Automated Detection of Information Leakage in Access Control

Charles Morisset[1] and Anderson Santana de Oliveira[2]

[1] SPI - LIP6 - Université Paris 6
104 av. du Président Kennedy
F-75016 Paris, France
charles.morisset@lip6.fr
[2] INRIA & LORIA, campus scientifique, BP 239
F-54506 Vandœuvre lès Nancy, France
santana@loria.fr

Abstract. The prevention of information flow is an important concern in several access control models. Even though this property is stated in the model specification, it is not easy to verify it in the actual implementation of a given security policy. In this paper we model-check rewrite-based implementations of access control policies. We propose a general algorithm that allows one to automatically identify information leakage. We apply our approach to the well-known security model of Bell and LaPadula and show that its generalization proposed by McLean does not protect a system against information leakage.

Keywords : Access Control, Bell and LaPadula, Term Rewriting, Model-Checking, Information Flow

1 Introduction - Motivations

Data protection within information systems has become a major problem during the last years with the growth and the decentralization of computer systems. One of the key issues is to guarantee that information can be accessed and/or modified only by authorized users. A solution to this problem is to enforce an access control policy within the system, which specifies the requests for access that should be granted or denied. Moreover, an access control mechanism should also prevent leak of information. For instance, a user must be denied to copy a sensitive file, such as /etc/shadow in the UNIX file system, to a file with less restrictive rights. The mechanisms used to enforce this kind of policy can involve several methods, from abstract state machines, term rewriting systems, to static information flow analysis, etc. The latter approach consists in statically finding out every flow of data inside the code of a program, and then to avoid the ones considered to violate the security policy. This was done, for instance, by using some extension of λ-calculus as in [HR98], or directly using extensions of programming languages as in [Mye99]. Other works also use term rewriting

systems [BF06,dO07] to model access control, but they are slightly different since a rewrite system can specify both the security policy and the state transitions.

Several frameworks for defining access control assume that the security policy specifying the authorized accesses is correctly defined. Indeed, they suppose it is not possible to circumvent the access control mechanisms, and there does not exist a sequence of authorized operations which can lead to a forbidden situation.

In this paper, we show that this is not always the case. We propose a method in which policy specification, given as a state machine, is separated from its implementation, given as a term rewriting system. Then we present a method based on model-checking for automatically discovering information leakages, which uses specific rules to explicit the potential information flow. We show the usefulness of the technique by applying it on an example of policy that allows the leak of information [McL88].

This paper is organized as follows. Sect. 2 presents the model of Bell and LaPadula and its implementation using both state machines and term rewriting systems. Sect. 3 introduces the McLean model, seen as a generalization of Bell and LaPadula, together with the demonstration that this model allows a leak of information. Then, in Sect. 4 we define an algorithm based on model-checking to detect such leaks.

In this paper, we adopt the usual definitions and notation for term rewriting systems contained in [Ter02,BN98], which we assume the reader to be familiar with.

2 The Bell and LaPadula Security Model and its Rewrite-Based Implementation

2.1 A framework for access control policies

In order to express access control policies, we use the mathematical framework defined in [JM06]. We do not intend to present here the whole framework, but rather the notations used in the definition of an access control policy.

An access control policy is defined as:

$$\mathbb{P}[\rho] = (\mathcal{S}, \mathcal{O}, \mathcal{A}, \Sigma, \Omega)$$

where $\mathcal{S}$ is a set of subjects (the active entities), $\mathcal{O}$ is a set of objects (the passive entities), ρ is security information (any relevant information necessary to the definition of the security policy), $\mathcal{A}$ is the set of access modes (`read`, `write`), Σ is a set of states of the system where the policy is enforced, and Ω is the security predicate, defining the secure states. Roughly speaking, an access control policy specifies the system states, and which of the states are secure. We then need to specify how to move from one state to another. So we introduce the notion of access control model:

$$\mathbb{M}[\rho] = (\mathbb{P}[\rho], \llbracket \mathcal{R} \rrbracket_{\Sigma})$$

where $\mathbb{P}[\rho]$ is a security policy, $\mathcal{R}$ is a set of requests and $\llbracket\mathcal{R}\rrbracket_{\Sigma} \subseteq \mathcal{R}\times\Sigma$ is the semantics of requests. This relation allows to express the modification expected by the application of a request. For instance, a request R asking to add the access A should have a semantics defined such that for all $\sigma \in \Sigma$, if $(R,\sigma) \in \llbracket\mathcal{R}\rrbracket_{\Sigma}$, then $A \in \sigma$.

