Intermediate report on Potestat

IRISA - Distribcom, Lande, Vertecs   LSR - Vasco

VÉRIMAG - DCS

May 2006

1 Overview

The POTESTAT project addresses the problem of testing security policies for open networked systems. It is a joint project of 5 teams in 3 laboratories. Participants at the date of May 1st, 2006 are:

- Thierry Jéron (CR Inria, IRISA - Vertecs)
- Hervé Marchand (CR Inria, IRISA - Vertecs)
- Vlad Rusu (CR Inria, IRISA - Vertecs)
- Loïc Hélouët (CR Inria, IRISA - Distribcom)
- Thomas Jensen ((CR Inria, IRISA - Lande)
- Gurvan Le Guernic (PhD, IRISA - Lande, from Oct 2002-MENRT + mobility grant from MENRT)
- Jérémy Dubreil (PhD IRISA - Vertecs - from March 2006 - INRIA contract)
- Jean-Claude Fernandez (PR, Verimag - DCS)
- Laurent Mounier (MCF, Verimag - DCS)
- Cyril Pachon (PhD, Verimag - DCS, 2001–October 2005, supported by contract)
- Roland Groz (PRASS, LSR - Vasco)
- Marie-Laure Potet (PR, LSR - Vasco)
- Jean-Luc Richier (CR CNRS, LSR - Vasco)
- Keqin Li (Post-Doc CNRS, LSR - Vasco, Dec 2005-Nov 2006)
- Vianney Darmaillacq (PhD, LSR - Vasco, MENRT from Oct 2003)

The initial approach to the problem was based on previous experience of the partners. We had experience on the use of formal models either to test the conformance of a distributed implementation to a specification (conformance testing for network protocols) or to analyse downloaded code (where testing can complement static analysis techniques). Based on this background, we proposed four directions of investigation.

1. Executable models of security policies: our concern was to identify formal models for security policies, to capture behaviours that could be exercised with active tests.

2. Methodology for testing policy enforcement: the goal was to define and formalize the concepts of testing and the associated methodology to relate requirements, policies, actual systems etc; in the case of protocol conformance testing, the Z.500 standard provides a formalization of the methodology defined in IS 9646. But the test of security is not well defined or establish, and we need a clarification of concepts.

3. Test selection/generation: we have some experience with test generation from formal models. However, in the case of security policies, testing cannot be based on the same principles. First because security is part of non-functional requirements, and a functional model would generally not be available and it would have to span various astraction levels. Second, the selection of test data in the absence of model may be crucial to uncover violations of security.

4. Conformance of external code to a given policy: in the case of open networked systems, not only the system must abide by a given policy, but it is crucial to check whether code incorporated by dynamic loading conforms to the policy.

Our preliminary investigations have led us to reorganize the projects along three main directions. Actually, we have looked into several models, but there is no clearly identified executable model that could serve as a reference. Contrary to e.g. reactive systems, where the scientific community was able to establish long ago typical benchmarks and theoretically sound bases for models, there is no such consensus for network policies. Since testing is a new concern, testable models have not been identified in the state of the art. This also affects the theorization of testing in that context. On the other hand, links with other techniques have been identified, and we have reorganized our investigations in the following directions.

- Non-interference. Three groups in the project address the use of testing techniques to assess non-interference on three complementary goals: confidentiality, secrecy and a more qualitative approach with the analysis of covert channels.

- Diagnosis. Whereas protocol testing is usually done through active tests, it turns out that passive testing techniques may be better related to the control of security requirements, through monitors or access controllers for instance.

- Generation of attacks. IRISA and the Grenoble labs have investigated complementary approaches on this issue: based on combination of properties with symbolic models in Rennes, based on direct generation from security rules in Grenoble.

In the following sections, we present those three main directions. Although this project has already given birth to a number of scientific publications, we are still in an early phase, where we investigate different approaches. We started from a background in functional (dependability) validation, and we are integrating the specificities of addressing security requirements. Although we did not expect to develop tools within the project, we are now contemplating small scale experimentations to see whether the techniques can actually be used to detect violations. It is as yet premature to compare to other approaches in security (e.g. Intrusion Detection Systems and associated models).

Apart from publications, the POTESTAT project has also given birth to a collaborative project with industrial (and other academic) partners: POLITESS, started in march 2006, has been accepted as a RNRT project. It incorporates elements that come from our first findings about models, the use of passive testing, test generation for policies etc. Another project, POSE, started in January 2006, has been accepted by RNTL. It involves only one partner of POTESTAT, but it also entails modelling of security policies and test generation for them, in the context of smartcard applications.

