Modeling and Testing Secure Web-Based Systems: Application to an Industrial Case Study

Wissam Mallouli, Mounir Lallali, Gerardo Morales and Ana Rosa Cavalli
Institut Telecom / Telecom SudParis, CNRS / SAMOVAR, France
{wissam.mallouli, mounir.lallali, gerardo.morales, ana.cavalli}@it-sudparis.eu

***Abstract*—Ensuring that a Web-based system respects its security requirements is a critical issue that has become more and more difficult to perform in these last years. This difficulty is due to the complexity level of such systems as well as their variety and increasing distribution. To guarantee such a respect, we need to test the target Web system by applying a complete set of test cases covering all the possible scenarios. To reach this aim, we first specify the Web system behavior from its functional point of view using IF language. Second, this model is augmented by applying a set of dedicated algorithms to integrate timed security properties specified in Nomad language. This language is well adapted to express security properties with time constraints. Then, we use a dedicated tool called TestGen-IF, to perform an automatic test generation of test cases targeting security purposes. These test sequences are transformed in executable test cases that can be applied on a real Web application. We present in this paper an industrial Web-based system provided by France Telecom[1] as a case study to demonstrate the reliability of our framework.**

***Index Terms*—Web Applications, Timed Automata, Security Policy Specification, Nomad Language, Test Generation, Test Execution, Security Validation.**

I. Introduction

In recent years Web-based systems have become extremely popular and, nowadays, they are used in critical environments such as financial, medical, and military systems. As the use of Web applications for security-critical services has increased, the number and sophistication of attacks against these applications have grown as well. For this reason it is essential to be able to prove that the target Web-based system implements the security requirements it should respect (described generally using a security policy language) either by (i) injecting the security policy in the studied Web system or (ii) by formally specifying the target system and generating proofs demonstrating that it implements the security policy or (iii) by considering several strategies based on formal testing. This last methodology will be explored in this paper using, more precisely, model-based active testing techniques [6], [19].

Model-based testing consists in deriving a suite of test cases from a model representing the Web system behavior. Such a model can be generated from an informal specification of the system and designed by software engineers through the use of diagram manipulation tools. Moreover, Web applications can have a time dependent behavior as well as an increasing demand for security mainly due to their increased complexity and inherent distribution. Consequently, engineers developing these Web systems are not only confronted to functional requirements but also have to manage other kinds of requirements concerning security issues. Roughly speaking, by "functional requirements" we mean the services that a Web application has to offer to end users. Whereas, security rules denote the properties that a system has to fulfil so that it is always in a safe state and guarantees service quality.

To tackle this problem we rely, in this paper, on a formal approach to integrate elaborated security rules involving time constraints into a formal specification of the system based on communicating extended timed automata [3] supported by the IF (Intermediate Format) language [2]. The derivation of the test cases can be done automatically, providing generic test cases described in a standard language. By executing the model-based test cases, the conformance of the implemented system to its specification can be validated. More precisely, the main contributions of this paper are:

- The specification of Web application features using the IF language. This language is well-adapted to formally describe Web systems features such as hyperlinks, sending and receiving data and client-server communications, etc.
- The definition of Web system security requirements based on the Nomad [8] (stands for Non Atomic Actions and Deadlines Model) formal language. Nomad allows specifying, unambiguously, security rules (such as permissions, prohibitions and obligations) in specific contexts that include time constraints.
- The integration of the security rules within the functional IF model to obtain a secure specification that takes into account the security requirements. This integration is performed according to the algorithms presented in [16].
- The automatic generation of test cases targeting security constraints. This generation is performed using TestGen-IF tool based on Hit-or-Jump algorithm [4].
- These test cases are instantiated into TCL script language [22] and are applied, in an automated manner, on a real Web-based system (Travel Web application provided by France Telecom) to check whether its behavior respects the security requirements. The application and the analysis of the designed test cases are performed by tclwebtest tool [23].

This paper is organized as follows. In section II we discuss the related work on formal modeling and testing of secure systems with timed constraints. Section III exposes an overview

[1]France Telecom is the main telecommunication company in France

of our methodology to specify and test the security of Web-based systems. In sections IV, V, VI and VII, we apply our methodology to an industrial case study (called Travel) by, respectively, specifying its functional behavior, integrating its security requirements, automatically generating security-target test cases and automatically performing them using tclwebtest tool. Finally, section VIII presents the conclusion and introduces future work.

II. Related Work

Many models are proposed in the literature for the formal specification of Web applications from their functional point of view. These models sometimes include functional time constraints. In [21], for instance, the authors present a methodology that specifies Web-based systems using the SDL formal language [10]. This language is based on Extended Finite State Machines (EFSM) model [14] and is well adapted for describing communicating systems. Others studies are based on timed automata theory [1] and allows specifying functional system requirements with timed pre and post conditions. This paper is based on the IF language [3] because it allows providing the main concepts to design Web-based systems with time constraints. Moreover, several tools allowing the simulation and the generation of test sequences exist and are open source. However a main issue here to guarantee the system reliability, is that this functional specification has to be completed by integrating system security aspects.

To tackle this problem we have introduced in an earlier publication [17] a formal approach that permits to augment a functional description of a system with security rules expressed with the Or-BAC language [13] (Organisationnal Based Access Control). We described security rules that specify the obligation, permission or prohibition for a user to perform some actions under given conditions called *context*. This context does not involve time aspects since we only specified rules without considering them. We applied this approach to a Weblog case study in order to validate its security and the results were satisfactory.

To be able to include timed security rules (for instance, a file system may have to specify the prohibition for a user to access to a specific document if he/she is not authenticated or if his/her session of 10 minutes has been expired), we propose, in this paper to rely on the Nomad language [8] that supports time constraints. We integrate the defined Nomad security rules that the system should respect, with its IF formal specification through the use of specific algorithms described in [15], [16]. The obtained system specification is called a secure system specification.

