

Flow Based Interpretation of Access Control: Detection of Illegal Information Flows

Mathieu Jaume¹, Valérie Viet Triem Tong², and Ludovic Mé²

¹ University Pierre & Marie Curie, LIP6, Paris, France

² SUPELEC, SSIR Group (EA 4039), Rennes, France

Abstract. In this paper, we introduce a formal property characterizing access control policies for which the interpretations of access control as mechanism over objects and as mechanism over information contained into objects are similar. This leads us to define both a flow based interpretation of access control policies and the information flows generated during the executions of a system implementing an access control mechanism. When these two interpretations are not equivalent, we propose to add a mechanism dedicated to illegal information flow detection to the mechanism of access control over objects. Such a mechanism is parameterized by the access control policy and is proved sound and complete. Finally, we briefly describe two real implementations, at two levels of granularity, of our illegal flow detection mechanism: one for the Linux operating system and one for the Java Virtual Machine. We show that the whole approach is effective in detecting real life computer attacks.

Keywords: Access control models, Information flows.

1 Introduction

The classical approach to protect information in a system consists in defining a security policy and enforcing it with diverse security mechanisms. Among these mechanisms, access control, which allows to grant or revoke the rights for active entities (the subjects) to access some passive entities (the objects), is of particular importance. Unfortunately, classical access control mechanisms implemented nowadays in mainly used computer systems cannot always control how the information is used once it has been accessed. As an example, let us consider an access control policy where granted accesses are specified by the matrix:

	o_1	o_2	o_3	o_4
Alice	read, write		read	
Bob	read	read, write		
Charlie		read, write		write

(1)

In this example, an authorized user can pass information to other users that are not authorized to read or write it. Indeed, Alice can read o_3 and write information contained in o_3 into o_1 on which Bob can make a read access, even if Bob cannot read o_3 . Similarly, Bob can write some information into o_2 and then Charlie

can read o_2 and write this information into o_4 , even if Bob cannot write into o_4 . To overcome this difficulty, flow control has been proposed. On the contrary to access control that does not provide a sufficient protection of information as it is not aware of the *can flow* relationship between objects in the system, flow control ensures that data contained in a container cannot flow into other containers readable by users that cannot read the original container. The more noticeable model of that kind is of course the Bell & La Padula model [1] where high level information cannot flow into low level containers. However, flow control has not been widely used until now for everyday computers running Windows or Unix-like OS. We believe nevertheless that flow control is crucial for a system to be secured: many real world attacks imply a series of flows, generated by several subjects, each flow being legal by itself (and hence not stopped by access control), but the composition of the flows resulting in information disclosure or integrity violation. After a presentation of some related works, we present a framework that enables us to formally define information flows induced by an access control policy over objects (section 2). Then we characterize flow policies induced by access control policies over information (section 3) and a mechanism for detecting illegal information flows in an operating system (section 4). Lastly we present an implementation of the monitor of information flows (section 5).

Related Works. Denning & al were the first to propose a static approach to check the legality of information flows in programs [3,4]. The legality of information flows is expressed through a policy which is specified using a lattice of security labels. The static analysis of programs consists in checking the legality of information flows, with regard to the security labels that are associated with data containers of the programs. We follow a similar approach with reading and writing tags, but we aim at a dynamic detection, without a previous static analysis. Many previous works are related to the information flows that might appear during the lifetime of a system where access to information is under the control of an access control policy. In [15], Sandhu presents a synthesis of Lattice-Based Access Control Models (LBAC), allowing to ensure that information only flows in authorized security classes. Nowadays, the system SELinux descends from these works and enforces a Mandatory Access Control (MAC) policy based on the Bell & LaPadula model. Unfortunately, SELinux is notoriously hard-to-configure. We thus propose a system that can be configured directly from an access control policy. Furthermore, in a real system, the use of a strict LBAC model appears quite rigid since applications often need declassification mechanisms [13,12,11]. As systems grow larger and more complex, such rigid models are not often used, users simply preferring an access control model like HRU or RBAC[16] without any control of information flows. Hence, we believe it is important to provide another mechanism to ensure that information flows are consistent with the policy. With the same objective, Osborn proposes in [14] to check if a given access control policy expressed within the RBAC model has an equivalent information flow policy in the LBAC model and if this last model is consistent with the security policy. More precisely, [14] describes an algorithm to map a role-graph of a RBAC model to an information flow graph defining the *can flow* relationship of

the given role graph, thus characterizing the information flows that can occur. We follow such an approach to study the *can flow* relationship generated by an arbitrary access control model and we provide a formal framework allowing to express the desired properties at an abstract level. A similar approach can be found in [19], where Zimmermann & al have proposed to complement classical non-transitive access control by a dynamic detection mechanism able to identify illegal composed information flows. In this development a reading tag is associated with each information (the initial contents of each object) and a writing tag to each object. The initial values of these tags comes directly from the access control matrix which specify the security policy enforced on the system. When an information flow occurs between a read object o_1 and a written object o_2 , the reading tag of o_2 is updated according to the one of o_1 . Thanks to this propagation mechanism, the detector is able to identify accesses that transitively create illegal information flows. Nevertheless, this detection model can only be used with an access control expressed by an matrix of rights, and there is no proof of its soundness and its completeness. Following the same idea, in [10], Ko and Redmond introduce a policy-based intrusion detection system dealing with information flows and allowing to detect data integrity violations by monitoring the users' actions' effects on the contents of data containers (mainly files). The detection process is achieved by controlling noninterference between users and data. They show that although the noninterference model itself is not always an enforceable security policy [17], it can nevertheless be used as a foundation for intrusion detection by taking into account the semantics of individual actions (in their case, file-related Linux system calls). Our approach is similar but allows to express confinement, integrity and confidentiality properties (it is not limited to data integrity) and we are thus able to detect a wider range of attacks, while Ko and Redmond's IDS is limited by construction to race condition-based attacks. Moreover, from a theoretical point of view, they only give a proof of the soundness of their approach, while completeness is not studied.

2 Access Control Models and Induced Information Flows

Our objective is to propose a dynamic monitor of information flows for currently used computers whose aim is to raise an alert when observing an illegal information flow. Hence, this monitor must be able to deduce some information flows from the observation of accesses. An information flow from an object o_1 to an object o_2 can be either *elementary* when a user performs at the same time a read access on o_1 and a write access on o_2 or *composed* when a user performs a read or write access on an object and closes a read/write chain between o_1 and o_2 that allows information to flow from o_1 to o_2 . An elementary flow is always legal since it is directly permitted by the access control policy. We consider here that a composed information flow is legal if can be obtained as a (legal) elementary flow. Practically, our monitor tracks elementary information flows and checks dynamically if resulting composed flows are legal or not.