Finally, an implementation of a model $\mathbb{M}[\rho]$ is a pair (τ, Σ_I), where $\tau : \mathcal{R} \times \Sigma \rightarrow \mathcal{D} \times \Sigma$ is a transition function over states (and $\mathcal{D}$ is a set of decisions such yes or no) and Σ_I is a set of initial states.

2.2 The Bell and LaPadula model

The Bell and LaPadula model [LB96,BL73] was originally defined for the military domain. The main idea is that every subject and every object is associated with a security level, such as *Top Secret, Secret*, etc. The original model deals with discretionary access control (in order to access an object, any subject needs the authorization from the owner of the object) and with mandatory access control (an access is granted or not according to the security level of the subject and of the object). Since discretionary access control can be seen as a completely separated part of the policy, we have decided, without loss of generality, to focus only on the mandatory part in this paper, which is really specific to the Bell and LaPadula model.

More formally, we write $\rho_{BLP} = (\mathcal{L}, \preceq, \sqcup, \sqcap)$ the *lattice of security levels*, where $\mathcal{L}$ is a set of security levels, $\preceq$ is the order defined over this set, $\sqcup$ is the least upper bound and $\sqcap$ is the greatest lower bound. Then, a state $\sigma \in \Sigma_{BLP}$ is a tuple $\sigma = (m, f_s, f_o)$ where m is the set of current accesses and $f_s : \mathcal{S} \rightarrow \mathcal{L}$ (resp. $f_o : \mathcal{O} \rightarrow \mathcal{L}$) defines levels of security associated with subjects (resp. objects). An access is represented by a tuple (s, o, x) expressing that a subject s has an access over the object o according to access mode x.

The *Bell and LaPadula security policy* is specified by a predicate Ω_{BLP} as follows. Given a state $\sigma = (m, f_s, f_o)$, $\Omega_{BLP}(\sigma)$ holds iff the following two properties are satisfied:

$$\forall s \in \mathcal{S}\ \forall o \in \mathcal{O} \quad (s, o, \texttt{read}) \in m \Rightarrow f_o(o) \preceq f_s(s) \tag{1}$$

$$\forall s \in \mathcal{S}\ \forall o_1, o_2 \in \mathcal{O} \quad (s, o_1, \texttt{read}) \in m \wedge (s, o_2, \texttt{write}) \in m \Rightarrow f_o(o_1) \preceq f_o(o_2) \tag{2}$$

The property (2), also known as the $\star$-security property, allows to prevent the copy of an object to a lower security level by a malicious subject. We write $\mathbb{P}_{BLP}[\rho_{BLP}] = (\mathcal{S}, \mathcal{O}, \mathcal{A}, \Sigma_{BLP}, \Omega_{BLP})$ such an access control policy.

The set $\mathcal{R}$ of requests is defined as follows:

$$\mathcal{R} = \{\langle +, s, o, \texttt{read}\rangle, \langle +, s, o, \texttt{write}\rangle, \langle -, s, o, \texttt{read}\rangle, \langle -, s, o, \texttt{write}\rangle\} \tag{3}$$

The request $\langle +, s, o, x\rangle$ (resp. $\langle -, s, o, x\rangle$) is the request where the subject s asks for getting (resp. releasing) the access to the object o according to access mode x. The semantics of requests $\llbracket\mathcal{R}\rrbracket_{\Sigma}$ is defined as follows:

$$\begin{array}{l} (\langle +, s, o, x\rangle, \sigma) \in \llbracket\mathcal{R}\rrbracket_{\Sigma} \Leftrightarrow (s, o, x) \in \Lambda(\sigma) \\ (\langle -, s, o, x\rangle, \sigma) \in \llbracket\mathcal{R}\rrbracket_{\Sigma} \Leftrightarrow (s, o, x) \notin \Lambda(\sigma) \end{array} \tag{4}$$

where $x \in \{\texttt{read}, \texttt{write}\}$ and $\Lambda(\sigma)$ stands for the set of current accesses of σ.

We write $\mathbb{M}_{BLP}[\rho_{BLP}] = (\mathbb{P}_{BLP}[\rho_{BLP}], \|\mathcal{R}\|_{\Sigma})$ the Bell and LaPadula access control model.