Many tools [12], [26] allowing automatic test generation [18] from IF specifications exist in the literature. In this paper, we rely on our own generation tool, called TestGen-IF, that efficiently constructs tests sequences with high fault coverage. The tool avoids the state explosion and deadlock problems encountered in exhaustive or exclusively random searches used in classical tools. The execution of the generated tests are performed using tclwebtest tool [5], well adapted to check the stability and scalability of Web applications.

III. Testing Methodology Overview

In this section, we present the proposed testing methodology to test security rules. This methodology manipulates three different inputs:

- a functional specification of the Web application based on the IF formal language,
- a specification of the security policy that the application has to respect (based on the Nomad language),
- and finally an implementation of the Web-based system.

The aim is to generate a new specification of the Web-based system (called secure specification) that takes into account the security policy, and then to generate a complete test suite to check whether the implementation of the system conforms to this secure functional specification.

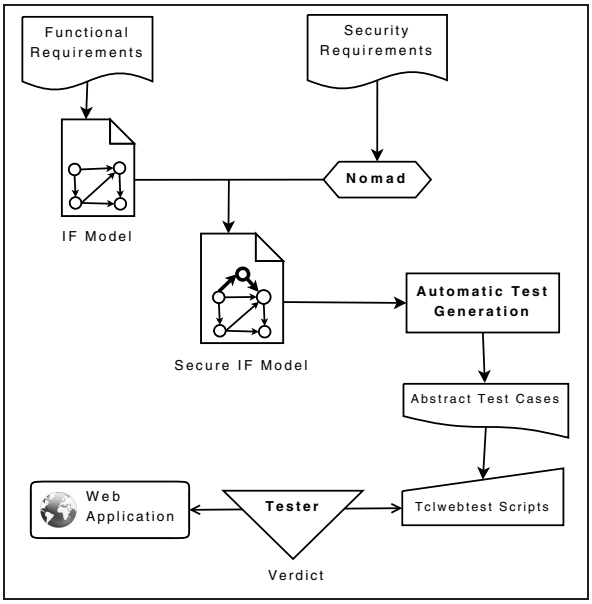

Figure 1. Testing Methodology Overview

To reach this goal we first define a set of algorithms to automatically integrate different types of security policy rules described using the Nomad language within the initial functional system specification. The integration process is twofold. First, the algorithm determines the rules to be applied on each transition of the IF specification. Then, it modifies each transition by adding clocks to represent the time progress and also by adding guards (or predicates), transitions and/or states to make the execution instance of a given action possible only under a specific clock valuation. More details on the algorithms used to perform the rules integration and their correctness proofs are given in [15], [16].

Finally, we define an end-to-end methodology for testing Web applications as presented in Figure 1. The main steps of our testing methodology are the *specification* of a secure system based on the IF language, the *automatic test generation* based on TestGen-IF tool (test cases produced are in aldebaran standard [9]), their instantiation to *executable test cases* and the *application* of these test cases on a real Web application using tclwebtest language. The analysis of the Web application is performed using a dedicated tool called ACS-Automated-Testing incorporated in the OpenACS platform [20].

IV. Functional Specification of Travel Web System

A. Travel Application Description

To prove the effectiveness of our framework we carried out a case-study using a Travel application which is an internal service used by France Telecom company to manage 'missions' (business travels) carried out by its employees. In our case study we only consider, at first, a simple Travel application where a potential traveler can connect to the system (using a dedicated URL) to request a travel ticket and a hotel reservation during a specific period according to some business purposes (called mission). This request can be accepted or rejected by his/her hierarchical superior (called validator). In the case it is accepted, the travel ticket and hotel room are booked by contacting a travel agency. The specification of this Travel Web application is performed using the IF language [2].

Further, we defined some specific security rules to boost the system security. These security rules are inspired from France Telecom's security test campaign in the context of POLITESS project[2]. The security rules are formally specified using the Nomad model.

B. Background on IF

The Intermediate Format (IF) language [3] can be considered as a common representation model for other existing languages. It was originally developed to act as intermediator between languages such as SDL, Promela or Lotos but it has been extended to tackle other notations, such as UML [7]. IF is based on communication timed automata and it is used to describe and also validate asynchronous systems.

A communicating system described using the IF language is composed of active process instances running in parallel and interacting asynchronously through shared variables and signals via *signalroutes* or by direct addressing. A process instance can be created and destroyed dynamically during the system execution. It has local data and a private FIFO buffer. Each IF process is described as a timed automaton extended with discrete data variables and clocks, communication primitives and urgency attributes on transitions. Each *transition* of this automaton has an enabling guard (on data variables and clocks) and a set of *actions* (i.e., *assignments*, signal *inputs* and *outputs*, process *creation* and *destruction*).

The clocks values are real numbers that can be set or reset. Time progresses in states whereas transitions take zero time to be executed. The transition urgency [3] is used to control the time progress: (i) *eager transition*: is urgent as soon as it is enabled, and blocks time progress; (ii) *lazy transition*: is never urgent, and never blocks time progress; (iii) *delayable transition*: allows waiting as long as time progress does not disable it.

IF is a language that is well adapted to specify Web systems. In particular, it can be used to define a set of communicating *processes* to describe different Web application entities and functionalities. More details about the IF language can be found in [2].

[2]http://www.rnrt-politess.info/

C. Travel IF Specification

Modeling Web applications allows software engineers to specify, understand and maintain their complex architecture in an optimized manner. To perform this formal specification, we use the IF language to model the functional behavior of the Travel communicating Web application. This specification provides the metrics in the table I.

