An Approach for Deploying and Monitoring Dynamic Security Policies

Jose-Miguel Horcas[1], Mónica Pinto[1], Lidia Fuentes[1], Wissam Mallouli[2], and Edgardo Montes de Oca[2]

[1] CAOSD Group, Universidad de Málaga, Andalucía Tech, Spain
{horcas, pinto, lff}@lcc.uma.es,
[2] Montimage, 39 rue Bobillot Paris 75013, France
{wissam.mallouli,edgardo.montesdeoca}@montimage.com

Abstract. Security policies are enforced through the deployment of certain security functionalities within the applications. When the security policies dynamically change, the associated security functionalities currently deployed within the applications must be adapted at runtime in order to enforce the new security policies. INTER-TRUST is a framework for the specification, negotiation, deployment and dynamic adaptation of interoperable security policies, in the context of pervasive systems where devices are constantly exchanging critical information through the network. The dynamic adaptation of the security policies at runtime is addressed using Aspect-Oriented Programming (AOP) that allows enforcing security requirements by dynamically weaving security aspects into the applications. However, a mechanism to guarantee the correct adaptation of the functionality that enforces the changing security policies is needed. In this paper, we present an approach based on the combination of monitoring and detection techniques in order to maintain the correlation between the security policies and the associated functionality deployed using AOP, allowing the INTER-TRUST framework to automatically react when needed.

Keywords: aspect-oriented programming, dynamic deployment, monitoring, security framework, security policies

1 Introduction

Future Internet (FI) systems encompass a set of pervasive computing devices (e.g., smartphones, tablets, vehicles, wearables, etc.) always connected to the Internet and continuously exchanging information with remote entities [1]. There are FI applications in different domains, such as smart cities, smart roads and smart homes, among others. Examples of these applications are monitoring the availability of parking spaces in a city [2], the tracking of vehicles and pedestrian levels to optimize driving and walking routes [3], or intelligent roads with warning messages or dynamic speed limits according to climate conditions [4, 5].

In order to ensure that the exchange of information is performed securely, the development of such systems requires a set of security mechanisms to be

conceived. These mechanisms are able to protect the system against different threats that may arise. For instance, in an Intelligent Transportation System (ITS), the communication is required to be secure since the transmitted information between the parties involved (vehicles and road infrastructures) may be critical in maintaining the safety of the vehicle's drivers/occupants.

As an example, let us consider an ITS application that dynamically recommends the speed limits of a road according to climate conditions and to unexpected events like accidents or traffic jams. This is done by collecting the information sent by both the vehicle's sensors (e.g., geolocation, proximity to other vehicles, current speed) and the road side sensors (e.g., weather conditions, traffic status). Then, using this information the new recommended speed limit is calculated and notified to the driver on his On Board Unit (OBU). Some of the security requirements that could be taken into account in the development of this application are: (1) the *user anonymity* must be ensured, otherwise, some users will not agree to send their current speed and location; (2) only *authorized users* subscribed to the service can send information to the ITS server and receive recommendations. Otherwise, malicious users may send fake information about the road status or about the weather conditions that could cause accidents, and (3) in some contexts (e.g., when a police car is pursuing an infractor) all the information sent by the police car should be *cyphered* in order to hide the information from the infractors.

The main problematic of enabling security in FI systems is the heterogeneity and dynamicity of the security policies that determine how the different parties need to interact with each other. First, the security policies can be heterogeneous because each user can customize his own security policies, which answer their security constraints. Moreover, these users' security policies can also be different from the security policies expected by the applications. Besides, the security policies can be dynamic and change over time to adapt to new requirements, new regulatory rules or new application contexts, for instance, moving from one country to another. In this context, existing approaches and techniques to design and development secure interoperable FI systems present several problems that prevent their correct application in security applicative domains [6, 7]. Firstly, there is a lack of sufficiently rich techniques to tackle the integration of all the tasks that cover security policy modeling, interoperability, deployment, enforcement and supervision. Moreover, focusing on dynamic security enforcement, there is also a lack of solutions that allow the dynamic adaptation of security to new application requirements and changes in the environment.

In order to solve these problems, the Inter-operable Trust Assurance Infrastructure (INTER-TRUST) framework [8] aims to deal with the problematic of enabling security in heterogeneous and pervasive systems, modeling secure interoperability policies with different constraints, and enabling the dynamic and secure establishment of trusted relationships between systems [9]. The main contributions of the INTER-TRUST framework are the *dynamic specification of security policies*, the *dynamic deployment of security policies*, the *dynamic monitoring of security policies* and the *fuzz and active testing of security policies*. In

this paper we focus on the second and third contributions. The dynamic deployment of security policies is performed by using one of the most used enhanced deployment mechanisms to inject dynamic behavior: Aspect Oriented Programming (AOP) [10]. AOP allows enforcing security requirements by dynamically weaving security aspects into applications, changing their configuration and behavior so that they respect the evolving requirements. AOP is used to add/implement security concerns (i.e., anonymity, availability, authentication, access control, integrity, encryption, enrolment, etc.) to application components at runtime so that applications can dynamically adapt their behavior for required/negotiated security policies. However, dynamic adaptation techniques can introduce new vulnerabilities and security risks [6, 11, 12]. Among other reasons [13], the industry is reluctant to adopt any of these dynamic adaptation mechanisms since hackers can exploit security gaps which are inherent to dynamic solutions. The most common technique for ensuring that a system under execution is not being altered is to monitor and test the behavior exhibited by the system. Concretely, INTER-TRUST incorporates dynamic monitoring and testing techniques to obtain enriched information of the system's execution, which is used to verify the conformity with the implementations, ensuring a secure interoperability between systems.

In this paper, we present a dynamic aspect-oriented approach for the deployment and monitoring of security policies. The approach maintains the correlation between the security policies negotiated between the communicating parties, the security aspects dynamically deployed into the applications in order to enforce those security policies, and the security properties that are monitored by the system. This is done by detecting contextual changes in the environment and by reacting to those changes once it is identified that the communicating parties are not respecting the negotiated security policies. The approach is independent from the security language used to define the security policies, and also from the AOP language used to implement the security functionalities as aspects, and thus, can be adapted to use with others security models and AOP languages/frameworks. The dynamic monitoring of the security policies allows FI applications to have a global understanding of the changes performed at runtime and can automatically react to new risk or threats that may arise. This approach represents a generic solution that can be applied to many types of pervasive applications. In this paper, our approach has been integrated as part of the INTER-TRUST framework. The INTER-TRUST framework is intended to be generic and to target different fields and domains. In this context, the framework has been successfully integrated in two real case studies with different security requirements: the ITS case study that we use throughout this paper and an online electronic voting (e-voting) case study.

The rest of the paper is organized as follows. Section 2 discusses the limitation of the related work. Section 3 briefly describes the software architecture of the INTER-TRUST framework. Section 4 explains the correlation between the security policies, the aspects, and the security properties. In Section 5 we present our approach to deploy the security policies and monitor that correlation. Sec-

tion 6 evaluates and discusses our approach. Finally, Section 7 concludes the paper and presents our future work.

2 Related Work

This section overviews existing work related to the dynamic adaptation and monitoring of security policies. First, we focus on security frameworks that deal with the dynamic deployment of security policies. Then, we discuss the advantages and limitations of the dynamic adaptation techniques, and in particular, AOP. Finally, we outline different techniques for the dynamic detection of vulnerabilities.

2.1 Security frameworks

The analysis of existing research work and standards in the domain of FI and pervasive systems reveals a common problem: the inexistence of a proper security framework to secure the communications flexibly and efficiently [7, 6]. In [7], the authors propose a framework for specifying, deploying and testing access control policies independently of the security model. For the specification phase, they define a generic meta-model for rule-based security formalisms. Then, following a model-driven approach, the generic meta-model is transformed into security policies for the eXtensible Access Control Markup Language (XACML) platform [14] by using the appropriate profile of the security model (e.g., OrBAC). The security test is done by applying mutation testing [15] — i.e., the security tests are qualified if they are able to detect any elementary modification in the security policy of the application (mutants). The main drawback to this approach is that the generic meta-model only supports access control policies, and thus, it is not possible to specify and deploy other security concerns such as integrity, encryption, or non-repudiation, as we propose in this paper with the INTER-TRUST framework. Moreover, although they can use any security model language, it requires the policy formalism (e.g., OrBAC) to be defined as an instance of the generic meta-model, restricting the expressiveness and versatility of the original model; in addition, the mutation operators for security policy testing are defined in terms of the concepts presented in the generic meta-model, complicating the specialization of the security tests. On the other hand, the modular architecture of the INTER-TRUST framework (see Section 3) allows decoupling the security functionality from the formalism used to specify the security policies, and also from the test procedures. Although the INTER-TRUST framework is based on the OrBAC model to specify the security policies, our approach to deploy the security policies and monitor the correlation between the security policies, the aspects, and the security properties is independent from the security model. Therefore, our approach can be adapted to use with others security models — e.g., Modular Access Control (ModAC) [16], or Ponder Policy Specification Language [17], by defining the relation between the security policies and the security functionalities to be deployed and the security properties

to be monitored. For instance, the use of ModAC instead of OrBAC facilitates the encapsulation of the access control functionality as an aspect [18]. However, obligation rules provided by OrBAC allow specifying requirements for any kind of functionality (e.g., encryption, integrity), not only access control functionality.