2 Non-interference

In this section, we report on three different research directions on non-interference. The general problem of non-interference consists in determining whether an attacker is able or not to discover some secret behavior of the system from its observations. In the first direction followed by the Lande team, one considers non-interference of one execution of a deterministic program. The second direction of work followed by the Vertecs team is concerned with a weak notion of non-interference of systems parametrized with a safety property (the secret). The third direction followed by the Distribcom team is concerned with the analysis of covert-channels.

2.1 Strict non-interference (confidentiality)

In this work, we have been looking at the respect of the strict *non-interference* property by programs. This property has been introduced by Cohen [19] and Goguen and Meseguer [21]. A program respecting the non-interference property is ensured to respect the *confidentiality* of its secret data. Basically, this property states that a variation in the inputs containing secret data has no effects on the public outputs of the program. Sabelfeld and Myers ([25]) state this property as follows:

$$\forall \sigma_1, \sigma_2 \in S, \sigma_1 =_L \sigma_2 \Rightarrow [\![C]\!]\sigma_1 \approx_L [\![C]\!]\sigma_2.$$

In this statement, C is a command (or program) respecting the non-interference property. The set S contains all the program states. The relation $=_L$ takes two program states as parameters; it is true whenever the two states differ only in their secret data. $[\![C]\!]\sigma$ is the "behavior" of the execution of the command C with the initial state σ. The relation $\approx_L$ is true whenever the two "behaviors" given in parameters do not differ at the level of their public outputs.

As in the rest of the project, we have been looking at the behavior of executions and not of the whole program. The standard approach for non-interference (i.e. static analyzes) is then not what interested us. We have been looking at what it means for an execution to be non-interfering and how to check or ensure it. Two approaches have been followed: either monitoring executions in order to ensure that it is non-interfering, or test executions of programs to gain some confidence that the program respects the confidentiality of its secret data. Some of this work have been published in [12].

Monitoring. Following some works by Schneider [26], Terauchi and Aiken [27], and Ligatti, Bauer and Walker [23], we developed an automaton tracking variables containing secret data during an execution. A semantics including this automaton has been formalized. Before executing any action, an input is sent to the automaton which checks the validity of this action, and either validates the execution of that action, or sends an other action to be executed instead of the one which should have. As non-interference is not a property of the sequence of actions of an execution, the automaton checks the commands contained in branches which are not executed. The correctness of the method has been proved.

Testing. The other approach considered involved the development of an extended semantics dealing with tagged data. Those tags evolve during the execution and keep track of the security level of the corresponding data. Based on the tags obtained at the end of the execution, it is possible to check if the corresponding execution respected the confidentiality of its secret data. As with monitoring, an analysis is used to get some inputs about the behavior of non-executed commands. The information acquired by this method is more precise than the one the monitoring method gives back. However, it can not be used for monitoring because of the creation of a new covert channel using the tags themselves. This method is then more appropriate for testing.

2.2 Weak non-interference (secrecy)

It is well known that the notion of non-interference introduced by Goguen and Meseguer [21] is often too restrictive for its application in real situations. In fact one would like to relax the non-interference property depending on what an external observer is authorized to learn about the internal behavior of the system.

In this work [5] we consider a system modelled as a symbolic transition system (STS) $\mathcal{S} = (V, \Theta, \Sigma, T)$ where V is a vector of variables, including locations, Θ is the initial location defined as a boolean expression on variables in V with a unique solution, $\Sigma = \Sigma_o \cup \Sigma_{uo}$ is the alphabet of actions partitionned into observable actions in Σ_o, and internal unobservable actions in Σ_{uo}. These actions may carry communication parameters $sig(a)$ in P, T is a set of transitions of the form (a, G, A) where a is an action, G is a boolean expression on V and $sig(a)$ called the guard, and A is the assignment of variables in V to expressions on $V \cup sig(a)$. The semantics of $\mathcal{S}$ is a labelled transition system (LTS) $S = (Q, Q_0, \Lambda, \rightarrow)$ where $Q = Dom(V)$ is the set of states, Q_0 is the unique solution of Θ, $\Lambda = \Lambda_o \cup \Lambda_{uo}$ is the set of valued actions (actions with values of parameters), partitionned according to the partition of Σ, and $\rightarrow$ is the transition relation. One can view the system as a set of executions in $Runs(\mathcal{S}) \subseteq Q_0.(\Lambda.Q)^*$, a language (set of sequences) $L(\mathcal{S}) = proj_\Lambda(Runs(S))$ or a set of traces $Traces(\mathcal{S}) = proj_{\Lambda_o}(Runs(S))$.