Processes	States	Transitions	Signals	Variables
basic_traveler	5	12	13	11
traveler_mission	7	12	11	8
basic_travel	2	7	7	8
travel_mission	9	11	14	6

Table I
IF Travel System Specification

The IF model is composed of four processes. Each process communicates with the other using a set of signals:

- *basic_travel* and *travel_mission* are two processes that describe the Travel system behavior. *basic_travel* allows to communicate with a basic user of the system whereas *travel_mission* allows to manage the 'missions' requested by a potential traveler.
- *basic_traveler* and *traveler_mission* are two processes that describe the user behavior. The first process simulates a basic traveler that can change its profile, delegate its rights, request the creation of a mission or its validation. Whereas *traveler_mission* describes a potential traveler that can choose the details of its business travel.

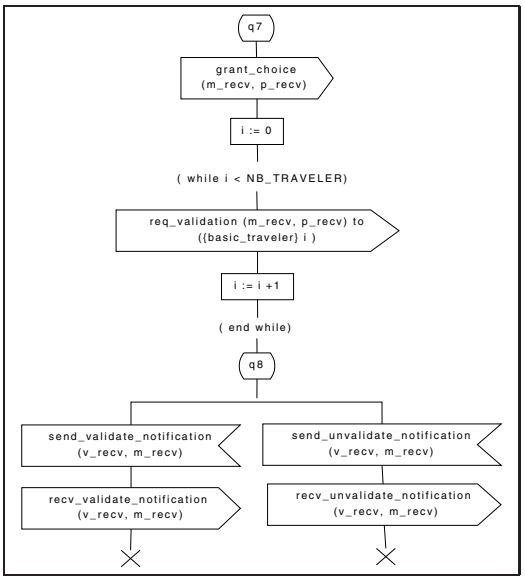

Figure 2. An IF State in the Travel_Mission Process: q7 and q8 States

The IF specification of the Travel system is finite but large. For matter of space, we only present, in Figure 2, two states (q_7 and q_8) from the *basic_travel* process. In the state q_7, the system asks for a validation relating to a mission request (output req_validation()). State q_8 has two transitions. The input in the left transition (resp. right transition) is received from the mission validator that sent an acceptance (resp. a reject) notification to the Web-based system. This notification is transmitted to the potential traveler using the output signal (output recv_(un)validate_notification()).

V. Security Integration for Travel Web System

A. Nomad Background

The Nomad formal language is used to specify, without any ambiguity, the set of security properties that the Travel Web system has to respect. The choice of this language was mainly motivated by the possibility offered by Nomad to describe permissions, prohibitions and obligations related to non-atomic actions within elaborated contexts and time constraints. By combining deontic and temporal logics, Nomad language can be used to describe conditional privileges and obligations with deadlines. To meet the requirements of the functional model of the system, we define atomic actions using the same concepts found in IF.

Definition 1: (Atomic action) We define an atomic action as one of the following actions: a variable assignment, a clock setting, an input action, an output action, a process creation or a process destruction

Definition 2: (Non-atomic action) If A and B are actions, then $(A;B)$, which means "A is followed immediately by B" is non-atomic action.

Definition 3: (Formulae) If A is an action then $start(A)$ (starting A), and $done(A)$ (finishing A) are formulae.

Here are some properties on actions and formulae:

- If α and β are formulae then $\neg\alpha$, $(\alpha \wedge \beta)$ and $(\alpha \vee \beta)$ are formulae.
- If α is a formula then $O^d\alpha$ (α was true d units of time ago if $d \leq 0$, α will be true after d units of time if $d \geq 0$) is a formula too.
- If α is a formula then $O^{<d}\alpha$ (within d units of time ago, α was possibly true if $d \leq 0$, α is possibly true within a delay of d units of time if $d \geq 0$) is a formula.
- If α and γ are formulae then $(\alpha|\gamma)$ is a formula whose semantics is: in the context γ, the formula α is true.

Notice also that using the Nomad language we deal with a discrete time. The choice of the unit of time can be very important and depends on the studied system. In our work, we use real time units (e.g. minutes).

Definition 4: (A security rule) If α and β are formulae, $\mathcal{R}$ $(\alpha \mid \beta)$ is a security rule where R denotes one of the following deontic operators: {$\mathcal{P}$, $\mathcal{F}$, $\mathcal{O}$}. $\mathcal{P}$ $(\alpha \mid \beta)$ (resp. $\mathcal{F}$ $(\alpha \mid \beta)$, $\mathcal{O}$ $(\alpha \mid \beta)$) means that it is permitted (resp. prohibited, mandatory) to execute α when context β holds.

More details about the syntax and semantics of Nomad are presented in [8].

B. Travel Security Specification Using Nomad Language

France Telecom proposed a preliminary version of the case study Travel in which some informal security requirements are provided. Based on these requirements, we formally specified a set of 34 security rules using the Nomad language. For matter of space, we only present the following three:

- Rule 1:

$$\mathcal{F}\ (start\ (output\ req\_create\_mission(t))| \\ O^{\leq -2min}\ done\ (output\ req\_create\_mission(t)))$$

  This first prohibition rule expresses that two missions requests of the same traveler must be separated by at least 2 minutes. This request can be performed in the *basic_traveler* process.
- Rule 2:

$$\mathcal{P}\ (start\ (output\ req\_proposition\_list(t,m))| \\ O^{\leq -10min}\ done\ (output\ req\_proposition\_list(t,m)))$$

  This permission rule expresses that a traveler can request for another list of travel propositions within a delay of 10 minutes if he/she already asked for a first list of travel propositions. This request can be performed in the *traveler_mission* process.
- Rule 3:

$$\mathcal{O}\ (start\ (output\ req\_validation())| \\ O^{-10080min}\ done\ (output\ req\_validation()) \\ \wedge\ O^{\leq -10080min}\ ((\neg\ done\ (input\ recv\_validate\text{-}\_notification())) \wedge (\neg\ done\ (input \\ recv\_unvalidate\_notification()))))$$

  This obligation rule expresses that if a traveler requested for the validation of his/her mission and if he/she did not received an answer, the system must send, as a reminder, another request to the potential mission validator. This reminder is sent within a delay of (10080 min = 7 days). The requests and answers are made in the *travel_mission* process.