In [6], the authors present an Aspect Oriented Permission System (AOPS) for runtime policy enforcement. The policy decisions are based on the execution History-Based Access Control (HBAC) model [19] and implemented in AspectJ following the Java permissions model but applied to AOP. Only security vulnerabilities related to access control permissions are considered (e.g., restricted rights to read and modify attributes of the base system by the aspects). Also, the approach assumes that the weaver as well as the execution environment are trusted, and that the weaver protects against scenarios in which untrusted aspects are incorrectly woven into the application code. The INTER-TRUST framework, in contrast, relies on dynamic deployment of security functionalities (e.g., encryption, digital signature, authentication) implemented as aspects that enforce the security policies after a negotiation phase. The application and the aspects are continuously monitored to guarantee the enforcement of the security policies by the aspects and to detect potential vulnerabilities and attacks.

2.2 Dynamic adaptation with Aspect-Oriented Programming

The dynamicity offered by AOP is similar to that offered by other dynamic adaptation techniques (e.g., interceptors in component-based application servers). AOP, however, offers a more flexible solution since applications do not necessarily need to be developed conforming to a particular component model or run on a particular application server container (e.g., JavaEE or Spring) in order to be adapted at runtime.

Vulnerabilities introduced by the dynamic adaptation of the applications are well-known and have been identified during the development activity [20, 11, 21, 22]. In [20], the authors present bug patterns in AspectJ and illustrate the symptoms of the patterns through examples. The security risks in using AOP to develop secure software are analyzed in [11] from a programming level point of view. An aspect permission system is also proposed to address some of the issues identified (e.g., parameter alteration, invocation hijacking, use of privileged aspects, etc.). In [21], the authors use a combination of static code analysis and protection code generation during the development phase. They focus on security vulnerabilities caused by missing input validation — i.e., the process of validating all the inputs for an application before using it. They analyze the source code and/or binary code without executing it and identify anti-patterns that lead to security bugs. The unexpected vulnerabilities that the dynamic weaving may introduce when the aspects are woven at runtime cannot be covered with the static analysis. In [22], aspect orientation is used to monitor the information flows between objects in a system for the purpose of detecting misuse. That is, identifying behavior that is close to some previously defined pattern signature of a known intrusion. The problem with misuse-based detections is that the

anomalies must be known in advance and cannot detect new vulnerabilities at runtime.

Finally, although AOP, as well as other existing dynamic adaptation mechanism (e.g., interceptors, component-based adaptation, reflection, etc.), may introduce vulnerabilities in the system, some papers support the idea that AOP can be a good technology for developing secure applications [12, 23, 24].

2.3 Dynamic detection of vulnerabilities

There are several techniques to perform dynamic detection of vulnerabilities. While monitoring (or passive testing) consists in observing, at runtime, whether the application behavior conforms to a set of formal properties, active testing [25] validates the application implementation by applying a set of test cases and analyzing its reaction. Fuzz testing [26] is also used in the INTER-TRUST framework. It consists in stimulating the application using random or mutated inputs in order to detect unwanted behavior, crashing, or security violation. However, fuzz and active testing techniques are applied at the testing phase, and thus, it is not possible to use them during the normal operation of the systems to detect the AOP vulnerabilities when aspects are deployed at runtime. In the INTER-TRUST framework, fuzz and active testing techniques are complemented with the monitoring techniques presented in this paper in order to help detect the vulnerabilities in the testing phase.

Our approach uses MMT-security properties [27] to formally specify security goals and attack behaviors related to the application or protocol under test. The originality of the MMT-security properties with respect to existing intrusion detection techniques, like for instance SNORT [28] and BRO [29], lies in that they are not based on just pattern matching (i.e., signatures) as in SNORT nor do they require writing executable scripts as in BRO. Instead, they allow a more abstract description of sequences of events that can represent normal/abnormal behavior. They can also integrate pattern matching, statistics and machine learning techniques; but describing this here is beyond the scope of this paper.

3 Background Information: The INTER-TRUST framework

The INTER-TRUST framework is a dynamic security framework that has been designed and implemented by the INTER-TRUST European Project consortium [8] to support trustworthy applications based on the negotiation, enforcement and dynamic adaptation of security policies. Figure 1 overviews the modular architecture of the INTER-TRUST framework, whose functionality is divided into four key blocks:

1. **Dynamic Specification of Security Policies**. The first step when using the INTER-TRUST framework is to specify the application's security policies. In INTER-TRUST, security policies rely on the OrBAC model [30]

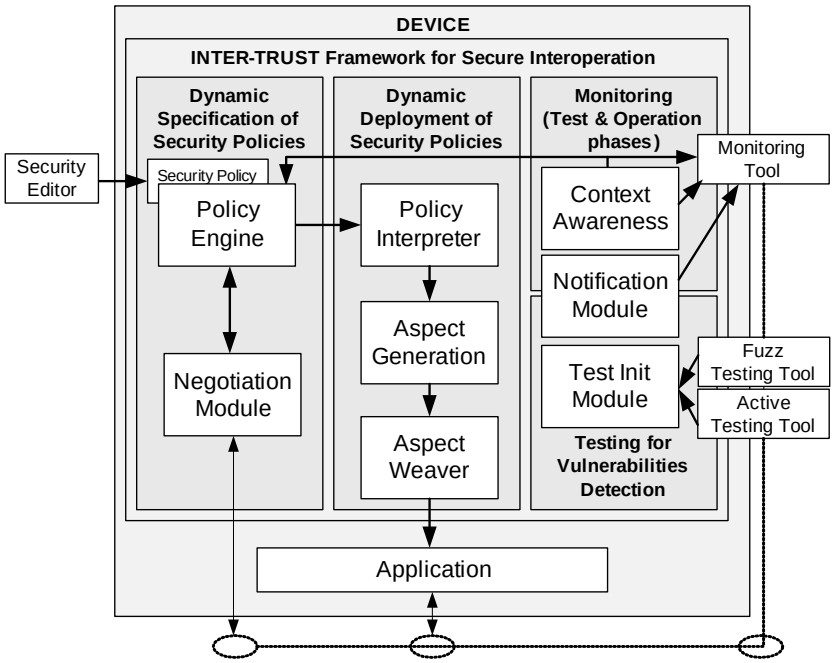

Fig. 1. Architecture of the INTER-TRUST framework.

and are first specified using a `Security Editor` (e.g., MotOrBac [31]) and then negotiated between the different parties in a communication using a `Negotiation` module (e.g., a vehicle and an ITS server in the context of a Vehicle-to-Infrastructure communication). Security policies support interoperability that includes: access control requirements, permissions and prohibitions, usage control requirements, obligations to respect, and delegation rules. It may also correspond to complex requirements or comprise temporal deadline conditions that specify what happens in the case of violation of any of the contracts (e.g., sanctions that are triggered when a violation is detected).

When the security requirements change at runtime the security policies are (re)negotiated. For instance, let's suppose that a user is not allowed to register in an ITS application because the provider requires different authentication credentials that the ones specified in the user's security policy. Since the user wants to register, the INTER-TRUST framework gives him the opportunity to renegotiate his security policy.

2. **Dynamic Deployment of Security Policies**. The negotiated security policies are analyzed and interpreted by the `Policy Engine` and the `Policy Interpreter` modules. These modules are responsible for identifying changes in the security policies that require the security concerns deployed inside the application to be adapted. Security policies are dynamically deployed,

and/or adapted at runtime using the `Aspect Generation` and the `Aspect Weaver` modules, which are in charge of receiving the information generated by the `Policy Interpreter` module and of incorporating or eliminating the corresponding security aspects in the application. Security aspects can be developed in any Java-based AOP language such as AspectJ, Spring AOP, CaesarJ, or JBoss. The detailed design of these modules is beyond the scope of this paper but can be consulted in Figure 6 of Appendix A.

3. **Monitoring (Test & Operation phases)**. Negotiated security policies are also sent to the `Monitoring Tool` in order to activate/deactivate the associated security properties that control the fulfillment of the security policies by the deployed aspects. Security properties are formally described as conditions in sequences of events [27] based on Linear Temporal Logic (LTL) [32] to define security rules (i.e., rules that should be respected) or attacks and misbehaviors [33]. The `Monitoring Tool` relies on an adaptation of the Montimage Monitoring Tool (MMT) [34] which is an online monitoring solution that allows a real-time network traffic, application, flow and user level visibility to be provided. The `Notification` and `Context Awareness` modules notify the `Monitoring Tool` about application's internal events and changes in the application context — e.g. network packets, battery of the device, CPU consumption, etc. The detailed design of the MTT tool is beyond the scope of this paper but can be consulted in Figure 7 of Appendix B.
4. **Testing for Vulnerabilities Detection**. Different fuzz [26] and active [25] testing techniques are also provided as part of the framework (`Fuzz Testing Tool` and `Active Testing Tool` modules) in order to test the application's security and robustness. In order to do that, during the testing phase the MMT tool monitors the traces automatically generated by the fuzz testing and active testing tools in order to simulate the application behavior.

Note that the INTER-TRUST framework is a modular solution with complementary functionalities that can be partially or completely deployed according to the developer's needs. This modular framework has several benefits for the following reasons: (1) the modules in charge of specifying, negotiating and interpreting the security policies are unaware of the use of AOP to enforce them. This allows the use of these modules in other contexts where other dynamic mechanisms (e.g., interceptors, reflection, component-based adaptation) may be used to deploy the security policies; (2) the modules in charge of weaving the aspects are unaware of the format used to specify the security policy, relying only on the format of the interpreted security policies. This decouples the aspects from the formalism used to specify the security policies and also allows using these modules in other contexts with other modules and languages; (3) the `MMT` tool is extensible and can support new analysis algorithms and definitions for parsing network packets and message formats [34]; (4) it is a distributed architecture where modules are decoupled among them and interact asynchronously by mean of a standard queue protocol (e.g., AMQP [35]), (5) the framework is extensible to use additional AOP weavers and only the `Aspect Weaver` module is affected. The rest of the modules can be reused as they are, and (6) the use of aspects

improve the separation between applications and security concerns. Thus, the applications can choose between using the INTER-TRUST framework or any other security solution and the change is not invasive.

