Notes on Information Flow Policies

CS 5430

May 1, 2017

Restrictions and Access Control

Data are associated with *restrictions.* These restrictions are usually expressed in terms of confidentiality (e.g., who can read data), in terms of integrity (e.g., how much trusted data is), or in terms of privacy (e.g., what operations can be applied on data).

Access control has been widely used to specify and enforce restrictions on data. However, access control alone is not enough. Consider a document *doc* associated with an access control policy P (e.g., only Alice can read *doc*). Assume that some computation is applied to *doc* producing several new documents that needs to be used later. What should be the access control policy on these new documents? A human should make this decision and should manually associate the desired policies to the new documents.

This manual work becomes even harder, when we scale up to a system that stores multiple pieces of data that are owned by multiple users, and supports rich interactions between data of different users. So, access control alone does not seem suitable for the *Big Data* era. Also, access control alone cannot prevent leaking information through metadata, shared resources, and other *covert channels*, which are channel not intended to convey information.

Information Flow Policies

Information Flow (IF) policies are proposed to address the limitations of access control. An IF policy associated with a piece of data d specifies restrictions on d, and on all data derived from d. For example, an IF policy for confidentiality could specify: value v and all its derived values is allowed to be read at most by Alice. Equivalently, we say that v is allowed *to flow* only to Alice.

An enforcement mechanism for IF policies automatically deduces the restrictions for derived data. For example, if a document *doc* is allowed to flow only to Alice, then any document derived from *doc* is allowed to flow only to Alice, too. Otherwise, the IF policy on *doc* would be violated. When documents with different policies are combined to produce a new document, that new document is associated with a policy that satisfies all policies of the constituent documents.

Labels for IF policies

We use *labels*, which are syntactic objects, to represent IF policies. *Classifications* can be used as labels for information flow policies. For example, labels U, C, S, and TS (i.e., Unclassified, Confidential, Secret, and Top Secret) or labels L and H (i.e., Low confidentiality, High confidentiality) can be used to represent IF policies. If some document *doc* is associated with C, it means that *doc* and all documents derived from *doc* are considered Confidential (i.e., tagged with C). *Sets of principals* can also be used as labels. For example, if some document *doc* is associated with $\{Alice\}$, it means that *doc* and all documents derived from *doc* can be read at most by Alice.

More expressive labels

The labels discussed above represent policies that are more restrictive than necessary, because they impose the same restrictions to all possible derived data. However, there are practical cases where restrictions on outputs of some operations are different (e.g., fewer or more) than the restrictions on inputs of these operations.

Consider, for example, operation *maj* that tallies votes for an election:

$$x := maj(\nu_1, \nu_2, \ldots, \nu_n).$$

Input plaintexts are usually considered *secret* (i.e., of high confidentiality), but the result x of the election is usually considered *public* (i.e., of low confidentiality). If each vote ν_i is tagged with label H (which represents that ν_i is of high confidentiality), then the output x, which is derived from all these votes, is required to be tagged with H, too. But this means that x cannot be considered public. So, there is a mismatch between the restrictions imposed by the labels of inputs to the output, and the desired restrictions to the output.

Similarly, for the encryption operation:

$$x := Enc(y; k).$$

Input votes are usually considered secret, but the ciphertext x is usually considered public. However, tagging y and k with H, implies that x is tagged with H. Again, there is a mismatch between the imposed and the desired restrictions on x.

Other operations may cause the restrictions imposed on outputs to increase comparing to restrictions imposed on inputs. For example, the list of students in CS and the list of addressed in Ithaca may be public, but the mapping of students to home addresses should not be public. So, here, if the list of students in CS and the list of addresses in Ithaca are tagged with L, then the resulting list will be tagged with L. Thus, fewer restrictions than desired will be applied to the resulting list.

So, there is a need for IF policies and labels to express how restrictions on derived data may change based on applied operations, or based on events that occur during execution, or based on ownership of this data.

Noninterference

Consider inputs and outputs of a program being tagged with label H or L. Inputs tagged with H are allowed to flow only to outputs tagged with H. Equivalently, inputs tagged with H are not allowed to flow to outputs tagged with L. This implies that *changing inputs tagged with H should not cause changes on outputs tagged with L.* This requirement is an instantiation of *noninterference.* Noninterference is a *semantic* guarantee that should be offered by the enforcement mechanism of IF policies. Access control does *not* offer a similar semantic guarantee.

Consider, for example, the program below

$$h' := h + l;\ l' := l + 1$$

where variables h, h' are tagged with H and variables l, l' are tagged with L. Here, l and h model inputs, while l' and h' model outputs. This program satisfies noninterference because changing values in h does not cause values in l' to change. However, program

$$l' := h * 2$$

does not satisfy noninterference because changing h causes l' to change. So, here h is *leaked* to l'.

More carefully, noninterference states that if two initial (or input) memories M_1, M_2 agree on variables tagged with L (i.e., $M_1 =_L M_2$), and if program C is executed on M_1 and M_2 to termination, then the corresponding outputs $C(M_1)$ and $C(M_2)$ should also agree on variables tagged with L (i.e., $C(M_1) =_L C(M_2)$). Specifically:

$$\text{if } M_1 =_L M_2, \text{ then } C(M_1) =_L C(M_2).$$

The above statement of noninterference handles only programs that terminate. What if a program does not terminate depending on inputs tagged with H? Consider the following example:

$$\begin{aligned}&\textbf{while } h > 5 \textbf{ do } \{\textbf{skip}\};\\ &l' := 4\end{aligned} \tag{1}$$

where command **skip** does nothing. If $h > 5$ is *false*, then l' becomes 4. If $h > 5$ is *true*, then no value is assigned to l'. Principals observing l' either observe 4 being assigned to l', or no value being assigned to l', depending on $h > 5$. So, $h > 5$ is leaked to principals observing l'.

Termination sensitive noninterference strengthens noninterference by requiring the termination behavior of the problem to not depend on secret values:

If $M_1 =_L M_2$, then
C terminates on M_1 iff C terminates on M_2 and
$C(M_1) =_L C(M_2)$.

Program (1) does not satisfy termination sensitive noninterference, but the following program satisfies termination sensitive noninterference:

$$\begin{array}{l}\textbf{while}\ l > 5\ \textbf{do}\ \{\textbf{skip}\};\\ l' := 4\end{array}$$

When information flow policies relax restrictions on derived data, and thus allow leaking information to the output of a certain operation, the previous statements of noninterference do not hold. Consider, for example, an information flow policy that allows an input h tagged with H to flow to an output l' tagged with L only through the operation *mod* 2. According to that policy principals observing l' are allowed to learn whether h is even or not, but they are not allowed to learn anything else about h. So, program

$$l' := h\ mod\ 2 \tag{2}$$

satisfies this policy, but program

$$l' := h\ mod\ 2 + h \tag{3}$$

does not, because this last program leaks more information to l' than just $h\ mod\ 2$. For this particular policy, noninterference should be restated as:

If $M_1 =_L M_2$ and $M_1(h)\ mod\ 2 = M_2(h)\ mod\ 2$, then $C(M_1) =_L C(M_2)$.

Notice that this statement of noninterference is satisfied by program (2), but it is not satisfied by (3).