C. Rules Integration Results

The integration of the security rules was performed based on the methodology and algorithms described in [15], [16]. The integration process is based on three aspects:

- The classification of the timed security rules into three distinguished classes. The first two classes denote basic rules including both atomic and non-atomic actions with simple contexts. A simple context only includes a single timed operator and no logical connectors. The third class is general and deals with elaborated security rules that include more complex contexts.
- The algorithms, to integrate security rules with an IF specification (according to each class), consist in adding clocks to represent the time progress and; also; in adding or modifying guards, transitions and even states to make the execution instance of a given action possible only under a specific clock valuation.
- The correctness proof of the integration algorithms, which demonstrates the accuracy of the integration procedure.

The following table II shows some metrics about the modifications after the integration of some specific rules: the modified and added transitions (M&A Transitions), the added variables and clocks (Added Var & Ck), the added processes (Added Proc).

Rule	M&A Transitions	Added Var & Ck	Added Proc
1	1+1	1	0
2	2+1	1	0
3	4+3	4	1

Table II
IF Travel System Modifications According to Each Rule

As an example, let us consider the Travel specification part described in Figure 2. These transitions are modified during

the integration of the third rule and leads to the creation of a new process forked in the transition triggered from state q_7. Figure 3 describes the resulting transitions. The new variable $wait\_a$ has 0 as a default value. If $wait\_a$ is positive, this means that the system is waiting for a mission validation. The process RHP (for Rule Handler Process) launches a clock in its first state. When the clock valuation reach the deadline of 7 days, the system verifies the $wait\_a$ value, if ($wait\_a$>0) the RHP process sends another validation request. Otherwise ($wait\_a \leq 0$ which means that the system got through one of the transitions of the state q_8 and already received an answer from the mission validator), the RHP process is stopped without performing any action.

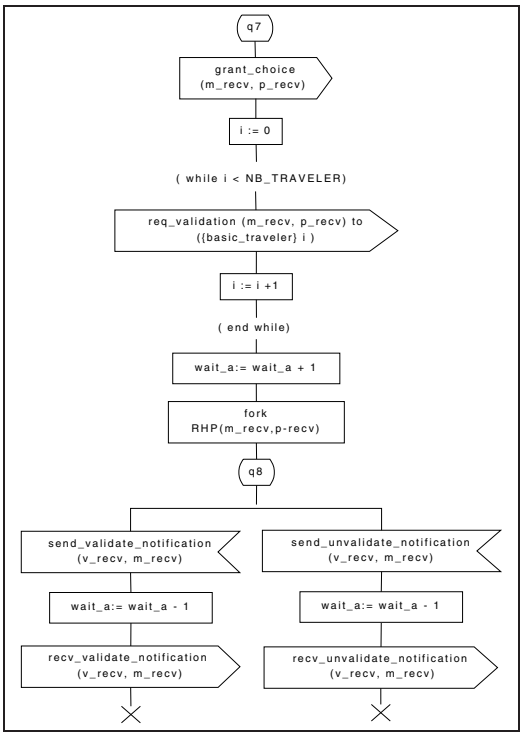

Figure 3. Resulting Transitions After Security Integration

VI. Test Generation

A. TestGen-IF tool

To automatically generate test cases from the secure specification of Travel, we use the *TestGen-IF* test generation tool developed in our laboratory.

1) Test Generation Algorithm: TestGen-IF implements a timed test generation algorithm based on a Hit-or-Jump exploration strategy [4]. This algorithm efficiently constructs test sequences with high fault coverage, avoiding the state explosion and deadlock problems encountered respectively in exhaustive or exclusively random searches. It allows to produce a partial accessibility graph of the system under test (SUT) specification in conjunction with the IF simulator [3].

At any moment, a local search is conducted from the current state in a neighborhood of the reachability graph. If a state is reached, and one or more test purposes are satisfied (a Hit), the set of test purposes is updated and a new partial search is conducted from this state. Otherwise, if a search depth limit is reached without satisfying any test purpose, a partial search is performed from a random graph leaf (a Jump). This algorithm terminates when all the test purposes are satisfied or when no transition is left to explore. The test case is the path constructed on the fly from the initial state of the SUT specification containing all the hit and jump states.

2) TestGen-IF Architecture: The active testing tool is illustrated by the Figure 4. The Properties (Test Purposes) box represents the timed system objectives to be tested (see Section VI-B1). The Automatic Test Generation box represents the test generation procedure combined with the IF specification (.if file) and the test purposes (.tp file). It is up to the user to choose the exploration strategy of the generated partial graph he wants to perform during the test generation: in depth (DFS) or in breath (BFS) [6]. During this generation, when a test purpose is satisfied, a message is displayed to inform the user. The number of test purposes already found and the number of those missing are also provided. Based on this approach, a test case is generated (represented by the Test Case box). A test suite is composed of a finite set of test cases (or scenarios) described in a standard format. It is used to stimulate the implementation under test (IUT) to validate its reaction.

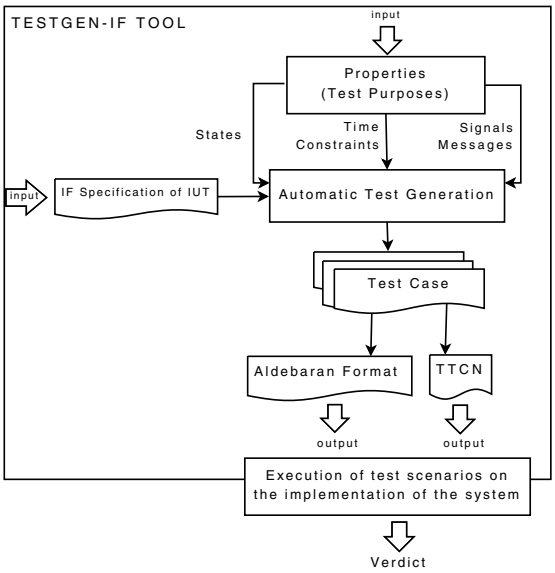

Figure 4. Basic architecture of the TestGen-IF tool.

