A Dynamic Information Flow Model of Secure Systems[*]

Jianjun Shen and Sihan Qing
Institute of Software, Chinese Academy of Sciences
Graduate University of Chinese Academy of Sciences
weishen_cu@yahoo.com.cn

ABSTRACT

We characterize the information flow features of an information system in a state machine model, which emphasizes on the subject properties of information flows. We ague that the legality of a flow mainly depends on the subjects exploit it rather than the variables it passes through. Within the model, security policies and access control are studied in terms of information flow. We also discuss information flow security analysis. The motivation of this work comes from our experience in covert channel analysis.

Categories and Subject Descriptors
D.4.6 [**Operating Systems**]: Security and Protection

General Terms
Security

Keywords
Information flow, Security policy, Access control, Covert channel.

1. General Model

1.1 Basic Concepts

Subject – A subject is the abstract delegate of a group of active entities with the same identity.

Primitive – Primitives comprise the interface of an information system to subjects. A subject can invoke a primitive, called one execution. The execution of a primitive may alter the system state.

We use *exec(s, p, ss)* to denote the execution of primitive *p* by subject *s* in system-state *ss*; *exec(s, p)* is also used when we do not care the system state.

State Transition – A transition of the system state takes place when an execution of a primitive returns.

State Variable – State-variables are information containers in an information system. A state-variable shall satisfy: 1) It is alterable and referenceable (in some primitive executions), that is, information is possible to flow in and flow out of a state-variable; 2) Its value keeps during a state-transition if it has not been deleted.

[*] This work was supported by the Natural Science Foundation of Beijing (No. 4052016); the National Natural Science Foundation of China (No. 60573042); the National Basic Research Program of China (No. G1999035802).

Note that *state-variable* does not equal *object* in our model. We view (see section 2) an object as an abstract entity which is the target of access control and is associated with a security label.

System State – A system-state corresponds to a set of "state-variable - value" pairs. It defines the value of each existing state-variable in the system at a given time point.

Input / Output Variable – Two special variables, V_I and V_O, represent the input from and output to the invokers of primitive executions, respectively.

1.2 Information Flow

Meta Flow – *exec(s, p)* may cause information to flow from variable *v1* to *v2*. This can be represented by a quadruple: *(s, p, v1, v2)*

Meta-flow (s, p, V_I, v) indicates that the value of state-variable *v* after *exec(s, p)* has relevancy to the input from *s*; while (s, p, v, V_O) indicates that *exec(s, p)* returns some information about *v*'s value to *s*. Actually, (s, p, V_I, v) exists so long as *s* is capable of altering *v* by executing *p*, for *s* can at least choose whether or not to invoke *p* to transfer information to *v*.

Definition 1-1. A sequence of meta-flows that occur in succession shape an **information flow**: $<(s_1, p_1, v_0, v_1)\ (s_2, p_2, v_1, v_2)\ \ldots\ (s_i, p_i, v_{i-1}, v_i)\ (s_i, p_i, v_{i-1}, v_i)\ \ldots\ (s_n, p_n, v_{n-1}, v_n)>$. We also write a flow sequence as $\underline{v_0\, s_1(p_1, v_1)\, s_2(p_2, v_2)\, \ldots\, s_n(p_n, v_n)}$ for short, where *s(p, v)* means: *exec(s, p)* extends the flow to a next variable *v*.

Definition 1-2. A **channel** is an information flow that can transmit information from one subject to another. It looks like: $\underline{V_I\, s_1(p_1, v_1)\, s_2(p_2, v_2)\, \ldots\, s_i(p_i, v_i)\, \ldots\, s_n(p_n, V_O)}$, $s_1 \neq s_n$.

Refer to [1] for example scenarios of information flows. Below, we introduce some operators to better describe flow sequences:

- *subj(mf)* gets the subject element of meta-flow *mf*; *prim(mf)* gets the primitive element of *mf*. *src(mf)* gets the source *variable* of *mf*; *dest(mf)* gets the destination variable of *mf*.
- *len(f)* gets the length of flow sequence *f*; *f[i]* gets the *i*'th meta-flow of *f*. *sender(f) = subj(f[1])*, *recver(f) = subj(f[len(f)])*; *src(f) = src(f[1])*, *dest(f) = dest(f[len(f)])*.

1.3 Information Flow System

Definition 1-3. An **information flow system *M*** is defined by: <S, P, V, SS, IN, *T*, *Mf*, Ss_0>.

- S is a set of subjects. P is a set of primitives. V is a set of state-variables. IN is a set of inputs.
- SS is a set of system-states.
  It defines the value space of state-variables.
- *T* is a state transition function: $S \times P \times SS \times IN \rightarrow SS$.
  T(s, p, ss, in) defines the new system-state after *exec(s, p, ss)* with input *in* returns.
- *Mf* is a function: $S \times P \times SS \rightarrow (V \cup \{V_I\} \times V \cup \{V_O\})^*$.
  Mf(s, p, ss) defines the meta-flows generated by *exec(s, p, ss)*.