In this paper, we focus on the dynamic deployment of the security policies (block 2 in Figure 1) and on the monitoring phase (block 3 in Figure 1), while the details of the dynamic specification of security policies [9, 36] and the testing phases [37, 38] are beyond the scope of this paper. The next section presents the correlation that must be maintained between the security policies, the aspects, and the security properties.

4 Correlation between Security Policies, Aspects and Security Properties

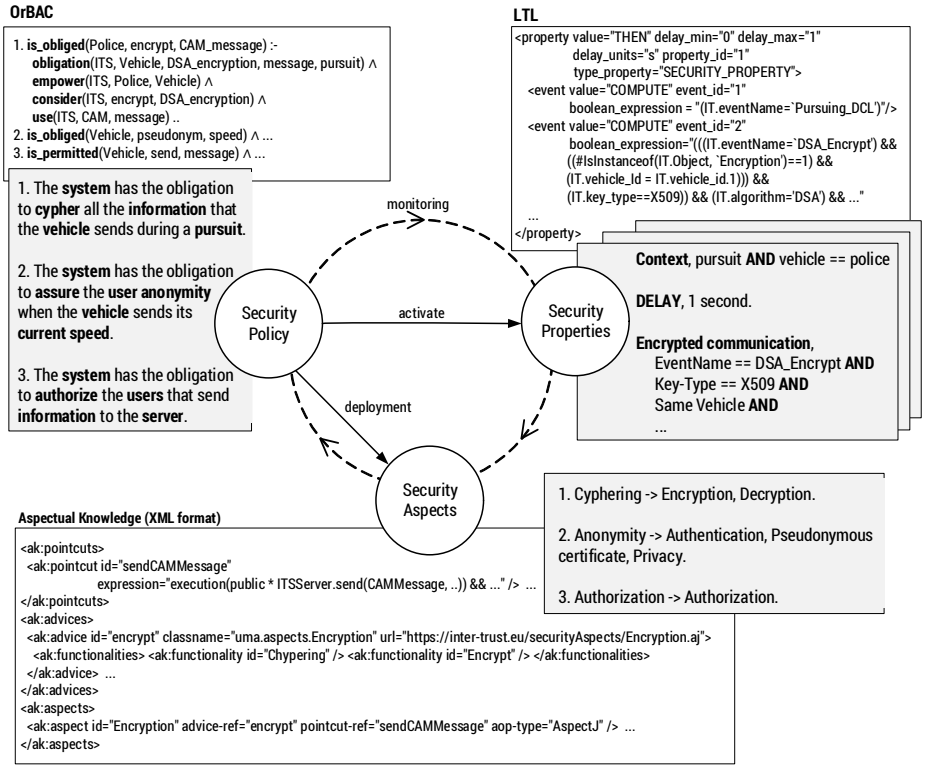

Fig. 2. Correlation of the security policies, the aspects, and the security properties.

The correct enforcement and dynamic adaptation of the security policies is based on two cornerstones (see Figure 2). The first is the correlation defined

between the *security policies* that need to be enforced, the *security aspects* that are deployed/undeployed in order to enforce those security policies and the *security properties* that are activated/deactivated in order to check whether or not the system is behaving according to the specified security policies. The second is the monitoring at runtime of this correlation in order to detect any attack that breaks it. These attacks could occur due to different kinds of security vulnerabilities (e.g., an attacker could send a huge number of legitimate requests to a server to monopolize its resources), or due to those vulnerabilities that are introduced by the dynamic deployment mechanism itself (e.g., a malicious aspect).

Let us illustrate the correlation with an example scenario of the ITS case study: a police vehicle communicates with the ITS central station by sending/receiving Cooperative Awareness Messages (CAMs). CAM messages contain a set of parameters describing the vehicle's status, among other information. When the police vehicle receives an emergency call and starts pursuing another vehicle, a context change is detected. As a result of the change in the context (pursuit context) a new security policy is negotiated between the police vehicle and the ITS central station. One of the statement of the new security policy is that the police vehicle has to encrypt the CAM messages to avoid the pursued vehicle to know its location, while still informing the ITS central station about its location for this to regulate traffic and facilitate the pursuit (rule 1 in the security policy of Figure 2). The required functionality (i.e., cyphering) must be deployed inside the application in order to encrypt the messages from the police vehicle and decrypt them into the ITS central station (`Encryption` and `Decryption` aspects of Figure 2). Also, a new security property is also activated in the monitoring tool to check that the encryption rule specified in the security policy is respected by the application when the new functionality is added.

In Figure 2 we have shown an example of the security property that needs to be verified to ensure that the messages are correctly cyphered. Also, for each rule in the security policy, a set of aspects that fulfill the required functionality are deployed inside the application. For instance, the encryption and decryption aspects are deployed to cypher the messages, the authentication, privacy and pseudonymous certificate aspects are deployed to ensure the user anonymity, and the authorization aspect are deployed to provide user authorization. The *aspectual knowledge* depicted in Figure 2 contains the functionality provided by the aspects for each security policy and the join points where the aspects can be deployed. Finally, the application with the aspects is monitored and the captured traces are sent to the monitoring tool that correlates the deployment of the aspects with the security properties. Note that this correlation must be maintained, both when the user joins the application for the first time (i.e., after the deployment of the initial security policies) and also at runtime, when the security policies are dynamically negotiated and adapted.

The *dynamic deployment of security policies* and the *dynamic monitoring of security policies* blocks of the INTER-TRUST framework implement the correlation described. The next section explains our approach to deploy the security policies and monitor the correlation.

5 Deployment and Monitoring Approach

Figure 3 provides a more detailed description of the dynamic deployment of security policies (activities labeled 1, 2, and 3) and the monitoring mechanism to maintain the correlation between the security policies, the security aspects, and the security properties (activities labeled 4, 5, and 6).

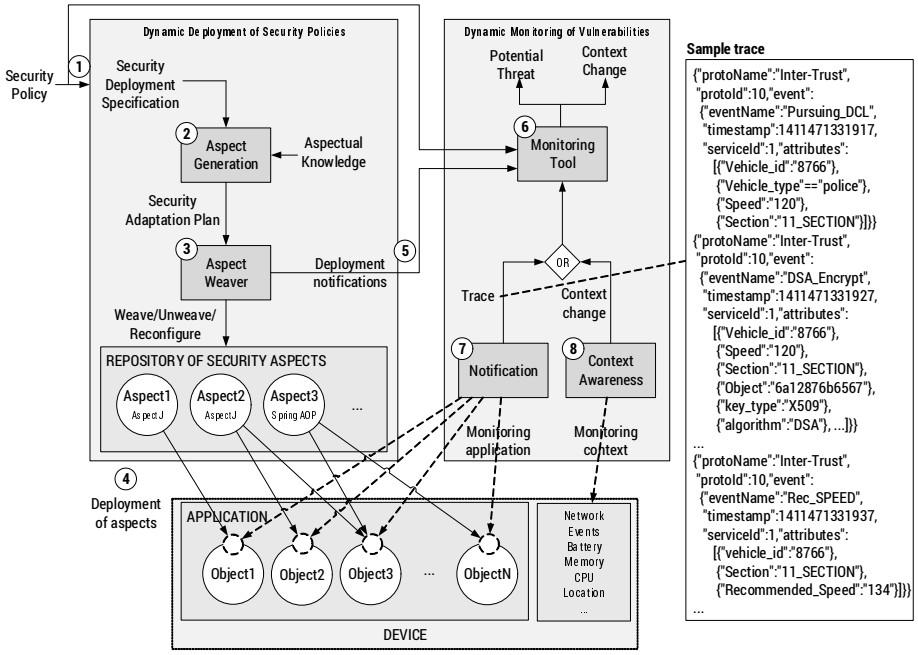

Fig. 3. Our approach for deploying and monitoring security policies.

5.1 Dynamic Deployment of Security Policies

When a security policy needs to be deployed inside the application at runtime (activity labeled 1 in Figure 3) — e.g., due either to the initial deployment or to a (re)negotiation of the security policy, the new security policy is sent to the modules of the framework in charge of: (i) the `Dynamic Deployment of Security Policies`, which will deploy/undeploy/reconfigure the aspects, and (ii) the `Dynamic Monitoring of Vulnerabilities`, which will activate/deactivate the corresponding security properties. In order to deploy the security policy, the `Aspect Generation` module receives a *security deployment specification* (activity labeled 2) that is the result of interpreting the security policy and contains the list of security aspects that must be deployed (woven), undeployed (unwoven), and reconfigured (i.e., changing the configuration parameters such as the

Listing 1.1. Aspectual Knowledge.

```
1 <ak:pointcuts>
2   <ak:pointcut id="sendCAMMessage" expression="execution(public *
        ITSServer.send(CAMMessage, ..)) && this(Vehicle) && args(message)"
        />
3   <ak:pointcut id="receiveCAMMessage" expression="execution(public *
        ITSServer.receive(CAMMessage, Vehicle, ..)) && args(message)" />
4   ...
5 </ak:pointcuts>
6 <ak:advices>
7   <ak:advice id="encrypt" classname="uma.aspects.Encryption" url="https:
        //inter-trust.eu/securityAspects/Encryption.aj"><
        ak:functionalities>
8     <ak:functionality id="Chypering" />
9     <ak:functionality id="Encrypt" />
10  </ak:functionalities></ak:advice>
11  <ak:advice id="decrypt" classname="uma.aspects.Encryption" url="https:
        //inter-trust.eu/securityAspects/Encryption.aj"><
        ak:functionalities>
12    <ak:functionality id="Chypering" />
13    <ak:functionality id="Decrypt" />
14  </ak:functionalities></ak:advice>
15  ...
16 </ak:advices>
17 <ak:aspects>
18    <ak:aspect id="Encryption" advice-ref="encrypt" pointcut-ref="
          sendCAMMessage" aop-type="AspectJ" />
19    <ak:aspect id="Decryption" advice-ref="decrypt" pointcut-ref="
          receiveCAMMessage" aop-type="AspectJ" />
20  ...
21 </ak:aspects>
```

digital certificate in an authentication aspect) within the application to enforce the new security policy. The `Aspect Generation` module also contains the required *aspectual knowledge* that encompasses the list of aspects available in the aspect repository of the framework.