3) TestGen-IF Implementation: The *TestGen-IF* tool is based on the IF-2.0 simulator [3] that allows to construct the accessibility graph from an IF specification. This simulator is developed by a research team at Verimag [25], for modeling and simulating asynchronous timed systems such as telecommunication protocols or distributed applications. *TestGen-IF* uses the IF-2.0 simulator libraries which provide some functionalities for on-the-fly state-space traversal. It is implemented in the same implementation language as the IF-2.0 simulator, i.e. C++ language.

As output of *TestGen-IF* tool, two files can be generated:

- The "*output.stat*" file containing statistics about the test generation process (number of jumps, visited states,

generation duration, test case length, depth limit value, strategy exploration, etc.);
- And the "*output.sequence*" file (in Aldebaran [9] or TTCN [11] format) containing the timed test sequence. This last output is represented in the Figure 4 by the Aldebaran format and TTCN boxes.

The test cases are generated from the output file "*output. sequence*" by filtering the generated test sequences according to the input actions, output actions and delays (i.e., progress of time) of the system under test. The test generation with *TestGen-IF* derives its benefits from *Hit-or-Jump* characteristics. It is faster than classical test generation tools and less memory consuming. In addition, it avoids the state explosion and deadlock problems.

B. Fixing the Test Objectives

1) Test Purposes Formulation: In order to formulate timed test purposes using TestGen-IF tool, several options are permitted:

- State constraint purposes: expressing that a system can be in a specific state;
- Action constraint purposes: corresponding, in particular, to signals actions (e.g., input signal, output signal) and describe that an action can be executed in a state (optionally) at a specific time;
- Clock constraint purposes: expressing that a clock can have a specific value, optionally in a specific state.

For instance, the constraint "*action* $=$ *input* sg *in state* $= s$ *when clock* $c = d$" describes that the signal sg is to be received in the state s when the clock $c = d$. Timeouts and deadlines can be usually described by clock constraint, whereas the flow requirements can be described using states and actions constraints.

2) Test Purposes for the Travel Application: The automatic test generation only targets security issues and, as a result, it is less time consuming. In this work we defined a set of test purposes describing security properties. In the following, we provide both informal and formal test purposes (according to the TestGen-IF formulation) relating to the rules described in section V-B.

- Rule 1: A potential traveler wants to request for two missions. He/She is obliged to perform the two requests within a delay greater than 2 minutes. The timed test purposes of the rule 1 are formulated as:

$TP_1 = \{tp_1, tp_2\}$
tp_1 = *{signal= output "req_create_mission"}*
tp_2 = *{signal= output "req_create_mission" when clock ck1 = 2}*

- Rule 2: A potential traveler tries to choose his/her mission parameters (date, flight, hour, etc.) among a set of propositions provided by the Travel Web application. In the case he/she requests for more propositions, the system allows him/her to request a new list within a delay of 10 minutes. Otherwise, his/her session will expire. The timed test purposes of the rule 2 are formulated as:

$TP_2 = \{tp_1, tp_2, tp_3, tp_4\}$
tp_1 = *{signal= output "req_create_mission"}*
tp_2 = *{signal= output "req_proposition_list"}*
tp_3 = *{signal= input "recv_proposition_list"}*
tp_4 = *{signal= informal "other_proposition_request"}*

- Rule 3: Once a mission is created, it has to be validated by a specific user called validator. The system sends the mission parameters to the validator and waits for his/her acceptance/rejection. If the validator does not send any notification within a delay of 7 days, the system generates a validation request reminder. The timed test purposes of the rule 3 are formulated as:

$TP_3 = \{tp_1, tp_2, tp_3, tp_4\}$
tp_1 = *{signal= output "grant_create_mission"}*
tp_2 = *{signal= input "req_proposition_list"}*
tp_3 = *{signal= output "req_validation"}*
tp_4 = *{signal= output "req_validation" in state relaunch}*

C. Test Generation with TestGen-IF

Our objective is to automatically generate test sequences according to our test purposes. To reach this aim we used, for Travel test generation, two users (one being the validator) and two missions. We also defined adequate interval values for data variables in order to reduce the accessibility graph size and to avoid state explosion problems.

A set of timed test cases are generated based on the IF specification of Travel Web application and the timed test purposes for each rule, using TestGen-IF. These test cases are then filtered according to the observable actions (input, output and delays) relating to the system under test. For instance, the filtered timed test case for the rule 3 is as follows:

TEST CASE FOR THE RULE 3

1) ?give_traveler_id{0} / !req_connect{0,0}
2) ?req_connect{0,0} / !grant_connect{0,0}
3) ?grant_connect{0,0}
4) !req_create_mission{0}
5) ?req_create_mission{0}
6) !grant_create_mission{{0,0,0,{traveler_mission}0,{travel_mission}0}}
7) ?grant_create_mission{{0,0,0,{traveler_mission}0,{travel_mission}0}}
8) !req_proposition_list{0,{0,0,0,{traveler_mission}0,{travel_mission}0}}
9) ?req_proposition_list{0,{0,0,0,{traveler_mission}0,{travel_mission}0}}
10) ?give_choice_list{{{0},1}} / !recv_proposition_list{0,{0,0,0,{traveler_mission}0,{travel_mission}0},{0},1}
11) ?recv_proposition_list{0,{0,0,0,{traveler_mission}0,{travel_mission}0},{0},1}
12) !req_choice{0,{0,0,0,{traveler_mission}0,{travel_mission}0},0}
13) ?req_choice{0,{0,0,0,{traveler_mission}0,{travel_mission}0},0}
14) !grant_choice{{0,0,0,{traveler_mission}0,{travel_mission}0},0} !req_validation{{0,0,0,{traveler_mission}0,{travel_mission}0},0}
15) ?grant_choice{{0,0,0,{traveler_mission}0,{travel_mission}0},0}
16) delay = 10080
17) !req_validation{{0,0,0,{traveler_mission}0,{travel_mission}0},0}