Access Control Policies. We introduce here an abstract specification of access control policies over objects together with operational mechanisms allowing to enforce them. By following the approach introduced in [9], a security policy is a characterization of secure elements of a set according to some security information. Hence, specifying a policy \mathbb{P} first consists of defining a set \mathbb{A} of “things” that the policy aims to control, called the security targets, in order to ensure the desired security properties (these “things” can be the actions simultaneously done in the system or some information about the entities of the system). When dealing with access control policies, targets are sets of accesses simultaneously done in the system and we represent accesses as tuples (s, o, a) expressing that a subject $s \in \mathcal{S}$ has an access over an object $o \in \mathcal{O}$ according to an access mode $a \in \mathcal{A} = \{\text{read}, \text{write}\}$. Hence, \mathbb{A} is the powerset of the cartesian product $\mathcal{S} \times \mathcal{O} \times \mathcal{A}$. Then, a set \mathcal{C} of security configurations is introduced: configurations correspond to the information needed to characterize secure elements of \mathbb{A} according to the policy. For access control policies, a configuration can be a set of granted accesses (for the HRU policy), a lattice of security levels (for a MultiLevel security policy), a hierarchy of roles (for the RBAC policy), etc. Last, a policy is defined by a relation \Vdash allowing to characterize secure targets according to configurations.

Definition 1. *A security policy \mathbb{P} is a tuple $\mathbb{P} = (\mathbb{A}, \mathcal{C}, \Vdash)$ where \mathbb{A} is a set of security targets, \mathcal{C} is a set of security configurations and $\Vdash \subseteq \mathcal{C} \times \mathbb{A}$ is a relation specifying secure targets according to configurations. An access control policy is a security policy such that \mathbb{A} is the powerset of the cartesian product $\mathcal{S} \times \mathcal{O} \times \mathcal{A}$.*

Example 1. We consider here the HRU policy $\mathbb{P}_H = (\mathbb{A}, \mathcal{C}_H, \Vdash_{\mathbb{P}_H})$, introduced in [7], which is a discretionary access control policy where $\mathcal{C}_H = \mathbb{A}$ (a configuration $m \in \mathcal{C}_H$ specifies a set of granted accesses). Secure targets are thus defined as sets of accesses which are granted: $m \Vdash_{\mathbb{P}_H} A$ iff $A \subseteq m$. In all this paper, we illustrate some of our definitions by considering the HRU policy applied with the configuration m defined by the access control matrix (1). We do not consider here the administrative part of this model allowing to change authorized accesses.

Information Flows. We formalize here information flows occurring during the lifetime of a system on which an access control policy applies.

Transition systems. We represent a system on which a policy $\mathbb{P} = (\mathbb{A}, \mathcal{C}, \Vdash)$ applies by a transition system whose states belongs to a set Σ . A state σ contains both the description of the security configuration of the policy used, written $\Upsilon(\sigma)$, and the set of current accesses done in the system, written $\Lambda(\sigma)$. Given a configuration $c \in \mathcal{C}$, we write

$$\Sigma|_c = \{\sigma \in \Sigma \mid \Upsilon(\sigma) = c \wedge c \Vdash \Lambda(\sigma)\}$$

the set of secure states according to c . We consider the language of requests $\mathcal{R} = \{\langle +, s, o, a \rangle, \langle -, s, o, a \rangle\}$, providing to the users a way to access objects: $\langle +, s, o, a \rangle$ (resp. $\langle -, s, o, a \rangle$) means that the subject s asks to get (resp. to release) an access

over the object o according to the access mode a . Then, transitions are defined by a transition function $\tau : \mathcal{R} \times \Sigma \rightarrow \mathcal{D} \times \Sigma$ (where $\mathcal{D} = \{\text{yes}, \text{no}\}$ are the answers given according to the policy). We assume here that this transition function is correct according to the policy and does not change security configurations, and that when the answer given is **no** the state is not modified:

$$\begin{aligned} \forall \sigma_1, \sigma_2 \in \Sigma \ \forall R \in \mathcal{R} \ \forall d \in \mathcal{D} \\ (\mathcal{Y}(\sigma_1) \Vdash \Lambda(\sigma_1) \wedge \tau(R, \sigma_1) = (d, \sigma_2)) \Rightarrow \mathcal{Y}(\sigma_2) \Vdash \Lambda(\sigma_2) \\ \forall \sigma_1, \sigma_2 \in \Sigma \ \forall R \in \mathcal{R} \quad \tau(R, \sigma_1) = (\text{no}, \sigma_2) \Rightarrow \sigma_1 = \sigma_2 \end{aligned}$$

We write $\sigma' = \sigma \oplus (s, o, a)$ (resp. $\sigma' = \sigma \ominus (s, o, a)$) the state obtained from σ by adding (resp. removing) the access (s, o, a) to its current accesses without changing security configuration: $\Lambda(\sigma') = \Lambda(\sigma) \cup \{(s, o, a)\}$ (resp. $\Lambda(\sigma') = \Lambda(\sigma) \setminus \{(s, o, a)\}$) and $\mathcal{Y}(\sigma) = \mathcal{Y}(\sigma')$. We assume that the transition function is correct according to this “semantics”:

$$\begin{aligned} \forall \sigma_1, \sigma_2 \in \Sigma \ \tau(\langle +, s, o, a \rangle, \sigma_1) = (\text{yes}, \sigma_2) \Rightarrow \sigma_2 = \sigma_1 \oplus (s, o, a) \\ \forall \sigma_1, \sigma_2 \in \Sigma \ \tau(\langle -, s, o, a \rangle, \sigma_1) = (\text{yes}, \sigma_2) \Rightarrow \sigma_2 = \sigma_1 \ominus (s, o, a) \end{aligned}$$

Given a set $\Sigma_I \subseteq \Sigma$ of initial secure states, we define executions of τ as follows:

$$Exec(\tau, \Sigma_I) = \bigcup_{n \in \mathbb{N}} \left\{ (\sigma_1, \dots, \sigma_n) \mid \sigma_1 \in \Sigma_I \right. \\ \left. \wedge \forall i (1 \leq i \leq n-1) \exists R \in \mathcal{R} \ \tau(R, \sigma_i) = (\text{yes}, \sigma_{i+1}) \right\}$$

Flows generated by sequences of states. During the lifetime of a system, transitions generate information flows. We introduce flow relations between entities of the system allowing to define flows generated by sequences of states (corresponding to executions of transition functions). We define subsets of $\overset{\circ\circ}{\Rightarrow} = \mathcal{O} \times \mathcal{O}$, $\overset{\circ\circ}{\Leftarrow} = \mathcal{O} \times \mathcal{S}$ and $\overset{\circ\circ}{\Rightarrow} = \mathcal{S} \times \mathcal{O}$ characterizing flows occurring in several contexts. An elementary flow of the information contained into an object o_1 to an object o_2 can occur iff there exists a subject s reading o_1 and writing into o_2 .

