the control connection and for each data
1260 connection separately, as they are all separate TCP or UDP connections.

1261 The only exception from the above is **Linux container setup** which uses Unix
1262 Domain Sockets as a trusted and reliable bottom transport layer instead of
1263 IPsec. In this case, there is no need for traffic control. Addressing and discovery
1264 of local domains in **Linux container setup** is based on common directories created
1265 and shared outside of the containers by the container manager.



1266

1267 Responsibilities for areas of code in the IDC architecture

1268 **Security domains**

1269 As process boundaries are the only way of enforcing trust boundaries, each
 1270 of these security domains corresponds to at least one separate process in the
 1271 system.

- 1272 • Inter-domain service in the automotive domain. We recommend that this
- 1273 remains a separate security domain from the rest of the services and soft-
- 1274 ware running in the AD. This allows it to be isolated from other compo-
- 1275 nents to reduce the attack surface exposed by the AD.
- 1276 • Rest of the automotive domain: as mentioned in **Security domains**, the
- 1277 automotive domain is essentially a black box.
- 1278 • Each application sandbox in the consumer-electronics domain.
- 1279 • Inter-domain service in the consumer-electronics domain.

- 1280 • Each service for an SDK API in the consumer–electronics domain. The
1281 trust boundaries between them may not be enforced strongly (as all ser-
1282 vices in the consumer–electronics domain are considered as trusted parts
1283 of the operating system), but their trust boundaries with the inter-domain
1284 service should be enforced, and the inter-domain service should consider
1285 them as potentially compromised.
- 1286 • Other devices on the in-vehicle network, and the outside world.
- 1287 • Hypervisor (if running as virtualised domains).

1288 Protocol design

1289 The protocol for communicating data between the domains has two *planes*: the
1290 control plane, and the data plane. They have different requirements, but both re-
1291 quire addressing, routing, mutual authentication of peers, confidentiality of data
1292 and integrity of data. In addition, the control plane must have bi-directional,
1293 in-order transmission, framing, reliability and error detection. Conversely, the
1294 data plane must have multiplexing, and the ability to apply traffic control to
1295 each of its connections ([Traffic control](#)).

1296 The control plane is used for sending control data between the domains — these
1297 are the method calls which form the majority of inter-domain communications.
1298 They require low latency, and are low bandwidth. The [control protocol][Control
1299 protocol] itself provides push and pull method call semantics, and allows for new
1300 data connections ([Data connections](#)) to be opened. Only one control connection
1301 exists between a pair of domains, and it is always connected.

1302 The data plane is used for high bandwidth data, such as video or audio streams,
1303 or Wi-Fi, 4G or Bluetooth downloads. The latency requirements are variable,
1304 but all connections are high bandwidth. The inter-domain communication sys-
1305 tem provides a plain stream for each data plane connection, and services must
1306 implement their own protocol on top which is appropriate for the specific type
1307 of data being transmitted (for example, audio or video streaming; or Wi-Fi
1308 downloads). Data connections are created between two domains on demand,
1309 and are closed after use.

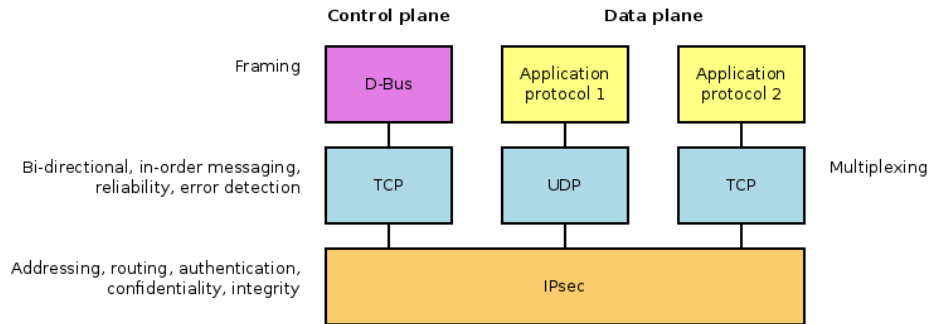
1310 IPsec versus TLS

1311 An important design decision is whether to use [IPsec](#)¹¹ or [TLS](#)¹² (and DTLS)
1312 for providing the security properties of the inter-domain connection.

1313 If IPsec is used (following figure), it forms the bottom layer of the protocol hierar-
1314 chy, and implements addressing, routing, mutual authentication, confidentiality
1315 and integrity for *all* connections in the control and data planes.

¹¹<https://en.wikipedia.org/wiki/IPsec>

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

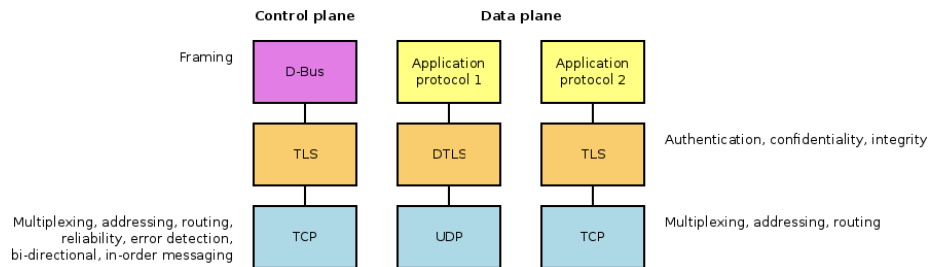


1316

1317 Protocol stacks for control and data planes if using IPsec.

1318 If TLS is used (Following figure), it forms the layer just below the application
 1319 protocols in the protocol hierarchy — the control plane would use a single
 1320 TLS over TCP connection; and the data plane would use multiple TLS over
 1321 TCP or DTLS over UDP connections. TLS (and hence DTLS — they have the
 1322 same security properties) implements mutual authentication, confidentiality and
 1323 integrity, but only for a single connection; each new connection needs a new TLS
 1324 session.

1325 The chief advantage of IPsec is its transparency: any protocol can be tunnelled
 1326 using it, without needing to know about the security properties it has. However,
 1327 to do this, IPsec needs to be supported by both the AD and CE kernels. Some
 1328 automotive operating systems may not support IPsec (although, as a data point,
 1329 QNX seems to).



1330

1331 Protocol stacks for control and data planes if using TLS.

1332 A [2003 review of the IPsec protocol](https://www.schneier.com/cryptography/archives/2003/12/a_cryptographic_eval.html)¹³ identified a number of problems with it.
 1333 However, since then, it has been updated by [RFC 4301](https://tools.ietf.org/html/rfc4301)¹⁴, [RFC 6040](https://tools.ietf.org/html/rfc6040)¹⁵ and [RFC](https://tools.ietf.org/html/rfc7619)
 1334 [7619](https://tools.ietf.org/html/rfc7619)¹⁶. These should be evaluated and the overall protocol security determined.
 1335 In contrast, the security of TLS has been well studied, especially in recent years
 1336 after the emergence of various vulnerabilities in it. TLS has the advantage that
 1337 it is a smaller set of protocols than IPsec, and hence easier to study.

¹³https://www.schneier.com/cryptography/archives/2003/12/a_cryptographic_eval.html

¹⁴<https://tools.ietf.org/html/rfc4301>

¹⁵<https://tools.ietf.org/html/rfc6040>

¹⁶<https://tools.ietf.org/html/rfc7619>

1338 **Open question:** What is the security of the IPsec protocol in its current (2015)
1339 state?

1340 Performance-wise, TLS requires a handshake for each new connection, which
1341 imposes connection latency of at least one round trip (assuming use of [TLS ses-](#)
1342 [sion resumption](#)¹⁷) for each new connection (on top of other latency such as the
1343 TCP handshake). It is not possible to use a single TLS session and multiplex
1344 connections within it, as this puts the protocol reliability (TCP retransmission)
1345 below the multiplexing in the protocol stack, which makes the multiplexed con-
1346 nection prone to [head of line blocking](#)¹⁸, which seriously impacts performance,
1347 and allows one connection to perform a denial of service attack on all others it
1348 is multiplexed with. IPsec has the advantage of not requiring this handshake
1349 for each connection, which significantly reduces the latency of creating new con-
1350 nections, but does not affect their overall bandwidth once they have reached a
1351 steady state.

1352 **Open question:** What is the performance of TCP and UDP over IPsec, TLS
1353 over TCP and DTLS over UDP on the Apertis reference hardware?

1354 Overall, we recommend using IPsec if it is expected to be supported by all
1355 automotive domain operating systems which will be used with Apertis systems.
1356 Otherwise, if an AD OS might not support IPsec, we recommend using TLS
1357 over TCP and DTLS over UDP for *all* configurations. We do *not* recommend
1358 providing a choice for OEMs between IPsec and TLS, as this doubles the possible
1359 configurations (and hence testing) of a part of the system which is both complex
1360 and security critical.

1361 The remainder of this document assumes that IPsec is chosen. Throughout,
1362 please read ‘IPsec’ as meaning ‘the IPsec protocol stack or the TLS protocol
1363 stack’.

1364 Configuration designs

1365 The physical links available between the domains differ between configurations of
1366 the domains, as do their properties. For some configurations ([Standalone setup](#),
1367 [Basic virtualised setup](#), [Linux container setup](#)) confidentiality and integrity of
1368 the inter-domain communications protocol are not strictly necessary, as the
1369 physical link itself cannot be observed by an attacker. However, for the other
1370 configurations, these two properties are important.

1371 Since the first two configurations are the ones which are typically used for devel-
1372 opment, we suggest implementing confidentiality and integrity for them anyway,
1373 regardless of the fact it’s not strictly necessary. This avoids the situation where
1374 the code running on production configurations is vastly different from that run-
1375 ning on development configurations. Such a situation often leads to inadequate
1376 testing of the production code.

¹⁷<https://tools.ietf.org/html/rfc5077>

¹⁸https://en.wikipedia.org/wiki/Head-of-line_blocking

1377 This should be weighed against the potential performance gains from eliminating
1378 encryption from those connections, and the potential gains in debuggability
1379 (for the [Standalone setup](#) and [Linux container setup](#)) by being able to inspect
1380 network traffic without needing to extract the encryption key.

1381 **Open question:** What trade-off do we want between performance and testa-
1382 bility for the different transport layer configurations?

1383 Standalone setup

1384 IPsec running on a [loopback interface](#)¹⁹ to a service running in the SDK which
1385 mocks up the inter-domain service running in the AD. The security properties it
1386 provides are technically not needed, as the standalone setup is for development
1387 and is ignored by the security model.

1388 Even though there are only two peers communicating, they will both have and
1389 use a full addressing scheme ([Addressing and peer discovery](#)).

1390 Basic virtualised setup

1391 A virtio-net connection must be set up in the CE and AD virtual guests, using
1392 a private network containing those two peers. If the AD cannot be modified to
1393 enable a virtio-net connection, a normal virtualised Ethernet connection must
1394 be used.

1395 Virtio-net is the name of the KVM paravirtualised network driver
1396 (<http://www.linux-kvm.org/page/Virtio>). Similar paravirtualised
1397 drivers exist for most hypervisors; so an appropriate one for the
1398 hypervisor should be used. For simplicity, this document will use
1399 ‘virtio-net’ to refer to them all.

1400 In either case, the transport layer will use IPsec between the two. The security
1401 properties it provides are technically not needed for a virtualised configuration,
1402 as the security model guarantees that the hypervisor maintains confidentiality
1403 and integrity of the connection.

1404 Even though there are only two peers on the network, they will both have and
1405 use a full addressing scheme ([Addressing and peer discovery](#)).

1406 Separate CPUs setup

1407 A normal Ethernet connection must be used to connect the AD and CE on a
1408 private network. IPsec will be used over this Ethernet link, providing all the
1409 necessary transport layer properties.

1410 Even though there are only two peers on the network, they will both have and
1411 use a full addressing scheme, described below.

1412 Separate boards setup

1413 Same as for the separate CPUs setup.

¹⁹https://en.wikipedia.org/wiki/Loopback#Virtual_loopback_interface

1414 Separate boards setup with other devices

1415 Same as for the separate CPUs setup.

1416 Multiple CE domains setup

1417 Same as for the separate CPUs setup. Each domain's address must be unique,
1418 and the use of addressing in this configuration becomes important.

1419 Linux container setup

1420 The communication is based on Unix Domain Sockets (UDS) shared between
1421 the counterpart domains; this means that a common directory must be shared
1422 for each pair of communicating domains. This directory must be writable by at
1423 least one container, such that its gateway layer or adapter layer can create the
1424 named unix domain socket file and listen on it, and must be readable on the
1425 other container, which will connect to the shared named unix domain socket
1426 file. The dedicated shared directory for communication may support space
1427 limits for writing and inodes creation, for example: dedicated `tmpfs` mount or
1428 `btrfs` subvolume quota, to prevent denials of service due to filesystem space
1429 exhaustion.

