Dynamic Deployment and Monitoring of 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. 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 with 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 automatically reacts when needed.

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

1 Introduction

Future Internet (FI) systems encompass a set of pervasive computing devices (e.g., smartphones, vehicles, wearables) always connected to the Internet and continuously exchanging information with remote entities [1]. In order to ensure that the exchange of information is performed securely, the development of such systems requires the creation of a set of security mechanisms that are able to protect the system against different threats that may arise. For instance, let us consider the following case study: an Intelligent Transportation System (ITS) application that dynamically recommends the speed limits for a road according to climate conditions and to unexpected events like accidents or traffic jams, collects the information sent by both the vehicle's sensors (e.g., geolocation, 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 assured, 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,and (3) in some contexts (e.g., when a police car is pursuing an offender) 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. On the one hand, the security policies can be heterogeneous because each user can customize his own security policies that answer their security constraints and they can also be different from the security policies expected by the applications. On the other hand, the security policies can be dynamic and can 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, there is a lack of sufficiently rich techniques to tackle the problem of 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 issues, the Inter-operable Trust Assurance Infrastructure (INTER-TRUST) framework [2] 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 [3]. 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) [4]. AOP is used to add/implement security aspects (i.e., anonymity, authentication, integrity, encryption, etc.) to application components at runtime so that applications can dynamically adapt their behavior for required/negotiated security policies. However, the dynamic deployment mechanism can introduce new vulnerabilities and security risks, and thus 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 an approach to detect changes in the environment and checking that the communicating parties respect the negotiated security policies by maintaining the correlation between the security policies, the security aspects, and the security properties of the monitoring tool. 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.

The rest of the paper is organized as follows. Section 2 explains the correlation between the security policies, the aspects, and the security properties; and briefly

overviews the INTER-TRUST framework. In Section 3 we present our approach to deploy the security policies and monitor that correlation. Section 4 evaluates the overhead performance of our approach and Section 5 discusses related work. Finally, Section 6 concludes the paper and presents our future work.

2 Correlation between Security Policies, Aspects and Security Properties

The correct enforcement and dynamic adaptation of the security policies is based on two cornerstones (see Figure 1). 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). For instance, in order to monitor the correct deployment of the security policy shown in Figure 1, with three rules that indicate that the system is required to cypher the messages, to ensure the user's anonymity and to allow only the interaction of authorized users, a set of security properties associated with these rules needs to be activated in the monitoring tool. In Figure 1 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. 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 modular architecture of the INTER-TRUST framework that implements the correlation described is shown in Figure 2. In INTER-TRUST, security policies rely on the OrBAC model [5], and are first specified using a `Security Editor` (e.g. MotOrBac [6]) and then negotiated between the different parties (e.g. a vehicle and an ITS server in the context of a Vehicle-to-Infrastructure communication) using a `Negotiation` module (see the **Dynamic Specification of Security Policies** block in Figure 2). 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

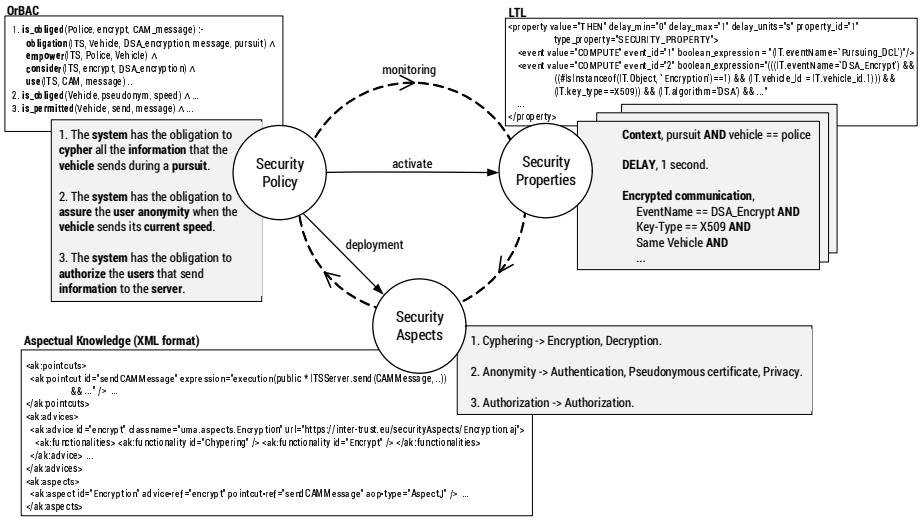

Fig. 1. Correlation of the security policies, the aspects, and the security properties.