In the following, we assume that a *secret* is specified by a safety property of runs (more general situations have not been investigated yet). This can be expressed by a deterministic $STS\ \Omega = (V \cup V_\Omega, \Theta_\Omega, \Sigma, T_\Omega)$ which non-intrusively observes the evolution of the variables of the system $\mathcal{S}$. Ω is equipped with a stable boolean variable *Accept* (once true, *Accept* remains true), confering it the status of recogniser of runs ($Runs_{Accept}(\Omega) = \{s \in Runs(\Omega) \mid \Theta_\Omega \xrightarrow{s} q \wedge q \models Accept = true\}$. By the stability of *Accept*, the set of runs $Runs_{Accept}(\Omega)$ is extension-closed i.e. $Runs_{Accept}(\Omega).(\Lambda.Q_\Omega)^* \subseteq Runs_{Accept}(\Omega)$, meaning that Ω specifies the violation of a safety property.

Now non-interference of $\mathcal{S}$ with respect to secret Ω is defined as follows :

$$\begin{aligned} NI(\mathcal{S}, \Omega) \quad = \quad & \forall s \in Runs(\mathcal{S}), proj^{-1}(s) \in Runs_{Accept}(\Omega), \\ & \exists t \in Runs(S), proj_{\Sigma_o}(s) = proj_{\Sigma_o}(t) \wedge proj^{-1}(t) \notin Runs_{Accept}(\Omega) \end{aligned}$$

Intuitively, this means that every run s of $\mathcal{S}$ which is accepted by Ω (the run exhibits the secret) is masked by another run with same trace, which does not exhibit the secret. This generalizes the definition of opacity given by [18, 15].

An attacker of the system is then a diagnoser (see section 3) whose aim is to diagnose membership of observed executions in $Runs_{Accept}(\Omega)$. Such a diagnoser is obtained as $Det(\mathcal{S} \times \Omega)$ (determinisation of the synchronous product between $\mathcal{S}$ and Ω) equipped with a function *Diag* on sets of states such that for $R \subseteq 2^{Q \times Q_\Omega}$, $Diag(R) = true$ if and only if $\forall q \in R, q \models Accept = true$. Now, the system is non-interferent if no such set of states is reachable in $Det(\mathcal{S} \times \Omega)$, meaning that the diagnoser will never succeed in determining that the secret has been exhibited by all runs that could produce the observation.

Constructing the diagnoser and verifying non-interference is relatively easy when $\mathcal{S}$ is finite state. However, it is much more difficult for infinite state systems. First $Det(\mathcal{S} \times \Omega)$ cannot always be constructed, but is possible with sufficient conditions on $\mathcal{S} \times \Omega$ [11]. Second, verifying non-interference relies on reachability and is in general undecidable. However, if non-interference can be proved on an over-approximation of $Det(\mathcal{S} \times \Omega)$ (e.g. using abstract interpretation), it is preserved on $Det(\mathcal{S} \times \Omega)$.

In the case where non-interference is false (or not proved), one should test if this attack corresponds to a real attack in the real system (see section 4).

2.3 Covert-channel, Distribcom

Covert channels are illegal information flows of information that are used secretely to transfer information. The main drawback of a covert channel is that it can be used to violate a security policy. Another drawback is that the covert mechanisms used to transfer information often use large amounts of systems resources, and hence impose a performance penalty to other users.

Covert channel identification has first been proposed for the classical Bell & La Padulla Model [17, 16]. The main idea was to detect potential flow in the transitive closure of an access matrix as in [22].

Covert channels have also been characterized in other models such as process algebra [24]. The current trend is to characterize covert channels as an interference property. The following interference formula defines non interference for a pair of users u, v that use a system S

$$u||S||v \approx S||v$$

This formula means that user v can not observe what u does (or even if u uses the system) through its use and observation of S. However, as stated in [24], an interference only characterizes the fact that two actors in the system may exchange at least one bit of information at one moment in time, but nothing more.

Another problem with the non-interference framework is that a covert channel is characterized when two actors u and v that are not allowed to communicate in the system's security policy violate the non-interference property. However, some covert channels are "legitimate", ie some hidden information can be passed from an actor to another over a legal flow of information.

Our work focuses on a notion of "iterated interference" to characterize the presence of a covert channel in a system, and on the extension to legitimate channels. A first framework based on games and scenarios has been proposed in [7, 6, 8], and a new characterization in alternating μ-calculus has been proposed by [4].

