A framework for testing and monitoring security policies: Application to an electronic voting system

Khalifa Toumi[1], Mohamed Aouadi[1], Ana R. Cavalli[1], Wissam Mallouli[2], Jordi Puiggal Allepuz[3] and Pol Valletb Montfort[3]

[1]Institut Mines-Telecom/ Telecom SudParis, EVRY - France
[2]Montimage, 39 rue Bobillot, Paris - France
[3]Scytl Secure Electronic Voting, S.A - Spain
Email: khalifa.toumi, mohamed.aouadi, ana.cavalli@telecom-sudparis.eu
wissam.mallouli@montimage.com
jordi.puiggali, pol.valletbo@scytl.com

Testing and monitoring the effectiveness of security policies under pervasive system architectures is still a major challenging problem for the research community as well as industrials. The inherent characteristics of these systems such us the heterogeneous communicating devices and the multiple used technologies make the burden more overwhelming when dealing with security measures and policies. This paper aims to bridge this gap through the introduction of a formal design of security policies to make security monitoring operation more efficient. Hence, a formal framework is proposed to actively test web based systems, as an example of these pervasive architectures. The goal of our technique is to check the compliance of the targeted web application to a set of generic security requirements such as confidentiality, integrity and availability as well as to a set of user-related security constraints. Our approach has been applied to a real industrial electronic voting application provided by the Scytl company. Several experiments show the merit of our technique in verifying the correctness of security measures of the targeted application. This framework is part of the INTER-TRUST[a] solution intended to ensure secure inter-operation between communicating systems and provide solutions to test and monitor them.

[a]INTER-TRUST is a project co-funded by the EC under the ICT theme of the 7th Framework Program for ICT-2011.1.4 Trustworthy ICT contract n. 317731. More details are given in www.inter-trust.eu.

Keywords: Formal testing; Security monitoring ; Validation ; Security policy

1. INTRODUCTION

Despite the advances made in the design and evolution of computer pervasive systems and services, new requirements imposed by emerging communication technologies necessitate more flexibility and interoperability when designing such systems. In fact, pervasive systems are usually composed of heterogeneous communication devices which have to exchange information and communicate in a safe mode. Thus, efficient interoperability is more and more needed in this type of services.

Nowadays, security policies are the key point of every modern infrastructure. The specification and the testing of such policies are the fundamental steps in the development of a secure system since any error in a set of rules is likely to harm the global security [?].To ensure that a certain level of security is always maintained, the system behavior must be restrained by a security policy. To ensure that a certain level of security is always maintained, the system behavior must be restrained by a security policy.

This latter can be described as a set of rules that regulates the nature and the context of actions that can be performed within a system, according to specific roles. As an example, such policy can tackle the interactions between a network infrastructure and Internet or manage accounts and rights toward an operating system or a database. Generally, a security policy is written by the mean of a natural language specification, containing statements such as this file must be accessible only to authorized users or all ports are closed except for 21 (ftp), 22 (ssh) and 80 (www).

Nevertheless, while security policies are useful in improving confidence in software, they cannot by themselves eliminate the possibility of attacks and malicious behaviors that violate the security policies. Such issues are induced by a variety of factors largely outside the control of the organization producing the software: unpredictable loads, variable resources, and malicious attempts to break a system [1]. Consequently, systems must have an active role in detecting potential problems using other validation techniques as testing and monitoring in order to insure that proper corrective measures are taken as quick as possible.

The challenge to design and test the security policies become more overwhelming when the studied use case is an adaptive system that is able to respond to environmental changes. This adaptability will make it harder to crack the system as compared to fixed never-changing security mechanisms. To address these new constraints, security has to be designed in an autonomous and spontaneous way. This requirement for autonomy and spontaneity must be addressed in all the steps of modeling and deployment of security requirements and especially for the testing and monitoring part. These mechanisms, if they are implemented, could sharply influence and make the testing and monitoring more complex for a manual testing.

All these problems motivate us to design a generic framework for the testing and monitoring of security policies. In our work, security testing and monitoring aim at:(1) to check the right implementation of the deployed security policies; (2) to simulate attacks in order to test the robustness of the system and (3) to detect and prevent anomalies.

In the literature, we find several security testing and monitoring tools [3, 4, 5, 6, 7] that could be used to achieve the same security testing and monitoring objectives. However, these solutions are mainly designed to be used by testing experts. Moreover, their processes are still more complex with an adaptive system since they may use dynamic security policies that are adapted to different usage contexts (device type, organization hosting the vote, etc.). The specification of test objectives and the generation of test scenarios to validate the security policies and to monitor the system behavior can be seen as a hard task in this dynamic context. In the counterpart, our framework aims at: (1) simplify and automate this task by the proposition of a list of test objectives based on the security policies; (2) automatically generate and execute test scenarios and (3) integrate a monitoring module in a testing tool. Consequently, the original contributions of this paper are as follows:

- A modeling system that integrates contextual security constraints. This model integrates the security policy to the system functional model. It allows also to provide a security property file that is a specific input to be used by both the test generator and the monitoring module.
- A test generation and execution component: it aims to generate a set of test scenarios in order to check the conformity of the implementation with the deployed security policy. Moreover, this module is responsible of the automatic execution of the generated scenario.
- A monitoring component: it allows to correlate notified events in order to detect security vulnerabilities as well as the non-compliance of security requirements during the testing and operation phases. It also generates warnings or alarms that will provoke the reaction countermeasures by enforcing secure interoperability and weaving contextual obligations during the operation phase.

In order to validate our approach, we decided to apply it with an e-voting system that provides us a secure and adaptive distributed system [8, 9]. E-voting systems could be presented as environments in which security requirements are complex and interacting with other requirements such as usability and verifiability. These requirements must be fulfilled despite the use of a secure voting channel. Moreover, e-voting system provides us with a very adaptive system. For example, in an e-voting environment, it is very important to guarantee that all the votes in the digital ballot box belong to an eligible voter being necessary at the same time to preserve the privacy of the voter (even in the counting process). If the voter uses a personal computer for casting a vote, this voter can be strongly authenticated using a personal digital certificate, such as an electronic ID card. However, personal digital certificates are not usually available in mobile phone platforms and therefore, an alternative authentication mechanism must be provided in this environment (e.g. SIM authentication or voice biometrics). This means that the security policies must be adapted depending on the device used by the voter and the authentication method available.

The remainder of the paper is organized as follows. Section 2 presents the related works. Then, the e-voting application provided in the INTER-TRUST project [10] is presented in Section 3. In Section 4, we present a detailed description of the proposed framework with the different modules and algorithms. Section 5 details the experimentation procedure and results of the approach. Finally, Section 6 concludes this paper.

2. RELATED WORK

In this section, we discuss the relationship between our approach and other related works on testing and monitoring techniques.