Given a set of initial states Σ_I^{BLP}, we introduce the implementation $(\tau_{BLP}, \Sigma_I^{BLP})$ of $\mathbb{M}_{BLP}[\rho_{BLP}]$ where τ_{BLP} is the transition function defined in Table 1 and corresponds to the restriction of the original function of Bell and LaPadula, without discretionary access control and only taking into account `read` and `write` access modes, since the access modes `append`, `control` and `execute` are not relevant in mandatory access control.

$$\tau_{BLP}(R, (m, f_s, f_o))$$

$$= \begin{cases} (\text{yes}, (m \cup \{(s, o, \texttt{read})\}, f_s, f_o)) \\ \text{if } \ R = \langle +, s, o, \texttt{read} \rangle \\ \wedge \ f_o(o) \preceq f_s(s) \wedge \{o' \in \mathcal{O} \mid (s, o', \texttt{write}) \in m \ \wedge \neg(f_o(o) \preceq f_o(o'))\} = \emptyset \\ \\ (\text{yes}, (m \cup \{(s, o, \texttt{write})\}, f_s, f_o)) \\ \text{if } \ R = \langle +, s, o, \texttt{write} \rangle \\ \wedge \ \{o' \in \mathcal{O} \mid (s, o', \texttt{read}) \in m \ \wedge \neg(f_o(o') \preceq f_o(o))\} = \emptyset \\ \\ (\text{yes}, (m \setminus \{(s, o, x)\}, f_s, f_o)) \\ \text{if } \ R = \langle -, s, o, x \rangle \\ \\ (\text{no}, (m, f_s, f_o)) \text{ otherwise} \end{cases}$$

Table 1. Implementation of the Bell and LaPadula policy

2.3 Rewrite implementation of the Bell and LaPadula model

Implementing the Bell and LaPadula policy according to the transition function defined in Table 1 leads to the following system:

- Let $\mathcal{F}$ be the signature:

$$\begin{array}{llll} \tau & : & \mathcal{R} \times M & \rightarrow M \\ req & : & Mode \times \mathcal{S} \times \mathcal{O} \times \mathcal{A} & \rightarrow \mathcal{R} \\ s & : & Id \times Level & \rightarrow \mathcal{S} \\ o & : & Id \times Level & \rightarrow \mathcal{O} \\ m & : & \mathcal{S} \times \mathcal{O} \times \mathcal{A} & \rightarrow Access \\ \wedge & : & Access \times Access & \rightarrow M \\ \texttt{read}, \ \texttt{write} & : & & \rightarrow \mathcal{A} \\ +, \ - & : & & \rightarrow Mode \\ \top, I, \bot & : & & \rightarrow Level \\ \text{yes}, \text{no} & : & & \rightarrow Access \end{array}$$

– The conditional rewrite rules implementing the policy contain the variables i_1, i_2, i_3 whose sort is *Id*, l_1, l_2, l_3 which have sort *Level*, x has sort *Access* and y is of sort M. Please remark that we consider the operator $\wedge$ to be associative and commutative. While analyzing the rules below the reader must be aware that there is an underlying strategy, which tries to apply the rules in the order they are presented, which is actually the semantics of several implementations of term rewriting systems.

$$
\begin{array}{lll}
(r_1) & \tau(req(+, s(i_1,l_1), o(i_2,l_2), \texttt{read}), x \wedge m(s(i_1,l_1), o(i_3,l_3), \texttt{write})) & \rightarrow \\
 & \text{no} \wedge x \wedge m(s(i_1,l_1), o(i_3,l_3), \texttt{write}) \qquad \textbf{if } (l_2 \preceq l_3) \rightarrow false & \\
(r_2) & \tau(req(+, s(i_1,l_1), o(i_2,l_2), \texttt{read}), y) & \rightarrow \\
 & \text{no} \wedge y \qquad \textbf{if } (l_2 \preceq l_1) \rightarrow false & \\
(r_3) & \tau(req(+, s(i_1,l_1), o(i_2,l_2), \texttt{read}), y) & \rightarrow \\
 & \text{yes} \wedge m(s(i_1,l_1), o(i_2,l_2), \texttt{read}) \wedge y & \\
(r_4) & \tau(req(+, s(i_1,l_1), o(i_2,l_2), \texttt{write}), x \wedge m(s(i_1,l_1), o(i_3,l_3), \texttt{read})) & \rightarrow \\
 & \text{no} \wedge x \wedge m(s(i_1,l_1), o(i_3,l_3), \texttt{read}) \qquad \textbf{if } (l_3 \preceq l_2) \rightarrow false & \\
(r_5) & \tau(req(+, s(i_1,l_1), o(i_2,l_2), \texttt{write}), y) & \rightarrow \\
 & \text{yes} \wedge m(s(i_1,l_1), o(i_2,l_2), \texttt{write}) \wedge y & \\
(r_6) & \tau(req(-, s(i_1,l_1), o(i_2,l_2), a), m(s(i_1,l_1), o(i_2,l_2), a) \wedge y) & \rightarrow \\
 & \text{yes} \wedge y &
\end{array}
$$

3 The McLean Model

The algebra of security [McL88] can be seen as a generalization of the model of Bell and LaPadula. One of the concepts introduced in this algebra is the one of joint access. Indeed, in some fields, as the military one, it could be useful to consider access as done by a group of subjects rather than by a subject alone. A typical example is the one where in order to launch a missile, two individuals have to push a button *at the same time*. Accesses are then represented as a tuple (S, o, x), where S is a non-empty set of subjects, o is an object and x is an access mode.