Definition 2. *Elementary flows generated by a set of accesses A are defined by:*

$$\mapsto_A = \left\{ o_1 \overset{\circ\circ}{\Rightarrow} o_2 \mid \exists s \in \mathcal{S} \ \{(s, o_1, \text{read}), (s, o_2, \text{write})\} \subseteq A \right\}$$

We can now define the relation $\overset{\circ\circ}{\Rightarrow}_\sigma \subseteq \overset{\circ\circ}{\Rightarrow}$ allowing to express that when the system is in a state σ , the information contained in the object o_1 flows into the object o_2 (which is written $o_1 \overset{\circ\circ}{\Rightarrow}_\sigma o_2$). Such a flow can occur iff there exists a chain of read and write accesses starting from a read access over o_1 and ending at a write access over o_2 . The flows between objects generated by the set of current accesses of σ are thus defined as the reflexive and transitive closure of $\mapsto_{\Lambda(\sigma)}$.

Definition 3. *The set of information flows generated by a state σ is denoted by $\overset{\circ\circ}{\Rightarrow}_\sigma$ and is defined by $\overset{\circ\circ}{\Rightarrow}_\sigma = \mapsto_{\Lambda(\sigma)}^*$.*

Hence, $o_1 \xrightarrow{\circ\circ}_{\sigma} o_2$ iff $o_1 = o_2$ or $\exists s_1, \dots, s_k, s_{k+1} \in \mathcal{S}, \exists o'_1, \dots, o'_k \in \mathcal{O}$ such that:

$$\left\{ \begin{array}{l} (s_1, o_1, \text{read}), (s_1, o'_1, \text{write}), (s_2, o'_1, \text{read}), (s_2, o'_2, \text{write}), \dots, \\ (s_i, o'_{i-1}, \text{read}), (s_i, o'_i, \text{write}), \dots, (s_{k+1}, o'_k, \text{read}), (s_{k+1}, o_2, \text{write}) \end{array} \right\} \subseteq \Lambda(\sigma)$$

We extend this definition for a set $E \subseteq \Sigma$ of states: $\xrightarrow{\circ\circ}_E = \bigcup_{\sigma \in E} \xrightarrow{\circ\circ}_{\sigma}$.

Example 2. By considering the HRU policy and the configuration m defined by (1), the set of flows generated by secure states is:

$$\xrightarrow{\circ\circ}_{\Sigma|m} = \left\{ \begin{array}{l} o_1 \xrightarrow{\circ\circ} o_1, o_2 \xrightarrow{\circ\circ} o_2, o_3 \xrightarrow{\circ\circ} o_3, o_4 \xrightarrow{\circ\circ} o_4, o_3 \xrightarrow{\circ\circ} o_1, o_1 \xrightarrow{\circ\circ} o_2, \\ o_2 \xrightarrow{\circ\circ} o_4, o_1 \xrightarrow{\circ\circ} o_4, o_3 \xrightarrow{\circ\circ} o_2, o_3 \xrightarrow{\circ\circ} o_4 \end{array} \right\}$$

Notice that we assume here the “worst” case: we suppose that if there is a potential for information flow then the flow actually occurs. However, it is possible to refine this definition by observing flows at the OS level. From definition 3, we can now define the relations characterizing flows generated by sequences of states.

Definition 4. Let $E \subseteq \Sigma$, and $F \subseteq E^*$ be a set of sequences of states in E . A sequence $(\sigma_1, \dots, \sigma_n) \in F$ generates several flows defined as follows.

1. Flows between objects are defined by composition as follows:

$$\xrightarrow{\circ\circ}_{(\sigma_1, \dots, \sigma_n)} = \begin{cases} \xrightarrow{\circ\circ}_{\sigma_1} & \text{if } n = 1 \\ \xrightarrow{\circ\circ}_{\sigma_{k+1}} \circ \xrightarrow{\circ\circ}_{(\sigma_1, \dots, \sigma_k)} & \text{if } n = k + 1 \end{cases}$$

2. Flows from objects to subjects characterize which subject can read (in a direct or indirect way) which information initially contained into an object:

$$\xrightarrow{\text{os}}_{(\sigma_1, \dots, \sigma_n)} = \bigcup_{i=1}^n \left\{ o_1 \xrightarrow{\text{os}} s \mid o_1 \xrightarrow{\circ\circ}_{(\sigma_1, \dots, \sigma_i)} o_2 \wedge (s, o_2, \text{read}) \in \Lambda(\sigma_i) \right\}$$

3. Flows from subjects to objects characterize which subject can write (in a direct or indirect way) into which object:

$$\xrightarrow{\text{so}}_{(\sigma_1, \dots, \sigma_n)} = \bigcup_{i=1}^n \left\{ s \xrightarrow{\text{so}} o_2 \mid (s, o_1, \text{write}) \in \Lambda(\sigma_i) \wedge o_1 \xrightarrow{\circ\circ}_{(\sigma_i, \sigma_{i+1}, \dots, \sigma_n)} o_2 \right\}$$

We extend these definitions as follows: $\xrightarrow{X}_Y = \bigcup_{y \in Y} \xrightarrow{X}_y$ where $X \in \{\text{oo}, \text{os}, \text{so}\}$ and $Y \in \{E, F\}$.

Expressing flows in a uniform way. It is possible to express all flows between subjects and objects as flows between objects. To achieve this goal, each subject $s \in \mathcal{S}$ is associated with an object $o_s \in \mathcal{O}_S$, where \mathcal{O}_S is the set of objects associated with subjects. Of course, objects occurring in the definition of the access control policy are only those belonging to \mathcal{O} . Furthermore, by following such an approach, for all state σ and all subject s , we have:

$$\{(s, o_s, \text{read}), (s, o_s, \text{write})\} \subseteq \Lambda(\sigma) \text{ and } ((s, o, a) \in \Lambda(\sigma) \wedge o \in \mathcal{O}_S) \Rightarrow o = o_s$$

Of course, for a sequence $(\sigma_1, \dots, \sigma_n)$ of states, we have:

$$\begin{aligned}\overset{\text{os}}{\rightsquigarrow}_{(\sigma_1, \dots, \sigma_n)} &= \{o_1 \overset{\text{oo}}{\rightsquigarrow}_{(\sigma_1, \dots, \sigma_n)} o_2 \mid o_1 \in \mathcal{O} \wedge o_2 \in \mathcal{O}_S\} \\ \overset{\text{so}}{\rightsquigarrow}_{(\sigma_1, \dots, \sigma_n)} &= \{o_1 \overset{\text{oo}}{\rightsquigarrow}_{(\sigma_1, \dots, \sigma_n)} o_2 \mid o_1 \in \mathcal{O}_S \wedge o_2 \in \mathcal{O}\}\end{aligned}$$

3 Flow Policies

We consider now access control over information contained into the objects of the system. This leads to view access control policies as information flows policies. A confidentiality policy defines which information can be accessed per users. This can be expressed in terms of flows from objects to subjects: a confidentiality policy can be expressed by a subset $\overset{\text{os}}{\rightsquigarrow}$ of $\overset{\text{os}}{\rightsquigarrow}$. An integrity policy defines who is allowed to modify a container of information and can be expressed by a subset $\overset{\text{so}}{\rightsquigarrow}$ of $\overset{\text{so}}{\rightsquigarrow}$. A confinement policy defines which information contained into an object can flow into another object and can be expressed by a subset $\overset{\text{oo}}{\rightsquigarrow}$ of $\overset{\text{oo}}{\rightsquigarrow}$.