Most work related to security policy can be mainly divided into two parts: the description of the policy itself and the testing and verification of the rules. Several studies have investigated the security policy design [11, 12, 13]. Each system has to define a security policy that specifies the desired behavior of the system when a user is using available resources. In the literature, there are four basic security access models [13] which are Mandatory Access Control (MAC), Discretionary Access Control (DAC), Role Based Access Control (RBAC) [11] and Organization Based Access Control (OrBAC) [12]. Many derivatives have been deduced from these models in order to resolve a specific need. Lately, solutions as RBAC and OrBAC have gain a lot of interest. This is due to the fact that these two models can manage easily larger number of users than the classical models (MAC and DAC). They have also various advantages such as their high description level of rules and their administration models.

RBAC has introduced a number of new concepts regarding the existing classical models (MAC and DAC) such as role, hierarchy and separation of duties. This model offers a higher level of abstraction than other classical models. RBAC uses also the concept of hierarchy which is a relation between two roles. It offers the possibility of rules derivation. A role A inherits from role B means that permissions given to the role B are automatically assigned to the role A. RBAC proposes two types of duties separation concept (Static Separation of Duties and Dynamic Separation of Duties) that deal with the problem of the assignment of two opposite roles to the same user. Our approach is based on the OrBAC security policy model, because OrBAC has many advantages compared to RBAC. Firstly, it has a higher level of abstraction thanks to the use of abstract entities. It can also describe dynamic security policies by relying on the context concept. OrBAC proposes thus a formal language to express contextual security policies including access control and usage policies. This language is based on a first order logic that allows expressing various types of security policies such as permission, prohibition and obligation in different types of contexts. To summarize, we can define OrBAC as an improvement of RBAC regarding to the dynamicity, abstraction, expressiveness of security policies design as well as the management mechanisms.

Concerning the verification of the security rules, several researches tended to concentrate their efforts on the verification of security rules in order to detect errors, attacks or misconfigurations, such as redundancy, contradictions or collisions [4, 5, 6, 14, 15, 16].

In [5] authors propose a testing strategy for automatically deriving test requests from a security policy and describe their pilot experience in test automation using this strategy. In [6] the authors give an overview of existing security testing approaches and, based on that, develop a novel classification for model-based security testing along two criteria: risk and automation of test generation. [7] provides a solution for testing XACML described security policies. The integration is also done automatically. Other researchers [14] propose to design a virtual testbed to facilitate intrusion detection for industrial control systems. In [16], authors propose a multi-phase testing approach that allows to assess the security of service platforms against DoS attacks. However, these solutions could only be used before the deployment of the system. Moreover, they do not propose any monitoring technique that may be used with the testing techniques.

Other approaches propose to focus on validation by checking the conformance of a system with respect to a security policy. In [15], authors provide a solution to verify the security policies modeled with the RBAC model in cloud systems. They have firstly designed a set of properties related to this environment and used a model checking technique to verify the correctness of the deployed policies. In [3], authors show how a network organization 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 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 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.

Based on the mentioned limitations of the approaches described above, our proposal stands as an efficient alternative since it offers the following features. Firstly, we check the correct implementation of an OrBAC based security policy which allow us to improve the correctness of this model. Moreover, we propose an end-to-end solution that integrates the security policy into a formal model, generates the test cases related to this policy, executes them and monitors their implementation. Our solution may be used during the development of the application to detect bugs and/or during its deployment to detect attacks by relying on the monitoring part of the proposed solution. In addition, our approach has addressed the critical issues related to the use of dynamic policies. Without this global framework, the classical testing process of an adaptive system should be hard and time consuming.

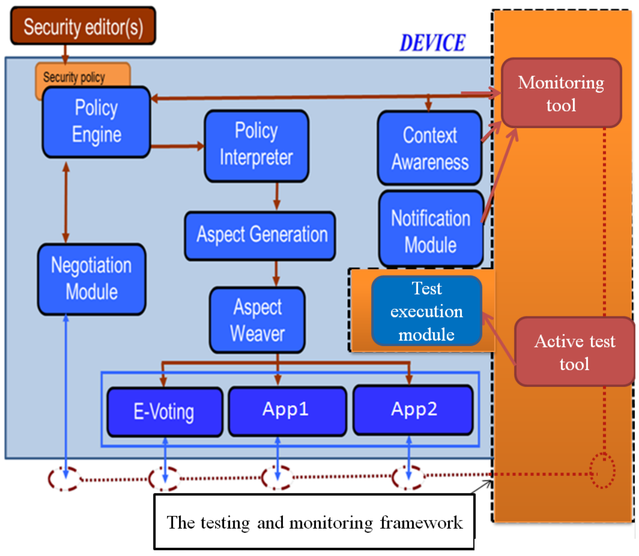


FIGURE 1. INTER-TRUST architecture

3. E-VOTING IN THE INTER-TRUST PROJECT

The main objective of INTER-TRUST project was to develop a dynamic and scalable framework to support security mechanisms to insure secure inter-operability between different organizations. One of the most important features of this framework is the dynamic characteristic of the security rules. The rules can dynamically adapt when changes occur. Moreover, the applications can delegate the security management to the INTER-TRUST management entity.

The INTER-TRUST solution facilitates the task of controlling and monitoring dynamic environments in which changes can occur, and can be applied on the fly. Section 3.1 describes the software architecture of our solution.

3.1. INTER-TRUST architecture

The INTER-TRUST framework is built using a modular scheme based on Aspect Oriented Programing. This new programming paradigm allows the developer to implement a really flexible application using the concept of aspects. Different aspects have been created and they will be deployed depending on the user interaction and/or the current context of the application, among other variables. Figure 1 describes the different modules of our solution. The main modules are the following:

- **Policy engine**: This module is responsible for taking decisions about the access control and for controlling the updating of security policies that we defined for the application.
- **Negotiation Module**: The negotiation module defines a communication process between two entities in order to agree on an inter-operable security policy.
- **Policy Interpreter**: This module is responsible for analyzing and interpreting the concrete security policy that has been derived by the Policy Engine. This security policy is based on the abstract specifications defined by users and negotiated between entities. The provided security policy will be used by the testing and monitoring framework to generate the test scenarios.
- **Aspect generation**: The Aspect Generation module is responsible for generating a Security Adaptation Plan (SAP), i.e., it is responsible for determining the list of aspects that need to be woven/unwoven (enabled/disabled) or reconfigured in order to enforce the received security rules. This is performed on the output received from the Policy Interpreter.
- **Aspect Weaver**: The Aspect Weaver module executes the SAP generated by the Aspect Generation module and it is responsible for adding, deleting and reconfiguring the list of aspects contained in that SAP.
- **Notification Module**: This module reports on application internal events (e.g. state changes, error conditions, etc.) to the monitoring server (the Monitoring Tool) that filters and analyses them for security checking during both testing phase and normal operation (i.e. at runtime).
- **Context Awareness**: This module is in charge of measuring application context parameters that are relevant for security analysis (e.g. network, CPU or memory consumption). It notifies the monitoring server (MMT) and the Policy Engine module in order to transmit the context changes to the rest of modules.
- **Testing and monitoring framework**: This framework will be described in detail in next section. Some modules of this framework are in the orange box of Figure 1.

All these modules need some communication mechanisms in order to transmit information between them. The selected technology to perform this communication is the Advanced Message Queuing Protocol (AMQP) [17]. An AMQP broker is the responsible of managing all the queues that the modules will use in order to exchange information. More details regarding the modules functionality and communication can be found in [18].

