Testing Security Rules with Decomposable Activities

Wissam Mallouli and Ana Cavalli
GET/INT, 9 rue Charles Fourier, 91011 Evry Cedex, France
{wissam.mallouli,ana.cavalli}@int-edu.eu

***Abstract*—Checking that a security policy has been correctly deployed over a network is a key issue for system administrators. Specification and testing of such policies constitute fundamental steps in the development of a secure system. To address both challenges, we propose a framework to describe how modalities such as permissions, prohibitions and obligations -involving decomposable activities- can be integrated in a functional EFSM specification of a system to obtain a new specification of the system that takes into account the security policy. Then, we propose a method to automatically derive test sequences to test the implementation, using a dedicated tool developed in our laboratory. Finally, we apply our framework to a Weblog system case study to demonstrate its reliability.**

***Index Terms*—Security Policy, Extended Finite State Machine, SDL, Verification and Testing, Test Generation.**

I. Introduction

In modern networks, the heterogeneity and the increasing distribution of the applications make security management complex. In order to give a global understandable view of network security, we make an abstraction from the technical constraints by using security policy rules. These rules express the security objectives of the network and specify the desired behavior of the system. In such networks, it is quite difficult to verify whether a system implementation conforms to its policy. However, if no one can ensure of this conformance, the global security can not be guaranteed anymore.

Most current works only concentrate on defining meta-languages in order to express security policies and provide unambiguous rules. [3] and [9] are typical examples of such generic policy description models. Indeed, they do not depend on the functional specification of the system. But, they suggest several concepts to describe the security policy independently of the system implementation. Once the security policy is formally specified, it is essential to prove that the target system implements this policy either by (1) injecting this policy in the studied system or (2) by formally specifying the target system and generating proofs that this system implements the security policy or (3) by considering several strategies of formal tests. This last methodology will be explored in this paper.

In this paper, we propose an approach that makes it possible to validate security rules. This approach manipulates three different inputs: a functional specification of the system based on a well-know mathematically-based formalism: Extended Finite State Machine [14], a specification of the security policy (based on the OrBAC model [3]) that we would like to apply on this system, and finally an implementation of the system. We want to obtain a new specification of the system that takes into consideration the security policy (we call it: secure functional specification), and then to generate tests to check whether the implementation of the system conforms to the secure functional specification.

This paper distinguishes itself from classical conformance testing work (see for instance [4]) by several significant differences. In fact, we propose an approach to integrate security rules involving decomposable activities within the functional specification of a system. Thus, we describe how modalities such as prohibitions, authorizations, obligations and delegation can be integrated in an EFSM, either by restricting predicates or by adding transitions and states. Then, we propose a method to automatically derive test sequences from a set of rules as well as an approach to restrict the number of test objectives required to perform verification. But, we do not address issues like checking the consistency of the security policy which is out of the scope of this paper. We assume that this issue has been checked. There are several techniques to achieve this goal (see for instance [7]).

The remainder of this paper is organized as follows. In section II, we discuss the related work tackling with the description and the validation of security policies. Section III presents the basic notions used for the management of security rules. In section IV, we expose the approach to integrate these security rules within an existing specification in EFSM as well as the relative algorithms. In section V, we present a case study: a weblog with security features, as well as the results through generated test objectives. In section VI, we present extensions of the approach. And finally, section VII concludes the paper and introduces the future work.

II. Related Work

Most work related to security policy can be divided into two parts: the description of the policy itself and the verification of rules. In many systems, there is no real policy specification outside of a description in terms of low-level mechanisms such as access control lists. Thereafter, the analysis of access-control leads to the definition of a number of access control models, which could provide a formal representation of security policies, and in some cases, it allows the proof of access control properties. With the great majority of models, security rules are defined with modalities like permission, prohibition and obligation that express the possible constraints on the behavior of the system [8]. Among these models, we can

mention for instance the Policy Description Language (PDL) [15], Ponder [9] and OrBAC (Organisationnal Based Access Control) [3].

Concerning the verification of rules, most work in this field deals with firewall rules testing. Early proposals consisted of performing testing of rules by hand. This implies that test construction is performed by human experts who focus on detecting traces of known attacks. Most recently, research tended to concentrate on the verification of security rules in order to detect errors or misconfigurations such as redundancy, contradiction or collision [12], [13].