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 (see the **Dynamic Deployment of Security Policies** block in Figure 2). Security aspects can be developed in any Java-based AOP language such as AspectJ, Spring AOP, CaesarJ, or JBoss. The *aspectual knowledge* depicted in Figure 1 contains the functionality provided by the aspects for each security policy and the join points where the aspects can be deployed.

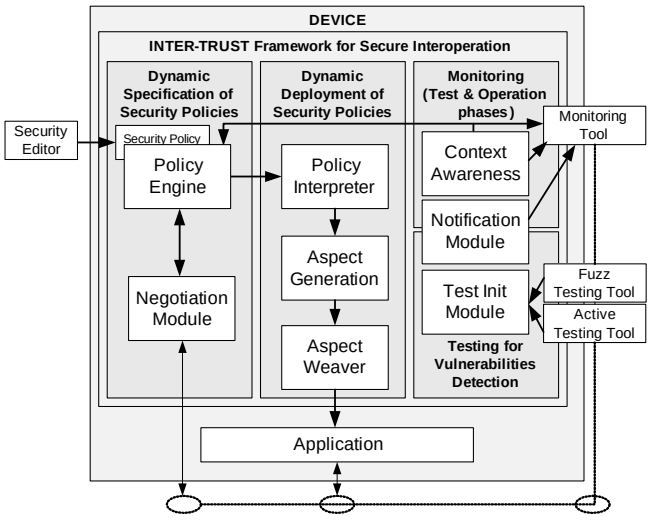

Fig. 2. Architecture of the INTER-TRUST framework.

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 [7] based on Linear Temporal Logic (LTL) to define security rules (i.e., rules that should be respected) or attacks and misbehaviors [8]. The `Monitoring Tool` relies on an adaptation of the Montimage Monitoring Tool (MMT) [9] 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. (see the **Monitoring (Test & Operation phases)** block in Figure 2). Finally, different fuzz [10] and active [11] 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. 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 (see the **Testing for Vulnerabilities Detection** block in Figure 2).

In this paper, we focus on the dynamic deployment of the security policies and on the monitoring phase, while the details of the dynamic specification of security policies and the testing phases are beyond the scope of this paper.

3 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).

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

The `Aspect Generation` module performs a mapping between the required security functionalities and the aspects that provide these functionalities. The

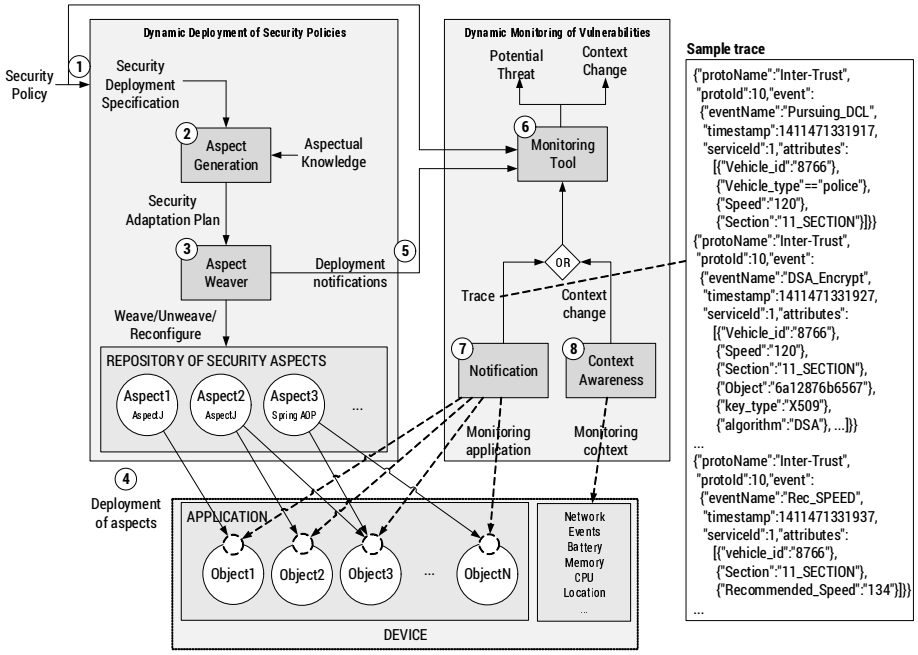

Fig. 3. Our approach for deploying and monitoring security policies.

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 the list of actions that must be performed over the aspects: weave, unweave, or reconfigure. 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). 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 (e.g., AspectJ does not support runtime weaving).

