Verification of Interoperability Security Policies by Model Checking

Mazen El Maarabani*, Ana Cavalli*, Iksoon Hwang* and Fatiha Zaïdi†
*IT/ TELECOM & Management SudParis, EVRY, F-91011
Email: {mazen.el_maarabani, iksoon.hwang, Ana.Cavalli}@it-sudparis.eu
†Univ. Paris Sud, LRI; CNRS Orsay, F-91405;
Email: Fatiha.Zaidi@lri.fr

Abstract—**Access control policies are the key point for a secured interaction in business community. In general, an information system has to include an interoperability access control security policy to regulate the access from other systems to its resources. The security policy specifies a set of rules that defines the privileges of any subject accessing to the information system resources. In this paper we provide an approach to verify the correctness of contextual based interoperability access control security policies which are integrated in a system model. Security rules are initially described using the organization to organization model (O2O). We first propose an approach to transform O2O security rules to the well known Linear Temporal Logic (LTL). In order to instantiate the LTL formulae from a set of O2O security rules, we provided a mapping between the elements of the O2O security rule and the elements of the functional model in which the security rules are integrated. The resulted LTL formulae are used to verify the correctness of the security rules by model checking.**

Keywords: OrBAC, model-checking, LTL, interoperability security policy.

I. Introduction

Access control security policies are the key point for modern infrastructures. Nowadays each organization defines an access control security policy that is responsible for efficiently and effectively managing the organization's resources on which its users can perform actions. An access control security policy is a set of rules that defines the desired behavior of a user within an organization. Accordingly, each action performed by a user on the organization resources is administrated by the system security policy.

Recently, there is an increasing need of interaction in business community due to the need of exchanging and sharing resources. Thus, each organization that wants to interoperate with other organizations has first to define its own interoperability access control security policy. The policy aims to regulate the access to an organization's resources when interacting with other organizations.

The local security objectives of an organization has to be maintained while exchanging and sharing resources with other organizations. In order to accomplish a secured interoperation, [4] has proposed solutions to specify the interoperability security policy without conflicts with the local one.

Specification of a security policy is actually no longer restricted to permissions. A security policy may include three different types of security rule: permission, prohibition and obligation. Intuitively, a prohibition is a negative permission implying that one must not perform some action. An obligation is associated with an action someone must perform and is usually triggered when some conditions are satisfied.

The increasing complexity in the requirements of secured interactions between systems urges security models to handle more flexible and dynamic security policies [5]: security rules in these policies are no longer static but dynamic, depending on the context. The advantage of such dynamicity is to provide a self-adaptive access control security policy that is aware of changes in the interoperability environment.

After the specification of the interoperability access control security policy using a security model, it is strongly necessary to verify whether it behaves as required by the original security requirements. Existing work in this domain can be divided in two approaches: 1) the first approach [10] aims to integrate the access control security policy in the functional model of the system. The resulted model can be used to verify security properties or to generate test cases by model based testing. 2) The other approaches [1], [9] model only the security policy using a formal model, Petri nets for instance, and verify some security properties in the model. This method could be less interesting when we consider contextual security policies which involves the functional behavior of a system.

In this paper we propose an approach to verify the correctness of contextual based interoperability access control security policy which is integrated in a functional model against its specification by model checking. We use O2O model to specify the interoperability security policy. This model cannot be directly used to verify the correctness of a security policy. Thus, we propose to transform the O2O rule into the Linear Temporal Logic. This logic allows to formalize properties that can be checked in a system execution by model checking such SPIN [8] to verify that all the executions of the model in which the policy is integrated respect these properties. The choice of the SPIN model-checker is motivated by the fact that it is an efficient and well accepted model-checker that relies on well-established algorithms that can manage huge specifications. It provides several means to handle the state space explosion such as compression algorithms, different levels of abstraction by states fusion, a bit-state space vector and so on. The contributions of the paper are manifold:

- We provide a method to translate from O2O the three types of security rules (which are permission, prohibition and obligation) into linear temporal logic formulae (LTL).
- We provide a method to instantiate the LTL formulae based on a set of O2O security rules.
- A case study is provided as a proof of concept of our method.