Some approaches propose to focus on validation by checking the conformance of a system with respect to a security policy. In [17], authors show how an organization's network security policy can be formally specified in a high-level way, and how this specification can be used to automatically generate test cases to test a deployed system. In contrast to other firewall testing methodologies, such as penetration testing, this approach tests conformance to a specified policy. These test cases are organization-specific - i.e. they depend on the security requirements and on the network topology of an organization - and can uncover errors both in the firewall products themselves and in their configuration. However, this model is limited to the network management and specifically to network and transport layer of the TCP/IP stack. Moreover, it is still a theoretical approach and there exists no tool yet to automate the testing process and to evaluate its effectiveness on a real case-study.

In [10], the authors choose another approach to test network security rules. They express the network behavior using labeled transition systems formulae. 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 for the whole rule to perform validation.

Our approach differs from these propositions by proposing a framework to formally specify rules that can be integrated within an EFSM. It generalizes our previous work that covers testing rules with atomic activities [16] by considering decomposed ones through the conception of new integration algorithms.

III. Preliminaries

In our approach, we consider two inputs: (1) the initial system[1] and (2) the security policy. Initial system refers to the functionalities with no security consideration. After that, a new context can be evolved to meet security considerations. In the latter, the initial system will not be valid anymore since it cannot satisfy the new requirements. It has to be completed with a security policy, to fit the new context. In this paper, we propose to automatically integrate the security policy rules into the initial specification in the form of an EFSM using a specific methodology. We assume that the specification of the security policy is correct which means that we do not need to search for conflicts or redundancies.

[1]Initial system is the system under test in its original state, i.e. before we consider security (non-functional) aspect

A. The Initial Functional System

In order to model the initial functional system, we choose to use the Extended Finite State Machine (EFSM) formalism. This formal description is used not only to represent the control portion of a system but also to properly model the data portion, the variables associated as well as the constraints which affect them.

Definition 1: (EFSM) An Extended Finite State Machine M is a 6-tuple $M = < S, s_0, I, O, \vec{x}, Tr >$ where S is a finite set of states, s_0 is the initial state, I is a finite set of input symbols (eventually with parameters), O is a finite set of output symbols (eventually with parameters), $\vec{x}$ is a vector denoting a finite set of variables, and Tr is a finite set of transitions. A transition tr is a 6-tuple $tr =< s_i, s_f, i, o, P, A >$ where s_i and s_f are the initial and final state of the transition, i and o are the input and the output, P is the predicate (a boolean expression), and A is an ordered set (sequence) of actions.

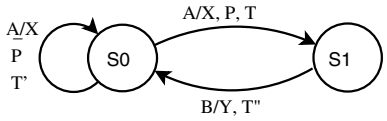

Figure 1. Example of a simple EFSM with two states.

We illustrate the notion of EFSM through a simple example described in Figure 1. The ESFM shown in Figure 1 is composed of two states S_0, S_1 and three transitions that are labeled with two inputs A and B, two outputs X and Y, one predicate P, and three tasks T, T' and T''. The EFSM operates as follows: starting from state S_0, when the input A occurs, the predicate P is tested. If the condition holds, the machine performs the task T, triggers the output X and passes to state S_1. If P is not satisfied, the same output X is triggered but the action T' is performed and the state loops on itself. Once the machine is in state S_1, it can come back to state S_0 if it receives input B. If so, task T'' is performed and output Y is triggered.

B. The Security Policy Description

A security policy is the set of laws and practices that regulates how an organization manages, protects, and distributes sensetive information.

Definition 2: (A Security Rule) is a relation between organizations, roles (sets of subjects with similar properties), views (sets of objects that satisfy a common property), activities[2] and context[2]. It is defined as a *role* having the permission, prohibition or obligation to perform within an *organization* an *activity* on a *view* in a given *context*. A *view* is a set of objects to which the same security rules apply. A typical security rule has the following form:

Obligation (S, R, A, V, C)

This rule means that within the system S, the role R is obliged to perform the activity A targeting the objects of view V in

[2]redefined later.

the context C. (The definition is similar for permission and prohibition)