1430 The container manager is responsible for the actions below when each container
1431 is started or stopped:

- 1432 • a shared storage space (a size-constrained `tmpfs` mount or `btrfs` subvol-
1433 ume) must be defined for each pair of containers on the host system, for
1434 instance `${IDC_HOST_DIR}/automotive-connectivity` for the link connecting
1435 the `automotive` and `connectivity` domains
- 1436 • the shared storage must be mounted by the container manager with
1437 read/write permissions on the first domain of the pair, for instance as
1438 `${IDC_DIR}/connectivity` in the `automotive` domain
- 1439 • the same shared storage must be mounted by the container manager
1440 with read permissions on the second domain of the pair, for instance as
1441 `${IDC_DIR}/automotive` in the `connectivity` domain
- 1442 • when the container is stopped, the shared storage and mounts associated
1443 with the container must be unmounted

1444 The variables `${IDC_HOST_DIR}` and `${IDC_DIR}` mentioned above represent the
1445 paths where the shared spaces are mapped on the host and containers filesys-
1446 tems respectively. By default, both variables `${IDC_HOST_DIR}` and `${IDC_DIR}`
1447 are defined in a common manner as `/var/lib/idc/`. OEM or developer's setup
1448 may require to redefine these paths for the customised environment.

1449 **Addressing and peer discovery**

1450 **Network addressing and peer discovery**

1451 Each domain will be identified by its IPv6 address, and domains will be dis-
1452 covered using the IPv6 protocol's secure [neighbour discovery](#)²⁰ protocol. As
1453 domains do not need to be human-addressable (indeed, the users of the vehicle
1454 need never know that it has multiple domains running in it), there is no need
1455 to use DNS or mDNS for addressing.

1456 The neighbour discovery protocol includes a feature called neighbour unreach-
1457 ability detection, which should be used as one method of determining that one
1458 of the domains has crashed. When a domain crashes, the other domain should
1459 poll for its existence on the network at a constant frequency (for example, at
1460 2Hz) until it reappears at the same address as before. This frequency of polling
1461 is a trade-off between not flooding the network with connectivity checks, but
1462 also detecting reappearance of the domain rapidly.

1463 When reconnecting to a restarted domain, the normal authentication process
1464 should be followed, as if both domains were starting up normally. There is no
1465 state to restore for the inter-domain link itself but, for example, SDK services
1466 may wish to re-query the automotive domain for the current vehicle state after
1467 reconnecting. They should do this after receiving an error response from the
1468 AD for an inter-domain communication which indicated that the other domain
1469 had crashed. Such behaviour is up to the implementers of each SDK service,
1470 and is not specified in this design.

1471 **Container-based addressing and peer discovery**

1472 Each container must be assigned a unique name on the filesystem to be used
1473 as domain identifier for addressing and peer discovery purposes.

1474 The `/${IDC_DIR}` directory in the container contains a directory entry for each
1475 associated domain to be connected through the inter-domain communication
1476 mechanism. As described in [Linux container setup](#), the container manager is
1477 responsible for mounting a dedicated shared space to host the socket for the
1478 container pairs.

1479 The name of mount point for the shared directory in the container should be the
1480 same as the name of counterpart peer. For example, to connect an `automotive`
1481 and a `connectivity` domain, the shared space must be mounted in the `automotive`
1482 container on the `/${IDC_DIR}/connectivity/` path and must be mounted in the
1483 `connectivity` container on the `/${IDC_DIR}/automotive/` path.

1484 On startup, each container in the pair must try to `unlink()` any stale file in
1485 the shared spaces and then create a Unix Domain Socket named `socket` there.
1486 Since the shared directory is mounted with write permissions only on a single
1487 domain, the `unlink()` and `bind()` calls on the unix socket file will fail on the
1488 other domain, which only has read permissions.

1489 Once it has removed any stale file and successfully created the socket, the first
1490 container in the pair must then `listen()` on it: for instance the `automotive`

²⁰https://en.wikipedia.org/wiki/Secure_Neighbor_Discovery

1491 domain must listen on the `/${IDC_DIR}/connectivity/socket` unix socket. The
1492 second container in the pair must instead wait for the `socket` file to be available
1493 and must connect to it as soon it is created: for instance the `connectivity` must
1494 wait for the `/${IDC_DIR}/automotive/socket` file to appear and connect to it.

1495 **Encryption**

1496 The confidentiality, integrity and authentication of the inter-domain commu-
1497 nications link is provided by IPsec in transport mode for networked setups,
1498 and by kernel-provided Unix Domain Sockets on [container-based setups][Linux
1499 container setup].

1500 **Open question:** What more detailed configuration options can we specify for
1501 setting up IPsec? For example, disabling various optional features which are
1502 not needed, to reduce the attack surface. What IKE service should be used?

1503 The system should use an IPsec security policy which drops traffic between
1504 the CE and AD unless IPsec is in use. The security policy should not specify
1505 behaviour for communications with other peers.

1506 Each domain must have an X.509 certificate (essentially, a public and private
1507 key pair), which are used for automatic keying for the IPsec connections. The
1508 certificates installed in the automotive domain must be signed by a certificate
1509 authority (CA) specific to the automotive domain and possibly the OEM. The
1510 certificates installed in the CE domain must be signed by a CA specific to the
1511 CE domain and possibly the OEM.

1512 A domain (automotive or CE) which is in developer mode must use a certificate
1513 which is signed by a developer mode CA, not the production mode CA. This
1514 allows a production mode domain to prevent connections from a developer mode
1515 domain.

1516 See [Appendix: Software versus hardware encryption](#) for a comparison of soft-
1517 ware and hardware encryption.

1518 In order to maintain confidentiality of the connection, the keys for the IPsec
1519 connection must be kept confidential, which means they must be stored in mem-
1520 ory which is not accessible to an attacker who has physical access to the system
1521 (see [Tamper evidence and hardware encryption](#)); or they must be encrypted
1522 under a key which is stored confidentially (a key-encrypting key, KEK). Such
1523 a confidential key store should be provided by the Secure Boot design — if
1524 available, confidentiality of the inter-domain communications can be guaran-
1525 teed. If not available, inter-domain communications will not be confidential if
1526 an attacker can extract the boot keys for the system and use them to extract
1527 the inter-domain communications keys.

1528 As of February 2016, the Secure Boot design is still forthcoming

1529 See section 8.15 for further discussion of the hardware base for confidentiality
1530 and integrity of the system.

1531 **Open question:** A lot of business logic for control over OEM licencing can
1532 be implemented by the choice of the CA hierarchy used by the inter-domain
1533 communication system. What business logic should be possible to implement?

1534 **Open question:** Consider key control, revocation, protocol obsolescence, and
1535 various extensions for pinning keys and protocols.

1536 **Open question:** What can be done in the automotive domain to reduce the
1537 possibility of exploits like [Heartbleed](#)²¹ affecting the inter-domain communi-
1538 cations link? This is a trade-off between the stability of AD updates (high; rarely
1539 released) and the pace of IPsec and TLS security research and updates and the
1540 need for crypto-agility (fast). Heartbleed was a bug in a bad implementation of
1541 an optional and not-very-useful TLS extension.

1542 **Control protocol**

1543 The control protocol provides push and pull method call semantics and a type
1544 system for marshalling method call parameters and return values — but it does
1545 not prescribe a specific set of APIs which it will transport. It must be flexible
1546 in the set of APIs which it transports.

1547 We suggest using D-Bus over TCP as the control protocol, using a private bus
1548 between the automotive domain and the consumer–electronics domain. For
1549 multiple CE domain configurations, each automotive—consumer–electronics do-
1550 main pair would have its own private bus.

1551 The transport should be implemented using D-Bus’ TCP [socket transport](#)²²
1552 mechanism. Authentication, confidentiality and integrity are provided by the
1553 underlying IPsec connection. D-Bus implements its own datagram framing on
1554 top of the TCP stream.

1555 On this bus, APIs from the automotive domain would be exposed as services;
1556 the CE domain can then call methods on those services, or receive signals from
1557 them.

1558 D-Bus was chosen as it implements the necessary functionality, reuses a lot of
1559 the technologies already in use in Apertis, is stable, and is familiar to Apertis
1560 developers. Note that we suggest D-Bus the *protocol*, not necessarily dbus-
1561 daemon the *message bus daemon* or libdbus the reference *protocol library*. D-Bus
1562 the protocol provides:

- 1563 • Method calls (pull semantics) with exactly one reply, supporting timeouts
- 1564 • Error responses
- 1565 • Signals (push semantics)
- 1566 • Properties

²¹<https://en.wikipedia.org/wiki/Heartbleed>

²²<http://dbus.freedesktop.org/doc/dbus-specification.html#transports-tcp-sockets>

- 1567 • Strong type system
- 1568 • Introspection

1569 There are several important points here: introspection means that the D-Bus
1570 services on the AD can send their API definitions to the CE at runtime if needed,
1571 so that the CE does not have to have access to header files (or similar) from the
1572 AD. It also means the API definition can change without needing to recompile
1573 things — for example, an update to the AD could expose new APIs to the CE
1574 without needing to update header files on the CE. Finally, method calls support
1575 ‘in’ and ‘out’ parameters (multiple return values) which allows for bi-directional
1576 communication in the control protocol.

1577 **Open question:** How should the multiple CE configuration ([Configuration de-](#)
1578 [signs](#) interact with D-Bus signals? Can the adapter layer perform the broadcast
1579 to all subscribers?

1580 The D-Bus protocol is stable, and has maintained backwards compatibility with
1581 all previous versions since 2006²³. If changes to the D-Bus protocol are intro-
1582 duced in future, they will be introduced as extensions which are used optionally,
1583 if supported by both peers on the bus. Hence backwards compatibility is main-
1584 tained.

1585 Data connections

1586 If a service wishes to send high-bandwidth data between the domains, it must
1587 open a new data connection. Data connections are created on demand, and
1588 are subject to traffic control, so the AD may, for example, reject a connection
1589 request or throttle its bandwidth in order to maintain quality of service for
1590 existing connections.

1591 The inter-domain communication protocol provides two types of data connec-
1592 tion: TCP-like and UDP-like. These are implemented as TCP or UDP connec-
1593 tions between the two domains, running over IPsec. IPsec provides the necessary
1594 authentication, confidentiality and integrity of the data; TCP or UDP provide
1595 the multiplexing between connections (see the IPsec protocol stacks figure in
1596 [IPSec versus TLS](#)). For [Linux container setup](#) a Unix domain socket is used as
1597 the IDC link; the local kernel provides the needed authentication, confidential-
1598 ity and integrity of the data. Services must implement their own application-
1599 specific protocols on top of the TCP or UDP connection they are provided. For
1600 example, a video service may use a lossy synchronised audio/video protocol over
1601 UDP for sending video data together with synchronised audio; while a down-
1602 load service may use HTTP over TCP for sending downloads between domains.
1603 (See [\[here\]\[Appendix: Audio and video decoding\]](#) for a discussion of options for
1604 implementing video and audio decoding.) Such protocols are not defined as part
1605 of this design — they are the responsibility of the services themselves to design
1606 and implement.

²³<http://dbus.freedesktop.org/doc/dbus-specification.html#stability>

1607 Data connections are opened by sending a request to one of the inter-domain
1608 services ([Protocol library and inter-domain services](#)), specifying desired charac-
1609 teristics for the connection, such as whether it should be TCP-like or UDP-like,
1610 its bandwidth and latency requirements, etc. The connection will be opened
1611 and a unique identifier and file descriptor for it returned to the requesting ser-
1612 vice. This service must then send the identifier over the control connection so
1613 that the corresponding service in the other domain can request a file descriptor
1614 for the other end of the connection from its inter-domain service.

1615 **Open question:** Could this be simplified by using D-Bus' support for file de-
1616 scriptor passing? D-Bus' TCP transport currently explicitly does not support
1617 file descriptor passing, so implementing it that way without introducing incom-
1618 patibilities requires planning.

1619 It is tempting to extend D-Bus' support for file descriptor (FD) passing so that
1620 it operates over TCP to provide these data connections. However, that would
1621 effectively be a fork of the D-Bus protocol, which we do not want to maintain as
1622 part of this system. Secondly, due to the way FD passing works, with the peer
1623 passing an FD to the dbus-daemon and asking for it to be forwarded — this
1624 would mean that the peer (i.e. an SDK or OEM service) has the responsibility
1625 for opening the data connection within the IPsec tunnel, which would be very
1626 complex.

1627 Instead, we recommend a custom API provided by the inter-domain service
1628 which an SDK or OEM service can call to open a new data connection, passing
1629 in the parameters for the connection (such as TCP/UDP, quality of service
1630 requirements, etc.). The inter-domain service would communicate over a private
1631 control API with the other inter-domain service to open and authenticate the
1632 connection at both ends, and return a file descriptor and cryptographic nonce
1633 (securely random value at least 256 bits long) to the original SDK or OEM
1634 service. This service can use that file descriptor as the data connection, and
1635 should pass the nonce over its own control protocol to the corresponding OEM or
1636 SDK service. This service should then pass the nonce to its inter-domain service
1637 and will receive the file descriptor for the other end of the data connection in
1638 reply.