3.2. Improvements on the e-voting platform

Nowadays, several e-voting platforms are able to perform an election guaranteeing voters' privacy, votes' integrity, and the most modern platforms are able to provide verifiability by the means of concepts such as cast as intended. All these security concepts are mandatory in a traditional election where, for instance, voters can vote via Internet without taking into account other variables as geographic position, device used to cast the votes, among many others. These modern e-voting schemes take advantage of advanced cryptographic mechanisms, such as zero knowledge proofs, in order to achieve robust protocols. However, these protocols require performing costly cryptographic operations, which might not be performed on mobile devices with low computing capacity. This is a big challenge since it could reduce the participation to people casting their votes via computers or high-end mobile devices.

In this context, our proposal stands as an efficient solution since it adds a dynamic managing of security requirements of the voting process. It might not be always necessary, or possible, to cast votes applying all the security concepts (privacy, integrity and verifiability). A voter that uses our solution is able to choose the security options regarding authentication, encryption and signature. We also provide a delegation protocol which allows voters, with a limited resources voting terminal, to vote delegating these operations to an external server.

This scenario where voters can select their security preferences is also dependent on the security requirements of the type of election where the voter wants to cast their votes. There are conditions regarding each election, which control the voters that can cast votes, depending on their security choices. These conditions are represented by **security policies** that will be created by the e-voting platform administrator using the MotOrBAC tool [13]. With these security policies, it is possible to decide whether a voter can vote or not. With the inclusion of a Negotiation Module flexibility is even more present. A voter which does not satisfy all the security policies for a certain election can be granted to vote, under a certain condition, if the Negotiation module reaches an agreement between the two entities. For instance, if an election can be only accessed by voters authenticated using digital credentials stored in a smartcard, but there is a voter that wants to authenticate himself using a login and a password, a negotiation process will start. This process may end with a resolution allowing the user to access to this election after answering some questions.

The addition of the INTER-TRUST framework over a traditional e-voting platform makes possible to extend the possibilities of the platform managing the security requirements of the electoral processes.

4. TESTING AND MONITORING FRAMEWORK

4.1. The security rules integration and implementation component

As it is shown in Figure 2, we have two inputs in our testing and monitoring framework: the initial model of the e-voting system and the security policy. Initial system refers to the functionality with no security consideration. The second input is a security policy. In the e-voting system, we may have several policies depending on the context, the used device, privacy options, etc. Therefore, the tester has only to choose one of the possible policies to validate the conformance of its implementation. Next, we give more details about the format of these two inputs:

The EFSM model: In order to model the initial e-voting system, we choose to use the Extended Finite State Machines (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: An Extended Finite State Machine M is a 6-tuple $M = < S, s_0, I, O, \overrightarrow{x}, Tr >$ 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), $\overrightarrow{x}$ 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 a predicate (a Boolean expression), and A is an ordered set (sequence) of actions.

We illustrate the notion of EFSM by a simple example described in Figure 3. The ESFM shown in figure 3 is composed of two states S_0 (vote fulfillment), S_1 (vote verification) 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 on the other hand, 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 receiving input B. If so, task T'' is performed and output Y is triggered.

4.2. The security policy

OrBAC (Organizational Based Access Control) [13] is an access control model that allows an organization to express its security policy. For this purpose, OrBAC defines two definition layers. The first one is called abstract and describes a rule as a role having the permission, prohibition or obligation to perform an activity on a view in a given context. A view is a set of objects to which the same security rules apply. The second level is the concrete one. It is derived from the abstract level and grants permission, prohibition or obligation to user to perform an action on an object. All rules in this paper are expressed with these few concepts. Thus, according to our syntax, three types of security rule may appear:

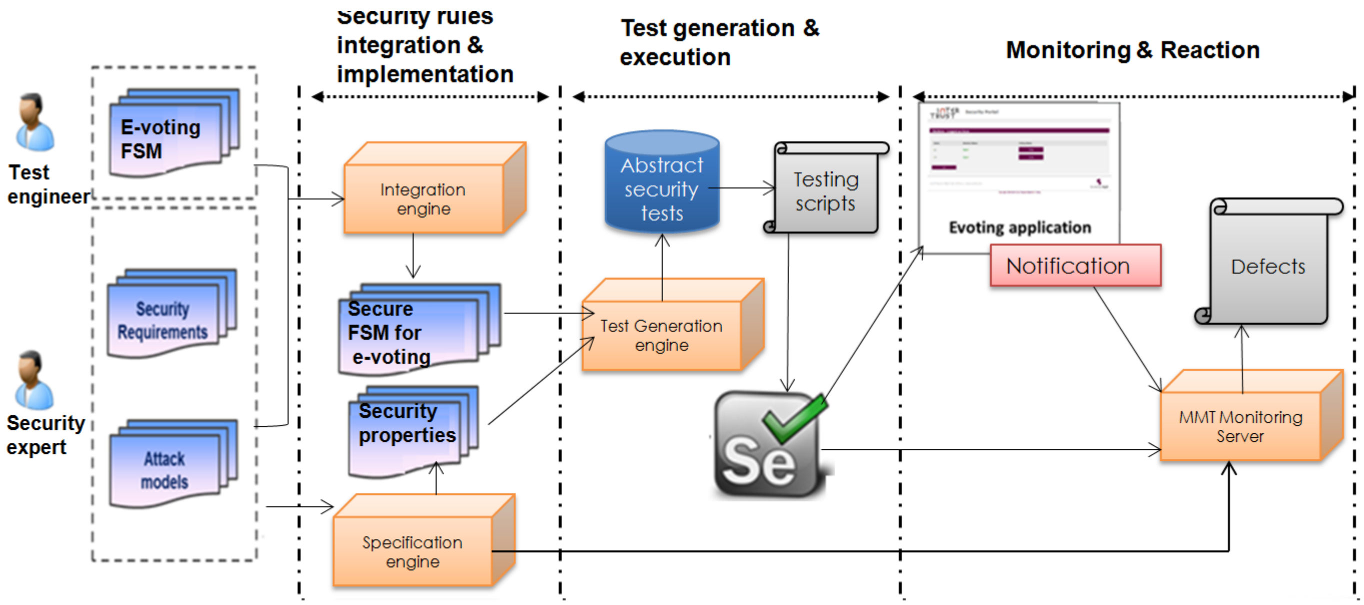


FIGURE 2. Our testing and monitoring framework for the e-voting systems

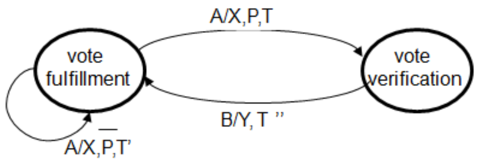


FIGURE 3. Example of a simple EFSM with two states.

$$SR\ (S,\ R,\ A,\ O,\ C\ )$$

a SR can be a permission, prohibition or an obligation

This rule means that within the system S, the role R is (allowed, prohibited or obliged) to perform the action A targeting the object O in the context C.