Definition 3: (A Rule Context) can be divided into two parts (each part may be empty):
(1)- An *EFSM context* with conditions related to the position in the EFSM.
(2)- A *variables context* with conditions related to variables values.
In this paper, we will manage specification written using SDL (Specification and Description Language) [5] based on the EFSM formalism. The EFSM and variables contexts are defined considering this language (SDL). Using an other specification language is possible since the definition can be easily adapted.
Definition 3.1 : (EFSM Context) The EFSM context denoted SDL_c is a formula generated by the following rules:
- True, false are formula;
- (sdl_c = $Instance$) is a formula, if sdl_c is an SDL command possible within a transition (like input, task, output , set , reset , export, create_request , procedure_call , or remote_procedure_call) and $Instance$ is an instance of the SDL command. For example, (input=message1) is an EFSM context formula;
- if p is a formula then $\neg p$ is a formula too;
- if p and q are formulas, $p \wedge q$ and $p \vee q$ are formulas too.

Definition3.2 : (Variables Constraint) The constraint on the variables v_i denoted VAR_c is a formula generated by the following rules:
- True, false and boolean variables are formulas;
- (v_i Op d_v) is a formula if Op is an SDL infix operator (like $<, \leq, >, \geq, =$ or $\neq$) and d_v belongs to the domain of variables values;
- if p is a formula then $\neg p$ is a formula too;
- if p and q are formulas, $p \wedge q$ and $p \vee q$ are formulas too.

Definition 4: (An Activity) refers to a possible action within the EFSM functional description of the system. It can be either atomic or decomposable activity. *Definition 4.1 :* (An Atomic Activity) is a basic part of an EFSM transition. It is defined as an SDL command like an input, a task or an output etc.
Definition 4.2 : (A Decomposable Activity) is an activity which can be composed of a set of atomic activities. It can be:
(1) Either a partial EFSM activity EFSM_ACT: A partial EFSM activity is a connected part of the initial EFSM functional specification (from the point of view of the graph theory). The cut points can be anywhere in the transitions. Only the EFSM_ACT activities[3] will be treated in this paper.
(2) Or a sequential activity SEQ_ACT: a sequential activity is a concatenation of atomic activities. For example, if α_i (1$\leq$ i $\leq$ n) are n atomic activities then (α_1; α_2; ... ; α_n) is a non atomic activity (α_1 followed by α_2 followed by ... followed by α_n). It refers to a workflow.

[3]The SEQ_ACT activities integration will be exposed in a next paper since the integration methodology is largely different.

Definition 5: (An EFSM_ACT) is partial EFSM activity. This partial EFSM can be composed of three different parts (may be empty): (1) the starting transitions set STS: includes the set of transitions that the system can follow to start the activity. It is the first transition(s) level in the EFSM activity, (2) the ending transitions set ETS: includes the set of transitions that the system can follow to end the activity. It the last transition(s) level in the EFSM activity, (3) the intermediate transitions set ITS: includes the remaining transitions.
The transitions that starts from states in the activity and finish in states beyond the activity are called outgoing transition set OTS.

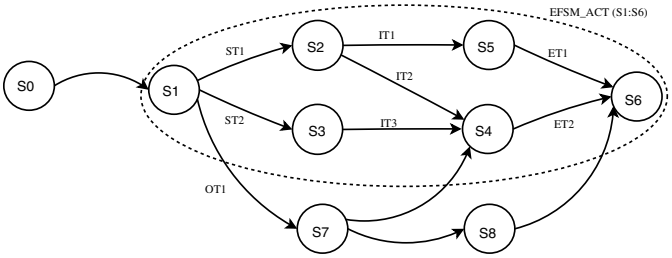

Figure 2. An example of a partial EFSM activity

In the Figure 2, we present an example of a partial EFSM activity composed of 6 states and 7 transitions. In this example, we can define 2 starting transitions ($ST1$ and $ST2$), 3 intermediate transitions ($IT1, IT2$ and $IT3$), 2 ending transitions ($ET1$ and $ET2$) and 1 outgoing transition ($OT1$).

Property 1: An activity within a permission or prohibition is an existing activity. It can correspond to one transition (we call it 1_Tr activity) or a sequence of transitions which represents a partial EFSM (we call it n_Tr activity).

Property 2: An activity within an obligation can be a new activity (refers to new states and transitions. It begins with an obligation state OS and ends with one or many end obligation states EOS_i), partially new, or an already existing activity contained in the initial system EFSM.