1639 Both inter-domain services should retain their file descriptors (which they have
1640 shared with the OEM and SDK services) for the data connection, so that if the
1641 kill switch ([Disabling the CE domain](#)) is enabled, they can call shutdown() on
1642 the data connection to forcibly close it.

1643 The inter-domain services must reserve all well-known names starting
1644 with `org.apertis.InterDomain` (for example, `org.apertis.InterDomain1` or
1645 `org.apertis.InterDomain1.DataConnections`), and similarly all D-Bus interface
1646 names. This means they must not allow these names to be used as part of the
1647 OEM API shared between the export and adapter layers ([Interaction of the](#)
1648 [export and adapter layers](#)).

1649 A data connection cannot exist without an associated control connection

1650 (though one control connection may be associated with many data connec-
1651 tions). As data connections are opened and controlled through APIs defined on
1652 the inter-domain services, there is no need for standard network-style service
1653 discovery using protocols like [DNS-SD](#)²⁴ or [SSDP](#)²⁵.

1654 **Time synchronization**

1655 As a distributed system, the inter-domain services may require a shared clock
1656 across the domains. Time synchronization is critical to correlate events and this
1657 is specially important when playing audio and video streams, for example. If
1658 those streams are decoded on the CE and needs to played by the AD, the AD
1659 and the CE should agree on the meaning of the timestamps embedded in the
1660 streams.

1661 For the synchronization, there are two suitable protocols:

- 1662 • [NTP](#)²⁶ is a well-known protocol to synchronise time among remote sys-
1663 tems. It provides millisecond or sub-millisecond accuracy over the Internet
1664 or local area networks respectively;
- 1665 • [PTP](#)²⁷ provides microsecond or sub-microsecond accuracy and is designed
1666 for local area networks.

1667 In terms of latency calculation, both protocols satisfy the requirements, but we
1668 recommends PTP for the following reasons:

- 1669 • NTP uses hierarchical time sources, whereas PTP has a simpler mas-
1670 ter/slave model. That means any system that is even untrusted domain
1671 in a network is able to be taken by the other CE domain as a NTP source;
- 1672 • PTP supports hardware assisted timestamps to improve accuracy. Un-
1673 der Linux, the PTP hardware clock (PHC) subsystem is used to produce
1674 timestamps on supported network devices.

1675 **Audio streams**

1676 To share audio streams [RTP](#)²⁸ and its companion protocol [RTCP](#)²⁹ are recom-
1677 mended both on networked and container-based setups, for encoded and decoded
1678 streams.

1679 They provide jitter compensation, out-of-sequence handling and synchronization
1680 across multiple different streams.

²⁴https://en.wikipedia.org/wiki/Zero-configuration_networking#DNS-SD

²⁵https://en.wikipedia.org/wiki/Simple_Service_Discovery_Protocol

²⁶https://en.wikipedia.org/wiki/Network_Time_Protocol

²⁷https://en.wikipedia.org/wiki/Precision_Time_Protocol

²⁸https://en.wikipedia.org/wiki/Real-time_Transport_Protocol

²⁹https://en.wikipedia.org/wiki/RTP_Control_Protocol

1681 In particular [multiplexed RTP/RCTP][Appendix: Multiplexing RTP and
1682 RTCP] can be used to multiplex both protocols over the kind of data
1683 connections described above.

1684 **Decoded video streams**

1685 A fully decoded video stream consumes large quantities of bandwidth and shar-
1686 ing it between domains using the same approach used by audio (RTP) can
1687 only work for very small resolutions (see [Memory bandwidth usage on the i.MX6](#)
1688 [Sabrelite](#) for the bandwidth limitations on one of the platforms targeted by
1689 Apertis).

1690 If a domain sends uncompressed 1080p video stream at 25fps in YUV422 for-
1691 mat to another domain it requires just a bit more than 100MB/s for just the
1692 stream transfer. This already makes it prohibitive on Gigabit Ethernet systems,
1693 which have a theoretical maximum bandwidth of 125MB/s, without including any
1694 framing overhead. Even for local transfers this is a significant portion of the
1695 total memory bandwidth, even more so if taking in account other activities in-
1696 cluding the actual decoding and playback, plus the need for the same memory
1697 bandwidth toward the GPU where the decoded frames need to be composed.

1698 To be able to handle 1080p video streams it is very important that zero-copy
1699 mechanisms are used for the transfer of frames, see [Appendix: Audio and video](#)
1700 [decoding](#) for further considerations about how a protocol can be defined to
1701 match such expectations.

1702 **Bulk data transfers**

1703 Data connections are suitable for transfers that involve large amounts of static
1704 contents such as firmware images.

1705 To avoid storing multiple copies of the same data on the limited local storage,
1706 for instance in cases where the contents are downloaded from the Internet from
1707 a lower-privilege domain before being handed over to a more isolated higher-
1708 privilege domain, validation of the data such as checksum verification should be
1709 done on the fly by the originator, and only the recipient should store the data
1710 on its local storage.

1711 Raw direct TCP connections over IPSec or raw UDP sockets can be suitable for
1712 the inter-domain data transfer, as they both provide reliability, integrity and
1713 confidentiality. The downside of this approach is that each application would
1714 need to handle data validation and resumable transfers on its own: for this
1715 reason it is preferable to handle basic data validation in the inter-domain com-
1716 munication layers and provide the data to the receiver only once it is complete
1717 and matches the specified cryptographic hashes.

1718 The basic API thus is aimed at senders downloading large contents from the
1719 Internet and directly streaming across the domains without storing them locally,
1720 doing on-the-fly cryptographic validation of the streamed data. The contents

1721 are received and re-validated on the destination domain, where they are stored
1722 in a file which is passed to the destination service once the transfer is complete
1723 and valid.

1724 When the destination service has received the file handle it must perform any
1725 additional verification of the contents. It can also link the anonymous file de-
1726 scriptor to a locally-accessible file path using the `linkat()`³⁰ syscall with the
1727 `AT_EMPTY_PATH` flag or use the `copy_file_range()`³¹ syscall to get a copy of the
1728 contents in the most efficient way that the kernel can provide.

1729 A different mechanism can be defined where the sender stores the contents in
1730 a private file and passes a file descriptor pointing to it to the inter-domain
1731 communication subsystem. The receiving side then uses the `copy_file_range()`
1732 syscall to get a copy of the data that cannot be altered by the sender and then
1733 validates the data. On filesystems that supports reflinks, `copy_file_range()` will
1734 automatically use them to provide fast copy-on-write clones of the original file:
1735 this would make the operation nearly-instantaneous regardless of the amount of
1736 data, and would avoid doubling the storage requirements. When reflinks can-
1737 not be used, `copy_file_range()` will do an in-kernel copy, avoiding unnecessary
1738 context-switches over normal user-space copy operations. Such approach can
1739 be used on container-based setups or when a cluster file system is shared across
1740 networked domains. Not many filesystems can handle reflinks, but Btrfs and
1741 the OCFS2 cluster filesystem support them.

1742 On systems set up such that reflinks can be used, this solution is much more
1743 efficient than the alternatives, but imposes constraints on the whole system
1744 that may not be acceptable, such as requiring filesystems that support reflinks
1745 (such as Btrfs or OCFS2) on all the domains and ensuring that the appropriate
1746 shared filesystem mounts are available to SDK services. For this reason, the
1747 socket-based approach is recommended in the general case.

1748 Data connections API

1749 This section defines the draft for a proposed D-Bus API that SDK services could
1750 use to request the creation of data channels separated from the control plane
1751 connection.

1752 The gateway and adapter layers are responsible for the creation and initialization
1753 of those channels, while other services and applications must not be able to
1754 directly create them.

1755 The gateway and adapter layers use instead file descriptors passing to share the
1756 channel endpoints with the requesting services and applications.

1757 The API drafted here is meant to only provide a very rough guideline for those
1758 implementing any real data channel API and it's not meant to be normative:
1759 real implementations can diverge from the interfaces described here and the

³⁰<https://manpages.debian.org/stretch/manpages-dev/link.2.en.html>

³¹https://manpages.debian.org/stretch/manpages-dev/copy_file_range.2.en.html

1760 actual API to be used by SDK services must be documented in a separate
1761 specification.

```

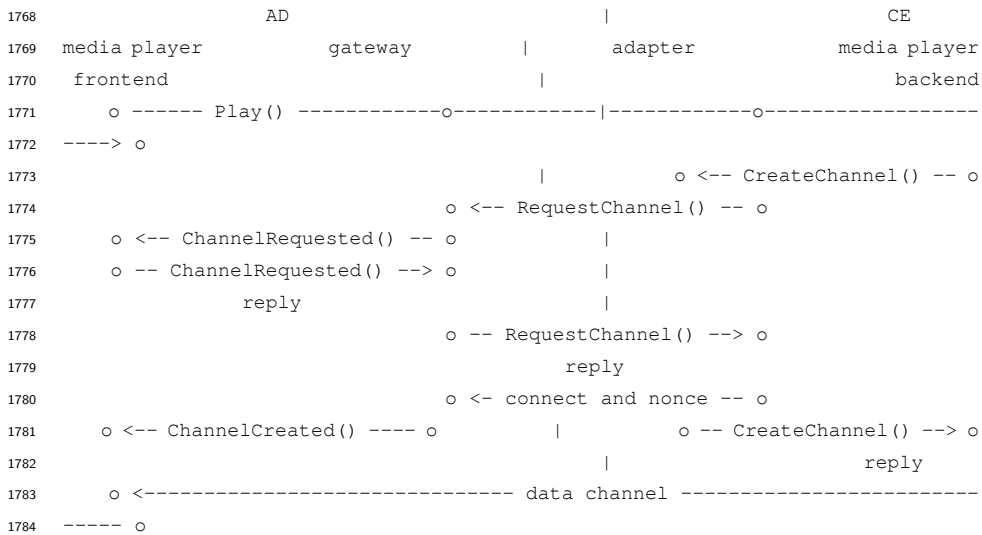
1  /* The interface exported by the adapter/gateway to SDK services to initiate channel creation. */
2  interface org.apertis.InterDomain.DataConnection1 {
3      /* @id: the app-specific unique token used to to identify and authorize the channel
4       * @destination: the bus name of the service which should be at the other end of the channel
5       * @type: the kind of data and the protocol to be used for the data exchange.
6       *
7       * Use 'audio-rtp' for multiplexed RTP/RFC5761.
8       * @metadata_in: a dictionary of extra information that can be used to authorize/validate the transfer
9       * @metadata_out: the @metadata_in dictionary with additional information
10      * @fd: the file descriptor for the actual data exchange using the protocol specified by @type */
11      method CreateChannel (in s id,
12                          in s destination,
13                          in s type,
14                          in a{sv} metadata_in,
15                          out a{sv} metadata_out,
16                          out h fd)
17
18      /* @id: see org.apertis.InterDomain.DataConnection1.CreateChannel()
19       *
20       * If the receiver was not able to validate the channel, the `org.apertis.InterDomain.ChannelError`
21       * error is raised. */
22      method CommitChannel(in s id)
23
24      /* @id: see org.apertis.InterDomain.DataConnection1.CreateChannel() */
25      method AbortChannel(in s id)
26
27      /* @refclk: the reference to the IDC shared clock, in the format of defined
28       * by the `clksrc` production of RFC7273 for the `ts-
29       refclk:` parameter */
30      method GetClockReference(out s refclk)
31  }
32
33  /* The interface to be exported by services that can handle incoming channels.
34   * Domains that do not use a local dbus-
35   daemon can implement a similar mechanism
36   * with the native IPC system. */
37  interface org.apertis.InterDomain.DataConnectionClient1 {
38      /* @id: see org.apertis.InterDomain.DataConnection1.CreateChannel()
39       * @sender: the bus name of the service which initiated the channel creation
40       * @type, @metadata_in, @metadata_out: see org.apertis.InterDomain.DataConnection1.CreateChannel()
41       * @proceed: true if the channel should be set up, false if it should be refused */
42      method ChannelRequested(in s id,
43                            in s sender,
44                            in s type,
45                            in a{sv} metadata_in,
46                            out a{sv} metadata_out,
47                            out b proceed)
48
49      /* @id: see org.apertis.InterDomain.DataConnection1.CreateChannel()
50       * @success: whether the connection has been successfully set up and @fd is usable
51       * @fd: the file descriptor from which to read the incoming data with the
52       previously agreed protocol
53      method ChannelCreated(in s id,
54                          in b success,
55                          in h fd)
56  }

```

1762 **Data channel API flow example for a media player streaming audio**

1763

1764 A possible use-case of the API is a Media Player frontend hosted on the AD
 1765 with the backend on the CE. The frontend requests the backend to decode a
 1766 specific stream using an application specific API and passing a token with the
 1767 request.