In this work, we start from a labeled transition system symbolizing the execution of a distributed system, and we chose a pair of users $p1, p2$. In this LTS, we identify some interfering places, i.e. states from which $p1$ may signal bit 0 or bit 1 to $p2$ (roughly speaking, there are at least two runs $r1, r2$ starting at interfering places such that the projection of $r1$ and $r2$ on actions observed by $p2$ are different). Being an interfering place can be defined with an alternating μ-calculus formula. Then the presence of a covert channel from $p1$ to $p2$ is characterized as the ability to pass infinitely often through interfering places. This can also be expressed as an alternating μ-calculus formula. We have shown that the satisfiability of such a formula is decidable, and implies the existence of a distributed strategy to pass infinitely often through interfering places.

3 Diagnosis of security policies specified by safety properties

The general problem is to detect, based on the observation whether or not a security policy, corresponding to a particular set of sequences, is violated in the system. In this section, we outline two different research directions on diagnosis of security policies. In the first one, the idea was to show how the diagnosis techniques traditionally used to supervise control-command systems, can be adapted to monitor a system on-line with respect to some securities policies corresponding to safety properties.

3.1 The diagnosability of discrete event-systems applied to security policies violation detection

The general diagnosis problem is to detect or identify patterns of particular (sequence of) events, corresponding to policy requirements, on a partially observable system. The aim of diagnosis is to decide, by means of a *diagnoser*, whether or not such a security policy pattern occurred in the system. Even if such a decision cannot be taken immediately after the occurrence of the pattern, one requires that this decision has to be taken in a bounded delay. This property is usually called *diagnosability.* This property can be checked *a priori* from the system model, and depends on its observability and on the kind of patterns which are looked for.

We assume that the system is modeled as a symbolic transition system (STS) $\mathcal{S} = (V, \Theta, \Sigma, T)$. We refer to Section 2.2 for the mathematical definition as well as the notations used here after. We formally introduce the notion of pattern as a means to define the security policies. A *security policy pattern* is specified by a safety property of sequences expressed by a deterministic *STS* $\Omega = (V_\Omega, \Theta_\Omega, \Sigma, T_\Omega)$ which observes the evolution of the system $\mathcal{S}$. Ω is equipped with a stable boolean variable *Violate* and we denote by $\mathcal{L}_\Omega(Violate)$ the set of sequences that violate the corresponding security policy. Note that these patterns are more expressive that the ones that are considered in [26] as they allow to specify properties on sequences (or even runs) and not only on traces.

Now, the Diagnosis Problem is expressed as the problem of synthesizing a function over traces, the *diagnoser*, which decrees on the possible/certain occurrence of the pattern on trajectories compatible with the trace. The diagnoser is required to fulfill two fundamental properties: *correctness* and *bounded diagnosability. Correctness* expresses that the diagnoser answers accurately and *Bounded Diagnosability* guarantees that only a bounded number of observations is needed to eventually answer with certainty that the pattern has occurred. *Bounded Diagnosability* is formally defined as the Ω-diagnosability of the system (where Ω is the security policy pattern). Formally, the *Diagnosis problem* can be stated as follows: given an STS G and given a security policy pattern Ω, decide whether there exists (and compute if any) a three valued function $Diag : Traces(\mathcal{S}) \rightarrow \{Yes,\ No,\ ?\}$ decreeing, for each observable trace μ of $\mathcal{S}$, on the membership in $\mathcal{L}_\Omega(Violate)$ of any trajectory compatible with μ. Formally,

- (Diagnosis Correctness) The function should verify
$$Diag(\mu) \quad = \begin{cases} Yes & \text{if } proj^{-1}(\mu) \cap \mathcal{L}(\mathcal{S}) \subseteq \mathcal{L}_\Omega(Violate) \\ No & \text{if } proj^{-1}(\mu) \cap \mathcal{L}(\mathcal{S}) \cap \mathcal{L}_\Omega(Violate) = \emptyset \\ ? & \text{otherwise.} \end{cases}$$

- (Bounded Diagnosability) As $\mathcal{S}$ is only partially observed, we expect in general situations where $Diag(\mu) = ?$. However, we require this undetermined situation not to last in the following sense: There must exist $n \in N$, the bound, such that whenever $s \in proj^{-1}(\mu) \cap \mathcal{L}(\mathcal{S}) \cap \mathcal{L}_\Omega(Violate)$, for all $t \in \mathcal{L}(\mathcal{S})/s \cap \Sigma^*.\Sigma^o$, if $\|proj(t)\| \geq n$ then $Diag(proj(s.t)) \ = \ Yes$.