- $Ss_0 \in$ SS is a constant. It defines the initial system-state.

Note that *Mf(s, p, ss)* does not depend on the input to *exec(s, p, ss)*. It indeed reflects the ability of subject *s* to arouse information flows by invoking primitive *p* in system-state *ss*.

1.4 Security Policy

Security policies are classified to static policies which do not vary with system states and dynamic policies which vary with system states. Given an information flow system ***M*** = <S, P, V, SS, IN, *T*, *Mf*, Ss_0>, let F_M denote the set of all the possible flow sequences and C_M denote the set of all the channels that can be constructed from <S, P, V>. Rss_M represents the set of ***M***'s runtime system-states that ***M*** can actually enter starting from Ss_0.

Definition 1-4. A **static security policy** *Sp* is defined by a set of information flows which are illegal in system ***M***.

To express dynamic policies, we append system-state constraints to an information flow sequence. A dynamic illegal flow is defined by a flow sequence together with a set of system-states. Only when its last meta-flow occurs in one of these system-states, the flow is actually illegal.

Definition 1-5. A **dynamic security policy** *Dp* is defined by a set of "information flow – constraint" pairs: $Dp \subseteq F_M \times SS^*$. ***M*** is **secure** iff: $(\forall (f, pc) \in Dp)$ $(\forall ss \in Rss_M)$ $[IFc(f, ss, pc) = False]$.

In plain words, system-state *ss* is a secure state iff no illegal flow can arise in *ss*, and system ***M*** is secure iff all its runtime states are secure states. A most important illegal flow type is illegal channel. In fact, a large proportion of security policies are indeed about illegal channels. As an example, below we present a simple static policy. A typical kind of dynamic policy is discretionary security policy. Section 2.2 shows a some "radical" one.

Example 1. Information downgrading

A multi-level security (MLS) system forbids information to flow from a high-level subject to another low-level subject, unless the communication is authorized by an arbitrator *sa* through a downgrading primitive *pa*. The policy is formally: $\{c \mid c \in C_M \wedge sl(sender(c)) > sl(recver(c)) \wedge \neg (\exists mf \in c) [(subj(mf), prim(mf)) = (sa, pa)]\}$, where *sl(s)* gets the security level of subject *s*.

2. Access Control

Access control covers all the mechanisms that make the effects of an identical action by two groups of subjects differ. The "identical action" here means: execution of the same primitive, in the same system state, with the same input; and "effects" means: the new system state after the execution and the meta-flows generated.

2.1 Preliminary Concepts

Object – Objects are the targets of access control. An object is indeed an abstract entity which is (explicitly or implicitly) associated with a security label. An object can be mapped to a set of state-variables which are deemed to have the same security label as the object. They are called the attributes of the object.

For example, a file can be viewed as an object. Its attributes include the state-variables for its control structure, contents, access-control-list, etc. We denote object *o*'s attribute variable *va* as *o.va*.

Access Operation – Access-operations are the spots that access control mechanisms take effect. Some kinds of access checks are performed at the entrance of an access-operation based on the security labels of the invoker and target object.

Access control restricts a large proportion of information flows. We use *(s, p, v1, v2) [a, o]* to represent a meta-flow which occurs only when access-operation *a* on object *o* passes the access control check. Some simplification is made here in that: a meta-flow may be restricted by multiple checks on different objects, but we only remain one restriction.

2.2 Access Control System

Definition 2-1. An **access control system *AM*** is defined by: <S, P, V, SS, IN, A, O, *T*, *AMf*, Ss_0>.

- A is a set of access-operations. O is a set of objects.
- *AMf*: $S \times P \times SS \rightarrow (V_S \cup \{V_I\} \times V_S \cup \{V_O\} \times A \times O)^*$.

Flow function *AMf* distinguishes the meta-flows bearing different access restrictions (or no restriction). *(v1, v2, a, o) ∈ AMf(s, p, ss)* indicates that *exec(s, p, ss)* generates meta-flow *(s, p, v1, v2) [a, o]*. Such a definition may not be good for understanding, so we introduce an access check function *Ac*: $(S \times A \times O \times SS) \rightarrow$ Boolean. *Ac(s, a, o, ss)* prescribes whether subject *s* is authorized to perform access-operation *a* on object *o* in system-state *ss*.

- $Ac(s, a, o, ss) = False \Rightarrow \neg (\exists (v1, v2, p) \in V_S \cup \{V_I\} \times V_S \cup \{V_O\} \times P) [(v1, v2, a, o) \in AMf(s, p, ss)]$;
- Or, $(\exists (v1, v2, p) \in V_S \cup \{V_I\} \times V_S \cup \{V_O\} \times P) [(v1, v2, a, o) \in AMf(s, p, ss)] \Rightarrow Ac(s, a, o, ss) = True$.