1785 The Media Player frontend initially calls the application-specific `Play()` method
 1786 on its backend, with the IDC system transparently proxying the request across
 1787 domains. This call must also carry an application-specific token that will be
 1788 used to identify the request during the channel creation procedure.

1789 Once the Media Player backend has gathered some metadata about the stream
 1790 to be played, it requests the creation of an `audio-rtp` channel directed to the Me-
 1791 dia Player frontend by calling the `org.apertis.InterDomain.DataConnection1.CreateChannel()`
 1792 on the local adapter service.

1793 The adapter service will then access the inter-domain link by calling the
 1794 `org.apertis.InterDomain.DataConnectionInternal1.RequestChannel()` method of
 1795 the remote gateway peer.

1796 The gateway service on the AD notifies the Media Player frontend that a channel
 1797 has been requested, passing the request token and other application-specific
 1798 metadata. If the token matches and the metadata is acceptable, the Media
 1799 Player frontend replies to the gateway service telling it to proceed.

1800 Once the request has been accepted by the destination, the gateway service
 1801 creates a listening socket for the requested channel type and returns the infor-

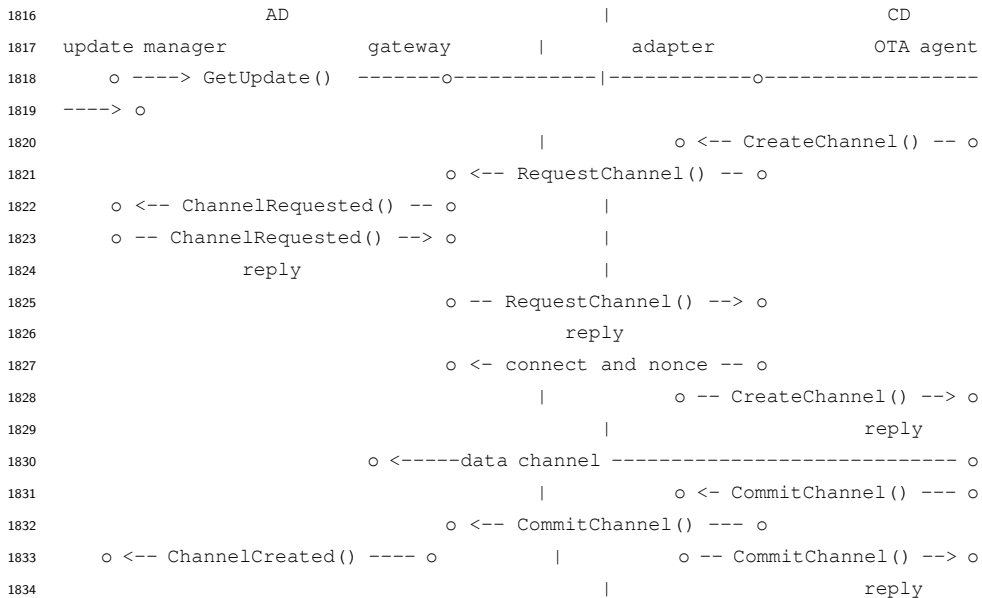
1802 mation needed to connect to it to the remote adapter peer, including a nonce
 1803 to authenticate the connection.

1804 As soon as the adapter gets the socket information it connects to it and sends
 1805 the nonce over it. On the other side the gateway will read the nonce and if does
 1806 not matches it immediately closes the connection.

1807 Once the connection has been set up and the nonce has been successfully shared,
 1808 the adapter and gateway services will hand over the file descriptors of the sockets
 1809 that have been set up.

1810 **Data channel API flow example for an update manager sharing**
 1811 **firmware images**

1812 The bulk data transfer API is meant to be useful for update managers where
 1813 an agent in the Connectivity Domain fetches firmware images from the Internet
 1814 and shares them with the update manager in the AD which has access to the
 1815 devices to be updated.



1835 The update manager calls the `GetUpdate()` method of the agent, with a to-
 1836 ken identifying the request. The OTA agent retrieves the metadata of the
 1837 file to be shared, in particular the size and a set of cryptographic hashes.
 1838 With that information, it requests the creation of a `bulk-data` channel with
 1839 the `org.apertis.InterDomain.DataConnection1.CreateChannel()` method of the lo-
 1840 cal adapter service. The OTA agent must specify the `size` parameter and a
 1841 known cryptographic hash such as `sha512` in the `metadata_in` parameter. It must
 1842 then check in the `metadata_out` for the `offset` parameter to figure out if it must
 1843 resume an interrupted download.

1844 The adapter service accesses the inter-domain link by calling the `org.apertis.InterDomain.DataConnectionInternal`
1845 method of the remote gateway peer.

1846 The flow is analogous to the one in the [streaming media player case][Data
1847 channel API flow example for a media player streaming audio] until the point
1848 where the inter-domain socket is created: while the receiving end of the socket
1849 in the streaming case is meant to be used by the receiving service, in the bulk
1850 data case it is used directly by the gateway, which stores the received data in a
1851 local file.

1852 While it sends data through the socket, the OTA agent is expected to perform
1853 on-the-fly data validation by computing cryptographic hashes on the streamed
1854 contents: once it has sent all the data the agent can close the socket and call
1855 `org.apertis.InterDomain.DataConnectionInternal1.CommitChannel()` to signal that
1856 all the data has been shared successfully and that the computed hashes match,
1857 or `AbortChannel()` otherwise.

1858 Upon receiving the `CommitChannel()` message, the gateway checks that the file size
1859 and cryptographic hashes match the expected values and raises the `ChannelError`
1860 error otherwise. If and only if the data is valid it instead shares the file descriptor
1861 pointing to the file to the OTA updater with a `ChannelCreated()` call.

1862 Traffic control

1863 Traffic control³² should be set by the inter-domain service (**Protocol library**
1864 **and inter-domain services**) in the CE domain, using the standard Linux traffic
1865 control functionality in the **kernel**³³. As the control connection and each data
1866 connection are separate TCP or UDP connections, they can have traffic controls
1867 applied to them individually, which allows different quality of service settings for
1868 individual data connections; and allows the control connection to have a higher
1869 quality of service than all data connections, to help ensure it has guaranteed
1870 low latency.

1871 Applying traffic control in the CE domain has the advantage of knowing what
1872 kernel functionality is available — if it were applied in the automotive domain,
1873 its functionality would be limited by whatever is provided by the automotive
1874 OS (for example, QNX). It has the disadvantage, however, of being vulnerable
1875 to the CE domain being compromised: if an attacker gains control of the inter-
1876 domain service in the CE domain, they can disable traffic control. However, if
1877 they have gained control of that service, the only remaining mitigation is for the
1878 automotive domain to shut down the CE domain, so having control over traffic
1879 policy has little effect.

1880 The specific traffic control policies used by the inter-domain service can be
1881 determined later, based on the relative priorities an OEM assigns to different
1882 types of traffic.

³²https://en.wikipedia.org/wiki/Network_traffic_control

³³<http://tldp.org/HOWTO/Traffic-Control-HOWTO/intro.html>

1883 Protocol library and inter-domain services

1884 The inter-domain communications protocol should be implemented as a library,
1885 containing all layers of the protocol. The particular domain configuration which
1886 the library targets should be a configure-time option, though the library must
1887 support enabling the [Standalone setup](#) transport in conjunction with another
1888 transport, when in developer mode (see [Mock SDK implementation](#)).

1889 By implementing the protocol as a library, it can be tested easily by being
1890 linked into unit tests — rather than trying to wrap the entire inter-domain
1891 service daemon in a test harness. Internally, the library should implement all
1892 protocol layers separately and expose them to the unit tests so that they can
1893 be tested individually.

1894 Furthermore, this allows the protocol code to be reused between the inter-
1895 domain service in the automotive domain, and the inter-domain service in the
1896 CE domain.

1897 The main advantage of implementing the protocol as a library is the flexibility
1898 this provides for integrating it into different automotive domain implementations
1899 — it can be integrated into an existing system service (bearing in mind the
1900 suggestion to keep it in a separate trust domain, [Security domains](#)), or could be
1901 used as a stand-alone service daemon.

1902 A reference implementation of such a stand-alone inter-domain service program
1903 should be provided with the protocol library. This should provide the necessary
1904 systemd service file and AppArmor profile to allow itself to be strictly confined
1905 if the automotive domain OS supports this.

1906 As the inter-domain communications protocol uses D-Bus, the protocol library
1907 must contain an implementation of the D-Bus protocol. Note that this is *not*
1908 a D-Bus daemon; it is a D-Bus library, like libdbus or GDBus. See [Appendix:
1909 D-Bus components and licensing](#) for details about the different components in
1910 D-Bus and their licensing.

1911 Apart from its D-Bus library dependency, the protocol library should be de-
1912 signed with minimal dependencies in order to be easily integratable into a va-
1913 riety of automotive domain operating systems (from Linux through to other
1914 Unixes, QNX or Autosar). If the chosen D-Bus library is available as part of
1915 the automotive OS (which is more likely for libdbus than for other D-Bus li-
1916 braries), it could be linked against; otherwise, it could be statically linked into
1917 the protocol library.

1918 libdbus itself is already quite portable, having been known to work on Linux,
1919 Windows, OS X, NetBSD and QNX. It should not be difficult to port to other
1920 POSIX-compliant operating systems.

1921 [Rate limiting on control messages](#) should be implemented in the protocol li-
1922 brary, so that the same functionality is present in both the automotive and CE
1923 domains.

1924 The protocol library should expose the encryption keys for the IPsec connection
1925 used in the inter-domain communications link, including signals for when those
1926 keys change (due to cookie renegotiation on the link). The keys must only be
1927 exposed in development builds of the protocol library. See [Debuggability](#) for
1928 more details.

1929 **Non Linux-based domains**

1930 The suggested implementation uses D-Bus the *protocol*, not necessarily dbus-
1931 daemon the *message bus daemon* or libdbus the *protocol library*.

1932 This means that for inter-domain communications purposes, only the serial-
1933 ization format of D-Bus is used as a well defined RPC protocol. There's no
1934 requirement that domains run `dbus-daemon` or that they use a specific D-Bus
1935 implementation to talk to other domains.

1936 Several implementations of the D-Bus serialization format exists and their use
1937 is strongly encouraged rather than reimplementing the protocol from scratch:

- 1938 • [GDBus](#)³⁴ is a GTK+/GNOME oriented implementation of the D-Bus pro-
1939 tocol in GLib
- 1940 • [QtDBus](#)³⁵ is Qt module that implements the D-Bus protocol
- 1941 • [node-dbus](#)³⁶ is a D-Bus protocol implementation for NodeJS written in
1942 pure JavaScript
- 1943 • [libdbus](#)³⁷ is the reference implementation of the D-Bus protocol
- 1944 • [dbus-sharp](#)³⁸ is a C#/.net/Mono implementation of the D-Bus protocol
- 1945 • [pydbus](#)³⁹ is a python implementation of the D-Bus protocol

1946 On networked setups the D-Bus-based protocol is transported over TCP, relying
1947 on IPSec for authentication, confidentiality and reliability.

1948 If IPSec nor TLS are available, those properties cannot be guaranteed, and thus
1949 such setup is strongly discouraged. In that case every input should be treated
1950 as potentially malicious: the trusted domains must export only a very reduced
1951 set of interfaces, which must be conceived in a way that any kind of misuse does
1952 not lead to harm.

1953 **Service discovery**

1954 Accordingly to the use of the D-Bus serialization protocol, each service
1955 exported over the inter-domain communication channels is identified by

³⁴<https://developer.gnome.org/gio/stable/gdbus.html>

³⁵<http://doc.qt.io/qt-5/qtdbus-index.html>

³⁶<https://github.com/sidorares/node-dbus>

³⁷<https://dbus.freedesktop.org/doc/api/html/>

³⁸<https://github.com/mono/dbus-sharp>

³⁹<https://github.com/LEW21/pydbus>

1956 a well-known name subject [specific constraints](#)⁴⁰, starting with the re-
1957 versed DNS domain name of the author of the service (for instance,
1958 `com.collabora.CarOS.ClimateControl1` for a potential service written by
1959 [Collabora](#)⁴¹.

1960 Only one service at a time can own such names on each domain, but the owner-
1961 ship is not tracked across domains and collision may happen due to a transitional
1962 state during an upgrade or other causes: each setup is thus responsible to define
1963 a deterministic collision resolution procedure should two domains export the
1964 same service name.

1965 The adapter layer is responsible to inspect on which channel each service is
1966 available. The `NameOwnerChanged` signal⁴² must be used by the adapter layer to
1967 track the availability of services on each connection and to detect when a service
1968 is no longer available or changed ownership (for example because it has been
1969 restarted). The `org.freedesktop.DBus.ListActivatableNames()`⁴³ message can be
1970 used to gather the initial list of available services.