Such a diagnoser is obtained as $Det(\mathcal{S} \times \Omega)$ equipped with a function *Diag* on sets of states such that for $R \subseteq 2^{Q \times Q_\Omega}$, $Diag(R) = \ Yes$ (resp. No) if and only if $\forall x \in R, x \in Q \times Violate$ (resp. $\notin Q \times Violate$ and ? otherwise). Verifying the Bounded Diagnosability is performed on $\epsilon(\mathcal{S} \times \Omega) \times \epsilon(\mathcal{S} \times \Omega)$ where ϵ corresponds to the epsilon closure of the STS $\mathcal{S} \times \Omega$. The idea is to test the presence of some undetermined states cycles in this STS. If any, then the system is not diagnosable with respect to the security policy pattern.

As for the non-interference problem, tackled in the previous section, constructing the diagnoser and verifying the bounded Diagnosability property is easy when S is finite state [9, 10]. For general infinite state systems, as the determinization is necessary to construct the diagnoser, the product $\mathcal{S} \times \Omega$ should fulfill some conditions [11]. Now, if the bounded Diagnosability property is proved on an over-approximation then it is preserved on the system.

Finally, one can adapt the results presented in Section 2.1 (paragraph Monitoring) in order to stop the execution of the system whenever the violation of the safety properties has been discovered by the diagnoser. In a strict security point of view, one can imagine to use this for intrusion detection. The policy describes behaviors one wants to prevent and the diagnoser plays the role of an intrusion detection system (IDS). The first advantage of the approach is that, when the system is diagnosable, the IDS can work on-line with logs restricted to observable actions, thus preventing the explosion in the size of logs. Moreover, the diagnosability bound n can serve to forget old events past the $n + 1$ last ones.

3.2 Refinement

A classical problem in network security is that security requirements are expressed either in a very abstract, either in a very concrete way.

For examples security policy defined by the management of an organization express what individuals or rôles are allowed to do, when, where, etc and can describe complex behaviours.

This allows to abstract from technical constraints caused by configuration, hardware, software, and to express security in rather broad terms of assets, individuals, data, confidentiality. But it makes it difficult to apply these requirements to a real system.

At the other end of the spectrum, security policies for administrators are state-of-the-art guidelines, classical configuration files for servers or architectural considerations. Easy to apply, but it is difficult to establish a link between the abstract and the concrete level. The mapping is up to the administrator to be done.

There is no sure way to know if the mapping is correct. There is a consensus in the security world that it is, the only problem remaining being to maintain your software and firmware up to date (antivirus, ...). In practice, errors and misconfigurations are very frequent, and administrators wait for users to signal problems to them.

A goal of Potestat is to resolve this problem by testing the conformance of a network to an abstract security policy. To attain this goal we refine an abstract policy into concrete data that could be used at a more concrete level shall be provided.

We already publish some work [3] in this area. [3] proposes a method to verify the conformance of a network to an abstract security policy by monitoring data flows.

The security policy is a collection of authorizations and prohibitions expressed over individuals, roles in the organization, and resources. Resource can be some data items, or a service provided by the network. Examples of rules are that *John is authorized to send mail* (in this case the resource is the process in the network used to send electronic mails, or *John can access the Intranet of the organization.*

We use the B method [14] to develop the monitor. This method allow to build a software by consecutive refinements of data and process. Each of the refinement is proved.

We decide that each of our B refinement will be used to refine both description of the environment (the network) and the refinement of the abstract security policy. The levels of description are approximately the same that those of the TCP/IP model.

A set $Event_i$ represents all possible events at the i^{th} description level. $Event_0$ is the level of the abstract security policy (not defined in the TCP/IP model). A relation, $mapping_i$, allow to link events from the concrete level i to the corresponding events at the abstract level, at which the security policy is defined.

A major problem is the limited observability of data flows in a network. Due to performance issues all informations about low-level events cannot be collected. For example it is not unusual that from a IP packet, it is not possible to obtain information about the applicative layer.

This makes impossible, in some cases, to determine exactly the link between the observed concrete event and an abstract event described in the security policy. An over- approximation shall be used. This causes conflicts, as for some concrete events the monitor can make a link with a set of abstract events, some of them being authorized and others prohibited. For this reason we create new values to indicate if there is a doubt or if something happened we do not understand. The monitor logs all these alerts.

Even when a direct link can be established between high and low level, subsists the problem of the trust in the link, very frequent in security. For example, such a link is one that indicates if one computer can be used by one individual (it can be its workstation). What happened if another individual use the computer? If this can happen the link between the levels is not sure. More or less trust can be placed in the link, depending on the security measures associated with the link (authentication, ...) and the security degree wished for the monitor.