Access Control over Information. An access control policy can be used to ensure confinement and/or confidentiality and/or integrity in an information system. Such policies, expressed in terms of information flows, can be defined from the access control policy as follows.

Definition 5. *Flow policies over Σ associated with a configuration $c \in \mathcal{C}$ of an access control policy $\mathbb{P} = (\mathbb{A}, \mathcal{C}, \Vdash)$ are defined as follows:*

$$\begin{aligned}\text{Confidentiality Policy: } \overset{\text{os}}{\rightsquigarrow}_{\mathbb{P}[c]} &= \{o \overset{\text{os}}{\rightsquigarrow} s \mid \exists \sigma \in \Sigma|_c (s, o, \text{read}) \in \Lambda(\sigma)\} \\ \text{Integrity Policy: } \overset{\text{so}}{\rightsquigarrow}_{\mathbb{P}[c]} &= \{s \overset{\text{so}}{\rightsquigarrow} o \mid \exists \sigma \in \Sigma|_c (s, o, \text{write}) \in \Lambda(\sigma)\} \\ \text{Confinment Policy: } \overset{\text{oo}}{\rightsquigarrow}_{\mathbb{P}[c]} &= \left\{ \begin{array}{l} o_1 \overset{\text{oo}}{\rightsquigarrow} o_2 \mid o_1 = o_2 \vee \exists \sigma \in \Sigma|_c \exists s \in S \\ (s, o_1, \text{read}) \in \Lambda(\sigma), (s, o_2, \text{write}) \in \Lambda(\sigma) \end{array} \right\}\end{aligned}$$

The whole flow policy induced by the access control is composed by the confidentiality, integrity and confinement policies. Using the expression of flows in a uniform way as defined in page 77, it is represented by:

$$\rightsquigarrow_{\mathbb{P}[c]} = \left\{ \begin{array}{l} o_1 \overset{\text{oo}}{\rightsquigarrow} o_2 \mid o_1 = o_2 \\ \vee (\exists s \in S \exists \sigma \in \Sigma|_c \quad o_2 = o_s \in \mathcal{O}_s \wedge (s, o_1, \text{read}) \in \Lambda(\sigma)) \\ \vee (\exists s \in S \exists \sigma \in \Sigma|_c \quad o_1 = o_s \in \mathcal{O}_s \wedge (s, o_2, \text{write}) \in \Lambda(\sigma)) \\ \vee (\exists s \in S \exists \sigma \in \Sigma|_c \quad (s, o_1, \text{read}), (s, o_2, \text{write}) \in \Lambda(\sigma)) \end{array} \right\}$$

Example 3. If we consider the configuration m defined in (1) for HRU, we have:

$$\begin{aligned}\overset{\text{os}}{\rightsquigarrow}_{\mathbb{P}_H[m]} &= \left\{ o_1 \overset{\text{os}}{\rightsquigarrow} \text{Alice}, o_3 \overset{\text{os}}{\rightsquigarrow} \text{Alice}, o_1 \overset{\text{os}}{\rightsquigarrow} \text{Bob}, o_2 \overset{\text{os}}{\rightsquigarrow} \text{Bob}, o_2 \overset{\text{os}}{\rightsquigarrow} \text{Charlie} \right\} \\ \overset{\text{so}}{\rightsquigarrow}_{\mathbb{P}_H[m]} &= \left\{ \text{Alice} \overset{\text{so}}{\rightsquigarrow} o_1, \text{Bob} \overset{\text{so}}{\rightsquigarrow} o_2, \text{Charlie} \overset{\text{so}}{\rightsquigarrow} o_2, \text{Charlie} \overset{\text{so}}{\rightsquigarrow} o_4 \right\} \\ \overset{\text{oo}}{\rightsquigarrow}_{\mathbb{P}_H[m]} &= \left\{ o_1 \overset{\text{oo}}{\rightsquigarrow} o_1, o_3 \overset{\text{oo}}{\rightsquigarrow} o_1, o_1 \overset{\text{oo}}{\rightsquigarrow} o_2, o_2 \overset{\text{oo}}{\rightsquigarrow} o_2, o_2 \overset{\text{oo}}{\rightsquigarrow} o_4, o_3 \overset{\text{oo}}{\rightsquigarrow} o_3, o_4 \overset{\text{oo}}{\rightsquigarrow} o_4 \right\}\end{aligned}$$

Access Control over Objects Versus over Information. We can now define in a formal way the properties expressing that a set of flows (occurring for a set of states E or occurring during sequences of states in a set F) is consistent according to the flow policies obtained from the access control policy.

Definition 6. Let $\mathbb{P} = (\mathbb{A}, \mathcal{C}, \vdash)$ be an access control policy, $c \in \mathcal{C}$, and X be a set of states or a set of sequences of states. X is said to be consistent according to:

- the confidentiality policy $\overset{\text{os}}{\rightsquigarrow}_{\mathbb{P}[c]}$ iff $\overset{\text{os}}{\hookrightarrow} X \subseteq \overset{\text{os}}{\rightsquigarrow}_{\mathbb{P}[c]}$,
- the integrity policy $\overset{\text{so}}{\rightsquigarrow}_{\mathbb{P}[c]}$ iff $\overset{\text{so}}{\hookrightarrow} X \subseteq \overset{\text{so}}{\rightsquigarrow}_{\mathbb{P}[c]}$,
- the confinement policy $\overset{\text{oo}}{\rightsquigarrow}_{\mathbb{P}[c]}$ iff $\overset{\text{oo}}{\hookrightarrow} X \subseteq \overset{\text{oo}}{\rightsquigarrow}_{\mathbb{P}[c]}$,
- the information flow policy $\rightsquigarrow_{\mathbb{P}[c]}$ iff X is consistent according to $\overset{\text{os}}{\rightsquigarrow}_{\mathbb{P}[c]}$, $\overset{\text{so}}{\rightsquigarrow}_{\mathbb{P}[c]}$ and $\overset{\text{oo}}{\rightsquigarrow}_{\mathbb{P}[c]}$.