The remainder of the paper is organized as follows. Section II presents the basic concepts needed for this work. In section III and IV we describe our approach to transform the O2O contextual based access control security rules into LTL formulae. In section V, the verification framework is defined. It also gives the experimental results to show the applicability of our approach. Section VI discusses the related work and section VII concludes the paper.

II. Preliminaries

A. Organization to Organization Model

The organization access control language is an access and usage control model based on a sorted first order logic [3]. Thus, operators such as conditional (if/then) statement ($\leftarrow$), conjunction ($\wedge$), disjunction ($\vee$) and negation ($\neg$) has the same meaning as in a first order logic. The concept of organization is central in OrBAC. Intuitively, an organization is any entity that is responsible for managing a security policy. Each organization can use OrBAC to specify its own security policy at the organizational level that is abstracted from the implementation of this policy. Thus, instead of modeling the policy by using the concrete and implementation-related concepts of subject, action and object, the OrBAC model suggests reasoning with the roles that subjects, actions or objects are assigned in the organization. Thus, a subject is abstracted into role which is a set of subjects to which the same security rule applies. Similarly, an activity and a view are respectively a set of actions and objects to which the same security rule applies.

Definition 1: A security rule is a relation between an organization role, views, activities and context. It is defined as a role having permission, prohibition or obligation to perform an activity on a view within an organization.
A security rule is expressed as follows:
$SecurityRule(org, modality(role, activity, view, context))$
Where *modality* belongs to {permission, prohibition, obligation}. OrBAC defines two types of context [5]: a state based context and an event based context. The state based context specifies boolean conditions. The event based context describes an event that has to be performed in order to activate the security rule. OrBAC defines logical conditions to characterize when contexts are active:
$Hold(org, subject, action, object, context)$
means that context $context$ holds between $subject$, $action$ and $object$ within org.

Or-BAC model supports the O2O approach to manage interoperability between components having their own policies defined by different organizations [4]. The O2O approach relies on the concept of virtual private organization (VPO) to designate the sub-organization in charge of the interoperability access control. To explain the basic principles of O2O, let us consider that a given organization A wants to interoperate with another organization B. In this case, each organization has to define a Virtual Private Organization (VPO) respectively called A2B and B2A. The VPO A2B is associated with a security policy that manages how subjects from the grantee organization A, O_grantee, may have an access to the grantor organization B, O_grantor. We say that the VPO A2B manages the interoperability security policy from organization A to organization B. The VPO B2A is similarly defined to control accesses of subjects from organization B to organization A.

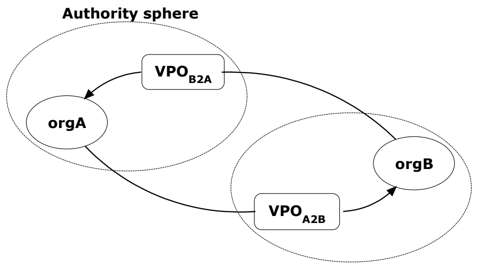

Figure 1. Interaction Between Two Organizations in O2O Framework

Thus, in a VPO, the O_grantor organization can define roles, activities and views and associate these roles, activities and views with contextual security rules as in a classical organization of the OrBAC model. In O2O, the concepts of authority sphere and management sphere specifying who creates and manages the interoperability security policy are introduced. Figure 1 illustrates the O2O framework.

B. Linear Temporal Logic

In this section we will introduce the linear temporal logic (LTL) [8]. The logic allows to formalize properties of a path in a computation tree unambiguously and concisely with the help of a small number of special logical operators and temporal operators. Given a set of atomic propositions $\wp$ where $\{\phi, \varphi\} \subseteq \wp$, LTL formulae are constructed inductively as follows. If ϕ and φ are formulae then:
Logical operators:

1) $\phi \wedge \varphi$ (logical conjunction), $\phi \vee \varphi$ (logical disjunction) are also formulae and $\neg\phi$ (logical negation) is a formula.
2) $(\phi \longrightarrow \varphi)$ (logical implication) and $(\phi \longleftrightarrow \varphi)$ (logical equivalence) are formulae.
3) $\top$ and $\bot$ denotes true and false respectively.