We are currently working on the merging of these results on refinement of security data with other works in Potestat about active conformance security testing of networks.

4 Generation of attacks from security policies

4.1 Test Generation for Network Security Rules

Our main goal is to define a methodoly for testing policy enforcement. Some sort of testing is actually performed by system administrators to check for known vulnerabilities : portscans and password crackers fall into that category. However, such tests are usually limited to just a single security mechanism. With existing techniques, it is difficult to address the issue of consistency of configurations on distributed devices. Our approach is to consider that testing a given network configuration for compliance with a stated policy is a kind of conformance testing. Therefore, we aim at deriving tests from a formal and global specification of the security policy to check whether the implementation is correct.

4.1.1 Approach for test generation

There are a number of significant differences with the framework of protocol conformance testing. Protocol conformance testing is done on a well defined protocol level (normally one at a time), with a given interface and associated

PDU definitions. We do not assume any formal model of the network and the policy is not described by a comprehensive model such as a global LTS or state machine, but by a collection of rules. The policy is described at a much higher level than the actual events that can be observed or controlled in the network. Security policies are often implemented with a mixture of mechanisms at various levels of communication and O/S inter- faces. Moreover, security rules use deontic modalities, such as authorization, denial and obligation. Testing such modalities is not straightforward and the use of deontic logic raises a number of paradoxes.

We propose a method to derive tests from a policy expressed as a set of rules. We consider this as a first step to define a global methodology for automatic derivation of tests suites as well as the execution of these tests suites. In order to identify typical requirements and the corresponding tests, we start from a case study, which is to identify the security policies used in the IMAG network, which connects our laboratories. Most of the rules in the case study (which actually covers much more than e-mail) express some constraints about the possible behaviour of the system. More specifically, they are of the form "*Mod* *P*", where *Mod* is a modality among obligation, permission or interdiction, and *P* is either a predicate on the system or a behaviour. We need to establish a correspondence between the basic predicates appearing in a rule and sequences of events that can represent a test or instanciations of such predicates. There are different types of predicates. Some may have to be tested dynamically through interaction with the system. Others might be checked without PDU exchanges, for instance if we have access to the configuration files of the system when the test is set up.

For the moment, we do not address all issues. We identify a language with a restricted form of rules that cover most of the rules that can be found in texts describing security policies, keeping in mind that we are focusing on rules that are translated into actual configurations and behaviours or network security devices. This restricted set of rules allows us to design a "tile-based" generation method. For each element of our language and each type of rule, we propose a pattern of test, which we call a tile. The simple form of rules makes it possible to compose a global test by simply combining the tiles associated to the elements. We have proposed [2, 1] a formal description of the rules: each rule is expressed by a logical formula, built upon literals. With each literal is associated a tile. Based on an operational semantics of our logic, we have defined a test generation function, which combine the tiles.

4.1.2 Perspectives

In our tile-based approach, complete test cases (dedicated to a whole formula) are obtained by combinations of more elementary ones (the tiles), following a syntax driven approach (a test combinator is associated to each logical operator of the formula). Elementary test cases, allowing to test basic events or predicates appearing in the security policy, have to be provided by the user. Our test generation method is based on the fact that security policies are most of the time expressed by rules which can be captured by a restricted logic as the one we used. Therefore it should be extended into several directions. First of all, the test cases produced are still very *abstract*. Turning them into executable test cases needs to take into consideration a concrete test architecture. Assuming that each elementary test case complies with this architecture, it would remain to ensure that it is also the case for the complete test case (or alternatively to take this architecture into account during the combination process). Moreover, these abstract test cases also need to be *instantiated* with concrete data (e.g. by selecting particular machines of the network). Suitable selection strategies should therefore be investigated (for instance a test could focus on the more recent changes in a network configuration, as in regression testing). Furthermore, the proposed generation technique itself could be improved. the test architecture).

Another improvement could be to extend the formalism we considered to specify the security rules. This initial choice was motivated by our case study, and it was sufficient to demonstrate the effectiveness of the approach. However, it is clear that this formalism may be not sufficient to deal with arbitrary security rules, and that more specific operator/modalities need to be considered. One can think for instance of a triggered obligation bounded by an event (and not by an arbitrary timeout), or of some of the general operators proposed in the NOMAD logic [20]. Further work remains to be done in order to check which of these operators could be supported by our tile-based approach.

Finally, we also intend to evaluate this work on other case studies, and to prototype it on a real network.

4.2 Generation of attacks with test generation techniques

In this work, we investigate the use of test generation techniques for the generation of attacks from security policies and network models.