In order to take into account joint access, the $\star$-security property, corresponding to the property 2, is defined in [McL88], as follows:

> "*a state is $\star$-secure if for any subjects S_1, S_2 and objects o_1, o_2, if $(S_1, o_1, \texttt{read}) \in m$ and $(S_2, o_2, \texttt{write}) \in m$ and the classification of o_1 dominates that of o_2, then $S_1 \cap S_2 = \emptyset$* "

A direct translation of this sentence leads to the following logical formula:

$$
\begin{array}{l}
\forall S_1, S_2 \ \forall o_1, o_2 \in \mathcal{O} \\
\quad ((S_1, o_1, \texttt{read}) \in m \wedge (S_2, o_2, \texttt{write}) \in m \wedge f_o(o_2) \prec f_o(o_1)) \\
\Rightarrow S_1 \cap S_2 = \emptyset
\end{array}
$$

which is logically equivalent (by contraposition) to:

$$
\begin{array}{l}
\forall S_1, S_2 \ \forall o_1, o_2 \in \mathcal{O} \\
\quad ((S_1, o_1, \texttt{read}) \in m \wedge (S_2, o_2, \texttt{write}) \in m \wedge S_1 \cap S_2 \neq \emptyset) \\
\Rightarrow \neg (f_o(o_2) \prec f_o(o_1))
\end{array} \tag{5}
$$

If we do not take into account joint access, every set of subjects S can be reduced to a singleton set $\{s\}$, and the property (5) is equivalent to:

$$\begin{array}{l}\forall s\ \forall o_1, o_2 \in \mathcal{O} \\ (\{s\}, o_1, \texttt{read}) \in m \wedge (\{s\}, o_2, \texttt{write}) \in m \Rightarrow \neg(f_o(o_2) \prec f_o(o_1))\end{array} \qquad (6)$$

With this new definition of the $\star$-property, the rewrite implementation of the McLean model can be obtained from the one of Bell and LaPadula, by replacing the rule (r_1) by

$$\begin{array}{l}(r_7)\ \tau(req(+, s(i_1, l_1), o(i_2, l_2), \texttt{read}), x \wedge m(s(i_1, l_1), o(i_3, l_3), \texttt{write})) \rightarrow \\ \quad \text{no} \wedge x \wedge m(s(i_1, l_1), o(i_3, l_3), \texttt{write}) \qquad \text{if } (l_3 \prec l_2) \rightarrow true\end{array}$$

and the rule (r_4) by

$$\begin{array}{l}(r_8)\ \tau(req(+, s(i_1, l_1), o(i_2, l_2), \texttt{write}), x \wedge m(s(i_1, l_1), o(i_3, l_3), \texttt{read})) \rightarrow \\ \quad \text{no} \wedge x \wedge m(s(i_1, l_1), o(i_3, l_3), \texttt{read}) \qquad \text{if } (l_2 \prec l_3) \rightarrow true\end{array}$$

Clearly, since $\preceq$ is only a partial order defining the lattice of security levels, the properties (2) and (6) are not equivalent (they become equivalent when $\preceq$ is a total order, but most of the time this is not the case).

Moreover, with such a definition of the security policy, a leak of information is possible. Indeed, let us consider the lattice $\mathcal{L} = \{min, \bot, I, \top, max\}$, where $\bot \preceq \top$, I is not comparable to $\bot$ or $\top$ and for all x in $\mathcal{L}$, $min \preceq x \preceq max$. Let $\mathcal{O}$ be the set of objects $\{o_1, o_2, o_3\}$ and f_o the security function such that $f_o(o_1) = \bot$, $f_o(o_2) = I$ and $f_o(o_3) = \top$. Let $\mathcal{S}$ be the set of subjects $\{s_1, s_2\}$ and f_s the security function such that $f_s(s_1) = \top$ and $f_s(s_2) = I$.

The state $\sigma_1 = (m_1, f_s, f_o)$, represented in Fig. 1, where $m_1 = \{(s_1, o_3, \texttt{read}), (s_1, o_1, \texttt{write})\}$, is clearly not secure, since $f_o(o_1) \prec f_o(o_3)$, and so the high-level information contained in o_3 cannot go to o_1.