Temporal operators:

1) $<> \phi$, $[\ ]\phi$ and $\phi U \varphi$ are formulae, where:

[ ] (always) the formula $[\ ]\phi$ holds if the formula ϕ holds in all the states of the path.
<> (eventually) the formula $<>\phi$ holds if the formula ϕ eventually holds in a state of the path.
U (until) The formula $\phi U \varphi$ holds, if ϕ holds until φ occurs, i.e., there is a state on the path at which φ holds, and at every state before ϕ holds.

III. Decomposition of the Transformation

An O2O security rule can be decomposed into the following three entities:

- An active security rule: the security rule is considered active which means that this rule is not constrained by a context or the context is already satisfied.
- A state based context.
- An event based context.

In this section we define the LTL formulae that describe each of the above entities. The motivation behind this decomposition is to show how each part of a contextual based security policy can be expressed by LTL formulae. The result of this transformation will be used to compose a contextual based security rule in LTL. Note that this decomposition is useful when not all the security rules are constrained with a state based or an event based context.

The differentiation between these two types of context depends on the level of abstraction in the system model. The condition of an event based context expresses a behavior in a system model that should be performed in order to activate the security rule. A state based context is represented in the system model with boolean variables or functions given by an oracle whether if some conditions are satisfied or not. In this paper it is assumed that the system model and the O2O security rules are defined in the same level of abstraction.

A. Security Rule Transformation

An O2O security rule is defined as follows:
$securityRule(VPO, modality(role, activity, view, context))$

We denote by S and O the formulae that represent respectively a subject of $role$ and an object of $view$. We denote by Req_r, the formula that represents the request to perform an action of $activity$. This request can generate several possible outputs in the functional model. For instance this request can be granted or denied. We denote by Ack_{Req_r} the formula that describes the expected output generated after the execution of the security rule. The other possible outputs that can be generated form the request are represented by the formula $NAck_{Req_r}$.

In the case where the $modality$ of the O2O security rule is a permission or a prohibition, the security rule is active when the value of the request, subject and object meets this rule. The secured system generates a specific response when the security rule is executed. Formally, this can be expressed with an LTL formula as follows:

$Permission\ or\ prohibition\ :\ Req_r \wedge S \wedge O\ \rightarrow <> (\neg NAck_{Req_r}\ U\ Ack_{Req_r})))$

An obligation rule represents an action that has to be performed by a subject on an object of the secured system. Thus, it is formally defined using LTL as follows:

$Obligation\ :\ (Req_r \wedge S \wedge O) \wedge ((Req_r \wedge S \wedge O)\ \rightarrow <> (\neg NAck_{Req_r}\ U\ Ack_{Req_r})))$

B. State based context Transformation

A state based context is defined in O2O as follows:
$Hold(VPO, subject, action, object, context_s) \leftarrow conditions$

This description means that $context_s$ is true for a specific subject object and action and when the logical $conditions$ are true. Let us denote by C_s the formula representing the logical $conditions$ of the state based context. Thus, the state based context can be described in LTL as follows:

$Context_s :\ C_s \wedge (C_s \rightarrow Req_r \wedge S \wedge O)$

Thus if the logical conditions specified in C_s does not occur in a system execution, $context_s$ remains false. In the case where the formula C_s is satisfied, the context is only true for a specific subject object and action which are respectively specified in the formulae S, O and Req_r.

C. Event based context Transformation

In the case of a permission rule, an event based context expresses a pre-obligation which specifies an action that has to be performed in order to activate the permission rule. In the case of a prohibition rule, an event based context specifies that after performing a specific activity, a user will be prohibited to perform another one on the secured system. An obligation rule can be triggered when specific action is executed on the secured system. O2O defines an event based context as follows:

$Hold(VPO, subject, action, object, context_e) \leftarrow$
$After\ do(subject', action', object')$

Thus, in this case the context $context_e$ is true for $subject$, $object$ and $action$ in the state which follows the occurrence of the action $do(subject', action', object')$. We denote by C_e the formula that describes this action. The event based context is represented in LTL as:

$context_e :\ C_e \wedge (C_e \rightarrow Req_r \wedge S \wedge O)$

Thus if the action described in C_e does not occur in a system execution, $context_e$ remains false. In the case where the formula C_e is satisfied, the context is only true for a specific subject object and action which are respectively specified in the formulae S, O and Req_r.