Property 3: If the activity within an obligation is an existing activity, it can correspond to one transition (1_Tr activity) or a set of transitions which represents a partial EFSM (n_Tr activity).
Property 4: An atomic activity is 1_Tr activity.

C. Specific Modalities

Modality 1: If a security policy rule introduces a new role and/or new variables (not already defined in the initial functional specification), precise definitions have to be provided to enable a correct integration of the rule within the functional specification (type of variable, default value, etc.).
Modality 2: The EFSM context SDL_v is mandatory in an obligation. It permits to determine the transitions to revise in the initial system EFSM.

IV. Integration Methodology

In this section, we define algorithms to automatically integrate the security policy rules into the initial specification in the form of an EFSM. It is possible that the security policy defines some new concepts that cannot be directly incorporated into the initial specification. In particular, a rule can express an activity that does not exist in the specification (new role, different object, new action, etc.). In that case, the new activity must firstly be created in the specification. That means some new states might be created in order to make the EFSM accept the new elements. Moreover, for each n_Tr activity, the algorithm creates a new Boolean variable Act_i that has $fault$ as a default value.

The beginning of the algorithm is the same for the three kinds of rules (permission, prohibition and obligation). It parses the EFSM specification and for each transition, it identifies the rules that map the activity and the *EFSM context* (only the *EFSM context* in the case of a new activity). If no rule maps the transition, the default one will be applied. After rules have been identified for each transition, we can proceed to their integration. Notice that several rules may apply to the same transition. In this case, the algorithm is recursively applied to each relevant rule.

A. Permissions Integration

Permissions relate to activities which already exist in the initial system. The algorithm 1 shows the permissions integration within the EFSM system.

Algorithm 1 Permission integration

Require: The permission with role R, variable context VAR_c and activity i that maps the transition(s).

1: **if** (1_Tr activity) **then**
2: Revise the associated predicated to the transition: $P := P \wedge (VAR_c \wedge R)$
(Note that if no predicate is associated to this transition, we create a new one $P := VAR_c \wedge R$)
3: **end if**
4: **if** (n_Tr activity) **then**
5: Add the task $'Act_i := true;'$ to the STS.
6: Add the task $'Act_i := false;'$ to the OTS
7: Duplicate the ETS into ETS_1 and ETS_2
8: Revise the associated predicated to the ETS_1: $P := P \wedge Act_i \wedge (VAR_c \wedge R)$
9: Revise the associated predicated to the ETS_2: $P := P \wedge (Act_i = false)$
10: Add the task $'Act_i := false;'$ to the ETS_1.
11: **end if**

Considering the EFSM, each activity corresponds to one (or many) transitions. (1) In the case of *1_Tr* activity, only one transition will be modified. If this transition contains no predicate, a predicate has to be added. On the other hand, if a predicate is already defined in the specification, it only needs to be further restrained (the condition is stronger). (2) If we are dealing with an n_Tr activity, the restriction will be only in the ending transitions set. The STS, ITS and OTS will only mention that we are dealing the right activity or not.

If many permissions can be applied the same transition, the predicate is restrained using a logical sum: $P := P \wedge (\vee_i (VARc_i \wedge R_i)$.

B. Prohibitions Integration

Like permissions integration, prohibitions integration consists either of adding a new predicate or restraining an existing one (it becomes stronger).

Algorithm 2 Prohibition integration

Require: The permission with role R, variable context VAR_c and activity i that maps the transition(s).

1: **if** (1_Tr activity) **then**
2: Revise the associated predicated to the transition: $P := P \wedge (\neg VAR_c \vee \neg R)$
(Note that if no predicate is associated to this transition, we create a new one $P := \neg VAR_c \vee \neg R$)
3: **end if**
4: **if** (n_Tr activity) **then**
5: Add the task $'Act_i := true;'$ to the STS.
6: Add the task $'Act_i := false;'$ to the OTS
7: Duplicate the ETS into ETS_1 and ETS_2
8: Revise the associated predicated to the ETS_1: $P := P \wedge Act_i \wedge (\neg VAR_c \vee \neg R)$
9: Revise the associated predicated to the ETS_2: $P := P \wedge (Act_i = false)$
10: Add the task $'Act_i := false;'$ to the ETS_1.
11: **end if**

If many permissions can be applied the same transition, the predicate is restrained using a logical sum: $P := P \wedge (\vee_i (\neg VARc_i \vee \neg R_i)$.