Notice that the input/output signals described in each test case are relative to the system under test. In our case it is the Travel system designed by the two processes *basic_travel* and *travel_mission*. The test cases generated by TestGen-IF tool are abstract but usable; they are produced in aldebaran standard notation facilitating their portability and their automatic execution. Some metrics about this test generation relating to three rules are presented in Table III

Rule	Strategy	Maxdepth	Jumps	Test Case Length	Visited States	Duration
1	BFS	10	0	9	291	0.2s
2	BFS	10	1	16	7844	10s
3	BFS	10	2	23	26552	1m25s

Table III
SOME TEST GENERATION METRICS

VII. TEST CASES INSTANTIATION AND EXECUTION

In order to execute the generated test cases to a real Web application, they need to be transformed into an executable script capable of communicating via http (or https) with the implementation under test. In this paper, we use the tclwebtest tool to describe the executable test cases and apply them on Travel Web application.

A. Tclwebtest tool

Tclwebtest [23] is a framework to build tests for Web applications. It provides an API for issuing http requests and processing results. It assumes specific response values, while taking care of the details such as redirects and cookies. It has the basic HTML parsing functionality to provide access to elements of the resulting HTML page that are needed for testing, mainly links and forms.

The execution of a test case written in *tclwebtest* simulates a user that is interacting with the Web application through a Web browser. By executing the instantiated test cases, it is possible to add, edit or delete data of the Web application, fill some forms or follow a specific link. Figure 5 illustrates the *tclwebtest* code for logging into the Travel application by requesting the register page, then filling the e-mail and password, and submitting this information.

```
::twt::do_request "http://tavel-example.org/register"
tclwebtest::form find ~n login
tclwebtest::field find ~n username
tclwebtest::field fill "user@mymail.com"
tclwebtest::field find ~n password
tclwebtest::field fill "mypassword"
tclwebtest::form submit
```

Figure 5. Example of Tclwebtest Code.

B. Test Cases Instantiation

The test cases generated by TestGen-IF tool are provided in aldebaran format and are a set of:

- Delays: a delay represents an amount of time that the tester has to wait for, before performing any input action.
- Input signals: the tester has to stimulate the Web application by applying a set of http(s) requests called inputs.
- Output signals: the tester has to access to the Web system answer to analyze it and check if it conforms to the expected reaction as described in the formal specification of the system.

1) Delay Instantiation: A delay in the test case can be easily translated in TCL script language. It is transformed directly into the code '*after* n' where n is in millisencond (ms). For instance, '*delay 10; !req_validation(m)*' is translated into '*after 10*1000*60 req_validation_script*' if we consider a delay of ten minutes. (See Algorithm 1, lines 3 to 5)

2) Input Instantiation: To automatically instantiate the abstract test cases provided by TestGen-IF tool, it is mandatory to know the types of HTML elements that correspond to input signals of the Web system under test (that correspond to the output signals of the tester). In this work, the Web system will be limited to receive just three types of inputs from a user via a regular browser: (i) a URL set in the address bar, (ii) a link in the body of the page or (iii) the submission of a form in the body of the page. Actions such as drag-and-drop and other Ajax functionalities are not considered in this paper.

The first step of our methodology consists in mapping the signals into the three types of inputs that the Web application can receive. It is important to highlight that some IF signals can be mapped just to one single interaction with the Web application, e.g. following a link, but other signals are mapped to a set of interactions, e.g. submitting a form.

For example, considering the form submission, there are several interactions that must be performed by tclwebtest, i.e. filling text fields, selecting radio buttons, selecting checkboxes and finally submitting the form. In these cases the signal is mapped to an HTML element (e.g. a form) but also to each signal parameter (e.g. text fields).

To perform the mapping, we propose two tables containing required information of the input signals and their parameters to transform them into tclwebtest script asking to follow a link, submit a form or request a new URL. Both, the *signal_info_table* and the *parameter_info_table* are illustrated in Tables IV and V. To access the information they contain, we can take advantage of standard SQL queries. For each signal, the table *signal_info_table* stores the following data:

- *signal:* the name of the input signal in the IF specification,
- *html_element:* the HTML type of the element that correspond to the signal,
- *html_name:* the name or id of the HTML element. For example, in the case of a link, the name is the link caption.

signal	html_element	html_name
req_connected	form	login
req_disconnect	link	logout

Table IV
SIGNAL_INFO_TABLE EXAMPLE

Then, for each parameter of an input signal, the information stored in this table V is:

- *parameter:* name of the parameter signal in the IF specification,
- *of_signal:* name of the input signal that uses this variable,
- *html_element:* the HTML type of the element that correspond to the parameter,
- *type:* the type of the variable expected by the html_element, e.g. integer, string, etc.

- *html_name:* the name or id of the html_element.

parameter	of_signal	html_element	type	html_name
user	req_connected	textfield	integer	username
password	req_connected	texfield	integer	pass

Table V
PARAMETER_INFO_TABLE EXAMPLE

The second step of the methodology is to translate the system inputs to tclwebtest script for each test case. These inputs are built dynamically and can be divided into three categories: following a link, submitting a form or setting a URL in the browser. The inputs translation methodology is presented in pseudo-code in Algorithm 1 (lines 6 to 36). By performing this algorithm the following parts of the test case will be built:

- The script of the test preamble: a sequence of inputs (operations) that will lead the system to a state where the test case can be executed. During this preamble, system outputs are not analyzed. For example, to test the creation of a mission, the user needs to be authenticated by the Travel system.
- The script that will stimulate the system to test it.