In this paper, we propose to automatically integrate the security policy rules into the initial specification in the form of an EFSM using our integration engine.

4.3. The integration engine

In this section, we describe the process to automatically integrate the security policy rules into the initial specification, which has the form of an EFSM, by the application of a specific methodology. The process is simple and twofold. At first, the algorithm seeks for the rules to be applied on each transition of the specification, and derives a simple automaton from this set of rules. Then, in the second time, it integrates the automaton within the initial specification. At the end of the process, this integration will generate a new specification that takes into account the security requirements.

4.3.1. Permission integration

The permission integration is the easiest modality to integrate. By definition, a permission does not define what is **possible** to do but instead, what is **permitted** to do. Thus, permissions relate to activities which already exist in the initial system. Considering a EFSM, a permission corresponds to one (or many) transitions. If the transition related to a permission 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 restrained (the condition is stronger).

Algorithm 1 Permissions integration

```
Require: The transition $Tr$ that maps the permissions. Each $permission_i$ is applied to a $role_i$ and
    probably to a "$variables\ context_i$".
1: if ($\exists$ associated predicate $P$) then
2:     $P := P \wedge (\vee_i(\text{"}variables\ context_i\text{"} \wedge role_i))$
3: else
4:     create predicate $P := \vee_i(\text{"}variables\ context_i\text{"} \wedge role_i)$
5: end if
```

An example of this rule is given in Figure 4. In the left transition, the system can pass from S_1 to S_2 if P is true, performing the task T. If the permission involves a role R, allowed to perform task T in the context C, the transition will be modified by strengthening the predicate, as in the left transition.

4.3.2. Prohibitions integration

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

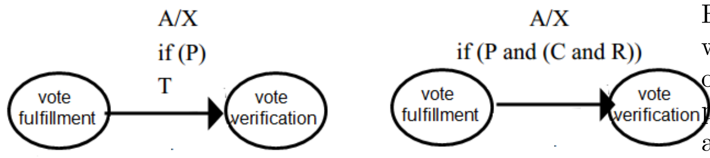


FIGURE 4. Permission (S, R, T, _ , C)

Algorithm 2 Prohibitions integration

```
Require: The transition $Tr$ that maps the prohibition(s) i.
1: if ($\exists$ associated predicate $P$) then
2:     $P := P \wedge_i (\neg$"variables context$_i$" $\vee \neg role_i)$
3: else
4:     create predicate $P := \wedge_i(\neg$"variables context$_i$" $\vee \neg role_i)$
5: end if
```

An example to illustrate this rule is given in Figure 5. In the left transition, the system can pass from S_1 to S_2 if P is true, performing the task T. If the prohibition involves a role R, prohibited to perform task T in the context C, the transition will be modified by the restriction of the predicate, as in the left transition.

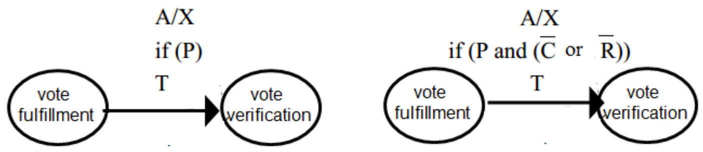


FIGURE 5. Prohibition (S, R, T, _ , C)

4.3.3. Obligation integration

In our model, we take as an assumption that an obligation implies the creation of a new activity (if an activity already exists in the initial specification, it only has to be allowed or denied). This new activity describes a new functional feature of the system. To make this possible, the new activity has to be initially expressed through a partial EFSM so that the algorithm can perform an automatic integration of the rule within the EFSM. In this manner, an obligatory activity is a partial EFSM which begins by a *starting obligation state* and ends with an *end obligation state*. Thanks to the "*EFSM context*" of the obligation, the algorithm identifies the transition which will be split into two, to insert the partial EFSM 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 is 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 (towards 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).