C. Obligation Integration

We will only consider in this paper the EFSM_ACT activities. These activities begins by a starting obligation state OS and ends with a set of end obligation states EOS_i. The OS and EOS_i can be part of the initial functional system specification. Thanks to the EFSM context of the obligation, the algorithm identifies the transition which will be split into two (pre/post transitions), to insert the EFSM_ACT of the obligation. Then, the algorithm needs to know how the components of this transition will be distributed relatively to the obligation (pre/post transitions). This can be determined using the *cut point*, that corresponds to the *last component* of the initial transition (state, input, task or output, but not a predicate) which maps the EFSM context. Each component until this *cut point* (included) will be attributed to the pre-transition (through the obligation) while other ones will be attributed to the post-transition. Finally, a last transition has to be added to bypass the obligation in the case the initial predicate is not satisfied (see Algorithm 3).

The Figure 3 shows an example of the process. In this case, the initial transition is "$A/X, if(P), T$". The EFSM_ACT

Algorithm 3 Obligation integration

Require: The transition $tr =< S_1, S_2, A, X, P, t_1...t_n >$ that maps the obligation with an activity specified by the mean of an EFSM with OS as a first state and EOS_i as a last one

for all (transitions from OS) **do**
  if ($\exists$ associated predicate Q) **then**
    $Q := Q \vee (VAR_c \wedge R)$
  end if
end for
determine the cut point C_utP_oint
delete the transition tr
create transitions C_1, C_2 and C_3 such that
if ($C_utP_oint == S_1$) **then**
  $C_1 :=< S_1, OS, -, -, -, - >$
  if ($\neg$ $\exists$ EOS state in M) **then** $C_2 :=< EOS_i, S_2, A, X, P, t_1...t_n >$ **end if**
  $C_3 :=< OS, S_2, A, X, \neg VAR_c \vee \neg R, t_1...t_n >$
else
  if ($C_utP_oint == A$) **then**
    $C_1 :=< S_1, OS, A, -, P, - >$
    if ($\neg$ $\exists$ EOS_i state in M) **then** $C_2 :=< EOS_i, S_2, -, X, -, t_1...t_n >$ **end if**
    $C_3 :=< OS, S_2, -, X, \neg VAR_c \vee \neg R, t_1...t_n >$
  else
    if ($C_utP_oint == t_i$ where $i \in \{1, ..., n\}$) **then**
      $C_1 :=< S_1, OS, A, -, P, t_1...t_i >$
      if ($\neg \exists$ EOS state in M) **then** $C_2 :=< EOS_i, S_2, -, X, -, t_{i+1}...t_n >$ **end if**
      $C_3 :=< OS, S_2, -, X, \neg VAR_c \vee \neg r, t_{i+1}...t_n >$
    else
      if ($C_utP_oint == X$) **then**
        $C_1 :=< S_1, OS, A, X, P, t_1...t_n >$
        if ($\neg \exists$ EOS state in M) **then** $C_2 :=< EOS_i, S_2, -, -, -, - >$ **end if**
        $C_3 :=< OS, S_2, -, -, \neg VAR_c \vee \neg R, - >$
      end if
    end if
  end if
end if
minimize the resulting EFSM by deleting silent transitions (without input nor output nor action)

activity is a new partial EFSM with two states (OS and EOS) and one transition characterized by the input B, the task T' and the output Y. According to the EFSM context, the *cut point* is the Input A. By the following, the transition $C1$ (pre-transition) is defined by the input A and the predicate P. The transition $C2$ (post-transition) is defined by the task T and the output X. Obligation integration is shown in the Algorithm 3.

