Information Flow Control (IFC) Model

Entities

Information flow control in the proposed model is based on assigning security and confidentiality labels to various system entities. We consider three types of entities:

1. *Applications*: Third-party software installed inside the home network.
2. *External communication channels*: any descriptor that identifies a range of external connections, e.g.,
    a. External IP address or subnet mask
    b. DNS domain
    c. <IP address, protocol, port> tuple
    d. VxLAN tunnel
    e. etc.
3. *Devices*: sensors and actuators

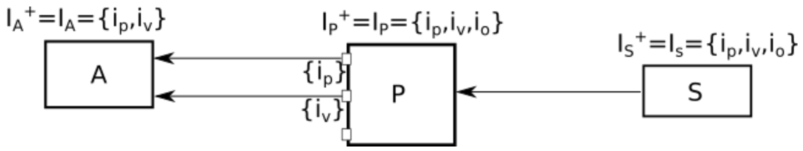
Confidentiality labels

Any entity E is assigned a *maximal confidentiality level* C_E^+, which is a set of *confidentiality labels* $C_E^+ = \{c_1,..,c_n\}$. For devices, confidentiality labels correspond to the various data streams generated by the device, e.g., an audio or video feed or a stream of notifications. Each device has its own unique set of labels, e.g., video feeds from different webcams are assigned different labels. For applications and external channels, labels describe data streams that can be observed by this application or channel. Maximal confidentiality levels are used to define the high-level security policy. For example a safe default policy may assign an empty set of labels to all untrusted Internet connections and the set of all existing confidentiality labels to each application. This policy allows applications to access sensor readings inside the house, but prevents them from sending these readings (or any data derived from them) to the outside world.

At runtime, security policies are enforced by assigning *current confidentiality levels* C_E to entities. For a device, the current confidentiality level is equivalent to the maximal confidentiality level: $C_E C_E^+$. For applications and external channels, the current level is computed based on the *connec tivity graph* of the system. A connectivity graph is a directed graph where vertices represent entities and edges represent communication channels. Edge E_1 E_2 specifies that E_1 can *read from* E_2, i.e., data is allowed to flow from E2 to E1. The effective label of a non-device entity E is computed as a union of effective labels of all successors of E: $C_E =_{Eisuc(E)}(C_{Ei})$. A connectivity graph satisfies the currently installed security policy iff for all entities E: $C_E C_E^+$. Otherwise, the security policy is violated.

Example 1: *Audio stream from the microphone may not leak to the Internet.*

When a new application is installed inside the home, it requests a number of capabilities. The user matches these requests to devices deployed inside the home network, thus establishing a set of bindings between applications and devices. These bindings become edges of the connectivity graph. Consider an intercom application A that requests access to a microphone (M) feed and to the Internet (I), and the resulting connectivity graph:



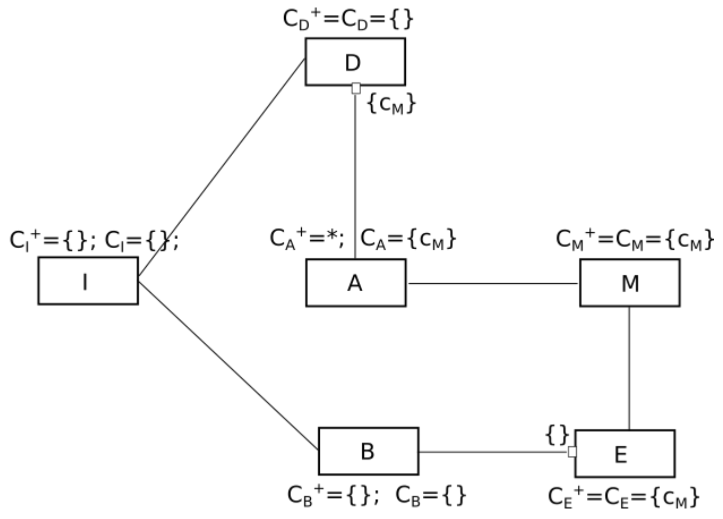
Here the edge IA violates the security policy. The system notifies the user about the violation and visualizes the possible data path along which sounds from their home can be streamed on the Internet. The user is then able to remove the Internet connectivity capability, add a security exception, or re-bind A to a trusted microphone proxy as described below.