We assume the existence of a model of the network. It is unrealisitc to assume that the model will completely describe the network behaviour, but one can imagine to model an abstraction of this behaviour or an abstraction of some elements of the network. We consider that the model of the network is given by a symbolic transition system N (STS, see 2.2) that allows to mix control and data aspects.

Even if this is not the most general form of security policy, we assume that a security policy is given in terms of a safety property (or a conjunction of safety properties). This can be easily modelled by an STS Ω equipped with a stable set of "bad" locations (or equivalently with a boolean variable *bad*). This STS accepts runs (or sequences of behaviours) that violate the safety property. This allows to generalize automata à la Schneider [26].

The principle of attack generation is then to identify behaviours of the network model that violate the security policy. This can be done by analysis of the reachability of "bad" states in the STS $N \times \Omega$ which precisely identify executions of N violating the security property. The generation of attacks is then very similar to test generation for safety properties [13].

There are still some problems to solve. First reachability in $N \times \Omega$ is not decidable. However, one can resort to over-approximations (using e.g. abstract interpretation techniques). If "bad" states are unreachable in the over-approximation, this remains true in $N \times \Omega$, thus in the real system (with the assumption that N is an over-approximation). If some "bad" states are reachable in the over-approximation, one can try to check if these are real or spurious counter-examples in $N \times \Omega$. This can be done by refinement of the approximation in order to eliminate these counter-examples. If this fails (either there exists real counter-examples or the refinement did not allow to eliminate all counter-examples), as again we assumed that N is an over-approxmation of the real system, this does not prove that this correponds to a real attack of the network. One should then test these counter-examples on the real system in order to see if these correspond to real attacks or not.

5 Facts and Data on the project

5.1 Plenary Meetings held

Plenary meetings with all the partners have been held every 3 months alternatively in Grenoble and Rennes. Each plenary meeting is held on two days, with an overnight stay.

- Grenoble 15-16/9/2004
- Rennes 20-21/12/2004
- Grenoble 21-22/03/2005
- Rennes 16-17/06/2005
- Rennes 20-21/12/2005
- Grenoble 6-7/04/2006

Apart from those plenary meetings, many local working meetings have been held in Grenoble between LSR and Vérimag, and in Rennes between Distribcom, Lande and Vertecs. POTESTAT also participated (and presented) in the two annual gatherings of the ACI programme on Security: in Toulouse in 2004 and in Bordeaux (PariSTIC) in 2005.

5.2 Contributing students

We list here only students, and neither PhD nor Post-doc that have already been mentioned on the first page. In the first year of the project (2004-2005):

- Sébastien Boutonnet (INPG Telecom)
- Aldric Degorre (ENS Ker Lann)

In the second year (2005-2006):

- Mohammed Abdelmoula (M2R Grenoble)
- Ylies Falcone (M2R Grenoble)
- Jérémy Dubreil (internship ENST Bretagne, then INRIA supported PhD from 1/3/2006)

5.3 Publications

[1] V. Darmaillacq, J-C. Fernandez, R. Groz, L. Mounier, and J-L. Richier. Éléments de modélisation pour le test de politiques de sécurité. In *Colloque sur les RIsques et la Sécurité d'Internet et des Systèmes, CRiSIS*, Bourges, France, 2005.

[2] V. Darmaillacq, J-C. Fernandez, R. Groz, L. Mounier, and J-L. Richier. Test Generation for Network Security Rules. In *18th IFIP International Conference, TestCom 2006*, New York, May 2006. LNCS 3964, Springer.

[3] V. Darmaillacq and N. Stouls. Développement formel d'un moniteur détectant les violations de politiques de sécurité de réseaux. In *AFADL2006 - Approches Formelles dans l'Assistance au Développement de Logiciels*, Paris, March 2006.

[4] A. Degorre. Caractérisation de canaux cachés en logique temporelle alternante. Master's thesis, ENS Cachan, antenne de Bretagne, 2005.

[5] J. Dubreil. Non-interference on symbolic transition systems. Master's Thesis, Uppsala University, Enst Brest and Irisa, 2006.

[6] L. Hélouët. Finding covert channels in protocols with message sequence charts: the case of rmtp2. In *Proc. of SAM'2004, 4th conference on SDL and MSC*, June 2004.

[7] L. Hélouët, C. Jard, and M. Zeitoun. Covert channels detection in protocols using scenarios. In *Proceedings of SPV'2003, Workshop on Security Protocols Verification*, 2003. Satellite of CONCUR'03. Available at `http://www.loria.fr/~rusi/spv.pdf`.