Listing 1.1 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 point[3]. This mechanism increases the degree of dynamicity by coding patterns that can dynamically support enabling and disabling advice in aspects [12]. In our example, the `AspectsStatus` class contains the configurations and status

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

Listing 1.1. 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 }
```

(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). That is, to notify whether or not the deployment was successfully carried out and which aspects were deployed/deployed/reconfigured.

3.2 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. 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. 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.

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) — 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 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 (Figure 1). 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, maintainig 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 occurr, 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.

4 Evaluation

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` and the `Aspect Weaver`)[4], the monitoring modules (the `Notification` and the `Context Awareness`)[5] as well as the Monitoring Tool[6] 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 case study.

4.1 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`. 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 experiments were done on a laptop Intel Core i3 M350, 2.27GHz, 4 GB of memory, and with 1.7 JVM. Aspects were implemented in AspectJ and Spring AOP. The results are summarized in Figure 4 where the performance presents a linear increment of the overhead over the number of aspects. 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 with AOP at runtime does not suppose a high overhead.

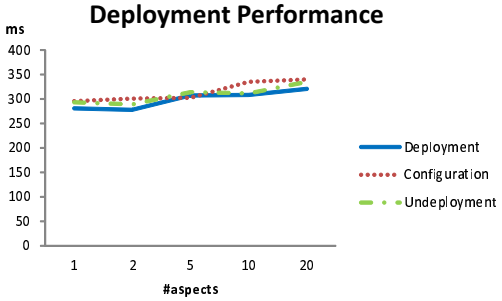

Fig. 4. Performance of deployment security policies.

4.2 Performance of Monitoring

The performance overhead of the dynamic monitoring considers the time overhead introduced at runtime when the `Notification` and `Context Awareness` modules are integrated as aspects inside the application. We evaluated the time overhead for generating the traces for the most expensive monitor annotation — i.e., the recursive 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

[4] https://github.com/Inter-Trust/Aspect_Generation/tree/demonstrator-version

[5] https://github.com/Inter-Trust/Notification_Module

[6] https://github.com/Inter-Trust/MMT_Security

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.

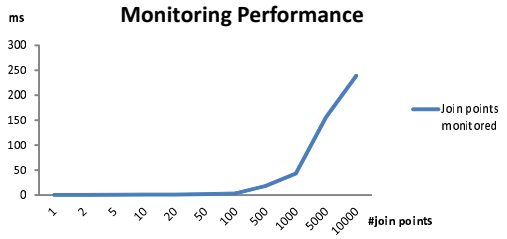

Fig. 5. Performance of monitoring join points at runtime.

5 Related Work

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 ([13, 14]). In [13], the authors propose a framework for specifying, deploying and testing access control policies independently of the security model. 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 the INTER-TRUST framework can. In [14], an Aspect Oriented Permission System (AOPS) for runtime policy enforcement is presented. The policy decisions are based on the execution history-based access control model (HBAC) [15] 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.

AOP vulnerabilities are well-known and have been identified during the development activity [16–19]. In [16], 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 [17] 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 [18], 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 [19], 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.

Apart from monitoring, there are several techniques to perform dynamic detection of failures in the deployment of security policies such as active testing [11] (to validate the implementation by applying a set of test cases and analyzing its reaction) or fuzz testing [10] (to detect unwanted behaviors or security violation by using random or mutated inputs). However, although these testing techniques are incorporated in the INTER-TRUST framework, these are not suitable to use at runtime as monitoring can be, but are applied at the testing phase.

Finally, the modular architecture of the INTER-TRUST framework allows its integration with different middlewares such as FamiWare [20] in order to provide security and privacy to wireless sensor networks; and with security adaptation services such as a MAPE-K loop approach [21].

6 Conclusions and Future Work

We have defined an approach to maintain the correlation 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. 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).

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, 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. FP7 European Project INTER-TRUST: Interoperable Trust Assurance Infrastructure. `http://www.inter-trust.eu/`
3. 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

4. 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
5. 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)
6. 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)
7. 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
8. Mallouli, W., Wehbi, B., de Oca, E.M., Bourdelles, M.: Online network traffic security inspection using MMT tool. In: System Testing and Validation. (2012)
9. Wehbi, B., de Oca, E., Bourdelles, M.: Events-based security monitoring using MMT Tool. In: Software Testing, Verification and Validation. (2012)
10. Howard, M., Lipner, S.: Inside the windows security push. IEEE Security Privacy **1**(1) (2003) 57–61
11. 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)
12. 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
13. 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)
14. De Borger, W., De Win, B., Lagaisse, B., Joosen, W.: A permission system for secure aop. In: Aspect-Oriented Software Development. (2010)
15. 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
16. 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
17. 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
18. 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)
19. 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
20. 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
21. 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