Listing 1.1 excerpts the part of the aspectual knowledge related to the encryption and decryption aspect. For each aspect, the aspectual knowledge includes: (1) the provided advice with the functionality that it implements, and (2) the associated pointcut where the aspect will be incorporated into the application. For instance, the encryption aspect (line 18) has associated an advice (line 7) with the functionality to encrypt messages (lines 8 and 9) and also the pointcut (line 2) where messages will be encrypted. Similarly, the decryption aspect (line 19) has associated with it, an advice (line 11) to decrypt messages (lines 12 y 13) and the associated pointcut (line 3) that indicates where messages will be decrypted. Both advice are implemented in the same AspectJ file (see Listing 1.3) of the INTER-TRUST aspect repository. This is indicated by the *url* attribute (lines 7 and 11) defined for each advice.

The `Aspect Generation` module performs a mapping between the required security functionalities and the aspects that provide these functionalities. The output of this mapping is a new configuration that is analyzed to: (1) obtain the differences between the new and the current configurations of the aspects deployed within the application, and (2) generate a *security adaptation plan* with

Listing 1.2. Security Adaptation Plan.

```
1 <sap:ADD>
2   <sap:aspect id="Encryption" ... />
3   <sap:aspect id="Decryption" ... />
4   ...
5 </sap:ADD>
6 <sap:REMOVE>
7   ...
8 </sap:REMOVE>
9 <sap:RECONFIGURE>
10  <sap:aspect id="Encryption" ><sap:configuration>
11    <sap:parameter name="algorithm">DSA</sap:parameter>
12    <sap:parameter name="key">
13      CN="VEHICLE_1", OU="ITS-VEHICLES", O="DIRECCION GENERAL DE TRAFICO
            ", C="ES"</sap:parameter>
14    <sap:parameter name="key-type">X509</sap:parameter>
15    ...
16  </sap:configuration></sap:aspect>
17  ...
18 </sap:RECONFIGURE>
```

the list of actions that must be performed over the aspects: weave, unweave, or reconfigure. Listing 1.2 shows an example of a security adaptation plan where we indicate that the encryption and decryption aspects need to be deployed inside the application (lines 1 to 5) and configured with parameters such as the encryption algorithm or the key values (lines 9 to 18). Also, we indicate the aspects that need to be undeployed (lines 6 to 8).

The security adaptation plan generated by the `Aspect Generation` module is sent to the `Aspect Weaver` module that is in charge of executing the actions by interacting directly with the aspects (activity labeled 3 in Figure 3). The `Aspect Weaver` module is a wrapper that translates the list of actions received as input (which is specified independently of a particular AOP language/framework) to the particular syntax of the AOP weaver being used. This means that we provide different instantiations of the `Aspect Weaver` module for using different AOP weavers, since the use of a unique AOP solution does not cover all the dynamicity, expressiveness, and performance requirements that the applications may need. For instance, AspectJ allows dynamically activating/deactivating at runtime the aspects woven when deploying the application, while Spring AOP allows directly weaving/unweaving new aspects at runtime. However, Spring AOP only supports interceptions of method executions, in contrast to AspectJ that can intercept any point in the execution of the application (e.g., calls and executions of methods and constructors, references and assignments of fields, handler of exceptions, etc.).

Listing 1.3 shows an example of an encryption aspect using the AspectJ language. The aspect defines two main pointcuts: encrypt (line 5) and decrypt (line 6). Each pointcut defines the points where the messages will be encrypted (line 2) or decrypted (line 3). To control the activation of the pointcuts we use the `if()` pointcut constructor that AspectJ provides to define a conditional pointcut expression which will be evaluated at runtime for each candidate join

Listing 1.3. Encryption aspect in AspectJ language.

```
1 public aspect Encryption {
2   pointcut sendCAMMessage(CAMMessage message): execution(public *
        ITSServer.send(CAMMessage, ..)) && this(Vehicle) && args(message);
3   pointcut receiveCAMMessage(CAMMessage message): execution(public *
        ITSServer.receive(CAMMessage, Vehicle, ..)) && args(message);
4
5   pointcut encrypt(CAMMessage m): if(AspectsStatus.isEnabled("ENCRYPT"))
        && sendCAMMessage(m);
6   pointcut decrypt(CAMMessage m): if(AspectsStatus.isEnabled("DECRYPT"))
        && receiveCAMMessage(m);
7
8   Object around(CAMMessage m): encrypt(m) {
9   ChyperingModule chyper = new ChyperingModule(AspectsStatus.getParams("
        ENCRYPT"));
10  CAMMessage chyperedMessage = chyper.encrypt(m);
11    proceed(chyperedMessage);
12  }
13
14  Object around(CAMMessage m): decrypt(m) {
15  ChyperingModule chyper = new ChyperingModule(AspectsStatus.getParams("
        DECRYPT"));
16  CAMMessage clearMessage = chyper.decrypt(m);
17    proceed(clearMessage);
18  }
19 }
```

point[3]. This mechanism increases the degree of dynamicity by coding patterns that can dynamically support enabling and disabling advice in aspects [39]. In our example, the `AspectsStatus` class contains the configurations and status (enabled/disabled) of the aspects that are changed at runtime by the `Aspect Weaver` module. The aspect defines two advice associated with the encrypt and decrypt pointcuts: one for encrypting (line 8) and one for decrypting (line 14) CAM messages. The advice use a `CypheringModule` object that provides the functionality for encryption and decryption and is configured with the algorithm and parameters indicated in the `AspectsStatus` class (lines 9 and 15).

Once the aspects have been adapted, the `Aspect Weaver` module notifies the `Monitoring Tool` in order to inform about the status of the deployment (activity labeled 5 in Figure 3). That is, to notify whether or not the deployment was successfully carried out and which aspects were deployed/deployed/reconfigured.

5.2 Dynamic Monitoring of Security Policies

In order to maintain the correlation between the security policies, the aspects, and the security properties, the application and the aspects are monitored at runtime by the `Notification` module. The `Notification` module reports the application's internal events (e.g., traces with state changes, error conditions, timestamps, method status, etc.) to a monitoring server (the `Monitoring Tool`) (activity labeled 7 in Figure 3). To operate at runtime, the `Notification` module is introduced into the target application as an aspect in the instantiation phase.

[3] `http://eclipse.org/aspectj/doc/released/progguide/index.html`

The target source code is annotated, using standard Java annotations, to specify the measurement points (or meters) that generate the monitored data. These annotations are also incorporated using AOP without manually modifying the source code of the application — e.g., using the *declare annotation* of AspectJ[4]).

While the target application is operating, the `Notification` module produces a stream of log messages. Measurement points can be attached to classes, methods and attributes, and work on two different levels of scope: local and recursive. Meters operating at the local scope level are always marked by an annotation. Only annotated elements are effected by local scope meters (e.g., calls to nested methods are not tracked). In the next scope level, recursive monitoring, beside the annotated code, all code reachable through control flow is monitored, up to the available call depth. Recursive monitoring may cause a significant performance overhead, so this kind of monitoring should be used by annotating only relevant data for security analysis. Call depth is limited by the available source code, because static aspects operate by modifying accessible source code. The instrumentation therefore does not penetrate pre-compiled classes, such as .class files or system libraries. Table 1 summarizes the monitoring annotations that the `Notification` module supports. Table 1 shows for each annotation the scope (local or recursive), the location or context of the annotation (i.e., before class, method, or attribute declarations), the meaning of the annotation, and the output information that is provided. The output is made up of key-value pairs. The `Notification` module appends status information to each logged event. Apart from the specific output information of each annotation, the status information string contains the following generic data:

- Monitored object name, which can be a class name, method name or attribute name.
- Source file name and source line number.
- Thread id of current thread.
- Total number of threads.
- Total number of tracked objects.
- Time stamp.

Furthermore, the `Context Awareness` module notifies the `Monitoring Tool` but, in contrast to the `Notification` module, the `Context Awareness` monitors changes in the environment (activity labeled 8 in Figure 3) — i.e., contextual changes that are external to the application such as packets over the communication network, battery status of the device, CPU consumption, etc. Both traces and context changes are sent to the `Monitoring Tool` that interprets them (activity labeled 8 in Figure 3) so it can react to changes or adapt the security rules with the negotiation of a new security policy.

The right-hand side of Figure 3 shows an excerpt of a sample trace received by the `Monitoring Tool` with three events generated from the `Notification` module. For instance, the first event (event with name `Pursuing_DCL`) provides

[4] `https://eclipse.org/aspectj/doc/next/adk15notebook/annotations-declare.html`

Table 1. Monitoring annotations.