D. Integration Result

The security rules integration allows us to obtain a new specification of the system that takes into account the security policy. This formal specification is described in the EFSM formalism that is a well adapted one to model communicating

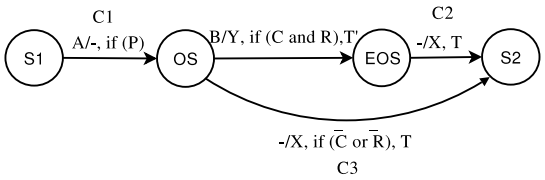

Figure 3. Obligation (S, R, new activity, _ , (Input = A) and C).

systems. Using SDL (Specification Description Language) [5] based on this formalism we can easily derive test sequences to check whether the implementation of the system conforms to the secure functional specification. The classical test generation methodology is presented in the ISO9646 standard [2].

V. Case-study: A Weblog

A. Weblog Description

To prove the effectiveness of our framework, we choose to carry out a case-study on a weblog (also called blog): a blog is a website used to post stories or news (such as in a journal or diary) to make them available for reading by any other party. Here, we consider at first, a simple weblog with various features, such as those commonly used on the World Wide Web. First of all, the service is open. That means anyone cannot only read but also post content in the form of news or stories. Then, other readers might add comments relating to any content. Possibly, a blogger[4] might take the decision to delete a posted content depending of its freshness or its relevance. The weblog can thus be seen as a mutable list of stories, which constitute themselves a content associated with a mutable, possibly empty, list of comments.

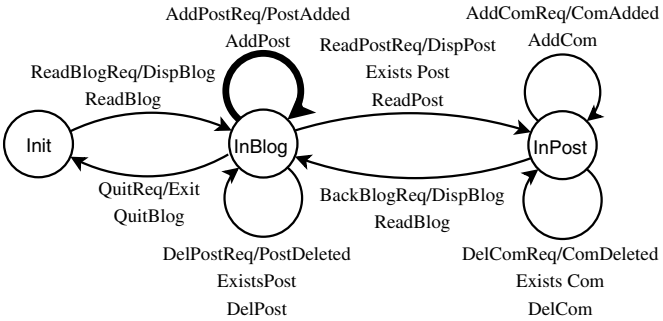

Figure 4. The initial system EFSM.

As one can notice, the initial model is voluntarily open and as a consequence, presents obvious security flaws. Indeed, no authentication is performed so that any user can delete numerous posted contents, which leads to a kind of denial of service attack. To tackle this problem, we specify a security rule whose goal is to protect the information within the organization, by preventing illegitimate users to delete any content.

For this purpose, the security rules will define a hierarchy of users, by defining three different roles. The first one is the

[4]Contraction for weblog user.

administrator (*admin*). It has the responsibility to maintain the global organization of the website. It is the only one authorized to delete a posted content or to suspend the activity of the website (in the case of maintenance for example). Beside the administrators, the policy will define another role: the bloggers. Bloggers are users which can post stories and also commentaries, relating to their own content or not. Moreover, they are allowed to perform the *delete* action but only on their own content. Finally, the normal users (called visitors or anonymous) can only read stories and commentaries; that means *delete* and *write* actions are prohibited for them.

B. The formal specification

To specify the Web application, we used *ObjectGEODE* [19] and *SIRIUS* [6] tools. They are based on a language specifically dedicated to the formal specification of interactive systems: SDL which is a specification and description language standardized by the ITU[5] (International Telecommunication Union). The key features of the language are:

- The ability to be used as a wide spectrum language from requirements to implementation;
- Suitability for real-time, stimulus-response systems;
- A model based on communicating processes (extended finite state machines);
- Object oriented description of SDL components.

C. The Security Policy Specification and Integration

The global system on which relate all modalities in the security policy is the *Website*. The objects will naturally correspond to each of the components defined initially, that are the *blog*, the *posts* and the *comments*. In the same manner, the first actions will be chosen among the existing in the initial system: *read*, *write* and *delete*. As for the context, it defines the set of conditions expressed by a rule, which have to hold to allow an activity. Based on these considerations, we specified 24 rules constituting the policy that manages the security of our weblog. Here are 3 different examples of theses rules:

1) Obligation(Website, anonymous, Authentication, _ , input = AddPostReq)
2) Permission(Website, admin, 'Reading Blog', blog,_)
3) Prohibition(Website, anonymous, 'Adding Post', Post, _)

After the specification of all rules, we step to the second phase of the process, which is their integration within the extended finite state machine. This integration process respects the methodology described in section 4 and leads to a secure functional specification.