Let us now consider the state $\sigma_2 = (m_2, f_s, f_o)$, represented in Fig. 2, where $m_2 = \{(s_1, o_3, \texttt{read}), (s_1, o_2, \texttt{write}), (s_2, o_2, \texttt{read}), (s_2, o_1, \texttt{write})\}$. According to the McLean policy, this state is secure since $\neg(f_o(o_2) \prec f_o(o_3))$ and $\neg(f_o(o_1) \prec f_o(o_2))$. So we clearly have a leak of information since this state is secure in spite of the fact that information contained in o_3 can go to o_1 passing through o_2.

We propose in the next section a method that allows us to find out such leaks of information, based on model-checking of term rewriting systems.

4 Model-Checking Access Control Policies

Model-checking is well adapted for searching software flaws when proof methods cannot be fully automated for a given property. This is likely the situation for proving that one implementation of a certain security model meets its specification. When an access control policy is implemented by a term rewriting system,

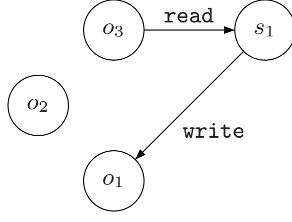

Fig. 1. Insecure state

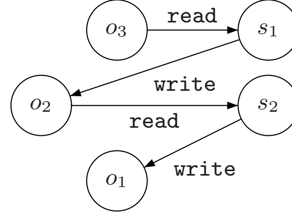

Fig. 2. Secure state

this kind of verification can be performed by using a technique that checks the reachability of a certain term, given a term rewriting system and an input term.

Reachability over term rewriting is a very general approach which constructs approximations of the set of normal forms w.r.t. a term rewriting system [Gen98]. This is done through the use of tree automata. Since the "target" terms represent unwanted situations in general, if they are captured by an approximation, then they will appear in one of the derivations of the actual term rewriting system. In our work, we use a simplified technique using the exact sets of normal forms, since computing the approximations is only necessary in a general framework for term rewriting systems.

Roughly speaking, the algorithm we propose here consists in making explicit every implicit information flow and it checks if this information flow is correct according to the security policy. An information flow is implicit when one can consider that a subject actually has access to an object, but the corresponding access tuple is not present in the set of current accesses. As an example of implicit information flow, we can consider the following situation: if a subject s_1 is reading an object o_1 and writing to an object o_2, and at the same time, a subject s_2 is reading the object o_2, then one can consider that s_2 is also reading the object o_1. In other words, if the accesses $(s_1, o_1, \texttt{read})$, $(s_1, o_2, \texttt{write})$ and $(s_2, o_2, \texttt{read})$ belong to a set of accesses, then we have to add the access $(s_2, o_1, \texttt{read})$ to this set. In this paper, we add this access thanks to a term rewriting system $\mathcal{R}'$, represented in Fig. 3. Note that adding a new access in a set of accesses may lead to the creation of new implicit information flows.

$$
\begin{array}{l}
m(s_1, o_1, \texttt{read}) \wedge m(s_1, o_2, \texttt{write}) \wedge m(s_2, o_2, \texttt{read}) \wedge x \quad \rightarrow \\
m(s_1, o_1, \texttt{read}) \wedge m(s_1, o_2, \texttt{write}) \wedge m(s_2, o_2, \texttt{read}) \wedge m(s_2, o_1, \texttt{read}) \wedge x \\
\hfill \text{if } m(s_2, o_1, \texttt{read}) \notin x
\end{array}
$$

Fig. 3. The Rewrite System $\mathcal{R}'$

The system $\mathcal{R}'$ consists of only one rule, and can be applied only if the implicit flow detected does not already belong to the set of accesses.

In order to define the algorithm of detection of information leakage, we introduce the function App, which takes a term and a (confluent and terminating) term rewriting system, and which returns the term obtained after application of one rule of the system. In other words, App(t, $\mathcal{R}$) = t' if $t \rightarrow_{\mathcal{R}} t'$. If no rule can applied to the term, the function App raise the exception *Fail*.

The algorithm of detection of information leakage relies on two sub-algorithms. The first one, represented by the Algorithm 1, makes explicit every implicit information flow, and checks that none of these flows violates the security policy.