Annotation	Scope	Context	Description and Output
@Monitor	Local	Method	Measures the execution time and monitors its exit status. Output: Start event, end event, duration, exit status, status information.
@Monitor	Local	Attribute	Captures changes made to attributes and logs the new value. Output: Attribute value on each assignment operation
@Monitor	Local	Class	Combines method and attribute monitoring applied to all methods and attributes of the class. Output: Combines the method and attribute monitoring output.
@Count	Local	Class	Tracks the number of objects on the memory heap. Output: Global object counter.
@Ping	Local	Method	Generates a signal whenever a function is called. Output: A 'Ping' message is added to the log on each invocation.
@Taint	Recursive	Method	Tracks the control flow of methods. Output: Same as method monitoring for each method encountered by the control flow.
@Exclude	Recursive	Method, attribute, class	Methods, attributes, and classes are excluded from the monitoring scope. Output: 'Negative output', curtails output of impacted code.

Listing 1.4. MMT-security property.

```
<property value="THEN" delay_min="0" delay_max="1" delay_units="s"
    property_id="1" type_property="SECURITY_PROPERTY"
          description="In a pursuit context, CAM messages communicated
              by a vehicle must be encrypted.">
    <event value="COMPUTE" event_id="1"
    description="A pursuit context is detected. This is done by the
        reception of an event declared by the police vehicle by pushing
        a specific button."
    boolean_expression = "(IT.eventName='Pursuing_DCL')"/>
    <event value="COMPUTE" event_id="2"
          description="CAM messages are encrypted. The encryption
              algorithm used is DSA. The type of the key is X509, using
              SHA1 as hashing algorithm with the format ETSI. The key
              is (CN=VEHICLE_1, OU=ITS-VEHICLES, O=DIRECCION GENERAL DE
              TRAFICO, C=ES)"
 boolean_expression="(((IT.eventName='DSA_Encrypt') && ((#IsInstanceof(
     IT.Object, 'Encryption')==1) && (IT.vehicle_Id = IT.vehicle_id.1))
     ) && (IT.key_type==X509)) && (IT.algorithm='DSA') && ..."
  ...
</property>
```

the values of the attributes captured by the monitoring annotation. When the first event arrives, the `Monitoring Tool` checks whether it fits one or more of the events defined in the security property (Listing 1.4). In the example, the first event received fits the event of the property `event_id="1"` that corresponds with a change in the context. The second event received with the name `DSA_Encrypt` fits the event `event_id="2"` of the property by checking the values of the attributes received in the event with the boolean expression defined in the property. The class object captured is an instance of the `DSAEncryption` aspect that is deployed inside the application of the police vehicle and is using the `DSA` algorithm to encrypt the messages. Other attributes such as the key and the type of the key are also checked against the rule defined in the security property. As the two events received have a delay of less than one second as defined by the security property, the two events consecutively match the rules of the security property. So, in this example the `Monitoring Tool` checks that the CAM messages sent by the police vehicle are being encrypted in the context of a pursuit, and verifies the correct deploying of the encryption aspect required by the security policy, maintaining the correlation between the three parts. A non-match condition in the boolean expression of the rules in the security property, for instance, if the event with the name `DSA_Encrypt` does not occur, or if the `algorithm` attribute is different to DSA. This means the non-match of the entire security property, and thus the detection of a gap in the correlation between the security policy, the aspects and the security property.

6 Evaluation and Discussion

In this section we quantitatively evaluate the performance overhead of the dynamic deployment of security policies and the dynamic monitoring of the application. Also, as part of our participation in the INTER-TRUST project, the deployment modules (the `Aspect Generation`[5] and the `Aspect Weaver`[6]), the monitoring modules (the `Notification` and the `Context Awareness`)[7] as well as the MMT tool[8] presented in this paper have been used to implement a demonstrator of the project that provides dynamic adaptation of security policies for two real case studies: the ITS case study presented in this paper and an online electronic voting (e-voting) case study. This demonstrator has been evaluated both quantitatively, by controlled tests performed for the implementation of the modules, and qualitatively, by collecting the opinion of software developers with different expertise on both security and AOP. The main results of this evaluation are discussed in this section.

The evaluated scenario focuses on the communication between an ITS central station and a vehicle ITS station (called OBU for On Board Unit) via a Roadside ITS station (called RSU for Road Side Unit) based on CAM messages. CAM

[5] `https://github.com/Inter-Trust/Aspect_Generation`
[6] `https://github.com/Inter-Trust/Aspect_Weaver`
[7] `https://github.com/Inter-Trust/Testing_and_Monitoring_related_modules`
[8] `https://github.com/Inter-Trust/MMT_Security`

messages are part of the implementation of several services provided by the ITS central like the CSA service (stands for Contextual Speed Advisory service) or the DRP service (stands for Dynamic Route Planning service) that constitute the target of the evaluation in the context of INTER-TRUST project. These CAM messages are filled with a set of parameters describing the vehicle's status and are sent to the ITS central station. When a negotiation between the parties involved ends by an agreement or after the detection of a context change by the `Context Awareness` module, new security aspects (e.g., providing an obfuscation algorithm to hide the current location in the CSA service, or providing an encryption algorithm to maintain private the route in the DRP service) need to be deployed at runtime and the `Monitoring Tool` needs to monitor the application in order to detect possible attacks that break the correlation between the security policy, the security aspects, and the security properties.

6.1 Performance Evaluation

We have measured the performance overhead introduced by the dynamic deployment process and the performance overhead introduced by the dynamic monitoring of the application. The experiments were done on a laptop Intel Core i3 M350, 2.27 GHz, 4 GB of memory, and with 1.7 JVM. Aspects were implemented in AspectJ and Spring AOP.

Performance of Deployment. The performance overhead of the deployment process considers the time from the reception of a security deployment specification in the `Aspect Generation` module to the execution of the adaptation plan by the `Aspect Weaver`. This time includes the generation of the adaptation plan by the `Aspect Generation` and the interaction with the aspects in order to weave, unweave, and/or reconfigure them. We consider the number of aspects that need to be dynamically adapted (i.e., woven, unwoven, or reconfigured) in order to fulfill the required functionality specified in the security policy.

The results are summarized in Figure 4 where the performance presents a linear increment of the overhead over the number of aspects. We observe that the overhead introduced by the adaptation process is lower than the one initially expected. For instance, the adaptation process takes 320 milliseconds for deploying 20 aspects specified in the security policy. Reconfiguring aspects takes 20 milliseconds more on average than deploying them, while undeploying aspects takes 15 milliseconds more than deploying them. The results indicate that adapting security policies at runtime does not suppose a high overhead taking into account the use of aspects.

Performance of Monitoring. The performance overhead of the dynamic monitoring considers the time overhead introduced in the application during operation when the `Notification` and `Context Awareness` modules are integrated as aspects inside the application. We evaluated the time overhead of generating the traces for the most expensive monitor annotation presented in Table 1 — i.e., the

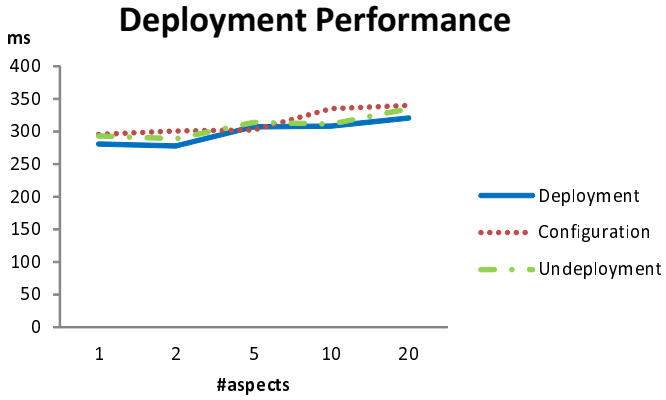

Fig. 4. Performance of deploying, reconfiguring, and undeploying security policies.

recursive @Taint annotation that tracks all methods encountered by the control flow from the annotated method. Figure 5 shows the time overhead based on the number of join points captured. We can observe that the performance presents a linear increment of the overhead over the number of join points while this number is lower than 100. Then, from 100 join points, the increment is higher but still linear. In all cases, the results obtained do not suppose a significative overhead. For instance, monitoring 10,000 join points in the control flow of a method takes 250 milliseconds on average. The analysis of the generated traces is carried out by the `Monitoring Tool` which is independent of the application and can reside in a different computer, and thus, the analysis of the traces does not affect the application's performance.

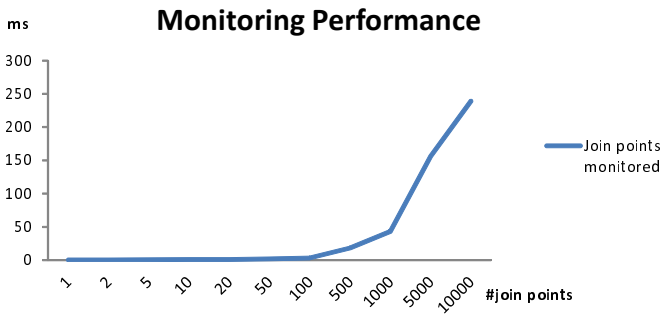

Fig. 5. Performance of monitoring join points at runtime.