Let us denote by S', O' and Req_e the formulae representing respectively $subject'$, $object'$ and $action'$. The formula Ack_{Req_e} represents the expected output specifying that the action of the event based context is successfully performed. The formula $NAck_{Req_e}$ describes the other possible output that can be generated by the secured model in response to this request. The formula C_e is satisfied only when the behavior that it describes is performed. Thus, we define C_e with the following formula:

$(Req_e \wedge S' \wedge O') \wedge ((Req_e \wedge S' \wedge O')\ \rightarrow <> (\neg NAck_{Req_e}\ U\ Ack_{Req_e}))$

IV. Contextual Based Security Rule Transformation

In this section we define the LTL formulae describing the contextual based interoperability access control security rules. The notation of an O2O security rule is given in the following:
$securityRule(VPO, modality(role, activity, view, context_s \wedge context_e))$

We aim to verify the correctness of contextual based security rules. Thus, The LTL formulae that describe a security rule have to satisfy the following:

- If in a system execution the request (described by $Req_r \wedge S \wedge O$) that activates a security rule is not satisfied, then the two LTL formulae describing this security rule has to be satisfied as in this case the correctness of the security rule could not be violated.
- If the event based context and the state based context are satisfied, then the security rule should be activated.

Accordingly, when the security rule is executed, a specific response should be generated by the system.

- If the state based context or the event based context are not satisfied, then the security rule should not be active. Thus, this rule cannot be executed.

In the case of a non active permission rule or prohibition rule, this rule cannot be executed. Thus, the output described in Ack_{Req_r} should not be generated in response to $Req_r \wedge S \wedge O$. We describe this behavior in LTL as follows:

$not\_active\_permission\ or\ not\_active\_prohibition:$
$Req_r \wedge S \wedge O \rightarrow <> \neg Ack_{Req_r}$

The behavior described in an obligation rule is executed only when the conditions related to its execution are satisfied (these conditions can be defined in a state based context or event based context). Thus the LTL formula that describes a non active obligation rule is defined as follows:

$not\_active\_obligation: \ \neg\ Req_r \wedge S \wedge O$

A contextual based permission security rule is described with two LTL formulae:

$F1: \ context_e \rightarrow (context_s \rightarrow permission)$
$F2: \ (\neg context_e \vee \neg context_s) \rightarrow not\_active\_permission$

The formula $F1$ describes that a permission security rule can be executed only if the two contexts are satisfied. In the case where one or both of the contexts are not satisfied, the formula F2 verifies that the permission rule is not active. In the case where one or both of the context types are not required for a permission rules, we simply replace the formulae $context_e$ and/or $context_s$ with the formula $\top$. The formulae defining a contextual based prohibition or obligation rule are similar to the permission rule case.

A. LTL Formulae Instantiation

In this section we will provide a way to instantiate the LTL formulae based on a set of O2O security rules. This approach is in two steps. First, we relate the O2O parameters to elements that correspond in the functional model. Then, we use the obtained values to instantiate the LTL formulae.

The action of an O2O security rule can be related to a specific request and the possible outputs that can be generated by this request. Then it is possible to narrow down the number of outputs to one specific output by considering the security rule modality. The subject and object of the security rule are represented as variables in the functional model or parameters in the request message. A state based context is represented with functions or boolean variables in the functional model. It gives an oracle whether the context is satisfied or not. Therefore it is possible to map a state based context to a set of boolean functions or boolean variables. An event based context represents a behavior that has to be performed in order to activate the security rule. Thus, the action of the context is related to a request and an output describing that the action of this event is successfully performed. The subject and object of the context are described with variables or messages parameters in the functional model. This mapping can be done by using a table that relates each element of an O2O security rule to the element that corresponds in the functional model.

In the next step we use the mapping result to instantiate LTL formulae. For instance, the formula $Req_r \wedge S \wedge O$ represents a request sent from a specific subject targeting a specific object. Thus, using the mapped values of the action, the subject and the object of an O2O security rule, it is easy to define the LTL formula that verifies this property.