These properties are useful to analyze flows generated by secure states of access control policies ($X = \Sigma|_m$) or flows generated by executions ($X = Exec(\tau, \Sigma_I)$).

Example 4. For HRU, there exists configurations m for which the sets of flows occurring during sequences in $Exec(\tau_H, \Sigma_H^I)$ are neither consistent according to $\overset{\text{os}}{\rightsquigarrow}_{\mathbb{P}_H[m]}$ nor to $\overset{\text{so}}{\rightsquigarrow}_{\mathbb{P}_H[m]}$ nor to $\overset{\text{oo}}{\rightsquigarrow}_{\mathbb{P}_H[m]}$. Indeed, if we consider the configuration m defined in (1), we have:

$$o_3 \overset{\text{os}}{\hookrightarrow}_{Exec(\tau_H, \Sigma_H^I)} \text{Bob}, \text{Bob} \overset{\text{so}}{\hookrightarrow}_{Exec(\tau_H, \Sigma_H^I)} o_4, o_3 \overset{\text{oo}}{\hookrightarrow}_{Exec(\tau_H, \Sigma_H^I)} o_2$$

but we don't have:

$$o_3 \overset{\text{os}}{\rightsquigarrow}_{\mathbb{P}_H[m]} \text{Bob}, \text{Bob} \overset{\text{so}}{\rightsquigarrow}_{\mathbb{P}_H[m]} o_4, o_3 \overset{\text{oo}}{\rightsquigarrow}_{\mathbb{P}_H[m]} o_2$$

Of course, when m satisfies some “good” properties (transitivity), the consistency properties hold. More precisely, we have proved that:

1. $\overset{\text{os}}{\hookrightarrow}_{Exec(\tau_H, \Sigma_H^I)}$ is consistent according to $\overset{\text{os}}{\rightsquigarrow}_{\mathbb{P}_H[m]}$ iff:

$$\forall s \in \mathcal{S} \forall o_1, o_2 \in \mathcal{O} \quad (o_1 \overset{\text{oo}}{\rightsquigarrow}_{\mathbb{P}[m]} o_2 \wedge (s, o_2, \text{read}) \in m) \Rightarrow (s, o_1, \text{read}) \in m$$

2. $\overset{\text{so}}{\hookrightarrow}_{Exec(\tau_H, \Sigma_H^I)}$ is consistent according to $\overset{\text{so}}{\rightsquigarrow}_{\mathbb{P}_H[m]}$ iff:

$$\forall s \in \mathcal{S} \forall o_1, o_2 \in \mathcal{O} \quad (o_1 \overset{\text{oo}}{\rightsquigarrow}_{\mathbb{P}[m]} o_2 \wedge (s, o_1, \text{write}) \in m) \Rightarrow (s, o_2, \text{write}) \in m$$

3. $\overset{\text{oo}}{\hookrightarrow}_{Exec(\tau_H, \Sigma_H^I)}$ is consistent according to $\overset{\text{oo}}{\rightsquigarrow}_{\mathbb{P}_H[m]}$ iff $\overset{\text{oo}}{\rightsquigarrow}_{\mathbb{P}_H[m]} = (\overset{\text{oo}}{\rightsquigarrow}_{\mathbb{P}_H[m]})^*$ (i.e. $\overset{\text{oo}}{\rightsquigarrow}_{\mathbb{P}_H[m]}$ and its reflexive and transitive closure define the same relation)

This framework has been successfully used to analyse flow properties of several classical access control policies: the Bell & LaPadula policy [1] which has been (not surprisingly) proved consistent, the Chinese Wall model [2] which has been proved consistent according to the confidentiality and integrity flow policies, and the role-based access control model (RBAC) [5,16] which is not consistent.

4 Detecting Illegal Information Flows

As we said, access control mechanisms are specified in terms of granted accesses but don't always make explicit the information flows that they are supposed to permit. In fact, the notion of granted flows depends on the interpretation of the access control model. As shown in the above section, the flow based interpretation of an access control policy may forbid flows that can occur when applying a classical access control mechanism. We introduce here a flow detection mechanism allowing to deal with such problems. We define what a flow detection mechanism consists in and what properties we can express over such mechanism. Then, in this setting, we provide theoretical foundations for the intrusion detection systems introduced in [6,8,18,19], by formalizing it as a detection flow mechanism together with proofs of its soundness and completeness. Last, we show how such mechanism can be used in practice for the HRU model.

Specification of Flow Detection Mechanisms. We show here how to specify mechanisms allowing to detect illegal information flows according to a flow policy \rightsquigarrow during the lifetime of an information system. Hence, we introduce the formal specification of flow detection mechanisms allowing to monitor an information system in order to detect flows occurring in a set $\rightsquigarrow_{\mathbb{F}}$ of forbidden flows during the lifetime of the system. Such mechanisms are based on the definition of a predicate \mathcal{U} over states characterizing states occurring in a sequence of states generating at least a flow in $\rightsquigarrow_{\mathbb{F}}$. A state σ verifying $\mathcal{U}(\sigma)$ is called an alert state. The set of observable sequences of states can be the set of executions of an operational mechanism of an access control policy.

Definition 7. A flow detection mechanism parameterized by a set E of observable states, a set $F \subseteq E^*$ of observable sequences of states and a set $\rightsquigarrow_{\mathbb{F}}$ of flows is denoted by $\mathbb{F}[E, F, \rightsquigarrow_{\mathbb{F}}]$ and is defined by (Σ, \mathcal{U}) where Σ is the set of states of the system and \mathcal{U} is a predicate characterizing alert states.

We also introduce the properties over flow detection mechanisms allowing to express that alert states are exactly states occurring in a sequence of states generating at least a flow in $\rightsquigarrow_{\mathbb{F}}$. In the following definition, $\rightsquigarrow_{\mathbb{F}}$ can specify a set of flows between objects ($X = \text{oo}$), from subjects to objects ($X = \text{so}$), or from objects to subjects ($X = \text{os}$).

Definition 8. Let $\mathbb{F}[E, F, \rightsquigarrow_{\mathbb{F}}] = (\Sigma, \mathcal{U})$ be a flow detection mechanism.