3) System Output Instantiation: The last step of the methodology consists in developing the scripts that analyzes the response (or reaction) of the Web application (Algorithm 1, lines 37 to 49). This script also assigns the verdict (pass or fail). Basically it checks whether the platform did what it was supposed to do. The system outputs (or reaction) can be classified into two categories:

- Observable outputs: Tclwebtest is basically dedicated for testing Web application interfaces accessible by Web user. It offers some basic HTML parsing functionalities and commands for the manipulation of the HTML elements of Web pages (in our case, on consider system output pages). The reaction of the system can be provided in one or many HTML pages of the Web-based system. In general, it is a notification message that stipulates that the desired action succeeded or failed, for instance an authentication. Sometimes, this reaction can be more difficult to discover such as when reloading the current page or navigating to another Web application page. In all cases, we need to define the two tables IV and V for the output as well as their parameters and follow the same methodology developed in the input instantiation case. To analyze system Web pages, we use *response* and *find* commands to locate the HTML elements we want to test. Then we rely on *assert* command to compare the displayed Web page values and the output signal parameters and deduce the adequate verdict.
- Non observable outputs: the system may react to a user operation by performing an action which is non-observable from this user's point of view (and as a result of the tester). For example, we can consider the adding/edition/deleting of information in a specific data base or the sending a notification email to a specific user. In these cases, no automatic solution has yet been elaborated.

In the Travel case study, all our test cases consider observable system reactions that can be defined automatically.

Algorithm 1 Instantiation Methodology

Require: An abstract test case TC, signal_info_table and parameter_info_table tables. Let act be a delay or an observable action in TC and let $sg(d_1, d_2, ..., d_k)$ be a signal instance of $sg(x_1, x_2, ..., x_k)$. d_j is denoted in_j if sg is an input signal and out_j if sg is an output signal. $(0 < j < k + 1)$

1: **for each** $(act_i \in TC)$ **do**
2: /*(where $i \in N$, $0 < i < n + 1$ such that n is the number of actions and delays in TC)*/
3: **case** (act_i = delay n) **do**
4: *tcl_script:* after n;
5: **end case**
6: **case** (act_i = input $sg_i(in_1, in_2, , in_k)$) **do**
7: **if** (html_element(sg_i)= url) **then**
8: *tcl_script:* do request url;
9: **end if**
10: **if** (html_element(sg_i)= link) **then**
11: *tcl_script:* follow link;
12: **end if**
13: **if** (html_element(sg_i)= form) **then**
14: *tcl_script:* form find ~n html_name(sg_i);
15: **for** (**each** parameter x_j of sg_i) **do**
16: /*(where $j \in N$, $0 < j < k + 1$)*/
17: *tcl_script:* field find ~n html_name(x_i);
18: **case** (html_element(x_j) = textfield) **do**
19: *tcl_script:* field fill in_j;
20: **end case**
21: **case** (html_element(x_j) = textarea) **do**
22: *tcl_script:* field fill in_j;
23: **end case**
24: **case** (html_element(x_j) = checkbox) **do**
25: **if** (in_j = 1) **then**
26: *tcl_script:* field check html_name(x_j);
27: **else**
28: *tcl_script:* field uncheck html_name(x_j);
29: **end if**
30: **end case**
31: **case** (html_element(x_j) = radiobutton) **do**
32: *tcl_script:* field select in_j;
33: **end case**
34: **end for**
35: *tcl_script:* submit form;
36: **end if**
37: **end case**
38: **case** (act_i = output $sg_i(out_1, out_2, , out_k)$) **do**
39: **if** (html_element(sg_i)= link) **then**
40: *tcl_script:* assert {[response url] == html_name(sg_i)};
41: **end if**
42: **if** (html_element(sg_i)= form) **then**
43: *tcl_script:* response body;
44: *tcl_script:* form find ~n html_name(sg_i);
45: **for each** (parameter x_j of sg_i) **do**
46: *tcl_script:* assert {[field get_value find ~n html_name (x_j)]== out_j} ;
47: **end for**
48: **end if**
49: call deduce_verdict procedure;
50: **end case**
51: **end for**

C. Test Cases Execution

The test cases execution was performed on a prototype implementation of the Travel Web application (developed

on OpenACS platform) to verify that the specified security requirements are respected. It is important to highlight that some time settings in this prototype have been changed so that the application of the tests where faster than in the real system. For example, we changed 10080 minutes (7 days) in the third rule to 3 minutes to avoid waiting so long. Therefore in this case study we verify the behavior of the system concerning this rule using a delay of 3 minutes rather than using 7 days.

The execution of the test cases is performed using a dedicated testing tool proposed by the OpenACS community [20]. This tool is called the ACS-Automated-Testing tool that allows executing the instantiated test cases, interacting with the Web-based application under test and, also, displaying the verdict of each test case. The ACS-Automated-Testing tool is, in itself, a Web application but we will refer to it just as *the tester* to avoid confusions between this system and the Web application to be tested.

As a result of the execution of the designed test cases on the prototype, we obtained positive verdicts for thirty test objectives, while, four test objectives were violated (fail verdict). For example, a problem has been detected according to the system respect to the first rule that expresses a prohibition. If a potential traveler requests for a first mission and then waits for 2 minutes, he/she is not allowed by the system to request for another mission. We analyzed the implementation of the Web-based system and noticed that a mistake was encrusted in the code. Instead of 2 minutes, the Travel system waited much longer before allowing a second mission request.