6.2 Vulnerabilities Detection

A critical part of the evaluation of our approach is the evaluation of the effectiveness of the attacks detection by the monitoring tool. On the one hand, we evaluate the capability to detect that the correlation between the security properties, the security aspects and the security properties has been broken, by simulating different attack scenarios and intrusion attempts by relying on the Fuzz Testing [26] and the Active Testing [25] tools of the INTER-TRUST framework. These testing tools automatically generate the set of traces that are used to test the scenarios. On the other hand, to evaluate the effectiveness of the detection techniques, three metrics are proposed: (1) the *detection rate* of the MMT tool in detecting security vulnerabilities; (2) the *false positive rate* of the MMT tool (i.e., the ratio of vulnerabilities detected by the MMT tool when they are not present); and (3) the *detection coverage* of the MMT tool that can target the network, the application, or the system, for detecting vulnerabilities.

The attack model. We have identified a set of vulnerabilities that can break the correlation in our approach. Table 2 shows examples of attacks based on the identified vulnerabilities. For each attack, we present the result of the effectiveness in detecting the attack by the `Monitoring Tool`.

According to the vulnerabilities identified, we assume that an attacker is able to access and corrupt sensitive information that is interchanged and managed by the INTER-TRUST framework modules. This kind of attack would be possible due to vulnerabilities `V1-V4` and `V6-V7`. By exploiting these vulnerabilities, the attackers can follow a deceptive attack model [40], where deception can be defined as an interaction between two parties, a deceiver and a target, in which the deceiver successfully causes the target to accept as true a specific incorrect version of reality, with the intent of causing the target to act in a way that benefits the deceiver [41]. A specific class of deception attack is the false-data injection attack [41], where the attacker would inject false data into the INTER-TRUST framework.

Using the false-data injection attacks, one of the benefits that attackers can obtain by exploiting the vulnerabilities `V1-V4` of the INTER-TRUST framework is altering the aspects that are woven/unwoven/reconfigured, thereby the security level of the applications. For instance, the implementation class of an aspect may be changed by altering the information in the aspectual knowledge to provide a less secure authentication mechanism. Or, false functionality may be assigned to an aspect by altering the aspectual knowledge so an inappropriate aspect is woven into the system. Also, the pointcuts where an aspect is applied could be maliciously changed at runtime (when Spring AOP is used) to weave an aspect in more or fewer join points than expected by the application. Finally, a rule in the security policy may be maliciously deactivated by omitting the information about that rule in the security deployment specification that is received by the `Aspect Generation` module. This will avoid the weaving of the corresponding aspect.

Table 2. Vulnerabilities detection by the `Monitoring Tool`.

Vulnerability and Attack	Detected
Vulnerability V1: An attack changes the security policy before sending it, and thus the same malicious and potentially damaging security policy is sent to both modules. **Attack:** The rule that specifies the obligation to cypher is deleted. Both modules receive the security policy changed.	No
Vulnerability V2: An attack changes the security policy but only the one sent to one of the modules. **Attack:** The rule that specifies the obligation to cypher is deleted. Only one of the modules receive the security policy changed. The another one receives the original policy.	Yes
Vulnerability V3.1: Adding or changing the functionality provided by an existing aspect or adding new, untrustworthy aspects. **Attack:** A new aspect bypassing the user authentication is included.	Yes
Vulnerability V3.2: Adding or changing the pointcuts where the aspect will be injected. **Attack:** An encryption aspect encrypts a message that will not be understood by the receiver because it does not expect it cyphered.	Yes
Vulnerability V3.3: Deleting the information about existing aspects. **Attack:** An intruder deletes an existing pointcut in the aspectual knowledge (e.g., some messages that should be encrypted are not encrypted anymore), or deletes an existing advice or aspect in the aspectual knowledge (e.g., may force an error during the deployment of the aspects that make the application to remain in its last stable state).	Yes
Vulnerability V4: Changing the security adaptation file (e.g., modifying the actions that the `Aspect Weaver` will perform over the aspects). **Attack:** An intruder changes in the aspects deployed/undeployed/reconfigured in the security adaptation plan. Note that the consequences are the same as for vulnerabilities V3 and the examples are similar.	Yes
Vulnerability V5: Directly weaving/unweaving/reconfiguring at runtime an aspect inside the application without any previous change to the security policy (i.e., without following the workflow of the deployment process). **Attack:** A new aspect bypassing the user authentication is included directly through the Spring framework at runtime or an existing aspect is modified.	Yes
Vulnerability V6: Modifying the notification about the deployment status. **Attack:** A malicious user notify the deployment of a new aspect that has been directly woven into the application via vulnerability V5	Yes
Vulnerability V7: Changing the traces received and analyzed by the MMT tool. **Attack:** An intruder changes the contextual traces sent to the `Monitoring Tool` by the `Notification module` and the `Context Awareness` module.	No

The same benefit can also be obtained by exploiting the security vulnerability introduced by the Spring AOP runtime weaver (vulnerability V5), or any other AOP solution offering runtime weaving. Here the attack model is different because the attacker does not modify the information interchanged by the modules and instead directly injects new code into the running application by means of aspects. However, the consequence is the same: the security level of the application is altered (the same examples illustrated before apply here).

There is a particular situation in which the false-data injection attack can cause a denial-of-service attack, understanding that the service that is denied is the application security. This can be achieved through the vulnerability V3 that alters the information in the aspectual knowledge. This kind of attack requires the attacker to know that in the process of adapting at runtime a security policy is transactional and in consequence any error when deploying the security policy (e.g., a required aspect is not described in the aspectual knowledge or no implementation class can be found in the aspect repository for one required aspect) leaves the system in the previous stable state.

Finally, the information received by the monitoring modules in order to monitor the vulnerabilities and to detect the possible attacks could also be altered, affecting the results obtained (vulnerabilities V6-V7). The attack model is the same (the information interchanged by the INTER-TRUST modules is altered) but the motivation behind this kind of attack is different since the main goal would be to hide the attacks performed through the other vulnerabilities. Basically, not only if the security policy, the aspectual knowledge or the aspects' implementations are maliciously altered but also the generated traces are appropriately altered then the attack could be undetectable. In any case it is not straightforward and, in this case, a very detailed knowledge of the framework's behavior is needed to successfully perform the attack. Within INTER-TRUST, we consider that the risk of attacks exploiting V6-V7 is very low.

Effectiveness of the Monitoring Tool. We evaluate the effectiveness of the detection techniques with the help of questionnaires. The evaluation procedure involved 30 evaluators. The evaluators were selected mainly from among software developers and security experts, with different backgrounds and levels of knowledge, and experience in security modeling, testing and monitoring. They followed a set of instructions to run the monitoring tool to analyze both network traffic and the application's internal events. The evaluators were also requested to create new security properties and use the tools on different pre-collected traces (containing vulnerabilities or not).

Table 3 shows the results of the effectiveness of our approach in vulnerabilities detection. For each metric we show the expected value and the values obtained by our approach after the evaluators have analyzed the output of the MMT tool. For instance, 42% evaluators answer the MMT tool detected more than 80% vulnerabilities. The estimation of the detection rate is difficult to perform for a non-expert user. The testing report (i.e., outcome of monitoring experimentation) provides detailed results about the events that led to the testing verdict.

Table 3. Evaluation results.

Metric	Expected value	Evaluators Results
Detection rate	>80%	42%: >80% 25%: >60% and <80% 33%: No experience
False positive rate	<20%	50%: 0% 8%: <20% 42%: No experience
Detection coverage	33%	25%: 100% 50%: 66% 25%: No experience

This allows us to understand the testing report and to estimate the detection rate. The analysis of the results demonstrates that the evaluators were satisfied with the detection rate provided by the MMT tool. The detection rate (more than 80%) may be regarded as satisfactory, although improvements still need to be made.

False positive verdicts occur when the tool detects vulnerabilities that are not present. The false positive rate (less than 20%) is excellent, but improvements still need to be made. As for the previous metric, a non-expert evaluator could have difficulties in estimating the false positive rate.

The analysis of the evaluator's results demonstrate the reliability of the monitoring techniques used in INTER-TRUST. They found the tool easy to understand and configure. It provides readable and relevant reports that can be easily exploited by the developers in order to detect and correct potentially detected bugs and vulnerabilities. MMT has also been applied to high speed networks (up to 5 Gigabits/s) internally at the Montimage company and showed its scalability and capability to detect vulnerabilities by combining events coming from different sources (network traffic, application internal events or environmental context).

6.3 Threats to Validity and Lessons Learnt

In this section we discuss the threats to validity of the evaluation presented in this section and some lessons learnt during the development and evaluation of our approach.

Dynamicity offered by INTER-TRUST. There are many application domains that will benefit from using INTER-TRUST due to the dynamicity offered in order to negotiate and deploy runtime security policies using aspects. However, we have seen that new vulnerabilities are introduced in the application due to this dynamicity. This is not exactly a limitation of AOP or the INTER-TRUST framework since other existing dynamic deployment techniques presents the same vulnerabilities, but rather the price that it has

to be paid in order to have applications which are more adaptive to changing security requirements. Anyway, it is still definitively an issue to be taken into account. This is the reason why the use of a monitoring technique such as the one presented in the paper is so important in coping with this threat.

Monitoring capabilities. The INTER-TRUST monitoring capabilities depend on the list of notified events. The more events are notified, the higher the detection capabilities of the monitoring tool. However, here a trade-off needs to be considered between the detection capabilities and the performance of the monitoring tool, especially when the recursive monitoring annotations (see Table 1) are used. Although in the performance evaluation we showed that the overhead is not so significant, special care needs to be taken when annotating the applications for monitoring. Regarding the capabilities of the `Monitoring Tool` we would like to highlight that it can be used to detect multiple kinds of vulnerabilities, but in this paper we have focused only on those vulnerabilities related with maintaining the correlation between the security policies, the security properties and the security aspects deployed.