In fact, this relation can be used to define *Ac* from *AMf*. Access checks are divided into static ones and dynamic ones. Discretionary access control is typical dynamic access control.

Example 2. Discretionary flow policy (not an access control policy)

Information of an object can leak to a subject only when the object's owner permits. The policy is: $\{(f, pc) \mid f \in F_M \wedge (\exists o \in O) [(src(f) = o.content \wedge dest(f) = V_O \wedge (\forall ss \in SS) [ss \in pc \leftrightarrow \neg perm(recver(f) \leftarrow o, ss)])\}$, where *perm* prescribes whether the owner authorizes the information flow in a given system-state.

Proper access control can effectively eliminate illegal information flows. We can deduce a function *antinomy* (limited by pages, the formal definition is not given here) on flow sequences in ***AM***. *antinomy(f)* checks whether flow *f* may survive the access restrictions on it (or whether it is not a false flow).

3. Framework of Information Flow Analysis

We summarize the general process of analyzing flow security within an information system as: (1) Analyze every primitive and identify the meta-flows that it may generate. (2) Define illegal flows in the form of information flow sequence; extract the flow characteristics of illegal flows. (3) Build flow combination rules from the illegal flow characteristics; combine meta-flows under the supervision of these rules to construct potential illegal flows.

In step (1) we construct a (or part of) static model; in step (2) we define the flow security policy. Compared to previous flow analysis approaches, this framework does not require extensive labeling or typing of variables. In fact, security-labels are associated with abstract *subject* or *object* but not variables at implementation level. We believe a more practical methodology for checking flow legality is to inspect the subjects that may actually exploit the flow rather than the variables the flow passing through, while flow characteristics and the access control restrictions on objects can help optimize the analysis process.

4. Covert Channel

Covert channels are typical "channel-like" illegal flows. In [1] we present a taxonomy of covert channel, and discuss the characteristics and identification of various channels. This section reviews the previous work within a static access control system.

4.1 Classification of Covert Channel

The principle of our classification lies on the information flow characteristics of and the relations between covert channels. Another consideration is that the classification should benefit channel analysis and handling.

Definition 4-1. A **direct covert channel** (DCC) is a covert channel that consists of a single sending operation and a single receiving operation, that is, it includes only two meta-flows: $\underline{V_I\ sender(p_1, v_1)\ recver(p_2, V_O)}$. A covert channel which has more than two meta-flows is called an **indirect covert channel** (ICC).

Definition 4-2. A **SR$^+$ covert channel** (SR$^+$CC) is a covert channel that has only one sending operation: $\underline{V_I\ sender(p_1, v_1)\ recver(p_2, v_2)\ ...\ recver(p_m, V_O)}$.

Definition 4-3. Two channels *c1* and *c2*. ***c1*** $\subset$ ***c2*** iff the flow sequence of *c1* is a sub-sequence of *c2*, neglecting the source variable (V_I) and destination variable (V_O); ***c1*** $\angle$ ***c2*** iff the "operation - variable" sequence of *c1* is a sub-sequence of *c2*, regardless of subject elements, the source variable and destination variable.

- $\boldsymbol{c1 \angle c2} \Leftrightarrow c1 \neq c2 \wedge \exists\, i\ (0 \leq i \leq len(c2) - len(c1))\ [prim(c1[1]) = prim(c2[i+1]) \wedge \forall\ (2 \leq j \leq len(c1))\ [prim(c1[j]) = prim(c2[i+j]) \wedge src(c1[j]) = src(c2[i+j])]]$.

That is, assume $c2 = \underline{V_I\ sender(p_1, v_1)\ ...\ sender(p_k, v_k)\ recver(p_{k+1}, v_{k+1})\ ...\ recver(p_m, V_O)}$. If $c1 \subset c2$ then *c1* is like $\underline{V_I\ sender(p_i, v_i)\ ...\ sender(p_k, v_k)\ recver(p_{k+1}, v_{k+1})\ ...\ recver(p_m, V_O)}$; if $c1 \angle c2$ then *c1* is like $\underline{V_I\ sender(p_i, v_i)\ ...\ recver(p_m, V_O)}$.

Definition 4-4. Covert channel *acc* is an **atomic covert channel** (ACC) iff $\neg\,(\exists cc \in \text{CC})\ [cc \angle acc]$.

A real instance of atomic channel is the file node number channel we report in [1]: $\underline{V_I\ sender(Create, next\_num)\ recver(Create, File.num)\ [create, File]\ recver(Stat, V_O)\ [read, File]}$.