V. Security Rule Verification

As a proof of concept, we exercise our approach on a hospital network case study. A hospital network is a network or group of hospitals that work together to coordinate and deliver a broad spectrum of services to their community. We consider the case where the hospital network consists of two hospitals, hospital A and hospital B. It is assumed that each hospital has its local security policy to manage the privileges of its local users. The following *roles* exist in the two hospitals:

- doctor: A doctor on duty, he/she can have any speciality;
- nurse: A nurse on duty, a nurse is in charge of patients within the department he/she belongs;
- Aduser: A user in the administrative staff;
- ITuser: A user in the Information technology staff.

It is also assumed that in both hospitals a patient has a medical report, data related for payment and some sensitive data which is related to personal information (like previous medical report, insurance company, etc.). Moreover, it is possible for a security rule to consider the *roles* as *views*. In such cases the activity of the security rule will target these *roles*.

We consider only the one way of interaction, from hospital A to hospital B. Thus, the resources to be accessed by the subjects of hospital A, such as medical reports, sensitive data and payment files of patients are located in hospital B. We will index all the roles of hospital A with the letter A.

A. Building the Model

We initially defined an interoperability access control security policy as a set of requirements that defines the privileges of each role of hospital A. Here are three different examples of these requirements:

1) A doctor from hospital A must be able to access information about the medical report, located in hospital B, of all the patients.
2) A doctor in duty from hospital A is allowed to do any action on the medical report of a patient in emergency context.
3) The system is obligated to notify a doctor after editing one of his/her patient's medical reports by a doctor from hospital A.

We build a partial model of a hospital B that describes the behavior which is involved by the interoperability access control security policy. We based on the work done in [10] and [7] to integrate respectively the local and the interoperability security policy in the functional model of hospital B. We used PROMELA as specification language, this language is compatible with the SPIN model checker. For lack of room, we do not provide in this paper all the details of the modeling.

	O2O	Functional Model	LTL
Security rule action	notify_Edit	?notify_inCharge(s,o) !notification_confirmed(0,0)	fromEnv?[notify_inCharge(s,o)] toEnv?[notification_confirmed(0,0)]
Security rule subject Security rule object	$system$ O	message parameter s=0 message parameter o=hispatient[100-o']	message parameter s=0 message parameter o=hispatient[100-o']
State based context	-	-	$\top$
Event based context action	edit_file	?edit_file(s',o') !edit_grant(0,0)	fromEnv?[?edit_file(s',o')] toEnv?[edit_grant(0,0)]
Event based context subject Event based context object	S' x.pdf	message parameter s'∈ [1..10] message parameter o'∈ [100..150]	message parameter s'∈ [1..10] message parameter o'∈ [100..150]

Table II
THE MAPPING TABLE

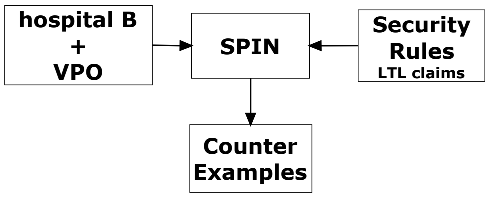

Figure 2. Verification Process

B. Defining the Verification Process

We aim to verify the correctness of the interoperability access control security policy which is integrated in the hospital B model against its specification modeled using O2O. Thus, the system model whose executions has to satisfy the access control security policy objectives consists of hospital B in which the interoperability access control security policy is integrated. The other entities that interact with the hospital B, such as the local entities of hospital B and the hospital A, are the environment. Figure 2 illustrates the verification process.

C. Description of the Interoperability Policy Using LTL

We specify the requirements of the interoperability security policy using the O2O model. As a result, we got twenty three O2O security rules.

Interoperability scurity policy	O2O	LTL
Set of requirements	23 security rules	46 formulae

Table I
RESULTS OF THE TRANSFORMATION OF THE INTEROPERABILITY SECURITY POLICY

We generated for each of the O2O security rules two LTL formulae (Table I summarizes the results). The LTL formulae are then instantiated with the values of subjects and objects that activate the original O2O security rule.