Algorithm 1 (Verification of a set of accesses)

```
Verification(M: set of accesses,
             R, R' : term rewriting systems)=
    M_d ← M;
    While App(M_d, R') ≠ Fail Do
       M_d ← App(M_d, R');
    End While;
    For each m(s,o,x) ∈ M_d \ M Do
       If App(τ(req(+,s,o,x),M), R) = no ∧ M Then
             Return false;
    End For;
    Return true;
End Verification;
```

The second sub-algorithm, represented by the Algorithm 2, takes a set of subjects, a set of objects and two term rewriting systems. For each subject and each object, the algorithm creates two requests corresponding to the read access and the write access, and tries to apply these two requests, in order to obtain a new set of accesses. The function Verification is then applied to each set of accesses. If one of them creates an information leakage, then the algorithm stops.

Algorithm 2 (Creation of sets of accesses)

```
Check(S: set of subjects,
         O: set of objects,
         R, R' : term rewriting systems)=
   M ← ∅;
   For each s ∈ S and each o ∈ O Do
       m_w ← req(+,s,o,write);
       m_r ← req(+,s,o,read);
       If App(τ(m_w,M), R) = yes ∧ x ∧ M Then
          M ← x ∧ M;
       If App(τ(m_r,M), R) = yes ∧ x ∧ M Then
          M ← x ∧ M;
       If Verification(M,R,R') = false Then
          Return "information leakage detected";
    End For;
    Return "no information leakage detected";
End Check;
```

The different sets of accesses created by the Algorithm 2 depend on the "order" of the sets $\mathcal{S}$ and $\mathcal{O}$. Indeed, let us consider for instance the sets $\mathcal{S} = \{s_1, s_2, s_3\}$ and $\mathcal{O} = \{o_1, o_2\}$. If the loop `For` considers each subject and each object in this order, then the first requests created are $m_w = req(+, s_1, o_1, \texttt{write})$ and $m_r = req(+, s_1, o_1, \texttt{read})$. If these two requests are granted, then, in the following, the set of accesses M will contain the access tuples $m(s_1, o_1, \texttt{write})$ and $m(s_1, o_1, \texttt{read})$. Hence, the algorithm will not verify other sets of accesses which do not contain both of these tuples. Since such sets of accesses can be reachable and respect the security policy, our algorithm is clearly not complete. Indeed, let us consider for instance that only the set of accesses:

$$m(s_2, o_1, \texttt{read}) \wedge m(s_2, o_2, \texttt{write}) \wedge m(s_3, o_2, \texttt{read})$$

creates an information flow, and that this set is secure and reachable. Let us also consider that the security policy is defined in a way that s_1 can read o_1, s_3 can read o_2 but these two accesses cannot appear at the same time. If the access $m(s_1, o_1, \texttt{read})$ is first added, and never removed, then the access $m(s_3, o_2, \texttt{read})$ will never be added, and so the information leakage is never detected. On the contrary, if the access $m(s_3, o_2, \texttt{read})$ is added first, then the information leakage can be detected. In order to obtain a complete algorithm, we need to call the function `Check` on every permutation of the sets $\mathcal{S}$ and $\mathcal{O}$. For instance, if $\mathcal{S} = \{s_1, s_2, s_3\}$ and $\mathcal{O} = \{o_1, o_2\}$, we need to call the function `Check` as described in Fig. 4.

$\texttt{Check}(\{s_1, s_2, s_3\}, \{o_1, o_2\}, \mathcal{R}, \mathcal{R}');$
$\texttt{Check}(\{s_1, s_3, s_2\}, \{o_1, o_2\}, \mathcal{R}, \mathcal{R}');$
$\texttt{Check}(\{s_2, s_1, s_3\}, \{o_1, o_2\}, \mathcal{R}, \mathcal{R}');$
$\texttt{Check}(\{s_2, s_3, s_1\}, \{o_1, o_2\}, \mathcal{R}, \mathcal{R}');$
$\texttt{Check}(\{s_3, s_1, s_2\}, \{o_1, o_2\}, \mathcal{R}, \mathcal{R}');$
$\texttt{Check}(\{s_3, s_2, s_1\}, \{o_1, o_2\}, \mathcal{R}, \mathcal{R}');$
$\texttt{Check}(\{s_1, s_2, s_3\}, \{o_2, o_1\}, \mathcal{R}, \mathcal{R}');$
$\texttt{Check}(\{s_1, s_3, s_2\}, \{o_2, o_1\}, \mathcal{R}, \mathcal{R}');$
$\texttt{Check}(\{s_2, s_1, s_3\}, \{o_2, o_1\}, \mathcal{R}, \mathcal{R}');$
$\texttt{Check}(\{s_2, s_3, s_1\}, \{o_2, o_1\}, \mathcal{R}, \mathcal{R}');$
$\texttt{Check}(\{s_3, s_1, s_2\}, \{o_2, o_1\}, \mathcal{R}, \mathcal{R}');$
$\texttt{Check}(\{s_3, s_2, s_1\}, \{o_2, o_1\}, \mathcal{R}, \mathcal{R}');$

Fig. 4. Calls of function `Check`

This method is clearly naive, because it leads to verifying several times the same set of accesses, but it allows to have a complete algorithm.