Privacy protection of the MMT tool. The monitoring of the application's internal events can engender several privacy issues, since sensitive data can be shared with MMT. Within INTER-TRUST, a privacy protection module is provided to ensure the anonymity of the shared data.

Implementation of security using aspects. There are some security concerns (e.g., authorization) that are more difficult to encapsulate as an aspect than others. In these cases additional components are needed in order to appropriately implement such security concerns. The approach that we followed in INTER-TRUST to implement the authorization aspect confirms this difficulty. Concretely, INTER-TRUST, as was illustrated in Section 3, is a framework specifically designed to incorporate security to applications and to be able to dynamically adapt it at runtime. This is achieved not only by the weaving/unweaving of aspects, but also by the security specific modules. Concretely, the authorization or access control aspect has been implemented in the context of the INTER-TRUST project for both the ITS and the e-voting case studies making use of the `Policy Engine` module (in a similar way that access control is aspectized in ModAC [16]). The `Policy Engine` acts as a PDP (Policy Decision Point — point which evaluates access requests against authorization policies before issuing access decisions[1]). The process is: the authorization rules are defined as part of the security policy and can be consulted at runtime through the `Policy Engine` module. The INTER-TRUST aspects encapsulate a predefined mechanism in order to access the modules of the INTER-TRUST framework in a decoupled and distributed way and that the authorization aspect uses this mechanism to access to the `Policy Engine` module.

Another issue of the implementation of security using aspects is that the aspect developers cannot be completely unaware of how applications are developed. A clear example is the case when reflection is used to implement

[1] See RFC2904 in `http://tools.ietf.org/html/rfc2904`

parts of the application. In this case, the pointcuts need to be written in a special way[1]. Otherwise, the classes created using reflection or the methods invoke using reflection will not be captured by the security aspects.

Undetectable attacks. The knowledge that the attackers have regarding how the INTER-TRUST framework works can make some of the attacks undetectable. Some of the situations have already being mentioned. For instance, the attacker is undetectable if the security policy is simultaneously altered in both the `Aspect Generation` module and the `Monitoring Tool` (vulnerability `V2`). Also if the attacker knows that the process of adapting a security policy at runtime is transactional and in consequence any error deploying the security policy (e.g., a required aspect is not described in the aspectual knowledge or no implementation class is found in the aspect repository for a required aspect) leaves the system in the previous stable state, vulnerability `V3` could be used to perform a denial-of-service attack by avoiding the correct deployment of the negotiated security policy. Thus, we still need to continue working to improve the monitoring and detection mechanisms in order to detect these undetectable or denial-of-service attacks.

Vulnerable points used to perform the attack. As discussed in the previous section, the `Monitoring tool` is able to detect the attacks performed by exploiting most of the vulnerable points introduced by the mechanism for the dynamic deployment of aspects. This is done basically by detecting that the correlation between the security policies, the security aspects and the security properties is broken. Moreover, since security is always enforced in INTER-TRUST by using aspects it is possible to identify that a required aspect is missing or that a woven aspect should not be in a specific join point. However, it is not straightforward to identify the precise vulnerable point that was used to perform the attack. In order to do that the application administrator needs to also consult the audit information generated by the implicated INTER-TRUST module. Although the description of this audit process is not in the scope of the paper it is important to highlight here that it complement the information generated by the monitoring tool.

Security expertise. A consequence of the high flexibility and adaptability offered by the INTER-TRUST framework is the difficulty to configure the different modules and to instantiate the security requirements into aspects. Also, the mapping of the security requirements to the security properties that are then checked by the monitoring tool is not a straightforward task. This instantiation work needs to be performed by a developer that has security expertise, which is not always easy to find in Industry.

7 Conclusions and Future Work

We have defined a dynamic aspect-oriented approach for the deployment and monitoring of security policies at runtime. The approach maintains the correlation between the security policies that need to be enforced, the security aspects

[1] `http://www.eclipse.org/aspectj/doc/released/faq.php\#q:reflectiveCalls`

that are deployed/undeployed in order to enforce those security policies and the security properties that are activated/deactivated in order to check whether or not the system is behaving according to the specified security policies. Following our approach, the adaptation of security policies at runtime and the monitoring of the correlation defined do not suppose a high overhead in the application's performance.

Our approach has been integrated as part of the INTER-TRUST framework, however, it can also be applied to many other types of pervasive systems in other contexts independently of the INTER-TRUST framework, and can also be used to adapt other functionalities implemented as aspects (not only security). INTER-TRUST has been successfully integrated in two real case studies: the ITS case study presented in this paper and an e-voting case study. Each case study is based on different technologies and has different security requirements. The flexibility exhibited by the INTER-TRUST framework allows its integration with different middlewares such as FamiWare [42] in order to provide security and privacy to wireless sensor networks; and with security adaptation services such as a MAPE-K (Monitor-Analyze-Plan-Execute over a shared Knowledge) loop approach [43].

The proposed solution is innovative since it relies on dynamic deployment of security policies after a negotiation phase and the continuous monitoring of resulting application to detect potential vulnerabilities and attacks. The solution allows to enforce context-awareness and dynamic adaptation of security in real world applications. The evaluation's results demonstrate the reliability of the monitoring techniques used in INTER-TRUST, regarding the effectiveness of our approach in vulnerabilities detection.

As for future work, we plan to complete our approach by dynamically generating the structure of the aspects and the security properties from the security policies minimizing the aspectual knowledge needed to maintain the correlation.

Acknowledgment

Work funded by the European INTER-TRUST FP7-317731 and the Spanish TIN2012-34840 (co-funded by EU with FEDER funds), FamiWare P09-TIC-5231, and MAGIC P12-TIC1814 projects.

References

1. Atzori, L., Iera, A., Morabito, G.: The internet of things: A survey. Computer Networks **54**(15) (2010) 2787–2805
2. Chinrungrueng, J., Sunantachaikul, U., Triamlumlerd, S.: Smart parking: An application of optical wireless sensor network. In: Applications and the Internet Workshops, 2007. SAINT Workshops 2007. International Symposium on. (2007) 66–66
3. Dornbush, S., Joshi, A.: Streetsmart traffic: Discovering and disseminating automobile congestion using vanet's. In: IEEE 65th Vehicular Technology Conference. VTC'07 (2007) 11–15

4. Varaiya, P.: Smart cars on smart roads: problems of control. IEEE Transactions on Automatic Control **38**(2) (1993) 195–207
5. Fu-Yuan, W., D, Z., Li, Y.: Smart cars on smart roads: An IEEE intelligent transportation systems society update. IEEE Pervasive Computing **5**(4) (2006) 68–69
6. De Borger, W., De Win, B., Lagaisse, B., Joosen, W.: A permission system for secure aop. In: Aspect-Oriented Software Development. (2010)
7. Mouelhi, T., Fleurey, F., Baudry, B., Traon, Y.: A model-based framework for security policy specification, deployment and testing. In: Model Driven Engineering Languages and Systems. (2008)
8. FP7 European Project INTER-TRUST: Interoperable Trust Assurance Infrastructure. `http://www.inter-trust.eu/`
9. Ayed, S., Idrees, M.S., Cuppens-Boulahia, N., Cuppens, F., Pinto, M., Fuentes, L.: Security aspects: A framework for enforcement of security policies using AOP. In: Signal-Image Technology & Internet-Based Systems. SITIS (2013) 301–308
10. Kiczales, G., Lamping, J., Mendhekar, A., Maeda, C., Lopes, C., Loingtier, J.M., Irwin, J.: Aspect-oriented programming. In: ECOOP — Object-Oriented Programming. Volume 1241. (1997) 220–242
11. De Win, B., Piessens, F., Joosen, W.: How secure is AOP and what can we do about it? In: Software Engineering for Secure Systems. (2006) 27–34
12. De Win, B., Vanhaute, B., De Decker, B.: Security through aspect-oriented programming. In: Advances in Network and Distributed Systems Security. Volume 78. (2002) 125–138
13. Steimann, F.: The paradoxical success of aspect-oriented programming. SIGPLAN Not. **41**(10) (2006) 481–497
14. Sun Microsystems, Inc.: Sun's XACML implementation. http://sunxacml.sourceforge.net/
15. DeMillo, R.A., Lipton, R.J., Sayward, F.G.: Hints on test data selection: Help for the practicing programmer. Computer **11**(4) (1978) 34–41
16. Toledo, R., Nunez, A., Tanter, E., Noye, J.: Aspectizing Java access control. IEEE Trans. Softw. Eng. **38**(1) (2012) 101–117
17. Damianou, N., Dulay, N., Lupu, E., Sloman, M.: The ponder policy specification language. In: Policies for Distributed Systems and Networks (POLICY). Volume 1995. Springer Berlin Heidelberg (2001) 18–38
18. Toledo, R., Tanter, E.: Secure and modular access control with aspects. In: Proceedings of the 12th Annual International Conference on Aspect-oriented Software Development. AOSD'13 (2013) 157–170
19. Abadi, M., Fournet, C.: Access control based on execution history. In: Proceedings of the 10th Annual Network and Distributed System Security Symposium (NDSS). (2003) 107–121
20. Zhang, S., Zhao, J.: On identifying bug patterns in aspect-oriented programs. In: 31st Annual International Computer Software and Applications Conference. Volume 1 of COMPSAC'07. (2007) 431–438
21. Serme, G., De Oliveira, A.S., Guarnieriy, M., El Khoury, P.: Towards assisted remediation of security vulnerabilities. In: 6th International Conference on Emerging Security Information, Systems and Technologies (SECURWARE). (2012)
22. Padayachee, K., Eloff, J.: An aspect-oriented model to monitor misuse. In: Innovations and Advanced Techniques in Computer and Information Sciences and Engineering. (2007) 273–278
23. De Win, B., Vanhaute, B., De Decker, B.: How aspect-oriented programming can help to build secure software. Informatica **26**(2) (2001) 141–149