Thus, we performed syntaxic and semantic verification of the new specification to ensure its correctness. Table I shows some metrics about the two weblog specifications (before and after security rules integration)

[5]http://www.itu.int

	States	Transitions	Signals	Nb of lines
Before	3	15	15	350
After	3	23	18	594

Table I
METRICS OF THE WEBLOG SPECIFICATION

D. Test Generation

1) Fixing the test objectives: In the Weblog case study, our aim was to test the security rules. The first idea was to define for each rule one or several test objectives. However, we noticed that one generated test sequence can verify more than one security rule. Then, we tried to minimize the number of the test objectives to test the entire modified transitions. We specified at the end 17 test objectives that represent more than 94% of the specification transitions.

2) Generation with SIRIUS: SIRIUS is based on Hit-or-Jump [6], an algorithm especially used for components testing to perform test sequences generation through the specification. This research is guided by objectives which are illustrated by predicates on transitions (and written in SDL). Research in the partial reachability graphs is performed in depth, width or both at the same time, and is restricted by a limited depth. In order to initialize the generation of test sequences, several parameters are necessary. Four main files must be developed. The first is the service specification (component to be tested), the second allows the initialisation of some variables if necessary, the third one mentions the stop conditions (i.e. test objectives) and finally the last one allows the expert to guide the system at the beginning of the simulation, this file is called preamble. This last one is very important; it allows reducing in a consequent way the length of the test sequence and the duration of its generation.

E. Discussions

The results are obtained after a BFS (breadth-first search) exploration of the reachability graph. This choice is due to the specificity of the weblog service which has to take into account, after each transition, all the possible inputs injected by the user to analyze them and generate the right output. The test objectives are reachable via short sequences according to the specification size of the system and do not need a DFS or BDFS exploration that tries to search in depth of the reachability graph. The generated test sequences are usable since they can be produced in TTCN [18] and MSC [1] standard notations facilitating their portability.

VI. EXTENSIONS

A. Delegation Integration

A delegation rule allows the handing of a task over to another person, usually a subordinate. It is the assignment of authority and responsibility to another person to carry out specific activities. It allows a subordinate to make decisions, i.e. it is a shift of decision-making authority from one organizational level to a lower one. We can consider the delegation as a permission given to a specific subject (and not all the subjects of a specific role). Thus, the integration algorithm for delegation rules is the same as for permission.

B. Security Policy Updating

In this paper, initial system refers to its functionalities (or behaviors) with no security consideration. We can revise this assumption, to consider an initial system as a system that takes into account an initial security policy. Later, a new context can evolve to meet new security considerations. Within this new one, the initial system is not valid anymore since it cannot satisfy new requirements. We can always use our methodology to integrate the new security rules to fit the new context.

C. Test Objectives Optimization

The integration of the security policy rules into the initial specification *Specinit* results in a new system specification $NewSpec$. If the system specification is modified, we need obviously to modify the implementation of the system considered as the new IUT (Implementation Under Test). The modification of the specification concerns only some transitions (some added or revised transitions). The automatic test generation can be targeted and as a result, easier and less time consuming. In fact, if we assume that the unmodified implementation $ImpIn$ has been tested and is equivalent to the unmodified specification $SpecIn$ and that only transitions that correspond to modified transitions of $SpecIn$ are modified in $ImpIn$, we are not obliged occording to [11] to check and test all the new implementation. It is even enough to test only the modified transitions when the final state of each modified transition has a correspondent state in the unmodified part of the modified $NewSpec$ and when each modified transition is reachable through unmodified transitions in the modified $NewSpec$. We call this method incremental testing.

The table below presents experimental results using different percentages of modification of the initial specification. H is the ratio calculated by the division of the length of test cases using HSI (for Harmonized State Identification) test generation method [20] by the length of test cases using incremental testing.

Modification	0-5%	5-10 %	10-15 %	15-20 %
H	36.0	11,3	6.1	4

We can clearly notice that the ratio H slightly increases when the number of transitions increases.

VII. Conclusion and Future Work

In this paper, we presented a framework for testing a security policy in a formal manner. We proposed an algorithm to automate the integration of security rules within an EFSM. Then, we presented a scheme to derive test objectives from the rules formal specification in order to test the conformance of a security policy with respect to its implementation. We described the process of our framework through a representative case-study. We showed that our approach allows the specification of various modalities such as obligation, permission, prohibition and delegation and makes it possible to obtain relevant test objectives. It is important to notice that our algorithm allows us to verify that the security policy has no gap (missing rules) when no rule is found to be applied on a functional specification transition.