In the case of the policy described in Sect. 3, information flow can be detected starting from $n_s = 2$ and $n_o = 2$, provided that at least three different confidentiality levels exist, in only four steps of the execution of the main loop, whose trace can be seen on Fig. 5.

Initialization.

Generation of the sets of $\mathcal{S} = \{s(0,\top), s(1,I)\}$, and $\mathcal{O} = \{o(0,\top), o(1,I)\}$.

The set of current accesses is empty: $M = \emptyset$

Step 1

Requests generated	Current accesses
$req(+, s(0,\top), o(0,\top), \texttt{read})$ $req(+, s(0,\top), o(0,\top), \texttt{write})$.	$m(s(0,\top), o(0,\top), \texttt{read})$ $\wedge\, m(s(0,\top), o(0,\top), \texttt{write})$

$M_d = M \Rightarrow M_d \setminus M = \emptyset$, no information flow detected.

Step 2

Requests generated	Current accesses
$req(+, s(0,\top), o(1,I), \texttt{read})$ $req(+, s(0,\top), o(1,I), \texttt{write})$	$m(s(0,\top), o(1,I), \texttt{write})$ $\wedge\, m(s(0,\top), o(0,\top), \texttt{read})$ $\wedge\, m(s(0,\top), o(0,\top), \texttt{write})$

$M_d = M \Rightarrow M_d \setminus M = \emptyset$, no information flow detected.

Step 3

Requests generated	Current accesses
$req(+, s(1,I), o(0,\top), \texttt{read})$ $req(+, s(1,I), o(0,\top), \texttt{write})$	$m(s(1,I), o(0,\top), \texttt{write})$ $\wedge\, m(s(0,\top), o(1,I), \texttt{write})$ $\wedge\, m(s(0,\top), o(0,\top), \texttt{read})$ $\wedge\, m(s(0,\top), o(0,\top), \texttt{write})$

$M_d = M \Rightarrow M_d \setminus M = \emptyset$, no information flow detected.

Step 4.

Requests generated	Current accesses
$req(+, s(1,I), o(1,I), \texttt{read})$ $req(+, s(1,I), o(1,I), \texttt{write})$	$m(s(1,I), o(1,I), \texttt{write})$ $\wedge\, m(s(1,I), o(1,I), \texttt{read})$ $\wedge\, m(s(1,I), o(0,\top), \texttt{write})$ $\wedge\, m(s(0,\top), o(1,I), \texttt{write})$ $\wedge\, m(s(0,\top), o(0,\top), \texttt{read})$ $\wedge\, m(s(0,\top), o(0,\top), \texttt{write})$

$M_d = m(s(1,I), o(0,\top), \texttt{read}) \wedge \ldots \wedge m(s(0,\top), o(0,\top), \texttt{write})$

$M_d \setminus M = m(s(1,I), o(0,\top), \texttt{read})$. When we apply the transition rule to this request we obtain $\tau(req(+, s(1,I), o(0,\top), \texttt{read}), M) = \text{no} \wedge M$ which characterizes the information flow. The execution then stops.

Fig. 5. Execution trace for $n_s = 2$ and $n_o = 2$

We implemented this algorithm in Tom[3] [KMR05,BBK+07]. The implementation follows the same general lines of the model-checking approach employed to find attacks in security protocols, as described in [CMR05], which explores the entire search space by applying all possible rules, over each successive state of the system, until the undesired situation is identified.

The technique we presented here can be adapted to check other access control models. It is is only necessary to substitute the policy implementation, with the corresponding implementation for the new model. However, if this algorithm does not capture an information flow to a certain instance of a policy model, as in any other model-checking technique, it does not mean that the implementation is secure with respect to the information flows.

5 Conclusion

In this paper we proposed a technique based on model-checking to detect information flow in access control models. We define the state transition function that implements a security policy using a term rewriting systems. The rule-based algorithm we provided explores the space of all possible requests in order to automatically identify information flow in these implementations. We illustrated the application of our approach over the McLean's security algebra, a generalization of the Bell and LaPadula security model. We showed that it fails to prevent information leakage even on a relatively small setting.