[8] L. Hélouët, M. Zeitoun, and A. Degorre. Scenarios and covert channels, another game... In *Proc. Games in Design and Verification, GDV '04, Satellite of Computer Aided Verification, CAV'04*, volume 119 of *Electronic Notes in Theoretical Computer Science*, pages 93–116. Elsevier, 2004.

[9] T. Jéron, H. Marchand, and M-O. Cordier. Motifs de surveillance pour le diagnostics de systèmes à événements discrets finis. In *15e congrès francophone AFRIF-AFIA Reconnaissance des Formes et Intelligence Artificielle*, Tours, France, January 2006.

[10] T. Jéron, H. Marchand, S. Pinchinat, and M-O. Cordier. Supervision patterns in discrete event systems diagnosis. In *Workshop on Discrete Event Systems, WODES'06*, Ann-Arbor (MI, USA), July 2006. Also Published in DX'06 and Irisa research report No 1784.

[11] T. Jéron, H. Marchand, and V. Rusu. Symbolic determinisation of extended automata. In *4th IFIP International Conference on Theoretical Computer Science*, IFIP book series, Stantiago, Chile, August 2006. Springer Science and Business Media. Also Irisa research report No 1776.

[12] Gurvan Le Guernic and Thomas Jensen. Monitoring information flow. In Andrei Sabelfeld, editor, *Proceedings of the 2005 Workshop on Foundations of Computer Security*. DePaul University, June 2005. LICS'05 Affiliated Workshop.

[13] Vlad Rusu, Hervé Marchand, and Thierry Jéron. Automatic verification and conformance testing for validating safety properties of reactive systems. In John Fitzgerald, Andrzej Tarlecki, and Ian Hayes, editors, *Formal Methods 2005 (FM05)*, LNCS. Springer, July 2005.

References

[14] J.R. Abrial. *The B-Book*. Cambridge University Press, 1996.

[15] Eric Badouel, Marek Bednarczyk, Andrzej Borzyszkowski, Benoît Caillaud, and Philippe Darondeau. Concurrent secrets. Technical Report 5771, INRIA, Novembre 2005.

[16] D. E. Bell and L. J. LaPadula. Secure computer system: A mathematical model. Technical Report MTR-2547, Vol. 2, The MITRE Corporation, 1973.

[17] D. E. Bell and L. J. LaPadula. Secure computer system: Mathematical foundations. Technical Report MTR-2547, Vol. 1, The MITRE Corporation, 1973.

[18] Jeremy W. Bryans, Maciej Koutny, Peter Y. A. Ryan, and Laurent Mazare. Opacity generalised to transition systems. In *Proceedings of Formal Aspects of Security and Trust (FAST'05)*, 2005.

[19] E. S. Cohen. Information transmission in computational systems. *ACM SIGOPS Operating Systems Review*, 11(5):133–139, 1977.

[20] Frédéric Cuppens, Nora Cuppens-Boulahia, and Thierry Sans. Nomad: A security model with non atomic actions and deadlines. In *18th IEEE Computer Security Foundations Workshop, (CSFW-18 2005)*, pages 186–196, Aix-en-Provence, France, 2005.

[21] Joseph A. Goguen and José Meseguer. Security policies and security models. In *Proc. IEEE Symp. Security and Privacy*, pages 11–20, 1982.

[22] R.A Kemerrer. Shared ressources matrix methodology: an approach to indentifying storage and timing channels. *ACM transactions on Computer systems*, pages 256–277, 1983.

[23] Jay Ligatti, Lujo Bauer, and David Walker. Enforcing non-safety security policies with program monitors. In Sabrina De Capitani di Vimercati, Paul F. Syverson, and Dieter Gollmann, editors, *ESORICS*, volume 3679 of *Lecture Notes in Computer Science*, pages 355–373. Springer, 2005.

[24] G. Lowe. Quantifying information flow. In *IEEE Computer Security Foundations Workshop*, pages 18–31, June 2002.

[25] Andrei Sabelfeld and Andrew C. Myers. Language-based information-flow security. *IEEE J. Selected Areas in Communications*, 21(1):5–19, January 2003.

[26] Fred B. Schneider. Enforceable security policies. *ACM Trans. Inf. Syst. Secur.*, 3(1):30–50, 2000.

[27] Tachio Terauchi and Alexander Aiken. Secure information flow as a safety problem. In Chris Hankin and Igor Siveroni, editors, *SAS*, volume 3672 of *Lecture Notes in Computer Science*, pages 352–367. Springer, 2005.