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 $EOS$ as a last one
1: for all (transitions from $OS$) do
2:     if ($\exists$ associated predicate $Q$) then
3:         $Q := Q \vee ($"variablescontext" $\wedge role)$
4:     end if
5: end for
6: determine the "cut point" $C_{ut}P_{oint}$
7: delete the transition tr
8: create transitions $C_1$, $C_2$ and $C_3$ such as
9: if ($C_{ut}P_{oint} == S_1$) then
10:    $C_1 :=< S_1, OS, -, -, -, - >$
       if ($\neg\ \exists\ EOS$ state in $M$) then $C_2 :=< EOS, S_2, A, X, P, t_1...t_n >$ end if
       $C_3 :=< OS, S_2, A, X, \neg$"variables context" $\vee \neg role, t_1...t_n >$
11: else
12:    if ($C_{ut}P_{oint} == A$) then
13:        $C_1 :=< S_1, OS, A, -, P, - >$
           if ($\neg\ \exists\ EOS$ state in $M$) then $C_2 :=< EOS, S_2, -, X, -, t_1...t_n >$ end if
           $C_3 :=< OS, S_2, -, X, \neg$"variables context" $\vee \neg role, t_1...t_n >$
14:    else
15:        if ($C_{ut}P_{oint} == t_i$ where $i \in \{1, ..., n\}$) then
16:            $C_1 :=< S_1, OS, A, -, P, t_1...t_i >$
               if ($\neg\ \exists\ EOS$ state in $M$) then $C_2 :=< EOS, S_2, -, X, -, t_{i+1}...t_n >$ end if
               $C_3 :=< OS, S_2, -, X, \neg$"variables context" $\vee \neg role, t_{i+1}...t_n >$
17:        else
18:            if ($C_{ut}P_{oint} == X$) then
19:                $C_1 :=< S_1, OS, A, X, P, t_1...t_n >$
                   if ($\neg\ \exists\ EOS$ state in $M$) then $C_2 :=< EOS, S_2, -, -, -, - >$ end if
                   $C_3 :=< OS, S_2, -, -, \neg$"variables context" $\vee \neg role, - >$
20:            end if
21:        end if
22:    end if
23: end if
24: minimize the resulting EFSM by deleting soundness transitions (without input nor output nor action)
```

Figure 6 shows an example of the process. In this case, the initial transition is "$A/X, if\ P, T$". The new activity is a 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 $C1$ is defined by the input A and the predicate P. The transition $C2$ is defined by the task T and the output X. Obligation integration is shown in the algorithm 3.

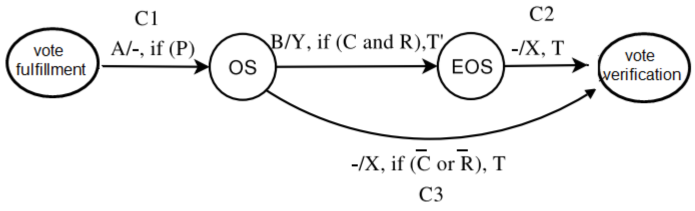


FIGURE 6. Obligation (S, R, new activity, _ , (Input = A) and C).

4.3.4. Complexity

One can notice that the integration of rules requires few transformations on the EFSM. At worst, a permission / prohibition only implies to modify one transition. In fact, the more costly operations are the obligations. As explained in the previous section, an obligation might lead to create new states and consequently, several transitions (two for one predicate depending if it is satisfied or not). Nevertheless, even in terms of transitions, the complexity of the algorithm remains linear (in $\mathcal{O}(n)$) since it is directly proportional to the number of rules in the policy.

Otherwise, as a border effect, the order of the rules might have a great impact on the time consumed by the integration process. The algorithm manages the rules one by one without analysing their relevance, since this role is devoted to the expert of the system.

4.4. Specification engine

This engine aims to translate each security rule to an understandable file for the test generation engine and the monitoring tool. The process is based on an embodiment algorithm that replaces each abstract term to a concrete one. This algorithm uses a mapping file that should be fulfilled by the e-voting developer. Based on our platform, a mapping interface is provided that requests the intervention of the developer to add some concrete information as the IP address of the server, a list of passwords and login that will be used for test, election names, etc. Later, this engine will be responsible for the organization of this information in order to construct an XML file. Each rule in this file is an IF-THEN property. It allows expressing specific constraints on network or applications events. Each event is a set of conditions on some of the field values of the exchanged packets.

4.5. The test generation and execution component

4.5.1. TestGen-IF tool presentation (the test generator)

The TestGen-IF tool was developed by a research team of Telecom SudParis [2, 20], for modeling and simulating asynchronous timed-systems such as distributed applications. It is based on active testing techniques, allowing automatic generation of test cases from a formal description of the studied system. The generation is made according to specific objectives called test purposes. The major advantage of this tool is that it avoids the exhaustive generation of the reachability graph of a system specified by building only partial graphs, allowing solving combinatorial explosion constraints. The automatic test generation module, written in C++ code, implements an automated test generation algorithm called Hit-or-Jump [21]. In particular it allows the automatic generation of test sequences for the test in context. This is very relevant mainly for the features integration in an operating platform. In addition to the generation of test sequences from test objectives, TestGen-IF also offers many other functionalities. It also allows:

- Automata minimization according to bi-simulation principles.
- Detection of sink nodes.
- Ability to execute abstract test cases.
- Very simple use (even for beginner) thanks to its user-friendly interface.

4.5.2. Execution engine

Nowadays, the importance of test automation in web engineering is evident considering the number of companies investing in automated testing frameworks and tools. This process is crucial for the success of large web applications and especially for the e-voting web applications. It reduces the testing time and enhances the application performances. In our platform, the execution engine is the responsible for the test instantiation and execution. This engine aims to transform the abstract test cases into executable scripts. It is based on the Selenium API that supports test automation. It implements the necessary scripts to run tests directly within a browser. This automation makes the testing process simple and transparent for the tester.

4.6. The monitoring component

For this component, we have chosen to rely on the MMT tool (Montimage Monitoring Tool). It is a monitoring solution that combines: data capture; filtering and storage; events extraction and statistics collection; and, traffic analysis and reporting. It provides network, application, flow and user level visibility. Through its real-time and historical views, MMT facilitates network

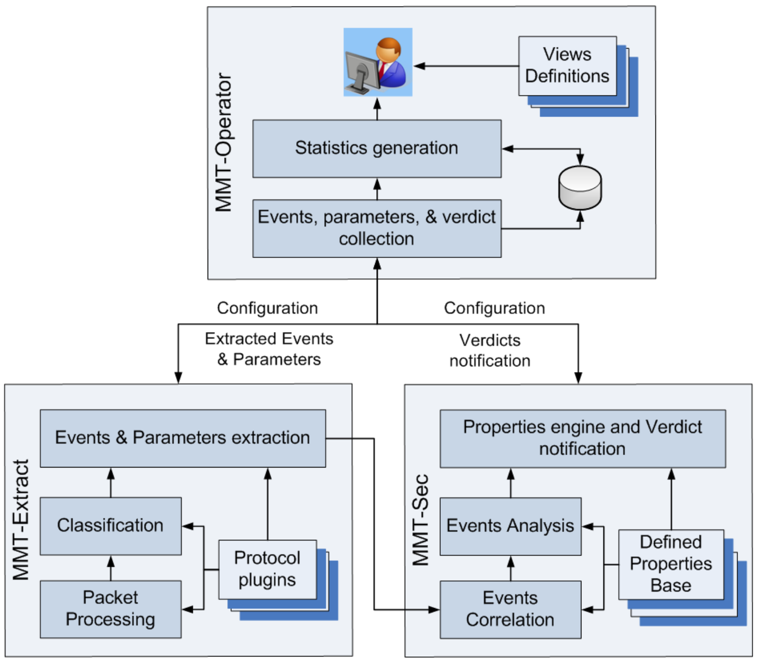


FIGURE 7. MMT Global architecture