1971 After an upgrade a domain may stop providing a specific service and
1972 another domain may start providing it instead: both the old and new
1973 domains must trigger the `NameOwnerChanged` signal⁴⁴ in response to the
1974 `org.freedesktop.DBus.ReleaseName()`⁴⁵ and `org.freedesktop.DBus.RequestName()`⁴⁶
1975 calls. No specific ordering is required and thus the service may be temporarily
1976 unavailable or the two domains may export the same service name at the same
1977 time: the collision resolution procedure must choose the one on the connection
1978 with the highest priority.

1979 In the simplest case, each domain must be given an unique priority with the
1980 AD having the highest priority. The relative priority between the CE domains
1981 is used to provide deterministic service access when a service name exists on
1982 multiple connections. As a result, the priority list must be static and the priority
1983 of CE domains can be assigned arbitrarily for each specific setup.

1984 When accessing a service name that exists on more than one connection, the
1985 service that exists on the connection with the highest priority must be given
1986 precedence by the adapter layer.

1987 CE domains should not be able to spoof trusted services exported by the AD:
1988 for this reason a static list of services meant to be exported only by the AD
1989 must be defined and the adapter layer must ignore matching services exported

⁴⁰<https://dbus.freedesktop.org/doc/dbus-specification.html#message-protocol-names-bus>

⁴¹<https://collabora.com>

⁴²<https://dbus.freedesktop.org/doc/dbus-specification.html#bus-messages-name-owner-changed>

⁴³<https://dbus.freedesktop.org/doc/dbus-specification.html#bus-messages-list-activatable-names>

⁴⁴<https://dbus.freedesktop.org/doc/dbus-specification.html#bus-messages-name-owner-changed>

⁴⁵<https://dbus.freedesktop.org/doc/dbus-specification.html#bus-messages-release-name>

⁴⁶<https://dbus.freedesktop.org/doc/dbus-specification.html#bus-messages-request-name>

1990 by other connections, even if the service is not currently available on the AD
1991 connection itself.

1992 Particular care must be taken to ensure each domain can be fully booted with-
1993 out blocking on services hosted on other domains, to avoid untracked circular
1994 dependencies.

1995 SDK services must access the above service names through the private bus
1996 instance exported by the adapter layer, which proxies them from all the inter-
1997 domain channels, abstracting the complexities of inter-domain communications.
1998 SDK services are not aware of the fact that the services are hosted on different
1999 domains.

2000 **Automotive domain export layer**

2001 To integrate the inter-domain communications system into an automotive do-
2002 main operating system, the APIs to be shared must be exported as objects on
2003 the D-Bus connection provided by the inter-domain service. This is done as an
2004 *export layer* in the inter-domain service in the automotive domain, customised
2005 for the OEM and their specific APIs. The export layer could be implemented
2006 as pure C calls from within the same process (no protocol at all), or D-Bus, or
2007 kdbus, or QNX message passing, or something else entirely. If D-Bus bus is
2008 used, a D-Bus daemon would need to be running on the automotive domain;
2009 otherwise, no D-Bus daemon would be needed.

2010 For example, if the automotive domain provides the APIs which are to be ex-
2011 posed over the inter-domain connection as:

- 2012 • C APIs in headers — the inter-domain service would call those APIs di-
2013 rectly, and the export layer would essentially be those C calls;
- 2014 • daemons with UNIX socket connections — the inter-domain service would
2015 connect to those sockets and run whatever protocol is specified by the
2016 daemons, and the export layer would essentially be the socket connections
2017 and protocol implementations;
- 2018 • D-Bus services — the inter-domain service would connect to a D-Bus dae-
2019 mon on the automotive domain and translate the services' D-Bus APIs
2020 into an API to expose on the inter-domain communications link (see be-
2021 low), and the export layer would be the D-Bus daemon, D-Bus library in
2022 the inter-domain service, and the code to translate between the two D-Bus
2023 APIs.

2024 The APIs must be exported under [well-known names](#)⁴⁷ formatted as reverse-
2025 DNS names owned by the OEM. For example, if the AD operating system
2026 was written by Collabora, APIs would be exported using well-known names
2027 starting with com.collabora, such as com.collabora.CarOS.EngineManagement1
2028 or com.collabora.CarOS.ClimateControl1.

⁴⁷<http://dbus.freedesktop.org/doc/dbus-specification.html#message-protocol-names-bus>

2029 The API formed by these exported D-Bus objects is vendor-specific, but should
2030 maintain its own stability guarantees — for every backwards-incompatible
2031 change to this API, there must be a corresponding update to the CE domain
2032 to handle it. Consequently, we recommend [versioning the exported D-Bus](#)
2033 [APIs](#)⁴⁸.

2034 APIs which the OEM does not want to make available on the inter-domain
2035 communications link (for example, because they are not able to handle untrusted
2036 data, or are too powerful to expose) must not be exported onto the D-Bus
2037 connection. This effectively forms a whitelist of exposed services.

2038 For each piece of functionality exposed by the AD, suitable safety limits must be
2039 applied ([Safety limits on AD APIs](#)). If the implementation of that functionality
2040 already applies the safety limits, nothing more needs to be done. Otherwise,
2041 the safety limits must be enforced in the interface code which exports that
2042 functionality onto the inter-domain D-Bus connection.

2043 Similarly, for each piece of functionality exposed by the AD, if it fails to respond
2044 to a call by the inter-domain service, the service must return an error to the
2045 CE over the inter-domain D-Bus connection, rather than timing out. This is
2046 especially important in systems where the export layer is a set of C calls —
2047 the implementation must take care to ensure those calls cannot block the inter-
2048 domain service.

2049 If the vendor wants to implement per-API kill switches for services exported
2050 by the automotive domain, these must be implemented in the export layer (see
2051 [Disabling the CE domain](#)).

2052 **Consumer-electronics domain adapter layer**

2053 Paired with the OEM-specific API export code in the automotive domain is an
2054 *adapter layer* in the CE domain. This adapts the API exported by the services
2055 on the automotive domain to the stable SDK APIs used by applications in the
2056 CE domain. The layer has an implementation in each of the SDK services in
2057 the CE domain.

2058 This adapter layer does not have a trust boundary — each part of it lies within
2059 the trust domain of the relevant SDK service.

2060 These adapters connect to a private D-Bus bus, which the inter-domain service
2061 in the CE domain is also connected to. The inter-domain service exports the
2062 OEM APIs from the automotive domain on this bus, and the adapters consume
2063 them.

2064 The private bus could be implemented either by running `dbus-daemon` with a
2065 custom bus configuration, or by implementing it directly in the inter-domain
2066 service, and having all adapters connect directly to the service. In both cases,

⁴⁸<http://dbus.freedesktop.org/doc/dbus-api-design.html#api-versioning>

2067 the trust boundary between the adapters (within the trust domains of the SDK
2068 services) and the inter-domain service are enforced.

2069 **Interaction of the export and adapter layers**

2070 The interaction between the export and adapter layers is important in main-
2071 taining compatibility between different versions of the AD and CE as they are
2072 upgraded separately. The CE is typically upgraded much more frequently than
2073 the AD. Both are customised to the OEM.

2074 **Initial deployment**

2075 The OEM develops both layers, and stabilises an initial version of their inter-
2076 domain API, using a version number (for example, 1). The export layer exports
2077 objects from the automotive domain, and the adapter layer imports those same
2078 objects. There may be functionality exposed on the objects which the SDK
2079 APIs currently do not support — in which case, the adapter layer ignores that
2080 functionality.

2081 **CE is upgraded, AD remains unchanged**

2082 A new release of Apertis is made, which expands the SDK APIs to support
2083 more functionality. The OEM integrates this release of Apertis and updates
2084 their adapter layer to tie the new SDK APIs to previously-unused objects from
2085 the inter-domain link.

2086 The version number of the inter-domain API remains at 1.

2087 **AD is upgraded, CE remains unchanged**

2088 The automotive domain OS is upgraded, and more vehicle functionality becomes
2089 available to expose on the inter-domain connection. The OEM chooses to expose
2090 most of this functionality using the inter-domain service. For some objects, this
2091 results in no API changes. For other objects, it results in new methods being
2092 added, but no old ones are changed. For some objects, it results in some old
2093 methods being removed or their semantics changed. For these objects, the
2094 OEM now exports *two* interfaces on the inter-domain service: one at version
2095 1, exporting the old API; and one at version 2, exporting the new API. The
2096 version number of other inter-domain APIs remains at 1.

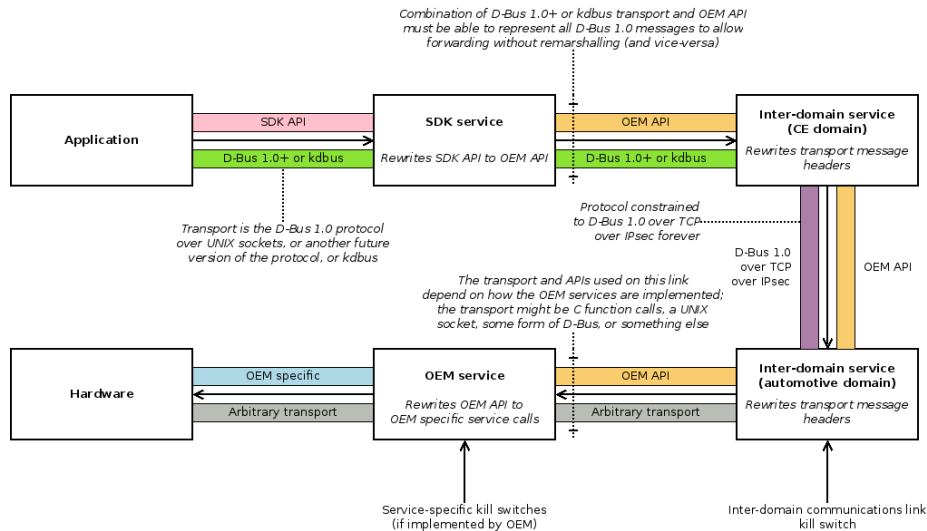
2097 The CE domain software remains unchanged, which means it continues to use
2098 the version 1 APIs. This continues to work because all objects on the inter-
2099 domain API continue to export version 1 APIs (in addition to some version 2
2100 APIs).

2101 **CE is upgraded again**

2102 The next time the CE domain is upgraded, its adapter layer can be modified by
 2103 the OEM to use the new version 2 APIs for some of the services. If this updated
 2104 version of the CE domain is guaranteed to only be used with new versions of
 2105 the AD, the adapter layer can drop support for version 1 APIs. If the updated
 2106 CE domain may be used with old versions of the AD, it must support version 1
 2107 and version 2 (or just version 1) APIs, and use whichever it prefers.

2108 **Flow for a given SDK API call**

2109 In the following figure, particular attention should be paid to the restrictions on
 2110 the protocols in use for each link. For the links between the application and the
 2111 inter-domain service in the CE domain, any version of the D-Bus protocol can be
 2112 used, including kdbus or another future version. This depends only on the dbus-
 2113 daemon and D-Bus libraries available in the CE domain. For the link between
 2114 the two inter-domain services, the protocol must always be at least D-Bus 1.0
 2115 over TCP over IPsec. If both peers support a later version of the protocol,
 2116 they may use it — but both must always support D-Bus 1.0 over TCP over
 2117 IPsec. For the link between the inter-domain service in the automotive domain
 2118 and the OEM service, whatever protocol the OEM finds most appropriate for
 2119 implementing their export layer should be used. This could be pure C calls
 2120 from within the same process (no protocol at all), or D-Bus, or kdbus, or QNX
 2121 message passing, or something else entirely.



2122

2123 Apertis IDC message flow, following a message being sent from ap-
 2124 plication to hardware; the message flow is the same in reverse for
 2125 message replies from the hardware

2126 **Trusted path to the AD**

2127 Providing a trusted input and output path between the user and the automo-
2128 tive domain is out of scope for this design — it is a problem to be solved by
2129 the graphics sharing and input handling designs. However, it is worth noting
2130 that the solution must not involve communication (unauthenticated, or authen-
2131 ticated via the CE domain) over the inter-domain link. If it did, a compromised
2132 CE domain could be used to forge this communication and gain control of the
2133 trusted path to the AD — which likely results in a large privilege escalation.

2134 A trusted path should be implemented by direct communication between the
2135 input and output devices and the automotive domain, or mediating such com-
2136 munication through the hypervisor, which is trusted.

2137 **Developer mode**

2138 In order to support connecting the CE domain from an SDK on a developer’s
2139 laptop to the automotive domain in a development vehicle, the ‘separate boards
2140 setup with other devices’ configuration must be used, with the CE domain and
2141 the automotive domain connected to the developer’s network (which might have
2142 other devices on it).

2143 In order to allow the SDK to connect, the vehicle must be in a ‘developer mode’.
2144 This is because the CE domain is entirely untrusted when it is provided by the
2145 SDK, because the developer may choose to disable security features in it (indeed,
2146 they may be working on those security features).