24. Pinto, M., Horcas, J.: How to develop secure applications with aspect-oriented programming. In: Risks and Security of Internet and Systems (CRiSIS), 2013 International Conference on. (2013) 1–3
25. Cavalli, A., de Oca, E., Mallouli, W., Lallali, M.: Two complementary tools for the formal testing of distributed systems with time constraints. In: Distributed Simulation and Real-Time Applications. (2008)
26. Howard, M., Lipner, S.: Inside the windows security push. IEEE Security Privacy **1**(1) (2003) 57–61
27. Morales, G., Maag, S., Cavalli, A., Mallouli, W., de Oca, E., Wehbi, B.: Timed extended invariants for the passive testing of web services. In: IEEE International Conference on Web Services. (2010) 592–599
28. Roesch, M.: SNORT - lightweight intrusion detection for networks. In: Proceedings of the 13th USENIX Conference on System Administration. LISA '99 (1999) 229–238
29. Paxson, V.: BRO: A system for detecting network intruders in real-time. Comput. Netw. **31**(23–24) (1999) 2435–2463
30. Kalam, A., Baida, R., Balbiani, P., Benferhat, S., Cuppens, F., Deswarte, Y., Miege, A., Saurel, C., Trouessin, G.: Organization based access control. In: Policies for Distributed Systems and Networks. (2003)
31. Autrel, F., Cuppens, F., Cuppens, N., Coma, C.: MotOrBAC 2: a security policy tool. Third Joint Conference on Security in Networks Architectures and Security of Information Systems (SARSSI) (2008)
32. Clarke, Jr., E.M., Grumberg, O., Peled, D.A.: Model Checking. MIT Press (1999)
33. Mallouli, W., Wehbi, B., de Oca, E.M., Bourdelles, M.: Online network traffic security inspection using MMT tool. In: System Testing and Validation. (2012)
34. Wehbi, B., de Oca, E., Bourdelles, M.: Events-based security monitoring using MMT Tool. In: Software Testing, Verification and Validation. (2012)
35. Vinoski, S.: Advanced message queuing protocol. IEEE Internet Computing **10**(6) (2006) 87–89
36. Horcas, J.M., Pinto, M., Fuentes, L.: Closing the gap between the specification and enforcement of security policies. In Eckert, C., Katsikas, S., Pernul, G., eds.: Trust, Privacy, and Security in Digital Business. Volume 8647 of Lecture Notes in Computer Science. Springer International Publishing (2014) 106–118
37. Aouadi, M., Toumi, K., Cavalli, A.: On modeling and testing security properties of vehicular networks. In: IEEE Seventh International Conference on Software Testing, Verification and Validation Workshops (ICSTW). (2014) 42–50
38. Aouadi, M., Toumi, K., Cavalli, A.: A formal approach to automatic testing of security policies specified in xacml. In: Foundations and Practice of Security. Volume 8930 of Lecture Notes in Computer Science. Springer International Publishing (2015) 367–374
39. Andrade, R., Rebelo, H., Ribeiro, M., Borba, P.: AspectJ-based idioms for flexible feature binding. In: Software Components, Architectures and Reuse (SBCARS), VII Brazilian Symposium on. (2013) 59–68
40. Ajay, V.: A survey on system attack models. Bonfring International Journal of Research in Communication Engineering **2**(1) (2012) 01–04
41. Teixeira, A., Pérez, D., Sandberg, H., Johansson, K.H.: Attack models and scenarios for networked control systems. In: Proceedings of the 1st International Conference on High Confidence Networked Systems. HiCoNS '12 (2012) 55–64
42. Pinto, M., Gámez, N., Fuentes, L., Amor, M., Horcas, J.M., Ayala, I.: Dynamic reconfiguration of security policies in wireless sensor networks. Sensors **15**(3) (2015) 5251

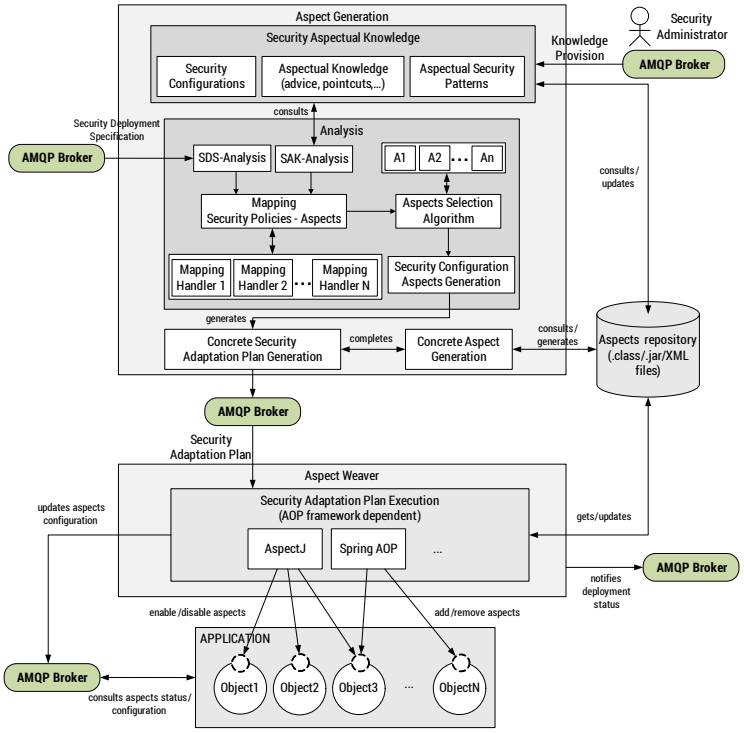

Fig. 6. Aspect Generation and Aspect Weaver modules.

43. Horcas, J.M., Pinto, M., Fuentes, L.: Runtime enforcement of dynamic security policies. In: Software Architecture. Volume 8627 of LNCS. Springer International Publishing (2014) 340–356

A Aspect Generation and Aspect Weaver

Figure 6 shows the detailed design of the `Aspect Generation` and `Aspect Weaver` modules.

The `Aspect Generation` module receives notifications about security policy updates that must be deployed, and dynamically generates an adaptation plan. An adaptation plan consists of a list of aspects, advices or pointcuts that need to be added or removed into the application. The `Aspect Generation` module contains all the information that the `Aspect Generation` and `Aspect Weaver` modules need in order to adapt the aspects deployed in the application to security policy changes —- i.e., the `Security Aspectual Knowledge`. The `Aspect Generation` module has the capability to incorporate (as part of the `Security Aspectual Knowledge`) the initial aspectual information and the capability to update that initial information at runtime. Moreover, this module also allows

adding new aspects (pointcuts and/or advices) to the aspect repository at runtime, in order to make them available to deploy in the applications. In order to generate the adaptation plan, the `Aspect Generation` performs the mapping process that matches the required functionalities specified in the security deployment specification with the functionalities provided by each aspect.

The adaptation plan is sent to the `Aspect Weaver` module, that is in charge of executing it by translating the list of aspects received as input (which is specified independently of a particular AOP framework) to the particular syntax of the particular AOP weaver being used. This means that different instantiations of the INTER-TRUST framework for using different AOP weavers will provide different implementations of this component. The output of this component is a direct interaction with the selected AOP weaver in order to interact with it and to weave/unweave/reconfigure the corresponding aspects into the applications.

B Montimage Monitoring Tool (MMT)

Figure 7 shows the detailed design of the `Montimage Monitoring Tool` (MMT).

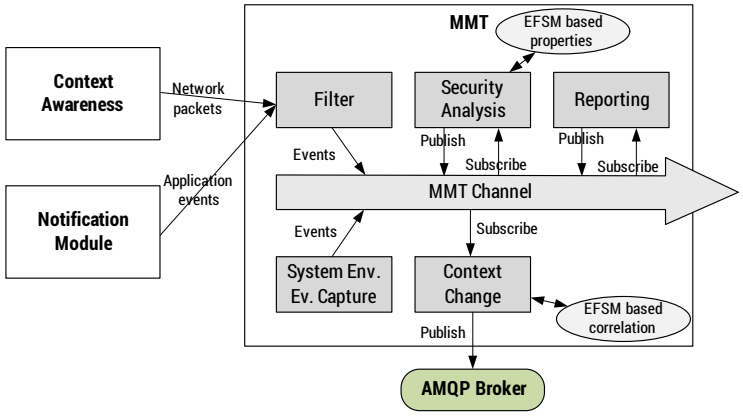

Fig. 7. INTER-TRUST Montimage Monitoring Tool (MMT).

The main objective of the MMT is to continuously capture observable information at different levels (e.g., application environment, network level, operating system, internal application events, etc.) and correlate them in order to detect potential vulnerabilities, security flaw and/or intrusion attempts. This module verifies application or protocol network traffic traces against a set of MMT-Security properties. The MMT can be used in the testing phase to complement the work already done by the testing tools, or during the application operation (i.e., at runtime) in order to detect live security issues. The MMT has two

main inputs: the negotiated security policy, and the security properties denoting known vulnerabilities and attacks to be checked on the system/application collected traces. The main output of this module is a security analysis report for each security property. The detection of the non-compliance of security properties at runtime generates warnings and alarms that may provoke the reaction (by mean of re-activating an obligation security policy) and enforce thus secure interoperability.