- $\mathbb{F}[E, F, \rightsquigarrow_{\mathbb{F}}]$ is sound iff each state σ verifying $\mathcal{U}(\sigma)$ is coming from a sequence in F generating at least one flow occurring in $\rightsquigarrow_{\mathbb{F}}$:

$$\forall(\sigma_1, \dots, \sigma_n) \in F \quad \mathcal{U}(\sigma_n) \Rightarrow (\overset{X}{\rightsquigarrow}_{(\sigma_1, \dots, \sigma_n)} \cap \rightsquigarrow_{\mathbb{F}} \neq \emptyset)$$

- $\mathbb{F}[E, F, \rightsquigarrow_{\mathbb{F}}]$ is complete iff each sequence in F generating a flow occurring in $\rightsquigarrow_{\mathbb{F}}$ ends with an alert state.

$$\forall(\sigma_1, \dots, \sigma_n) \in F \quad (\overset{X}{\rightsquigarrow}_{(\sigma_1, \dots, \sigma_n)} \cap \rightsquigarrow_{\mathbb{F}} \neq \emptyset) \Rightarrow \mathcal{U}(\sigma_n)$$

Of course, the parameter $\rightsquigarrow_{\mathbb{F}}$ of a flow detection mechanism $\mathbb{F}[E, F, \rightsquigarrow_{\mathbb{F}}]$ used to check that a flow policy \rightsquigarrow is satisfied must be such that:

- granted flows according to \rightsquigarrow are not detected by the detection mechanism:

$$\rightsquigarrow_{\mathbb{F}} \cap \rightsquigarrow = \emptyset \quad (2)$$

- flows that may occur and that are not authorized by \rightsquigarrow are detected:

$$(\overset{X}{\leftarrow}_F \setminus \rightsquigarrow) \subseteq \rightsquigarrow_{\mathbb{F}} \quad (3)$$

Thanks to these properties, for a sound and complete flow detection mechanism $\mathbb{F}[E, F, \rightsquigarrow_{\mathbb{F}}]$, alert states are exactly states coming from a sequence generating at least one flow that do not respect the flow policy \rightsquigarrow .

Definition of a Flow Detection Mechanism. We use the uniform representation of flows introduced page 77 and define a flow detection mechanism allowing to detect flows occurring during sequences of states that do not satisfy a reflexive information flow policy \rightsquigarrow . We use the relation $\rightarrow \subseteq \mathcal{P}(\mathcal{O}) \times \mathcal{O}$ such that $\{o_1, \dots, o_n\} \rightarrow o$ expresses that information initially contained in o_1, \dots, o_n are allowed to flow separately or together into o . The relation \rightarrow is defined as follow:

$$\rightarrow = \bigcup_{o \in \mathcal{O}} (\{o_i | o_i \rightsquigarrow o\}, o)$$

Observable sequences of states. The flow detection mechanism we define here is generic: it is parameterised by a set E of observable states and by a set $F \subseteq E^*$ of observable sequences of states such that for all sequences $(\sigma_1, \dots, \sigma_n) \in F$, the initial state σ_1 has an empty set of current accesses ($\Lambda(\sigma_1) = \emptyset$), and such that the state σ_{i+1} is obtained from σ_i either by adding or by removing an access:

$$\forall (\sigma_1, \dots, \sigma_n) \in F \quad \forall i \quad \sigma_{i+1} = \sigma_i \oplus (s, o, a) \vee \sigma_{i+1} = \sigma_i \ominus (s, o, a)$$

Definition of the set of forbidden information flows $\rightsquigarrow_{\mathbb{F}}$. Since we want to detect flows that do not satisfy the policy \rightsquigarrow , we define the set of forbidden information flows $\rightsquigarrow_{\mathbb{F}}$ as the set of all possible information flows except those in the policy:

$$\rightsquigarrow_{\mathbb{F}} = \overset{\circ\circ}{\leftarrow}_F \setminus \rightsquigarrow$$

Definition of alert states. The flow detection mechanism we define here is a tagging system for which we prove that it permits a sound and complete detection of illegal information flows defined in $\rightsquigarrow_{\mathbb{F}}$. For a state σ , and for each object $o \in \mathcal{O}$, we define a reading tag $T_{\sigma}^R(o)$ and a writing tag $T_{\sigma}^W(o)$. The reading tag denotes the policy related to the information really contained in the object. The writing tag denotes the policy related to the object viewed as a container of information. These tags are defined as follows:

- For all sequence $(\sigma_1, \dots, \sigma_n) \in F$ and for all $o \in \mathcal{O}$, the tags are initially defined for σ_1 as follows. The tag $T_{\sigma_1}^R(o)$ attached to o denotes the part of the flow policy \rightarrow related to the information initially contained in o :

$$T_{\sigma_1}^R(o) = \{(\mathcal{O}, o') \in \rightarrow \mid o \in \mathcal{O}\}$$

The tag $T_{\sigma_1}^W(o)$ attached to o denotes the part of the information flow policy \rightarrow related to o where o is viewed as a container of information:

$$T_{\sigma_1}^W(o) = \{(O, o') \in \rightarrow \mid o = o'\}$$

- At each transition from σ_i to σ_{i+1} , the writing tags don't change ($T_{\sigma_i}^W(o) = T_{\sigma_{i+1}}^W(o)$) and the reading tags of objects evolve in the following way:

$$T_{\sigma_{i+1}}^R(o) = \bigcap_{\{o_j \in \mathcal{O} \mid o_j \xrightarrow{\circ\circ} \sigma_{i+1} o\}} T_{\sigma_i}^R(o_j)$$

Notice that if o has not been modified by a flow then $\{o_j \in \mathcal{O} \mid o_j \xrightarrow{\circ\circ} \sigma_{i+1} o\}$ is simply $\{o\}$ (see definition 3) and the tag $T_{\sigma_{i+1}}^R(o)$ is exactly $T_{\sigma_i}^R(o)$.

The following lemma (proved by induction over n) states that $T_{\sigma}^R(o)$ exactly denotes the part of the policy \rightarrow shared by all information contained in o .

Lemma 1. $\forall (\sigma_1, \sigma_2, \dots, \sigma_n) \in F \ \forall o \in \mathcal{O}$

$$T_{\sigma_n}^R(o) = \{(O, o') \in \rightarrow \mid \forall o_i \in \mathcal{O} \ o_i \xrightarrow{\circ\circ}_{(\sigma_1, \dots, \sigma_n)} o \Rightarrow o_i \in O\}$$

We are now in position to define the predicate \mathcal{U} as follows:

$$\mathcal{U}(\sigma) \Leftrightarrow \exists o \in \mathcal{O} \ T_{\sigma}^R(o) \cap T_{\sigma}^W(o) = \emptyset$$

which allows to characterize alert states and leads to define a sound and complete flow detection mechanism (definition 8) as shown by the following lemma.

Lemma 2. $\mathbb{F}[E, F, \rightsquigarrow_{\mathbb{F}}] = (\Sigma, \mathcal{U})$ is both sound and complete.