2147 **Open question:** What cryptography should be used to implement this authen-
2148 tication, and the division of trust between development and production devices?
2149 A likely solution is to only have the AD accept the CE connection if it connects
2150 with a ‘production’ key signed by the vehicle OEM.

2151 **Mock SDK implementation**

2152 In order to allow applications to be developed against the Apertis SDK, imple-
2153 mentations of all the SDK APIs need to be provided as part of the official SDK
2154 virtual machine distribution. These implementations need to be fully featured,
2155 otherwise application developers cannot develop against the unimplemented fea-
2156 tures.

2157 There are two implementation options:

- 2158 1. Have an Apertis SDK adapter layer which provides the mock implemen-
2159 tations, and which does not use an inter-domain service or mock up any
2160 of the automotive domain.
- 2161 2. Write the mock implementations as stand-alone services which are logi-
2162 cally part of the automotive domain (even though there is no domain
2163 separation in the SDK). Expose these services on the inter-domain link
2164 using an Apertis SDK export layer; and adapt the services to the actual

2165 SDK APIs using an Apertis SDK adapter layer.
2166 The inter-domain services would be running in the same domain (the
2167 SDK) and would communicate over a loopback TCP socket (see [Stan-](#)
2168 [dalone setup](#)).

2169 Option #1 has a much simpler implementation, but option #2 means that the
2170 inter-domain communications code paths are tested by all application develop-
2171 ers. Similarly, option #1 introduces the possibility for behavioural differences
2172 between the mock adapter layer and the production inter-domain communica-
2173 tion system, which could affect how application developers write their applica-
2174 tions; option #2 reduces the potential for that considerably.

2175 As option #2 uses the inter-domain service in the CE domain, it also allows for
2176 the possibility of connecting the CE domain to a different automotive domain
2177 — rather than the mock one provided by the SDK, a developer could connect
2178 to the automotive domain in a development vehicle ([Developer mode](#)).

2179 Hence, our recommendation is for option #2.

2180 Debuggability

2181 The debuggability of the inter-domain communications link is important for
2182 many reasons, from integrating two domains to bringing up a new automotive
2183 domain (with its export and adapter layers) to developing a new SDK API.

2184 Referring to the figure in [Overall architecture](#), debugging of:

- 2185 • *applications and the SDK services* happens using normal tools and meth-
2186 ods described in the [Debug and Logging design](#)⁴⁹;
- 2187 • *communications between the dbus-daemon (private bus) and inter-domain*
2188 *service (CE domain)* happens using normal D-Bus monitoring tools (such
2189 as [Bustle](#)⁵⁰ or [dbus-monitor](#)⁵¹), though this requires the developer to gain
2190 access to the private bus' socket;
- 2191 • *communications between the inter-domain services* happens using a special
2192 debug option in the services (see below);
- 2193 • *the export layer and OEM services* happens using tools and methods spe-
2194 cific to how the OEM has implemented the export layer.

2195 If possible, all debugging should happen on the SDK side, in the adapter layer or
2196 above, as this allows the greatest flexibility in debugging techniques — none of
2197 the communications at that level are encrypted, so are accessible to a developer
2198 user with the appropriate elevated permissions.

2199 If the connection between the inter-domain services (the TCP/IPsec link be-
2200 tween domains) needs to be debugged, it can be complex, as any debugging

⁴⁹<https://sjoerd.pages.apertis.org/apertis-website/concepts/debug-and-logging/>

⁵⁰<http://willthompson.co.uk/bustle/>

⁵¹<http://dbus.freedesktop.org/doc/dbus-monitor.1.html>

2201 tool needs to be able to decrypt the IPsec encryption. Wireshark is [able to do](#)
2202 [this](#)⁵², if given the encryption key in use by the IPsec connection. This key may
2203 change over the lifetime of a connection (as the connection cookie is refreshed),
2204 and hence needs to be exported dynamically by the inter-domain service. In
2205 order to allow debugging both ends of the connection, it should be implemented
2206 in the protocol library ([Protocol library and inter-domain services](#)). In the CE
2207 domain, it should be exposed as a D-Bus interface on the private bus which is
2208 part of the adapter layer. This limits its access to developers who have access
2209 to that bus.

```
1  Interface org.apertis.InterDomainConnection.Debug1 {
2      /* Mapping from IKEv1 initiator cookie to encryption key. */
3      readonly property a{ss} Ike1Keys;
4      /* Mapping from IKEv2 tuple of (initiator SPI, responder SPI) to tuple
5       * of (SK_ei, SK_er, encryption algorithm, SK_ai, SK_ar, integrity
6       * algorithm). Algorithms are enumerated types, with values to be
7       * documented by the implementation. Other parameters are provided as
8       * hexadecimal strings to allow for varying key lengths. */
9      readonly property a((ss)(ssssussu)) Ike2Keys;
10 }
```

2210 A [new Lua plugin](#)⁵³ in Wireshark could connect to this interface and listen for
2211 signals of updates to the connection's keys, and use those to update Wireshark's
2212 IKE decryption table. Wireshark is the suggested debugging tool to use, as it is
2213 a mature network analysis tool which is well suited to analysing the protocols
2214 being sent over the inter-domain connection.

2215 In the automotive domain, the key information provided by the protocol library
2216 should be exposed in a manner which best fits the debugging infrastructure and
2217 tools available for the automotive operating system.

2218 In both domains, this interface must only be exposed in developer builds of the
2219 inter-domain services. It must not be available in production, even to a user with
2220 elevated privileges. To expose it would allow all inter-domain communications
2221 to be decrypted.

2222 **External watchdog**

2223 There must be an external watchdog system which watches both the automotive
2224 and consumer-electronics domains, and which restarts either of them if they
2225 crash and fail to restart themselves.

2226 In order to prevent one compromised domain from preventing a restart of the

⁵²<https://ask.wireshark.org/questions/12019/how-can-i-decrypt-ikev1-and-or-esp-packets>

⁵³<https://ask.wireshark.org/questions/44562/update-decryption-table-from-lua>

2227 other domain (a denial of service attack), each domain must only be able to send
2228 heartbeats to its own watchdog, and not the watchdog of the other domain.

2229 The implementation of the watchdog depends on the configuration:

- 2230 • Standalone setup: No watchdog is necessary, as the configuration is not
2231 safety critical.
- 2232 • Basic virtualised setup: The watchdog should be a software component in
2233 the hypervisor, exposed as virtualised watchdog hardware in the guests.
- 2234 • Separate CPUs setup: A hardware watchdog on the board should be used,
2235 connected to both domains. As an exception to the general principle that
2236 the CE domain should not be allowed to access hardware, it must be able
2237 to access its own watchdog (and must not be able to access the automotive
2238 domain's watchdog).
- 2239 • Separate boards setup: A hardware watchdog on each board should be
2240 used, connected to the domain on that board.
- 2241 • Separate boards setup with other devices: Same as the separate boards
2242 setup.
- 2243 • Multiple CE domains setup: Same as the separate boards setup.

2244 **Tamper evidence and hardware encryption**

2245 The basic design for providing a root of confidentiality and integrity for the
2246 system in hardware should be provided by the Secure Boot design — this design
2247 can only assume that some confidential encryption key is provided which is used
2248 to decrypt parts of the system on boot which should remain confidential.

2249 As of February 2016 the Secure Boot design is still forthcoming

2250 One possibility for implementing this is for a confidential key store to be pro-
2251 vided by the automotive domain, storing keys which encrypt the bootloader
2252 and root key store for the CE. When booting the CE, the AD would decrypt
2253 its bootloader and hence its root key store, making the keys necessary for inter-
2254 domain communications (amongst others) available in the CE's memory. Note
2255 that this suggestion should be ignored if it conflicts with recommendations in
2256 the Secure Boot design, once that's published.

2257 A critical requirement of the system is that none of the keys for encrypting inter-
2258 domain communications (or for protecting those keys) can be shared between
2259 vehicles — they must be unique per vehicle (**No global keys in vehicles**). This
2260 implies that keys must be generated and embedded into each vehicle as a stage
2261 in the imaging process for the domains.

2262 A corollary to this is that none of those keys can be stored by the vendor,
2263 trusted dealer or other global organisations associated with the vehicles; as
2264 to do so would provide a single point of failure which, if compromised by an

2265 attacker, could reveal the keys for all vehicles and hence potentially allow them
2266 all to be compromised easily.

2267 Tamper evidence is an important requirement for the system (**Tamper evi-**
2268 **dence**), providing the ability to determine if a vehicle has been tampered with
2269 in case of an accident or liability claim.

2270 The most appropriate way to provide tamper evidence for the hardware depends
2271 on the hardware and how it is packaged in the vehicle. Typical approaches to
2272 tamper evidence involve sealing the domain's circuitry, including all access and
2273 I/O ports, in a casing which is sealed with tamper evident **seals**⁵⁴. If a garage or
2274 trusted vehicle dealer needs to access the domain for maintenance or updates,
2275 they must break the seals, enter this in the vehicle's maintenance log, and replace
2276 the seals with new ones once maintenance is complete.

2277 Tamper evidence for software should be provided through the integrity proper-
2278 ties of the Secure Boot design, as in any **trusted platform module**⁵⁵ system.

2279 **Disabling the CE domain**

2280 The automotive domain must be able to disable the power supply to the CE
2281 domain (or otherwise prevent it from booting), and must be able to prevent
2282 inter-domain communications at the same time.

2283 Preventing inter-domain communications should be implemented by having the
2284 automotive domain inter-domain service read a 'kill switch' setting. If this is
2285 set, it should close any open inter-domain communication links, and refuse to
2286 accept new ones while the setting is still set.

2287 Preventing the CE domain from booting can be done in a variety of ways,
2288 depending on the hardware functionality available. For example, it could be
2289 done by controlling a solid-state relay on the CE domain's power supply. Or,
2290 if the CE domain implements secure boot, the boot process could require the
2291 automotive domain to decrypt part of the CE domain bootloader using a key
2292 known only to the automotive domain — if the kill switch is set, this key would
2293 be unavailable.

2294 **Open question:** What hardware provisions are available for controlling the
2295 power supply or boot process of the CE domain? How should this integrate
2296 with the secure boot design?

2297 The kill switch is intentionally kept simple, controlling whether *all* inter-domain
2298 communications are enabled or disabled, and providing no finer granularity.
2299 This is intended to make it completely robust — if support were added for
2300 selectively killing some of the control APIs or data connections on the inter-
2301 domain communications link, but not others, there would be much greater scope
2302 for bugs in the kill switch which could be exploited to circumvent it.

⁵⁴https://en.wikipedia.org/wiki/Security_seal

⁵⁵https://en.wikipedia.org/wiki/Trusted_Platform_Module

2303 If the OEM wants to provide finer grained kill switches for different APIs in
2304 the automotive domain, they must implement them as part of those services,
2305 or as part of the export layer which connects those services to the inter-domain
2306 service.

2307 **Reporting malicious applications**

2308 There are three options for reporting malicious behaviour of applications to the
2309 Apertis store:

- 2310 1. Report from the inter-domain service in the automotive domain, based on
2311 error responses from the OEM APIs.
- 2312 2. Report from the inter-domain service in the CE domain, based on error
2313 responses from the automotive domain.
- 2314 3. Report from the SDK API adapter layers, based on error responses from
2315 the automotive domain.

2316 They are presented in decreasing order of reliability, and increasing order of
2317 helpfulness.

2318 Option #1 is reliable (an attacker can only prevent a detected malicious action
2319 from being reported by compromising the automotive domain), but not helpful
2320 (the automotive domain does not have contextual information about the access,
2321 such as the application bundle which originally made the request — bundle iden-
2322 tifiers cannot be sent across the inter-domain link as that would mean partially
2323 defining the OEM APIs). This option has the additional disadvantage that it
2324 requires the AD to communicate directly with the Apertis store without going
2325 via the CE, which likely means the AD is on the Internet and could potentially
2326 be compromised by a Heartbleed-style vulnerability in a communication path
2327 that was intended to be secure. Options #2 and #3 do not have this disadvan-
2328 tage, because in those options it is the CE that needs to communicate on the
2329 Internet.

2330 Option #3 is unreliable (an attacker can prevent a detected malicious action
2331 from being reported by compromising that SDK service in the CE domain),
2332 but most helpful (the CE domain knows all contextual information about the
2333 access, including the application bundle identifier, parameters sent to the SDK
2334 API by the application, and the output of the adapter layer which was sent to
2335 the inter-domain link).

2336 We recommend option #3 as it is the most helpful, and we believe that the
2337 additional contextual information it provides outweighs the potential loss of
2338 reports from most severely compromised vehicles. This is one part of many
2339 which contribute to the security of the system.

2340 An alternative would be to implement two or all of the options, and leave it up
2341 to the Apertis store software to combine or deduplicate the reports.