Some related works use model-checking in the domain of access control policies. In [GRS04], the authors verify the satisfiability of a temporal logic formula in the policy model they presented, but it is not clear how they could address information flow. In [DFK06], the authors suggest to perform goal reachability over Datalog programs as a means to compare access control policies, but information flow is not addressed as well. The main advantage of our approach is that it can be easily adapted to several access control policies. Furthermore, one can use automated approaches to translate logic programs into term rewriting systems [SKGST07] and then apply the algorithm we introduce here.

As future work, we consider to extend our technique to check other properties such as answering more general queries, for instance, whether access is always denied for a given object, or whether access is granted and denied at the same time to a given request. We also intend to enhance the algorithm `Check`, in order to avoid superflous calls of this function on the several permutations of an access tuple.

Acknowledgments Many thanks to Horatiu Cirstea, Thérèse Hardin, Eric Jaeger, Mathieu Jaume and Claude Kirchner for enlightening discussions on this subject. We also thank Guillaume Burel. Oana Andrei, and Radu Kopetz for proof reading previous drafts of this paper and the anonymous referees for their very useful remarks.

[3] http://tom.loria.fr

References

[BBK+07] Emilie Balland, Paul Brauner, Radu Kopetz, Pierre-Etienne Moreau, and Antoine Reilles. Tom: Piggybacking rewriting on java. In *Proceedings of the 18th Conference on Rewriting Techniques and Applications*, Lecture Notes in Computer Science. Springer-Verlag, 2007.

[BF06] Steve Barker and Maribel Fernández. Term rewriting for access control. In *DBSec*, pages 179–193, 2006.

[BL73] D. Bell and L. LaPadula. Secure Computer Systems: a Mathematical Model. Technical Report MTR-2547 (Vol. II), MITRE Corp., Bedford, MA, May 1973.

[BN98] Franz Baader and Tobias Nipkow. *Term Rewriting and all That*. Cambridge University Press, 1998.

[CMR05] Horatiu Cirstea, Pierre-Etienne Moreau, and Antoine Reilles. Rule-based programming in java for protocol verification. *Electr. Notes Theor. Comput. Sci.*, 117:209–227, 2005.

[DFK06] Daniel J. Dougherty, Kathi Fisler, and Shriram Krishnamurthi. Specifying and reasoning about dynamic access-control policies. In Ulrich Furbach and Natarajan Shankar, editors, *IJCAR*, volume 4130 of *Lecture Notes in Computer Science*, pages 632–646. Springer, 2006.

[dO07] Anderson Santana de Oliveira. Rewriting-based access control policies. In *Proceedings of - SECRET'06*, ENTCS. Elsevier, 2007.

[Gen98] Thomas Genet. *Contraintes d'ordre et automates d'arbres pour les preuves de terminaison*. PhD thesis, Université Henri Poincaré - Nancy I, 1998.

[GRS04] Dimitar P. Guelev, Mark Ryan, and Pierre-Yves Schobbens. Model-checking access control policies. In Kan Zhang and Yuliang Zheng, editors, *ISC*, volume 3225 of *Lecture Notes in Computer Science*, pages 219–230. Springer, 2004.

[HR98] Nevin Heintze and Jon G. Riecke. The SLam calculus: programming with secrecy and integrity. In ACM, editor, *Symposium on Principles of Programming Languages, San Diego, California*, pages 365–377, New York, NY, USA, 1998. ACM Press.

[JM06] M. Jaume and C. Morisset. Towards a formal specification of access control. In P. Degano, R. Kusters, L. Vigano, and S. Zdancewic, editors, *Proceedings of FCS-ARSPA'06*, pages 213–232, 2006.

[KMR05] Claude Kirchner, Pierre-Etienne Moreau, and Antoine Reilles. Formal validation of pattern matching code. In Pedro Barahona and Amy P. Felty, editors, *PPDP*, pages 187–197. ACM, 2005.

[LB96] L.J. LaPadula and D.E. Bell. Secure Computer Systems: A Mathematical Model. *Journal of Computer Security*, 4:239–263, 1996.

[McL88] McLean. The algebra of security. In *Proc. IEEE Symposium on Security and Privacy*, pages 2–7. IEEE Computer Society Press, 1988.

[Mye99] Andrew C. Myers. JFlow: Practical mostly-static information flow control. In *Symposium on Principles of Programming Languages*, pages 228–241, 1999.

[SKGST07] P. Schneider-Kamp, J. Giesl, A. Serebrenik, and R. Thiemann. Automated termination analysis for logic programs by term rewriting. In German Puebla, editor, *LOPSTR*, volume 4407 of *Lecture Notes in Computer Science*, pages 177–193. Springer, 2007.

[Ter02] Terese. *Term Rewriting Systems*. Cambridge University Press, 2002.