Proof. The proof is done by contraposition, we prove that in a sequence of states $(\sigma_1, \dots, \sigma_n) \in F$ no illegal information flows has occurred if and only if for any object o the intersection between read and write tags is non-empty in σ_n (e.g. σ_n is not an alert state):

$$\forall (\sigma_1, \dots, \sigma_n) \in F \ \xrightarrow{\circ\circ}_{(\sigma_1, \dots, \sigma_n)} \cap \rightsquigarrow_{\mathbb{F}} = \emptyset \Leftrightarrow \forall o \in \mathcal{O} \ T_{\sigma_n}^R(o) \cap T_{\sigma_n}^W(o) \neq \emptyset$$

Let $(\sigma_1, \dots, \sigma_n) \in F$. According to (2) and (3) and the definition of $\rightsquigarrow_{\mathbb{F}}$, no illegal flow has occurred during $(\sigma_1, \dots, \sigma_n)$ iff only allowed flows has occurred:

$$\begin{aligned} & \xrightarrow{\circ\circ}_{(\sigma_1, \dots, \sigma_n)} \cap \rightsquigarrow_{\mathbb{F}} = \emptyset \\ \Leftrightarrow & \xrightarrow{\circ\circ}_{(\sigma_1, \dots, \sigma_n)} \subseteq \rightsquigarrow \\ \Leftrightarrow & \forall o \in \mathcal{O} \ \forall o' \in \mathcal{O} \ o' \xrightarrow{\circ\circ}_{(\sigma_1, \dots, \sigma_n)} o \Rightarrow o' \rightsquigarrow o \\ \Leftrightarrow & \forall o \in \mathcal{O} \ \forall o' \in \mathcal{O} \ o' \xrightarrow{\circ\circ}_{(\sigma_1, \dots, \sigma_n)} o \Rightarrow \exists (O, o) \in \rightarrow, \ o' \in O \text{ (by def. of } \rightarrow) \\ \Leftrightarrow & \forall o \in \mathcal{O} \ \left\{ (O, o) \in \rightarrow \mid \forall o' \in \mathcal{O} \ o' \xrightarrow{\circ\circ}_{(\sigma_1, \dots, \sigma_n)} o \Rightarrow o' \in O \right\} \neq \emptyset \end{aligned}$$

Hence, by lemma 1 and by definition of $T_{\sigma_n}^W(o)$, we have:

$$\forall o \in \mathcal{O} \ T_{\sigma_n}^R(o) \cap T_{\sigma_n}^W(o) = \left\{ (O, o) \in \rightarrow \mid \forall o_j \in \mathcal{O} \ o_j \xrightarrow{\circ\circ}_{(\sigma_1, \dots, \sigma_n)} o \Rightarrow o_j \in O \right\}$$

which allows to conclude, by definition. \triangleleft

Application. We describe now how to use our flow detection mechanism to detect illegal flows (w.r.t. $\rightsquigarrow_{\mathbb{P}_H[m]}$) generated by executions of $Exec(\tau_H, \Sigma_H^I)$. Parameters are instantiated as follows: E is the set $\Sigma|_m$ of secure states and F is the set $Exec(\tau_H, \Sigma_H^I)$ where $\Sigma_H^I \subseteq \{\sigma \in \Sigma \mid \Lambda(\sigma) = \emptyset\}$ and \rightsquigarrow is the relation $\rightsquigarrow_{\mathbb{P}_H[m]}$. According to lemma 2, alert states are exactly states coming from sequences of states generating flows that do not satisfy $\rightsquigarrow_{\mathbb{P}_H[m]}$.

Example 5. Let us consider again the HRU policy applied with the configuration m defined in (1). This policy induces an information flow policy \rightarrow_H defined as follows (the objects o_A, o_B, o_C stand for the objects associated with the subjects Alice, Bob and Charlie):

$$\rightarrow_H = \left\{ \left((\{o_1, o_3, o_A\}, o_1), (\{o_1, o_2, o_B\}, o_2), (\{o_2, o_C\}, o_2), (\{o_3\}, o_3), \right. \right. \\ \left. \left. (\{o_2, o_4, o_c\}, o_4), (\{o_1, o_3, o_A\}, o_A), (\{o_1, o_2, o_B\}, o_B), (\{o_2, o_C\}, o_C) \right) \right\}$$

Let $(\sigma_1, \sigma_2, \sigma_3, \sigma_4)$ be a sequence of states such that σ_1 has an empty set of current accesses, σ_2 is obtained by adding a read access on o_3 to Alice, σ_3 by adding a write access on o_1 to Alice and σ_4 by adding a read access on o_1 to Bob. None of the states σ_1, σ_2 and σ_3 are alert states, while σ_4 is an alert state since $T_{\sigma_4}^R(o_B) \cap T_{\sigma_4}^W(o_B) = \{(\{o_1, o_3, o_3\}, o_1), (\{o_3\}, o_3), (\{o_1, o_3, o_A\}, o_A)\} \cap \{(\{o_1, o_2, o_B\}, o_B)\} = \emptyset$.

5 Implementation

Detecting Illegal Information Flows at the OS Level. A first implementation of the tagging system, called Blare¹, has been realised at the OS level. Blare deduces information flows between typical OS containers (files, sockets or IPC) by observing system calls. Blare is implemented in the Linux kernel, with extra userland tools and associated API library. Figure 1 presents the general architecture of Blare. In order to detect accesses to the OS containers, hooks have been inserted in the kernel to avoid system call overloading. These hooks are principally located in the *Virtual File System* (VFS) layer of the Linux kernel. Linux provides access to containers through the file concept and the VFS is an abstraction of various container types in terms of files. In addition, some containers require operations outside of the scope of the VFS, such as socket connections or virtual consoles. Kernel code implementing these functions is thus also modified. The detection of illegal information flows engine features can be classified into four categories:

1. the *core detection engine* implements the detection algorithm and the tag propagation mechanism;
2. the *configuration management component* allows user space processes to manage the detection engine or to check some information such as the total number of alerts (this is done using the Linux sysfs interface and is useful for debugging or alert diagnosis, but can be locked for security reasons);

¹ Freely available at <http://www.rennes.supelec.fr/blare/>

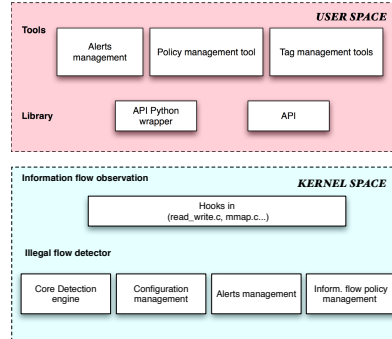


Fig. 1. Architecture of the illegal flow detector at the OS level (Blare)

3. the *information flow management component* allows users to specify a security policy \rightarrow ;
4. the *alert management component* increments the alert counter, notifies a userland tool whenever an alert is raised and can optionally and directly log alerts and diagnosis information into a file.

Tags are implemented by a specific structure containing two fixed size bitmap arrays, describing the reading and writing tags. Thanks to this data structure, checking the legality of flows has a small slowdown. In [8], the flow policy \rightarrow is computed from a free interpretation of access control rights, but we have also proposed to set \rightarrow manually [18] or to compute it from a MAC policy [6].