security and performance monitoring and operation troubleshooting. MMTs rules engine can correlate network and application events in order to detect operational, security and performance incidents. In this paper, we relied on MMT-Security that is a functional and security analysis tool (part of MMT) that verifies application or protocol network traffic traces against a set of MMT-Security properties. These properties are prepared by the specification engine.

Our monitoring solution can be composed of several probes monitoring different network interfaces and application internal events. They are connected to a central application correlating information to obtain a more global view of what is happening. The analysis for security purposes follows four steps:

1. The definition of the monitoring architecture: this architecture depends on the nature of the application and its deployment in the network. Capture engines (i.e. probes) are placed at relevant elements or links in the network to obtain real time visibility of the traffic to be analyzed.
2. The description of the system security goals and attacks based on the MMT security property format: the description specifies the security policies that the studied system has to respect (output of the negotiation mechanism) and/or the security attacks that it has to avoid. This task is realized by the specification engine.
3. The security analysis: based on the security property specification, the passive tester performs security analysis on the captured events (application events, network packets etc.) to deduce verdicts for each property. A plug-in was developed for the e-voting system in order to understand and to be able to analyze the log files and the key messages exchanged for this application.
4. The reaction: in case of a fail verdict, some reactions have to be undertaken, based on previously defined security strategies, e.g. to block any malicious behavior or the triggering of a renegotiation mechanism.

5. EXPERIMENTS AND DISCUSSION

To prove the effectiveness of our framework, we applied it to a real case study: An e-voting application provided by Scylt company. Thus, we show that our framework allow to detect potential attacks and verify the correct implementation of the proposed security rules.

5.1. e-voting modeling

Figure 8 presents the model of the e-voting application performed in the context of INTER-TRUST project. The EFSM contains 9 main states. The starting state is the **login** state, which represents the first page of the web application. In this step the user is asked to introduce his login and password. If there is an error in the login, an error message is displayed in this page. If the authentication is successful the system passes to the **privacy_options** state. As it is shown in Figure 9 in this state the user is asked to choose his privacy options. There are 3 types of options, for each type there are 2 choices. Theses options are related to the authentication, encryption and signature options.

- Authentication: The system forces the voter to enter recognized or advanced (user + password)

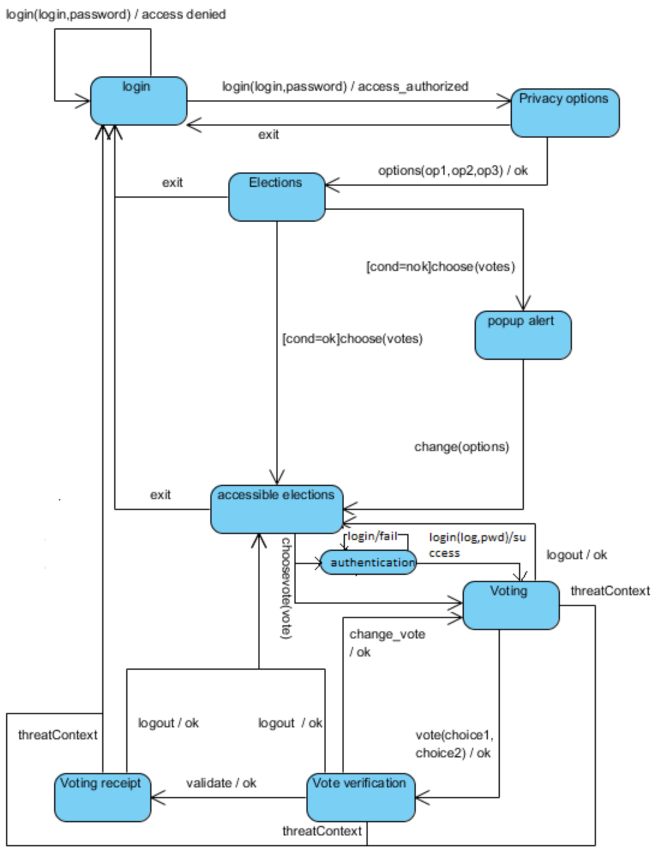
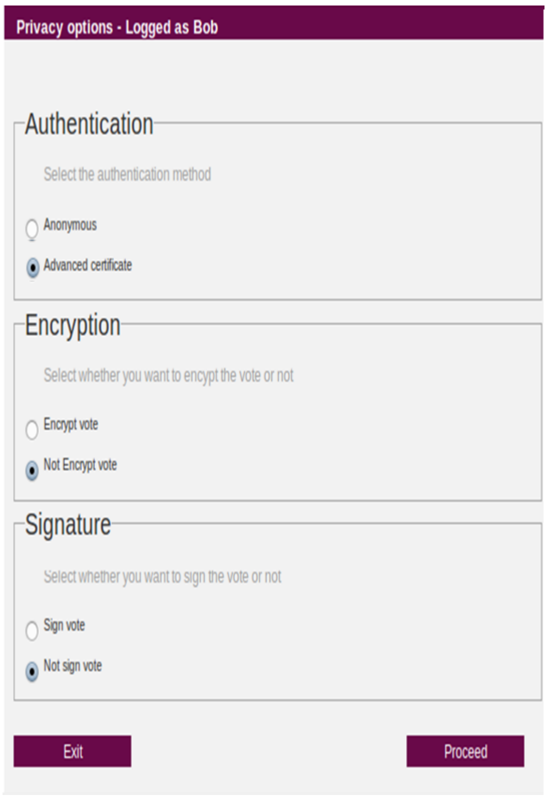


FIGURE 8. EFSM of the e-voting system



Privacy options - Logged as Bob

Authentication

Select the authentication method

Anonymous

[✓ Advanced certificate]

Encryption

Select whether you want to encypt the vote or not

Encrypt vote

[✓ Not Encrypt vote]

Signature

Select whether you want to sign the vote or not

Sign vote

[✓ Not sign vote]

FIGURE 9. List of privacy options

certificates in order to access to the platform. The recognized certificate can be any type of certificate installed in a smart card or in the voter's favorite browser, like electronic voter id card, company's personal smart card, etc. The advanced certificate is composed of a username and a password that can be used as a pseudonymous authentication mechanism.

- Encryption: Encryption is always done in the server-side. For the client, it can be delegated (No encrypt option is filled) if the vote is casted from a limited resources voting terminal (i.e. Smart phone).
- Signature: It is a mathematical technique used to validate the authenticity and integrity of the vote. The voter may choose to sign or not.

The voter has to choose his options that will affect the list of possible elections. When the user makes his choices, the system passes to the **elections** state. In this state, the user will choose a list of elections in which he will vote. Once done, if some predefined conditions are not respected a popup alert appear which means that the system is under the **popup_alert** state because there is a warning regarding the security policy. The user must choose other options. However if the conditions are respected the system passes under the **accessible_elections** state where the available elections from which the user can choose a vote are presented. The user can then choose a vote among them making the system pass to the state **voting**, a state in which the vote choices are displayed. The user have to fill the vote form. This step represents the effective vote. However, in some conditions the system cannot pass directly from the **accessible_elections** state to the **voting** state due to a higher level of security and the user will be obliged to authenticate using his login and password for the chosen vote. This step represents another **authentication** state of our model. For the **voting** state, when the user fill the voting form, he makes the system move to the state **vote_verification**. In this step the vote of the user is displayed. The user can verify his vote, then he can confirm (and pass to the **voting_receipt** state) or change his vote. In the **voting_receipt** the vote is validated. The user cannot modify his vote anymore. However he can choose another election or logout. We notice that in every state among the previous states the user can exit the vote, making the system move to the initial **login** state.

5.2. Attack simulation

The test objectives that guide the test generation can specify an intruder behavior. In the context of e-voting, a set of attack behaviors have been specified: brute force attack, deny of service attack at a strategic vote step,

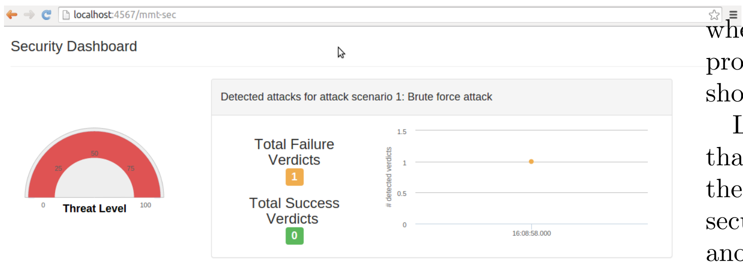


FIGURE 10. Attack report generated by MMT

splitting sessions to cast multiple votes, etc.

Let's take a simple example related to the brute force attack. This attack is an attempt to discover a password by systematically trying every possible combination of letters, numbers, and symbols in order to discover the correct combination that works. In our framework, a set of security properties representing this attack behavior have been specified as an input for MMT and the TestGen-IF tools. One of these properties is a prohibition rule (R_1).

(R_1)= Prohibition(Org1, voter, authenticate, elections, Malicious_behavior)