2342 Suggested roadmap

2343 One the design has been reviewed, it can be compared to the existing state of
2344 the inter-domain communication system, and a roadmap produced for how to
2345 reconcile the differences (if there are any).

2346 **Open question:** How does this design compare to the existing state of the
2347 inter-domain communication system?

2348 Requirements

2349 **Open question:** Once the design is finalised a little more, it can be related
2350 back to the requirements to ensure they are all satisfied.

2351 Open questions

- 2352 • **Existing inter-domain communication systems:** Are there any relevant
2353 existing systems to compare against?
- 2354 • **IPSec versus TLS:** What is the security of the IPsec protocol in its current
2355 (2015) state?
- 2356 • **IPSec versus TLS:** What is the performance of TCP and UDP over IPsec,
2357 TLS over TCP and DTLS over UDP on the Apertis reference hardware?
- 2358 • **Configuration designs:** What trade-off do we want between performance
2359 and testability for the different transport layer configurations?
- 2360 • **Configuration designs:** What more detailed configuration options can we
2361 specify for setting up IPsec? For example, disabling various optional fea-
2362 tures which are not needed, to reduce the attack surface. What IKE
2363 service should be used?
- 2364 • **Configuration designs:** A lot of business logic for control over OEM li-
2365 cencing can be implemented by the choice of the CA hierarchy used by
2366 the inter-domain communication system. What business logic should be
2367 possible to implement?
- 2368 • **Configuration designs:** Consider key control, revocation, protocol obsoles-
2369 cence, and various extensions for pinning keys and protocols.
- 2370 • **Configuration designs:** What can be done in the automotive domain to
2371 reduce the possibility of exploits like Heartbleed affecting the inter-domain
2372 communications link? This is a trade-off between the stability of AD
2373 updates (high; rarely released) and the pace of IPsec and TLS security
2374 research and updates and the need for crypto-agility (fast). Heartbleed
2375 was a bug in a bad implementation of an optional and not-very-useful TLS
2376 extension.
- 2377 • **Control protocol:** How should the multiple CE configuration (section
2378 8.3.2) interact with D-Bus signals? Can the adapter layer perform the

- 2379 broadcast to all subscribers?
- 2380 • **Developer mode:** What cryptography should be used to implement this
- 2381 authentication, and the division of trust between development and pro-
- 2382 duction devices? A likely solution is to only have the AD accept the CE
- 2383 connection if it connects with a ‘production’ key signed by the vehicle
- 2384 OEM.
- 2385 • **Disabling the CE domain:** What hardware provisions are available for
- 2386 controlling the power supply or boot process of the CE domain? How
- 2387 should this integrate with the secure boot design?
- 2388 • **Suggested roadmap:** How does this design compare to the existing state
- 2389 of the inter-domain communication system?
- 2390 • **Requirements:** Once the design is finalised a little more, it can be related
- 2391 back to the requirements to ensure they are all satisfied.

2392 Summary of recommendations

2393 **Open question:** Once the design is finalised a little more, and a suggested

2394 roadmap has been produced (**Suggested roadmap**), it can be summarised here.

2395 Appendix: D-Bus components and licensing

2396 The terminology around D-Bus can sometimes be confusing; here are some

2397 details of its components and their licensing.

- 2398 • *D-Bus* is a [protocol](#)⁵⁶ which defines an on-the-wire format for marshalling
- 2399 and passing messages between peers, a type system for structuring those
- 2400 messages, an authentication protocol for connecting peers, a set of trans-
- 2401 ports for sending messages over different underlying connection media,
- 2402 and a series of high-level APIs for implementing common API design pat-
- 2403 terns such as properties and object enumeration. It has a reference im-
- 2404 plementation (libdbus and dbus-daemon), but these are by no means the
- 2405 only implementations. The protocol has had full backwards compatibility
- 2406 since [2006](#)⁵⁷.
- 2407 • A *D-Bus daemon* (for example: dbus-daemon, kdbus) is a process which
- 2408 arbitrates communication between D-Bus peers, implementing multicast
- 2409 communications (such as signals) without requiring all peers to connect to
- 2410 each other. Different D-Bus daemons have different performance charac-
- 2411 teristics and licensing. For example, kdbus runs in the kernel to improve
- 2412 performance by reducing context switching overhead, at the cost of some
- 2413 features; dbus-daemon runs in user space with more overhead, but is still
- 2414 quite performant.

⁵⁶<http://dbus.freedesktop.org/doc/dbus-specification.html>

⁵⁷<http://dbus.freedesktop.org/doc/dbus-specification.html#stability>

- 2415 • A *D-Bus library* (for example: libdbus, GDBus) is a set of code which
2416 implements the D-Bus protocol for one peer, converting high-level D-Bus
2417 API calls into on-the-wire messages to send to another peer or a D-Bus
2418 daemon to send to other peers. Different D-Bus libraries have different
2419 performance characteristics and licensing.

2420 Licensing

- 2421 • The D-Bus Specification is freely licensed and has no restrictions on who
2422 may implement it or how those implementations are licensed.
- 2423 • libdbus and dbus-daemon are both licensed under your choice of the
2424 [AFLv2.1](#)⁵⁸, or the [GPLv2](#)⁵⁹ (or later versions).
- 2425 – Hence, if the AFL license is chosen, libdbus and dbus-daemon may
2426 be used in non-open-source products.
- 2427 • GDBus is part of GLib, and hence is licensed under the [LGPLv2.0](#)⁶⁰ (or
2428 later versions).

2429 Appendix: D-Bus performance

2430 libdbus and dbus-daemon are reasonably performant, having been used in vari-
2431 ous low-resource products (such as mobile phones) over the years. There have
2432 not been any quantitative evaluations of their performance in terms of latency
2433 or memory usage recently, but some have been done [in](#)⁶¹ [the](#)⁶² [past](#)⁶³.

2434 As indicative numbers *only*, D-Bus (using [dbus-python](#)⁶⁴ and dbus-daemon, not
2435 kdbus) gives performance of roughly:

- 2436 • 20,000 messages per second throughput
- 2437 • 130MB per second bandwidth
- 2438 • 0.1s end-to-end latency between peers for a given message
- 2439 – This is likely an overestimate, as ping-pong tests written in C have
2440 given latency of 200µs
- 2441 • 2.5MB memory footprint (RSS) for dbus-daemon in a desktop configura-
2442 tion

⁵⁸<https://spdx.org/licenses/AFL-2.1.html>

⁵⁹<http://spdx.org/licenses/GPL-2.0+>

⁶⁰<http://spdx.org/licenses/LGPL-2.0+>

⁶¹<https://desktopsummit.org/sites/www.desktopsummit.org/files/will-thompson-dbus-performance.pdf>

⁶²<http://blog.asleson.org/index.php/2015/09/01/d-bus-signaling-performance/>

⁶³<https://blogs.gnome.org/abustany/2010/05/20/ipc-performance-the-return-of-the-report/>

⁶⁴<http://www.freedesktop.org/wiki/Software/DBusBindings/>

2443 – So this could likely be reduced if needed — the amount of message
2444 buffering dbus-daemon provides is configurable

2445 Note that these numbers are from performance evaluations on various versions of
2446 dbus-daemon, so should be considered indicative of an order of magnitude only.
2447 As with all performance measurements, accurate values can only be measured
2448 on the target system in the target configuration.

2449 The most commonly accepted disadvantage of using D-Bus with dbus-daemon
2450 is the end-to-end latency needed to send a message from one peer, through the
2451 kernel, to the dbus-daemon, then through the kernel again, to the receiving
2452 peer. This can be reduced by using kdbus, which halves the number of context
2453 switches needed by implementing the D-Bus daemon in [kernel space](#)⁶⁵. However,
2454 kdbus has not yet been accepted into the upstream kernel, and (as of February
2455 2016) there is some concern that this might not happen due to kernel politics.
2456 It can be integrated into distributions as a kernel module, although it relies on a
2457 few features only available in kernel version 4.0 or newer. This means it should
2458 be straightforward to integrate in the CE, but potentially not in the AD (and
2459 certainly not if the AD doesn't run Linux — in such cases, dbus-daemon can
2460 be used).

2461 Overall, the performance of a D-Bus API depends strongly on the API design.
2462 Good [D-Bus API design] eliminates redundant round trips (which have a high
2463 latency cost), and offloads high-bandwidth or latency sensitive data transfer
2464 into side channels such as UNIX pipes, whose identifiers are sent in the D-Bus
2465 API calls [as FD handles](#)⁶⁶.

2466 Appendix: Software versus hardware encryption

2467 The choice about whether to use software or hardware encryption is a tradeoff
2468 between the advantages and disadvantages of the options. There are actually
2469 several ways of providing 'hardware encryption', which should be considered
2470 separately. In order from simplest to most complex:

- 2471 • **Encryption acceleration instructions** in the processor, such as the
2472 [AES instruction set](#)⁶⁷, [CLMUL](#)⁶⁸ or the [ARM cryptography extensions](#)⁶⁹.
2473 These are available in most processors now, and provide assembly instruc-
2474 tions for performing expensive operations specific to certain encryption
2475 standards, typically AES, SHA and Galois/Counter Mode (GCM) for
2476 block ciphers. Intel architectures have the most extensions, but ARM
2477 architectures also have some.
- 2478 • **Secure cryptoprocessor**⁷⁰. These are separate, hardened hardware de-

⁶⁵<http://www.freedesktop.org/wiki/Software/systemd/kdbus/>

⁶⁶<http://dbus.freedesktop.org/doc/dbus-specification.html#idp9446907251>

⁶⁷https://en.wikipedia.org/wiki/AES_instruction_set

⁶⁸https://en.wikipedia.org/wiki/CLMUL_instruction_set

⁶⁹<http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.ddi0514g/index.html>

⁷⁰https://en.wikipedia.org/wiki/Secure_cryptoprocessor

2479 vices which implement all encryption operations and some key storage
2480 and handling within a tamper-proof chip. They are conceptually similar
2481 to hardware video decoders — the CPU hands off encryption operations
2482 to the coprocessor to happen in the background. They typically do not
2483 have their own memory.

- 2484 • **Hardware security module**⁷¹ (HSM). These are even more hardened se-
2485 cure cryptoprocessors, which typically come with their own tamper-proof
2486 memory and supporting circuitry, including tamper-proof power supply.
2487 They handle all aspects of encryption, including all key storage and man-
2488 agement (such that keys never leave the HSM).

2489 **Software encryption (without encryption acceleration instructions)**

- 2490 • Lowest encryption bandwidth.
- 2491 • Highest attack surface area, as keys and in-progress encryption values have
2492 to be stored in system memory, which can be read by an attacker with
2493 physical access to the hardware.
- 2494 • Certain versions of some cryptographic libraries are **FIPS**⁷²-certified, but
2495 not all. GnuTLS has been FIPS certified in various devices, but is not
2496 **routinely certified**⁷³. OpenSSL is not routinely certified, but provides a
2497 OpenSSL FIPS Object Module which *is* **certified**⁷⁴ as a drop-in replace-
2498 ment for OpenSSL, provided that it's used unmodified. The Linux kernel's
2499 IPsec support has been certified in Red Hat Enterprise Linux 6, but is not
2500 **routinely certified**⁷⁵.
- 2501 • Cheaper than hardware.
- 2502 • Provides the possibility of upgrading to use different encryption algorithms
2503 in future.
- 2504 • Possible to check the software implementation for backdoors, although
2505 it's a lot of work. Some of this work is being done by **other users of open**
2506 **source encryption software**⁷⁶.

2507 **Software encryption (with encryption acceleration instructions)**

- 2508 • Same advantages and disadvantages as software encryption without en-
2509 cryption acceleration instructions, except that the use of acceleration gives

⁷¹https://en.wikipedia.org/wiki/Hardware_security_module

⁷²https://en.wikipedia.org/wiki/FIPS_140-2

⁷³http://www.gnutls.org/manual/html_node/Certification.html

⁷⁴<https://www.openssl.org/docs/fipsvalidation.html>

⁷⁵https://access.redhat.com/documentation/en-US/Red_Hat_Enterprise_Linux/6/html/Security_Guide/sect-Security_Guide-Federal_Standards_And_Regulations-Federal_Information_Processing_Standard.html

⁷⁶<http://www.zdnet.com/article/ncc-group-to-audit-openssl-for-security-holes/>

2510 a higher encryption bandwidth (on the order of a factor of 10 improve-
2511 ment).

- 2512 • Same software interface as without acceleration.
- 2513 • Both TLS and IPsec provide various cipher suite options, at least some of
2514 which would benefit from hardware acceleration — both use [AES-GCM](#)⁷⁷
2515 for data encryption, which benefits from AES instructions.