Definition 4-5. Covert channel *scc* is a **single covert channel** (SCC) iff $\neg\,(\exists cc \in \text{CC})\ [cc \subset scc]$. A covert channel that is not a single channel is called a **plural covert channel** (PCC).

A part of the flow sequence of a plural channel already forms a channel. It can be proved that DCC $\subseteq$ ACC $\subseteq$ SCC $\subseteq$ SR$^+$CC, and we conclude in [1]: to attain the completeness of covert channel analysis, at least all single covert channels shall be identified, or atomic covert channels shall be recursively identified and handled.

4.2 Covert Channel Identification

In this section, we discuss the optimization of covert channel identification guided by the information flow analysis framework, and mainly focus on flow characteristics analysis and meta-flow combination. An example of identifying atomic channels in the descriptive specification of a secure file system is presented in [1].

A covert channel $cc = \underline{V_I\ sender(p_i, v_i)\ ...\ recver(p_n, V_O)}$ (access-restrictions are omitted for short) presents following information flow characteristics:

a.1 The flow sequence extends to variable V_O and then terminated;

a.2 There is one subject switch in the flow sequence.

A SR$^+$ covert channel additionally presents:

b. A subject switch occurs between the first and second meta-flows in the flow sequence.

A single covert channel (*scc*) additionally presents:

c. The information flow is unable to find an outlet before its last operation: $\forall i\ (2 \leq i < len(scc))\ [\neg\,(\exists cc \in \text{CC})\ [cc = \underline{V_I\ sender(p_1, v_1)\ recver(p_2, v_2)\ ...\ recver(p_i, V_O)}]]$.

An atomic covert channel (*acc*) additionally presents:

d. $\forall i\ (2 \leq i < len(acc))\ [\neg\,(\exists cc \in \text{CC})\ [cc = \underline{(V_I\ sender(p_i, v_i)\ recver(p_{i+1}, v_{i+1})\ ...\ recver(p_m, V_O))}]]$.

Next, meta-flows are combined to construct potential covert channels. It is an essential step for reducing false illegal flows. There are various techniques that can be utilized to connect flows (e.g. Shared Resource Matrix). So we do not dwell on the whole process, but just discuss: given a flow *f* and a meta-flow *mf*, how to judge whether the combination of them can help construct a covert channel of the expected type. Figure 1 shows the scene: the extension of *f* to *mf* is supervised by a series of rules.

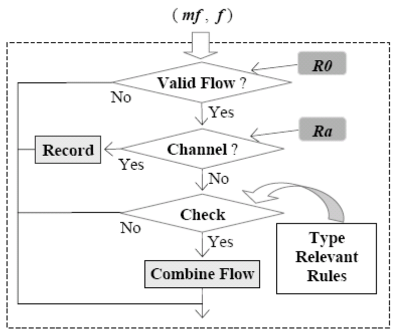

Figure 1. Flow combination control

An information flow sequence may be extended from its head to tail, or reversely form the tail to head. According to flow characteristic *a.1*, reverse extension can avoid some invalid steps if we start from an output meta-flow. Let *mf » f* get a new flow by inserting meta-flow *mf* to the head of flow *f*. Rule *R0* checks whether *mf » f* is a valid flow and terminates the extension if not; *R1* checks whether *mf » f* already forms a channel and records it as a potential covert channel if so.

- ***R0*** := $dest(mf) \neq src(f) \vee antinomy(mf \gg f) \vee \exists\, i\ (1 \leq i \leq len(f))\ [mf = f[i]]$ {abort;}
- ***Ra*** := $subj(mf) = sender \wedge src(mf) = V_I$ {record *mf » f*; abort;}

Type relevant rules check whether *mf » f* tends to form a channel of the specified type. They are deduced from the information flow characteristics of corresponding covert channel types. The combination rules for SR$^+$ channel (*Rb*), single channel (*Rc*), and atomic channel (*Rd*) are (incrementally):

- ***Rb*** := $subj(mf) = sender$ {abort;}
- ***Rc*** := $(\exists\ mf1 \in AMf_S(recver, prim(mf))\ [src(mf1) = src(mf) \wedge dest(mf1) = V_O \wedge restrict(mf) \subseteq restrict(mf1)]$ {abort;}

($restrict(mf) \subseteq restrict(mf1)$ means the access restriction on meta-flow *mf* is stricter than that on *mf1*, formalization elided here)

- ***Rd*** := $(\exists\ mf1 \in AMf_S(sender, prim(mf)))\ [src(mf1) = V_I \wedge dest(mf1) = dest(mf) \wedge \neg\, antinomy(mf1 \gg f)]$ {abort;}

5. References

[1] J. Shen, S. Qing, Q. Shen, and L. Li. "Optimization of Covert Channel Identification", In *Proc. of* 3rd *International IEEE Security in Storage Workshop*, San Francisco, CA, Dec. 2005.