Consider the requirement "*The system is obligated to notify a doctor after editing one of his/her patient's medical reports by a doctor from hospital A*". It specifies an obligation rule with an event based context. The O2O notation of this rule is given in the following:

$securityRule(VPO_{A2B}, obligation(system, notify, doctor, MedReportEdit))$

The context $MedReportEdit$ can be described as:

$hold(VPO_{A2B}, system, notify\_Edit, O, MedReportEdit) \leftarrow After\ do(S', edit\_file, x.pdf)$

In our PROMELA model each role is represented with a sequence of values which will be assigned to subjects of this role. For instance, a doctor of hospital A can have one of the values of [1..10]. The role system is composed of only one subject with the value 0. The values [100..150] are assigned to the medical reports. We also defined an array *hispatient* to specify who is the doctor in charge of each medical report. Table II gives the mapping that corresponds to this example. We defined the following formulae:

$$\begin{array}{ll}
obligation: & fromEnv?[notify\_inCharge(s,o)]\&\& \\
 & (fromEnv?[notify\_inCharge(s,o] \rightarrow <> \\
 & toEnv?[notification\_confirmed(0,0)]) \\
 & \\
C_e: & fromEnv?[edit\_file(s',o')]\&\& \\
 & (fromEnv?[edit\_file(s',o')] \rightarrow <> \\
 & toEnv?[edit\_grant(0,0)]) \\
 & \\
context_e: & C_e\&\&(C_e \rightarrow \\
 & fromEnv?[notify\_inCharge(s,o)]) \\
 & \\
not\_active\_obligation: & \neg fromEnv?[notify\_inCharge(s,o)]
\end{array}$$

The Two formulae describing the obligation rule are:

$O1: \ context_e \rightarrow obligation$

$O2: \ \neg context_e \rightarrow not\_active\_obligation$

Note that the provided LTL formulae respect the notation of the SPIN model checker. The parameters s, s', o and o' has to be instantiated with the values defined by the mapping in Table II. It is possible to use a script that automatically assigns s, s', o and o' the values that fits the mapping.

D. Verification of Correctness

The generated LTL formulae are used to define *claims* in the SPIN model checker. The *claims* represent properties that should be satisfied during the execution of a model. We verify that all executions of the model are accepted by these *claims*. If the execution of the claim does not match with the execution of the model, SPIN will produce a *counter example* that allows the execution to be replayed.

To show the applicability of our method we injected some faults in the functional model. For instance we investigate the case where the behavior of the obligation rule (specified in the above subsection) is removed from the functional model. Thus, after a medical file of a patient is edited by a doctor of hospital A the system does not notify the doctor in charge of this patient.

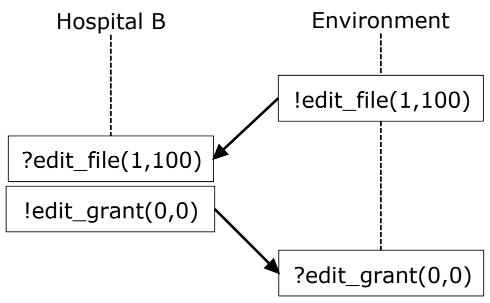

Figure 3. The Counter Example Generated by SPIN

When running the verification process a counter example is generated by SPIN when checking the formula $O2$. This counter example is given in Figure 3. We also verified whether the obligation action actually notify the right doctor by verifying if the value of o corresponds to the doctor in charge of the medical report represented in o'.

VI. Related Work

Several works have proposed solutions to specify security properties in a formal way in order to verify or test some security properties. In [7] the authors proposed a formal way to integrate O2O security rules in the functional model of a system. The system is modeled using extended automata. The resulted model is used to generate test cases using a dedicated tool. In another work [9] the authors proposed to verify some security policy properties using colored Petri net. They defined security properties such as completeness, consistency, termination and confluence in Petri net terminology. In [1] the authors proposed to model e-contract security rules such as permission, prohibition and obligation using timed automata. Then they verified that the automata will never reach an undesirable state. Ammar Masood and al. [11] constructed an FSM model of the RBAC (role based access control) policy and then generated test cases from the model using the well known W-method. The proposed technique provides complete fault detection with respect to the RBAC fault model that can be mapped to Chow's fault model. In [6], authors propose an approach to achieve testing of network security rules. They express the network behavior using labeled transition systems formula. The security rules are specified using a variant of the Linear Temporal Logic. Then, for each element of their language and each type of rule, they propose a pattern of test called a tile. Then, they combine those tiles into "complete" test cases. The test cases are used to test the conformance of the implemented rules with respect to a global specification of security policy for a network of interconnected systems. Also [2] has proposed an approach to security testing of web-based applications in which test cases are automatically derived from counterexamples found through model checking.