The Travel application was analyzed to detect the four error sources. Once the mistakes corrected, all the test cases were applied again on the Web application. This time, all the verdicts were positive which demonstrates the efficiency of our methodology.

VIII. Conclusion and Future Work

In this paper, we presented a framework for the modeling and the testing of Web applications from their security point of view. Our approach consists in automatically integrating security rules described in using the Nomad formal language within an IF specification. This integration leads to an IF secure specification that takes the system security requirements into account. Afterward, we presented an approach to derive test cases from this IF secure specification using TestGen-IF tool developed in our laboratory and to transform them into executable test cases (using TCL script language). We applied the generated test cases to an industrial Web-based system provided by France Telecom to study its respects to its security policy. Relying on our end-to-end framework, we discovered several security flaws that we was able to correct obtaining thus a secure Web system.

As future work, we want to extend the test purposes formulation by adding data constraints and complex clock constraints to express more elaborated test objectives. We also intend to adapt test generation algorithms to include these new test purposes types. In addition, we want to investigate the automatic analysis of non-observable system reactions in the context a white box testing [24].

References

[1] R. Alur and D. L. Dill. A Theory of Timed Automata. *Theoretical Computer Science*, 126(2):183–235, 1994.

[2] M. Bozga, S. Graf, and L. Mounier. IF-2.0: A Validation Environment for Component-Based Real-Time Systems. In *CAV*, pages 343–348, 2002.

[3] M. Bozga, S. Graf, I. Ober, I. Ober, and J. Sifakis. The IF Toolset. In M. Bernardo and F. Corradini, editors, *SFM*, volume 3185 of *Lecture Notes in Computer Science*, pages 237–267. Springer, 2004.

[4] 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.

[5] A. Cavalli, S. Maag, and G. Morales. Regression and Performance Testing of an E-learning Web Application: dotLRN. In *3rd ACM/IEEE International Conference on Signal-Image Technology and Internet-Based Systems (SITIS)*, Shangai, China, 2007.

[6] A. R. Cavalli, S. Maag, W. Mallouli, M. Marche, and Y.-M. Quemener. Application of Two Test Generation Tools to an Industrial Case Study. In *TestCom*, volume 3964 of *Lecture Notes in Computer Science*, pages 134–148. Springer, 2006.

[7] P. Chevalley and P. Thévenod-Fosse. Automated Generation of Statistical Test Cases from UML State Diagrams. In *COMPSAC*, pages 205–214, 2001.

[8] F. Cuppens, N. Cuppens-Boulahia, and T. Sans. Nomad: A Security Model with Non Atomic Actions and Deadlines. In *CSFW*, pages 186–196, 2005.

[9] J.-C. Fernandez, H. Garavel, A. Kerbat, L. M. R. Mateescu, and M. Sighireanu. CADP : A Protocol Validation and Verification Toolbox. In *The 8th Conference on Computer-Aided Verification, CAV'96*, New Jersey, USA, August 1996. Springer Verlag.

[10] E. Gaudin, E. Najm, and R. Reed, editors. *SDL 2007: Design for Dependable Systems, 13th International SDL Forum, Paris, France, September 18-21, 2007, Proceedings*, volume 4745 of *Lecture Notes in Computer Science*. Springer, 2007.

[11] J. Grabowski, D. Hogrefe, G. Rethy, I. Schieferdecker, A. Wiles, and C. Willcock. An Introduction to The Testing and Test Control Notation (TTCN-3). In *Computer Networks 42(3)*, pages 375–403, 2003.

[12] C. Jard and T. Jéron. TGV: Theory, Principles and Algorithms. *STTT*, 7(4):297–315, 2005.

[13] A. A. E. Kalam, S. Benferhat, A. Miège, R. E. Baida, F. Cuppens, C. Saurel, P. Balbiani, Y. Deswarte, and G. Trouessin. Organization Based Access Control. In *POLICY*, pages 120–135. IEEE Computer Society, 2003.

[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] W. Mallouli, A. Mammar, and A. R. Cavalli. Integration of Timed Security Policies within a TEFSM Specification. Technical Report TI-PU-08-868, Telecom SudParis, 2008.

[16] W. Mallouli, A. Mammar, and A. R. Cavalli. Modeling System Security Rules with Time Constraints Using Timed Extended Finite State Machines. In *the 12-th IEEE DS-RT proceedings*, Vancouver, British Columbia, Canada, October 27-29, 2008.

[17] W. Mallouli, J.-M. Orset, A. Cavalli, N. Cuppens, and F. Cuppens. A Formal Approach for Testing Security Rules. In *SACMAT*, Nice, France, 2007.

[18] M. G. Merayo, M. Núñez, and I. Rodríguez. Generation of Optimal Finite Test Suites for Timed Systems. In *TASE*, pages 149–158. IEEE Computer Society, 2007.

[19] M. G. Merayo, M. Núñez, and I. Rodríguez. Formal Testing from Timed Finite State Machines. *Computer Networks*, 52(2):432–460, 2008.

[20] OpenACS Community. http://www.openacs.org/.

[21] J. A. Syriani and N. Mansour. Modeling Web Systems Using SDL. In A. Yazici and C. Sener, editors, *ISCIS*, volume 2869 of *Lecture Notes in Computer Science*, pages 1019–1026. Springer, 2003.

[22] TCL Script Language . http://www.tcl.tk/.

[23] TCLWEBTEST Tool. http://tclwebtest.sourceforge.net/.

[24] J. Tuya, J. J. Dolado, M. J. S. Cabal, and C. de la Riva. A controlled experiment on white-box database testing. *ACM SIGSOFT Software Engineering Notes*, 11(1), 2008.

[25] Verimag Lab. http://www-verimag.imag.fr/ ˜ async/if/.

[26] E. R. Vieira and A. Cavalli. Toward Test Suite Automatic Generation with Delayable Transitions and Timing-Fault Detection. In *RTCSA*, 2007.