(R_1) is proposed to limit the number of the wrong authentication attempts. It is a prohibition for a simple voter to access any elections when the context Malicious_behavior is activated. Several attempts of a wrong authentication will enable this context. Therefore, the integration of this rule into the EFSM model is done based on our integration methodology. Next, an abstract case study was generated and translated to a set of HTTP requests based on the Selenium API. Finally, the execution of this attack is done automatically, where the algorithm will try to find the correct password.

In parallel, MMT tool intercepts the e-voting application internal events and correlate them to detect potential misbehavior or anomalies. In the case of the brute force attack, any bad login is notified to MMT using the notification module. If these events occurs more that 3 times for the same user name (application based info) or more that 5 times from the same use IP (network based info), then MMT raises an alarm. The attack is thus detected and the administrator is aware about this result (see Figure 10). In the context of INTER-TRUST, this set of counter-measures has been specified and implemented as aspects (using Aspect Oriented Paradigm) to block the misbehaving user. When a threat context is detected by MMT, the INTER-TRUST framework activates immediately the reaction that is dynamically deployed into the system.

5.3. Security policy specification

A set of security policies have been specified for the e-voting application to take into account different security deployment strategies and to fulfill the user preferences when it is possible (thank to a negotiation mechanism provided by the INTER-TRUST framework). Table 1 shows example of a such security policies.

Let us take the example of the security rule R_8 that stipulates that only authenticated users can cast their votes for any election with the type (L1). This security rule is activated when the user doesn't choose anonymous authentication. This security rule can be used as a test objective to guide the test generation. A set of 3 abstract test sequences are thus generated.

`- Access to the system, choose privacy options, good authentication, vote, cast vote` $\Rightarrow$ `pass verdict`

`- Access to the system, choose privacy options, bad authentication, vote, cast vote` $\Rightarrow$ `false verdict`

`- Access to the system, choose privacy options, vote, cast vote` $\Rightarrow$ `false verdict` (here we do not try to login)

The same property is used to monitor the security of the studied application. Figure 11 presents a testing report generated by MMT related to the verification of this security property.

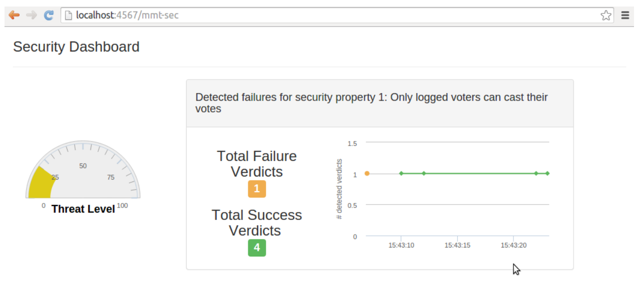


FIGURE 11. A security property verification report by MMT

In this section, we have presented an example of how to use our platform (1) to check the security properties deployed dynamically and (2) to simulate and detect attacks. This process was tested by 39 persons between security testers, developers and researchers [22]. We provide in the following some of their feedbacks about the use of our framework:

- Figure 12 shows the opinion of the experts related to the system dynamic adaptation to change in the environment context. 100% of testers believe that the solution is useful (5% slightly useful, 53% quite useful and 33.3% extremely useful).
- Figure 13 is related to the security property specification. We requested to the assessors to check the difficulty to write security properties based on the proposed format. Then, we find that 92% of the users confirm that it possible and not difficult and only 7% saying that it is possible but difficult.
- The third evaluation question, as shown in Figure 14 was related to the response time between

TABLE 1. An example of a server security policy

Security rule	Description
R2 = Permission (Org1, voter, access, L1, intranet_context)	A voter connecting from the intranet system is permitted to access to the first type of elections (named L1)
R3 = Permission (Org1, voter, access, L2, intranet_context)	A voter connecting from the intranet system is permitted to access to the second type of elections (named L2)
R4 = Permission (Org1, voter, access, L3, intranet_context)	A voter connecting from the intranet system is permitted to access to the third type of elections (named L3)
R5 = Permission (Org1, voter, access, L4, desktop_context)	A voter connecting from a specific desktop is permitted to access to the fourth type of elections (named L4)
R6 = Permission (Org1, voter, access, L5, desktop_context)	A voter connecting from a specific desktop is permitted to access to the fifth type of elections (named L5)
R7 = Obligation (Org1, server, authorize, election_lists, violation_context)	The sever is authorized to check the elections lists when a violation is detected
R8 = Obligation (Org1, voter, authenticate, L1, normal_context)	The voter should be authenticated to access to the first type of election (L1) when the normal context is activated (anonymous context is not activated)
R9 = Obligation (Org1, voter, encrypt, L1_vote, violation_context)	The voter device should encrypt the data related to any election with the type L1 when a violation is detected
R10 = Obligation (Org1, voter, authenticate, L3, violation_context)	The voter should authenticate to access the the third election type (L3) when a violation is detected
R11 = Obligation (Org1, voter, advanced_authenticate, L4, violation_context)	The voter should be authenticated based on advanced methods to access to the fourth type of elections (L4) when a violation is detected
R12 = Obligation (Org1, voter, encrypt, L4, violation_context)	The voter device should encrypt the data related to any election with the type L4 when a violation is detected
R13 = Permission (Org1, voter, update_vote, L1 + L2, already_voted_context)	A voter can update his/her vote in any election of type L1 or L2 if he/she already voted in this same election
R14 = Prohibition (Org1, voter, update_vote, L3 + L4 + L5, already_voted_context)	A voter cannot update his/her vote in any election of type L3 or L4 or L5 if he/she already voted in this same election
R15 = Prohibition (Org1, voter, vote, L1 + L2 + L3 + L4 + L5, already_voted_context)	A voter cannot vote again in any election (Types L1 to L5) if he/she already voted in this same election
R16 = Prohibition (Org1, voter, access, L1_vote + L2_vote +, L3_vote + L4_vote + L5_vote, different_voter_context)	A voter cannot access to the vote of other participants (voters) in any election (types L1 to L5)

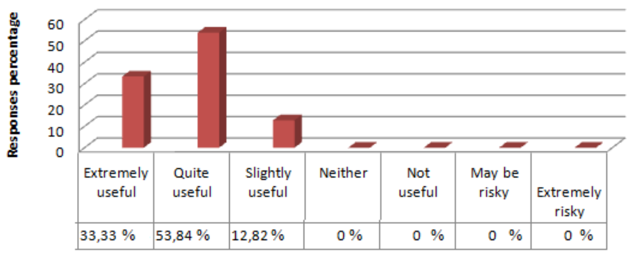


FIGURE 12. The systems dynamic adaptation to changes in the environment context

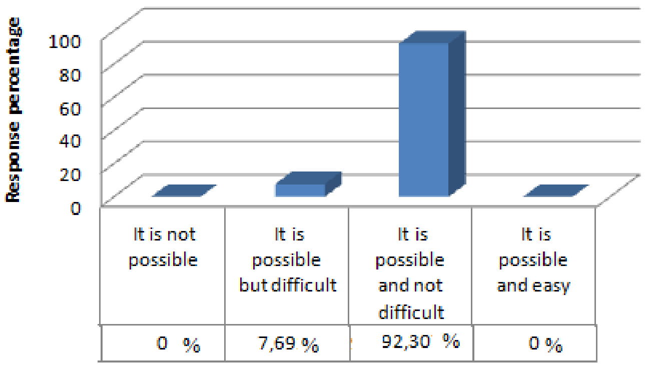


FIGURE 13. Is it easy to express security properties specification with our framework