VII. Conclusion

In this paper, a formal approach to verify the correctness of contextual based interoperability access control security policy is presented. The control access security policy is initially specified using O2O model. In order to verify the correctness of a security policy which is integrated in a functional model, we propose an approach to transform O2O security rules to the well known Linear Temporal Logic. We show that an O2O security rule can be described using two LTL formulae. In order to instantiate an LTL formula from a set of O2O security rules, we provided a mapping between the elements of the O2O security rule and the elements of the LTL formulae. Finally, we carried out a case study on a hospital network to show the applicability of our method.

As future work, we are investigating the possibility to adapt these results for monitoring testing. Thus, the traces of two systems that communicate according to an interoperability security policy will be collected. These traces will be checked with respect to the LTL formulae requirements and automatically a verdict of conformance of the implemented interoperability security policy with its requirements will be emitted. This will enable us to detect on real-time system crashes and security rules violations and most importantly to be able to stop this kind of malicious behaviors without any delay. Another possibility is to use the model checker to produce active test cases covering the requirements by using the counter-example provided by SPIN. In this case, the property will become a non desirable property to be checked. To perform such work, we will get inspired by [12].

References

[1] A. Abou El Kalam and N. Idboufker. Specification and verification of security properties of e-contracts. In *8th International Conference on Communications*, pages 427 – 430, 2010.

[2] A. Armando, R. Carbone, L. Compagna, K. Li, and G. Pellegrino. Model-checking driven security testing of web-based applications. In *Third International Conference on Software Testing, Verification, and Validation Workshops*, pages 361–370, Washington, DC, USA, 2010.

[3] C. Baral and J. Lobo. Formal characterization of active databases. In *Proceeding of international workshop on logic in databases*, pages 175–195, 1996.

[4] F. Cuppens, N. Cuppens, and C. Coma. O2O : Managing security policy interoperability with virtual private organizations. In *13th annual workshop of HP Openview University Association, HP-OVUA, may 21-24, Côte d'Azur, France*, 2006.

[5] F. Cuppens and N. Cuppens-Boulahia. Modeling contextual security policies. volume 7, pages 285–305, Berlin, Heidelberg, 2008.

[6] V. Darmaillacq, J.C. Fernandez, R. Groz, L. Mounier, and J.L. Richier. Test generation for network security rules. pages 341 – 356, New York, 2006. Springer.

[7] M. El Maarabani, I. Hwang, and A. Cavalli. A formal approach for interoperability testing of security rules. volume 0, pages 277–284, Los Alamitos, CA, USA, 2010. IEEE Computer Society.

[8] G. Holzmann. *Spin model checker, the primer and reference manual.* Addison-Wesley Professional, first edition, 2003.

[9] H. Huang and H. Kirchner. Formal specification and verification of modular security policy based on colored petri nets. volume 99, Los Alamitos, CA, USA, 2010. IEEE Computer Society.

[10] W. Mallouli, J. Orset, A. Cavalli, N. Cuppens, and F. Cuppens. A formal approach for testing security rules. In *Proceedings of the 12th ACM symposium on Access control models and technologies*, SACMAT '07, pages 127–132, New York, NY, USA, 2007. ACM.

[11] A. Masood, R. Bhatti, A. Ghafoor, and A. Mathur. Scalable and effective test generation for role-based access control systems. volume 35. IEEE Computer Society, 2009.

[12] F. Zaïdi, M. Lallali, and S. Maag. A component based testing technique for a manet routing protocol. In *IEEE AICSSA Conference*, pages 1–7, 2010.