Trusted proxies

Trusted proxies provide a flexible way to manage the tradeoff between functionality and security. They act as declassifiers that mediate access to confidential data sources. A trusted proxy is a special type of application that connects to a confidential data source and that exports restricted access to it to other applications that are not authorized to access the data source directly.

Example 2*: Securely storing encrypted audio to untrusted cloud*

In the figure below, application B stores microphone data in the cloud in encrypted form. To this end, it uses the Encryptor proxy (E) that provides unrestricted access to encrypted audio stream. Note that the output port of the encryptor has an empty set of labels, i.e., everyone is allowed to *read from it. In order to play back the recorded audio on demand, application A uses the Decryptor proxy (D) that downloads data from the cloud and labels is with microphone's confidentiality label c_M. Note that this may easily introduce a potential side channel from A to the Internet. This is one of the caveats of using proxies–programming them in a truly secure fashion is hard.*



***Example 3**: Temporally restricted streaming ((C) Iqbal)*

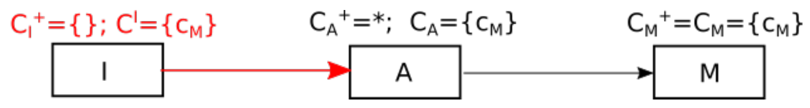
Consider a version of intercom application A that insists on sending audio to the cloud in order to perform speech processing on it and convert it into text messages sent to family members. The user considers this functionality important enough to justify the security risk. However, in order to do its job, this application does not require continues 24hr access to the microphone. Therefore, the user binds A to a proxy that only streams microphone data when the user presses the intercom button. The output port of this proxy is assigned a new confidentiality label $c_{M'}$. The user also adds an exception allowing data from the proxy to be sent to the application provider's cloud (by creating an external channel with maximal security label $\{c_{M'}\}$.

Integrity labels

Integrity labels are used to restrict control flow in a symmetric manner to how confidentiality labels restrict data flow. We define *maximal and effective integrity levels* (I_E^+ and I_E) as sets of integrity labels. An application can control an actuator iff the actuator's integrity label is contained in the application's effective integrity level. The effective integrity level of E is computed from the connectivity graph as a union of effective labels of all predecessors of E: $I_E{=}_{Eisuc(E)}(I_{Ei})$.

***Example 4**: Restricted control over speaker*

The intercom application from the above example requests access to speakers installed around the house. However, these speakers are also used by the home security monitor application as an alarm siren. The intercom should not be allowed to switch off the speakers or override volume settings of the security monitor. Such secure device sharing can be achieved with fine-grained capabilities. The speaker device exports a set of labels that correspond to its various capabilities: play sounds (i_p), control volume (i_v), volume override (i_o). The intercom app has access to i_p and i_v, but not i_o and is therefore not allowed to directly control the speaker. Instead, the user binds it to a trusted proxy (P) that provides access to individual device capabilities:



Implementation issues

Bidirectional communication

In practice, most communication links involve network packets flowing in both directions, even if the actual control or data only flow one way. Making on links bidirectional will force all entities to have identical security and confidentiality labels. There are several ways in which we can mitigate this. First, some links can really be unidirectional (e.g., a one-way UDP stream). We can also force uni-directional TCP by only allowing ACKs in the opposite direction (but of course this creates a covert channel). Third, we can trust proxies to enforce directionality at higher semantic levels. I.e., even though packets flow in both directions between the proxy and its client app, the proxy guarantees that useful data is only exchanged in one direction.