the detection time and the notification. As a conclusion, the average response time evaluated by the different users was less than the 0.2 seconds for the 98% of the testers and only 2% that have a response time between 0.2 seconds and 0.3 seconds which was also accepted.

- The last question was related to the detection delay during the active testing process, the average value was 0.15 seconds that was really less than the target value given as a requirement for the inter-trust project that was 0.7 seconds.

The performances evaluation corroborates the efficiency of our platform in preventing and revealing intrusions and attacks. The assessors agreed that the monitoring and detection mechanisms are sound and effective in the sens that they allowed the detection of known attacks and bugs. The monitoring module and the testing tools can be easily applicable to different use cases. It is required to configure properly the monitoring functions (to fit the security rules defined with OrBAC), and to model the system behavior test objectives for the testing tool. Moreover, as underlined by some testers, this proposal is a part of the INTER-TRUST framework which can be exploited separately from the rest of the framework.

6. CONCLUSION

A security testing and monitoring framework is proposed in this paper for adaptive pervasive systems. Our approach was applied to an electronic voting system to be validated. Several security requirements, such as privacy, confidentiality, integrity, have been taken into account when designing our solution.

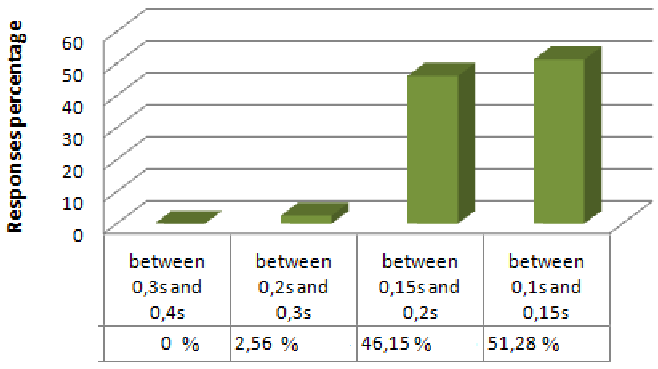


FIGURE 14. Detection delay during active testing

Moreover, aware of the complexity of designing a system that verifies the correctness of security mechanisms, our approach aims to become the panacea of such a challenging problem. Thus, a three-component based solution was presented in this paper. A first component called 'System modeling integrating security constraints component' was designed to integrate the specified security policy into the target system, here being the e-voting functional model. The second component, denoted as 'the test generation and execution module', is responsible of the generation of test cases based on security objectives and allows then the automatic execution of the test cases based on the Selenium IDE. The last component handles the detection of security vulnerabilities that will generate warnings, alarms and reaction countermeasures. This environment was designed and implemented in the context of the INTER-TRUST project and has been applied to a real industrial e-voting application provided by the Scytl company. The performances evaluation part has confirmed the efficiency of our approach.

REFERENCES

[1] Nez A. and Merayo M.G. (2014), A formal framework to analyze cost and performance in Map-Reduce based applications. Journal of Computational Science, 5, 106-118.

[2] Cavalli A. R., Oca E. M. D., Mallouli W., and Lallali M. (2008), Two complementary tools for the formal testing of distributed systems with time constraints, Proceedings of DS-RT 08, Vancouver, Canada, 27-29 October, pp. 315-318. IEEE Computer Society.

[3] Senn D., Basin D. A., and Caronni G. (2005), Firewall conformance testing. Proceedings of TESTCOM 05,Montreal, Canada May 31 - 2 June, pp. 226-241, Springer, Berlin.

[4] Shahriar, H. and Zulkernine, M. (2011) Taxonomy and classification of automatic monitoring of program security vulnerability exploitations. Journal of Systems and Software, 84(2), 250-269.

[5] Bertolino, A., Daoudagh, S., Lonetti, F., Marchetti, E. and Schilders, L. (2013). Automated testing of extensible access control markup language-based access control systems. IET software, 7(4), 203-212.

[6] Felderer, M., Agreiter, B., Zech, P. and Breu, R. (2011). A classification for model-based security testing. Proceedings of VALID 11, Barcelona, Spain,23-29 October, pp. 109-114, IARIA.

[7] Aouadi, M. H., Toumi, K. and Cavalli, A. (2014). Testing Security Policies for Distributed Systems: Vehicular Networks as a Case Study, Journal of Computer Science Issues, 11, pp.68-77.

[8] Kiraz, M. S. and Uzunkol, O. (2016). Efficient and verifiable algorithms for secure outsourcing of cryptographic computations. International Journal of Information Security, 15(5), 519-537.

[9] Baiardi, F., Falleni, A., Granchi, R., Martinelli, F., Petrocchi, M. and Vaccarelli, A. (2005). SEAS, a secure e-voting protocol: design and implementation. Computers and Security journal, 24(8), 642-652.

[10] Enrico M. et al. (2015) Interoperable Trust Assurance Infrastructure. Available: http://inter-trust.lcc.uma.es/documents

[11] Ferraiolo, D., Cugini, J. and Kuhn, D. R. (1995). Role-based access control (RBAC): Features and motivations. Proceedings of ASAC 95, New Orleans, Louisiana, 11-15 December, pp. 241-48, IEEE Computer Society.

[12] A.A.E Kalam et al.(2003), Organization based access control, Proceedings of POLICY 03, Villa Olmo, Lake Como, Italy, 4-6 June, pp. 120-131 , IEEE Computer Society.

[13] Toumi, K., Cavalli, A. and Maarabani, M. E. (2012). Role based interoperability security policies in collaborative systems. Proceedings of CTS 12, Denver, CO, USA, 21-25 May, pp. 471-477, IEEE Computer Society.

[14] Reaves, B. and Morris, T. (2012). An open virtual testbed for industrial control system security research. International Journal of Information Security, 11(4), 215-229.

[15] Gouglidis, A., Mavridis, I. and Hu, V. C. (2014). Security policy verification for multi-domains in cloud systems. International Journal of Information Security, 13(2), 97-111.

[16] Oliveira, R. A., Laranjeiro, N. and Vieira, M. (2015). Assessing the security of web service frameworks against Denial of Service attacks. Journal of Systems and Software, 109, 18-31.

[17] Information technology - Advanced Message Queuing Protocol (AMQP) v1.0 specification, ISO/IEC 19464,2014.

[18] INTER-TRUST team. (2014). D3.3 Guidelines for Deployment [Online]. Available: http://inter-trust.lcc.uma.es/documents

[19] Adida, B. (2008, July). Helios: Web-based Open-Audit Voting. In USENIX security symposium, San Jose, California, USA, July 28-August 1, pp. 335-348, USENIX association.

[20] Hwang, I., Cavalli, A. R., Lallali, M. and Verchere, D. (2012). Applying formal methods to PCEP: an industrial case study from modeling to test generation. Software Testing, Verification and Reliability, 22(5), 343-361.

[21] Cavalli, A., Lee, D., Rinderknecht, C. and Zadi, F. (1999). Hit-or-Jump: An algorithm for embedded testing with applications to IN services, Proceedings of FORTE 99, Beijing, China, 5-8 October, pp. 41-56. Springer USA.

[22] INTER-TRUST team. (2015). INTER-TRUST: Final Evaluation Report [Online]. Available: http://inter-trust.lcc.uma.es/deliverables