Detecting Illegal Information Flows within the JVM. Blare suffers from one major drawback: the semantics of the operations are not taken into account by the detection system. Hence, processes are viewed as black boxes: the reading tags that are modified by a syscall depend on *all* the reading tags of the objects that act as input for this syscall. This approximation is correct, but its coarse-grained nature sometimes leads Blare to trigger false positives. This reason has led to a second implementation, JBlare [8], refining the view of the OS implementation. Information flows are here observed at the language level (Java). The containers of information are variables and object attributes. Flows between these containers are those observed through method calls or affectation of variables. Java types are either primitive types or reference types and we only consider objects whose types are primitive data types (`int`, `char`, ...). We consider reference data types (such as classes or arrays) as an aggregation of primitive data types. The main goal of JBlare is to refine the computation of reading tags used by Blare. JBlare thus associates a reading tag to any variable whose type is a primitive type. JBlare reading tags are implemented as Java objects. The wrapper class `blareTag` encodes these tags as bitmap structures compatible with the Blare OS implementation. Whenever an OS object is read, corresponding tags are created on the corresponding java object. Tags are updated when a field is accessed or a method is called. Reading or modifying a

field involves a flow from the read field to the current method local variables or vice-versa. Method calls are handled according to the method signatures.

A Two-Level Cooperative Detector. Cooperation between JBlare and Blare is achieved by a class on the Java side which detects when a Java program performs some accesses to OS objects such as files or sockets. On the one hand, JBlare language level tags are initialized from Blare OS-level tags. For example when a file is read, the reading tag maintained by Blare is propagated to JBlare through the read security tag of the first instrumented method in the stack frame. On the other hand, JBlare security tags are propagated to Blare tags. For example, a write access to a file implies the propagation from JBlare to Blare of the reading tag related to the method accessing the file.

6 Conclusion

In this paper, we have proposed a formal framework allowing to study different access control and information flow policies. From an access control policy, we have defined three flows policies: a confinement policy which is the set of authorized information flows between objects, a confidentiality policy which is the set of authorized flows from objects to subjects, and an integrity policy which is the set of authorized flows from subjects to objects. Then, we have introduced the consistency property expressing that flows occurring during the life of a system implementing an access control policy are authorized by these flows policies. For example, this property holds for the Bell & LaPadula policy but is not satisfied by the HRU policy. Thus we have elaborated a general mechanism dedicated to illegal information flow detection (security tags propagation and evaluation of a predicate over the tags associated with a state). This mechanism can be used to detect attacks generating illegal information flows in a system (for example a HRU-like system such as Linux), leading actually to an intrusion detection system (IDS) that aims at both completeness and soundness, provided that information flows that violate the policy are detectable and discernible by the IDS. We have presented two implementations of our detection model in two concrete IDSes at two levels of granularity. The first IDS, Blare, is a patch of the Linux kernel and works at the OS level. The second IDS, JBlare, is a Java class that performs bytecode instrumentation on the classes of Java applications; this allows the modified application, once loaded in the JVM, to track the information flow resulting from its own execution. We now plan to extend our formalism to deal with two main features. First, we'd like to add the execute access mode in order to take into account executions of algorithms that can generate flows between its inputs and outputs. We also aim to consider systems for which the configurations can be modified (under the control of an administrative policy).

References

1. Bell, D., LaPadula, L.: Secure Computer Systems: a Mathematical Model. Technical Report MTR-2547 (Vol. II), MITRE Corp., Bedford, MA (May 1973)
2. Brewer, D.F.C., Nash, M.J.: The chinese wall security policy. In: Proc. IEEE Symposium on Security and Privacy, pp. 206–214 (1989)

3. Denning, D.E.: A lattice model of secure information flow. *Commun. ACM* 19(5), 236–243 (1976)
4. Denning, D.E., Denning, P.J.: Certification of programs for secure information flow. *Communications of the ACM* 20(7), 504–513 (1977)
5. Ferraiolo, D.F., Kuhn, D.R.: Role-based access control. In: *Proceedings of the 15th National Computer Security Conference* (1992)
6. Geller, S., Hauser, C., Tronel, F., Viet Triem Tong, V.: Information flow control for intrusion detection derived from mac policy. In: *IEEE International Conference on Communications, ICC 2011* (2011)
7. Harrison, M., Ruzzo, W., Ullman, J.: Protection in operating systems. *Communications of the ACM* 19, 461–471 (1976)
8. Hiet, G., Viet Triem Tong, V., Mé, L., Morin, B.: Policy-based intrusion detection in web applications by monitoring java information flows. In: *3rd International Conference on Risks and Security of Internet and Systems, CRiSIS 2008* (2008)
9. Jaume, M.: Security Rules *versus* Security Properties. In: Jha, S., Mathuria, A. (eds.) *ICISS 2010*. LNCS, vol. 6503, pp. 231–245. Springer, Heidelberg (2010)
10. Ko, C., Redmond, T.: Noninterference and intrusion detection. In: *IEEE Symposium on Security and Privacy*, pp. 177–187 (2002)
11. Myers, A.C.: Jflow: Practical mostly-static information flow control. In: *Proceedings of the 26th ACM on Principles of Programming Languages* (1999)
12. Myers, A.C., Liskov, B.: Complete safe information flow with decentralized labels. In: *IEEE Symposium on Security and Privacy* (1998)
13. Myers, A.C., Liskov, B.: A decentralized model for information flow control. *SIGOPS Oper. Syst. Rev.* 31(5), 129–142 (1997)
14. Osborn, S.L.: Information flow analysis of an RBAC system. In: *7th ACM Symposium on Access Control Models and Technologies SACMAT*, pp. 163–168 (2002)
15. Sandhu, R.S.: Lattice-Based Access Control Models. *IEEE Computer* 26(11), 9–19 (1993)
16. Sandhu, R.S., Coyne, E.J., Feinstein, H.L., Youman, C.E.: Role-based access control models. *IEEE Computer* 29(2), 38–47 (1996)
17. Schneider, F.B.: Enforceable security policies. *Information and System Security* 3(1), 30–50 (2000)
18. Viet Triem Tong, V., Clark, A., Mé, L.: Specifying and enforcing a fined-grained information flow policy: Model and experiments. *Journal of Wireless Mobile Networks, Ubiquitous Computing and Dependable Applications, JOWUA* (2010)
19. Zimmermann, J., Mé, L., Bidan, C.: An Improved Reference Flow Control Model for Policy-Based Intrusion Detection. In: Snekkenes, E., Gollmann, D. (eds.) *ESORICS 2003*. LNCS, vol. 2808, pp. 291–308. Springer, Heidelberg (2003)