As a future work, we are currently investigating several approaches to enhance this framework. At first, we consider extending our integration scheme to be able to take into account more complex modalities (such temporal rules (which denote actions limited in time) and interoperability of rules (that is testing if a rule can be deployed on several systems)). We are also working on a new approach to test of workflow represented by sequential activities.

References

[1] *IUT-T Rec. Z. 120 Message Sequence Charts, (MSC)*. Geneva, 1996.

[2] I. 9646-1. *Information Technology - Open Systems Interconnection - Conformance testing methodology and framework Part 1: General Concepts*.

[3] A. Abou El Kalam, R. E. Baida, P. Balbiani, S. Benferhat, F. Cuppens, Y. Deswarte, A. Miège, C. Saurel, and G. Trouessin. Organization Based Access Control. In *4th IEEE International Workshop on Policies for Distributed Systems and Networks (Policy'03)*, June 2003.

[4] A. V. Aho, A. T. Dahbura, D. Lee, and M. U. Uyar. An optimization technique for protocol conformance test generation based on uio sequences and rural chinese postman tours. pages 427–438, 1995.

[5] A. Cavalli and D. Hogrefe. Testing and validation of SDL systems : Tutorial. In *SDL'95 forum*, 1995.

[6] A. Cavalli, D. Lee, C. Rinderknecht, and F. Zaïdi. Hit-or-Jump: An Algorithm for Embedded Testing with Applications to IN Services. In *Formal Methods for Protocol Engineering And Distributed Systems*, pages 41–56, Beijing, China, october 1999.

[7] F. Cuppens, N. Cuppens-Boulahia, and M. B. Ghorbel. High-level conflict management strategies in advanced access control models. In *Workshop on Information and Computer Security (ICS)*, Timisoara, Roumania, September 2006.

[8] N. Damiannnou, A. Bandara, M. Sloman, and E. Lupu. *Handbook of Network and System Administration*, chapter A Survey of Policy Specification Approaches. Elsevier, 2007 (to appear).

[9] N. Damianou, N. Dulay, E. Lupu, and M. Sloman. The ponder policy specification language. In *POLICY '01: Proceedings of the International Workshop on Policies for Distributed Systems and Networks*, pages 18–38, London, UK, 2001. Springer-Verlag.

[10] V. Darmaillacq, J.-C. Fernandez, R. Groz, L. Mounier, and J.-L. Richier. Test generation for network security rules. In *TestCom*, pages 341–356, 2006.

[11] K. El-Fakih, N. Yevtushenko, and G. von Bochmann. Fsm-based incremental conformance testing methods. *IEEE Trans. Software Eng.*, 30(7):425–436, 2004.

[12] J. García-Alfaro, F. Cuppens, and N. Cuppens-Boulahia. Analysis of policy anomalies on distributed network security setups. In *ESORICS*, pages 496–511, 2006.

[13] J. García-Alfaro, F. Cuppens, and N. Cuppens-Boulahia. Towards filtering and alerting rule rewriting on single-component policies. In *SAFECOMP*, pages 182–194, 2006.

[14] D. Lee and M. Yannakakis. Principles and methods of testing finite state machines - A survey. In *Proceedings of the IEEE*, volume 84, pages 1090–1126, 1996.

[15] J. Lobo, R. Bhatia, and S. A. Naqvi. A policy description language. In *AAAI/IAAI*, pages 291–298, 1999.

[16] W. Mallouli, J.-M. Orset, A. Cavalli, N. Cuppens, and F. Cuppens. A formal approach for testing security rules. In *SACMAT*, Nice, France, 2007.

[17] D. Senn, D. A. Basin, and G. Caronni. Firewall conformance testing. In *TestCom*, pages 226–241, 2005.

[18] E. TTCN-3. *TTCN-3 – Core Language*.

[19] Verilog. *ObjectGEODE Simulator, Reference manual*, 1997.

[20] N. Yevtushenko and A. Petrenko. Synthesis of test experiments in some classes of automata. *Automatic Control and Computer Sciences.*, (4):pp : 50–55, 1998.