2516 **Secure cryptoprocessor**

- 2517 • Higher encryption bandwidth.
- 2518 • Reduced attack surface area, as keys and in-progress encryption values are
2519 handled within the encryption hardware, rather than in general memory,
2520 and hence cannot be accessed by an attacker with physical access. Keys
2521 may still leave the cryptoprocessor, which gives some attack surface.
- 2522 • Typical secure cryptoprocessors have tamper evidence features in the hard-
2523 ware.
- 2524 • Typically hardware is FIPS-certified.
- 2525 • More expensive than software.
- 2526 • Provides a limited set of encryption algorithms, with no option to upgrade
2527 them once it's fixed in silicon.
- 2528 • No possibility to audit the hardware implementation to check for back-
2529 doors, so you have to trust that the hardware vendor has not been secretly
2530 required to provide a backdoor by some government.
- 2531 • Typical cryptoprocessors originate from mobile or embedded networking
2532 hardware, both of which need to support TLS, and hence cryptoprocessors
2533 typically have support for AES, DES, 3DES and SHA. This is sufficient
2534 for accelerating the common cipher suites in TLS and IPsec.
- 2535 • Have to be supported by the Linux kernel crypto API (`/dev/crypto`) in
2536 order to be usable from software.

2537 **Hardware security module**

- 2538 • Highest encryption bandwidth.
- 2539 • Minimal attack surface area, with keys never leaving the HSM.
- 2540 • All hardware is tamper-proof and tamper-evident, and typically can de-
2541 stroy stored keys automatically if tampering is detected.
- 2542 • Hardware is almost universally FIPS-certified.
- 2543 • Most expensive.

⁷⁷https://en.wikipedia.org/wiki/Advanced_Encryption_Standard

- 2544 • Provides a range of encryption algorithms, but with no option to upgrade
2545 them.
- 2546 • No possibility to audit the hardware implementation to check for back-
2547 doors, so you have to trust that the hardware vendor has not been secretly
2548 required to provide a backdoor by some government.
- 2549 • Some modules can handle encryption of network streams transparently,
2550 taking a plaintext network stream as input and handling all TLS or IPsec
2551 operations for it with peers.

2552 Conclusion

2553 According to [one evaluation](#)⁷⁸, using encryption acceleration instructions should
2554 reduce the number of cycles per byte for AES encryption from 28 to 3.5. As-
2555 suming the inter-domain connection is being used to transmit a HD video at
2556 250kB·s⁻¹, that means encryption requires 7MHz of CPU compute without ac-
2557 celeration, and 875kHz with it. Performing symmetric encryption on a data
2558 stream doesn't significantly increase the required memory bandwidth compared
2559 to copying the stream around without encryption.

2560 Hence, overall, if we assume a peak bandwidth requirement on the inter-domain
2561 communications link on the order of 250kB·s⁻¹ then using software encryption
2562 with acceleration instructions should give sufficient performance.

2563 The hardware security (tamper-proofing) provided by a HSM is overkill for an
2564 in-vehicle system, and is better suited to data centres or military equipment.
2565 We recommend either using software encryption with acceleration, or a secure
2566 cryptoprocessor, depending on the balance of the advantages and disadvantages
2567 of the two for the particular OEM and vehicle. For the purposes of this design,
2568 both options provide all features necessary for inter-domain communications.

2569 Appendix: Audio and video streaming standards

2570 There are several standards to enable reliable audio and video streaming between
2571 various systems. These standards aim to address the synchronization problem
2572 with different approaches.

- 2573 • [AES67](#)⁷⁹: The AES67 standard combines PTP and RTP using PTP clock
2574 source signalling ([RFC7273](#)⁸⁰) to synchronize multiple streams with an
2575 external clock, focusing on high-performance audio based on RTP/UDP.
- 2576 • VSF TR-03: This is a technical recommendation from the [Video Service
2577 Forum](#)⁸¹ (VFS). The TR-03 standard is similar to AES67 in terms of using

⁷⁸<https://groups.google.com/forum/msg/cryptopp-users/5x-vu0KwFRk/COSUIzvgiKYJ>

⁷⁹<https://en.wikipedia.org/wiki/AES67>

⁸⁰<https://tools.ietf.org/html/rfc7273>

⁸¹<http://www.videoservicesforum.org/>

2578 PTP for clock synchronization, but it extends AES67 to cover other kinds
 2579 of uncompressed streams, including video and metadata.

- 2580 • [AVB⁸²](#): The Audio Video Bridging (AVB) is a small extensions to standard
 2581 layer-2 MACs and bridges. An advantage of AVB is that the time syn-
 2582 chronization information is periodically exchanged through the network
 2583 so it provides great synchronization precision. However, it requires to im-
 2584 plement AVB for all of devices in the network because the device should
 2585 allocate a fraction of network bandwidth for AVB traffic.

2586 The following comparison table depicts the characteristics of the standards.

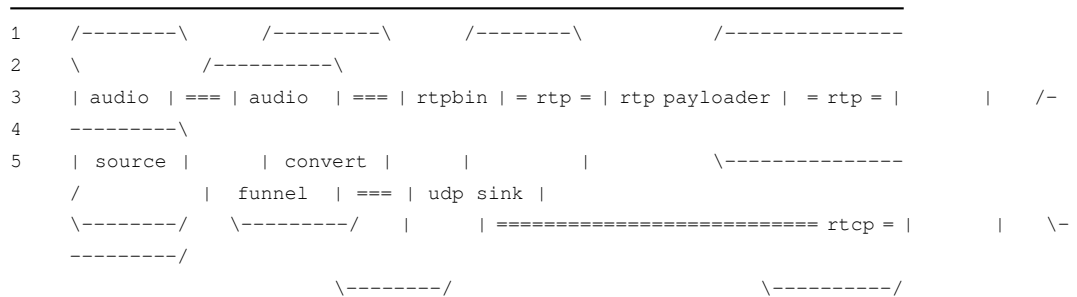
	AES67	VSF TR-03	AVB
Time synchronization	external (PTP)	external (PTP)	supported by the network
Kernel support	not required	not required	required
Transport protocol	RTP	RTP	RTP, HTTP(s), IEEE 1722
Related open source project	GStreamer	N/A	OpenAvnu

2587 Note that VFS TR-03 has no explicit open source implementation, but as it
 2588 combines RTP for transport and PTP for clock synchronization, it is generally
 2589 supported by GStreamer.

2590 Appendix: Multiplexing RTP and RTCP

2591 RTP requires the RTP Control Protocol (RTCP) to exchange control packets
 2592 and timing information such as latency and QoS. Usually RTP and RTCP use
 2593 two different channels on different network ports, but it is also possible to use
 2594 a single port for both protocols using the [RFC 5761⁸³](#) standard, supported by
 2595 the GStreamer `funnel` element.

2596 The following diagram shows how a RFC 5761 pipeline can be set up in
 2597 GStreamer:



⁸²https://en.wikipedia.org/wiki/Audio_Video_Bridging

⁸³<https://tools.ietf.org/html/rfc5761>

2598 Appendix: Audio and video decoding

2599 As a system which handles a lot of multimedia, deciding where to perform audio
2600 and video decoding is important. There are two major considerations:

- 2601 • minimising the amount of raw communications bandwidth which is needed
2602 to transmit audio or video data between the domains; and
- 2603 • ensuring that an exploit does not give access to arbitrary memory from
2604 either domain (especially not the automotive domain).

2605 The discussion below refers to video encoding and decoding, but the same con-
2606 siderations apply equally well to audio.

2607 Software encoding is a large CPU burden, and introduces quality loss into
2608 videos — so decoding and re-encoding videos in one domain to check their
2609 well-formedness is not a viable option. If decoding is being performed, the de-
2610 coded output might as well be used in that form, rather than being re-encoded
2611 to be sent to the other domain.

2612 In order to avoid spending a lot of CPU time and CPU-memory bandwidth on
2613 video decoding, it should be performed by hardware. However, this hardware
2614 does not necessarily have to be in the domain where the encoded video origi-
2615 nates. For example, it is entirely possible for videos to be sent from the CE to
2616 be decoded in the AD.

2617 The original designs which were discussed in combination with the GPU video
2618 sharing design planned to create a GStreamer plugin in the CE which treats the
2619 AD as a hardware video decoder which accepts encoded video, decodes it, and
2620 returns a handle which can be passed to the GL scene being output by the CE,
2621 via a GL extension (similar to `EXT_image_dma_buf_import`⁸⁴). This is the
2622 same model as used for ‘normal’ hardware decoders, and ensures that decoded
2623 video data remains within the AD, rather than being sent back over the inter-
2624 domain communications link (which would incur a very high bandwidth cost,
2625 which for uncompressed 1080p video in YUV 422 format at 60fps amounts to
2626 $16 \text{ bits/pixel} \times (1920 \times 1080) \text{ pixels/frame} \times 60 \text{ frames/s} = 1898 \text{ Mbit/s} = 237$
2627 MB/s).

2628 Regarding security, a hardware decoder is typically a `DMA`⁸⁵-capable peripheral
2629 which means that, unless constrained by an `IOMMU`⁸⁶, it can access all areas
2630 of physical memory. The threat here is that a malicious or corrupt video could
2631 trigger the decoder into reading or writing to areas of memory which it shouldn’t,
2632 which could allow it to overwrite parts of the (hypervisor) operating system or
2633 running applications. This concern exists regardless of which domain is driving
2634 the decoder. We highly recommend that hardware is chosen which uses an
2635 `IOMMU` to restrict the access a video decoder has to physical memory.

⁸⁴https://www.khronos.org/registry/egl/extensions/EXT/EGL_EXT_image_dma_buf_import.txt

⁸⁵https://en.wikipedia.org/wiki/Direct_memory_access

⁸⁶https://en.wikipedia.org/wiki/Input-output_memory_management_unit

2636 Note that the same security threat applies to the GPU, which has direct access
2637 to physical memory (if shared with the CPU — some systems use dedicated
2638 memory for the GPU, in which case the issue isn't present). GPUs have a much
2639 larger attack surface, as they have to handle complex GL commands which are
2640 provided from untrusted sources, such as WebGL.

2641 We recommend investigating the hardening and security applied to video de-
2642 coders on the particular hardware platforms in use, but there is not much which
2643 can be done by software to improve their security if it is lacking — the perfor-
2644 mance cost is too high.

2645 **Memory bandwidth usage on the i.MX6 Sabrelite**

2646 This section refers to some benchmarks evaluating the available memory band-
2647 width on the i.MX6 Sabrelite platform used in the reference hardware for Aper-
2648 tis. This data is very system dependent, but the order of magnitude should
2649 provide a general guide for evaluating approaches.

2650 The [iMX6 memory bandwidth usage benchmark](#)⁸⁷ describes some tools that can
2651 be used to measure how memory is used, and reports that a [1080p @ 60fps](#)
2652 [loopback pipeline](#)⁸⁸ using GStreamer requires up to 1744.46 MB/s of memory
2653 bandwidth.

2654 Another useful benchmark is the one evaluating [the cost of memory copies](#)⁸⁹
2655 done with the `memcpy()` function. The effective usable memory bandwidth mea-
2656 sured with this test amounts to roughly 800 MB/s.

2657 **Security Vulnerabilities in GStreamer**

2658 To list vulnerabilities by type we can refer to the statistics available from the
2659 [CVE](#)⁹⁰ data source.

2660 According to the [CVE Details](#)⁹¹ website, a third party that provides summaries
2661 of CVE vulnerabilities, GStreamer had [total 17 vulnerabilities](#)⁹² since 2009.

2662 Examining the DoS and Code Execution vulnerability types, the statistics
2663 showed different characteristics for demuxers and decoders. There have been
2664 12 DoS vulnerabilities affecting demuxers, but only one issue could lead to
2665 Code Execution. For decoders, all the the 5 DoS issues which were found can
2666 be escalated to Code Execution as well.

⁸⁷https://developer.ridgerun.com/wiki/index.php?title=IMX6_Memory_Bandwidth_usage

⁸⁸https://developer.ridgerun.com/wiki/index.php?title=IMX6_Memory_Bandwidth_usage#1080p60_loopback

⁸⁹<https://community.nxp.com/thread/309197>

⁹⁰<http://cve.mitre.org/>

⁹¹<https://www.cvedetails.com>

⁹²<https://www.cvedetails.com/vendor/9481/Gstreamer.html>

2667 This report indicates that demuxers might have a smaller attack surface than de-
2668 coders from the arbitrary code execution viewpoint. However, it is also possible
2669 to have a security hole similar to [Video or audio decoder bugs](#).

2670 Both demuxing and possibly even decoding in the CE can help to mitigate the
2671 described attacks. If the CE is responsible of demuxing the AD does not need
2672 to deal with content detection and container formats, and the CE provides a
2673 kind of partial verification of the stream even without decoding it.

2674 Decoding in the CE poses some challenges in terms of bandwidth, as the amount
2675 of data generated by fully decoded video streams is very high. It's not going to
2676 be a viable solution on ethernet-based setups, and advanced zero-copy mecha-
2677 nisms to transfer frames are recommended on single board setups (virtualised
2678 or container-based).