Personalized Security in Mobile Environments Using Software Policies

Mhammed Chraibi
School of Science and Engineering
Al Akhawayn University in Ifrane
Ifrane, Morocco
M.Chraibi@aui.ma

Hamid Harroud
School of Science and Engineering
Al Akhawayn University in Ifrane
Ifrane, Morocco
H.Harroud@aui.ma

Abdelilah Maach
Ecole Mohammadia des Ingenieurs
University Mohamed V
Rabat, Morocco,
maach@emi.ac.ma

***Abstract* - With the advance of technology and the widespread of mobile devices that enable users to have access to a wide range of services wherever they are, and whenever they want, many security issues arise. Both users and service providers feel the need to protect themselves from the large number of threats that are present on every network. Some time ago, users could have access to services only if they were physically present in a certain, predefined, area. This gave a lot of user personal information to the service providers which helped them secure their systems and their transactions with users. Now, it is not anymore the case. Therefore, the need arose for a novel way, for mobile users and service providers, to secure their information and their transactions. In this paper, we show that combining software policies and context information provides users and service providers with confidentiality, data integrity, data availability, and accountability.**

Keywords- mobility; security; software policies; context

I. INTRODUCTION

With the emergence of mobile technologies and the perpetual improvement of context aware technologies, users make use of their small devices, such as smartphones, laptops, and personal digital assistants (PDAs) and take advantage of the surrounding services in their environment that they need to achieve their everyday life tasks. To be able to receive the most appropriate and personalized services, users build their own profiles within which they find themselves obliged to disclose personal information as in [1]. There is obviously a threat to privacy as not all the service providers need access to all the information available in the profile. A tradeoff between the amount of personal information released through the profile and user privacy has to be made.

Another aspect that makes it even more important to protect the user's information is mobility. Ideally, the user must be able to move from one environment to the other and still receive the same services if not more services that are adapted to his profile while being protected. In this paper, we tackle the security issues that rise from user's mobility, and show how software policies can be used to enforce security in mobile environments. The fact that users can transport their policies with them wherever they go, added to the fact that users can express their security needs in terms of policies make software policies a suitable solution for mobile users. In addition, users can decide which specific information to disclose to a specific service provider. Moreover, well designed policies enable users to take advantage of context information to enhance security. Combining the rules with context information allows the user not only to take advantage of his knowledge of the specificities of the action that he will be conducting, but also the knowledge of environment conditions that he might not be aware of. The user, the service provider, and the security management component, each must have their own policies that will help regulate and secure any transaction and/or action that takes place.

The rest of this paper is organized as follows: In Section 2, there is an overview of the work that has been done on security in mobile environments and policy-based systems. In Section 3, we present the policies that we have designed and show how context information can be incorporated. Section 4 contains a thorough description of the different components of our policy-based security system and how it achieves security. Section 5 contains a scenario that takes places at Al Akhawayn University and that shows the functioning of the policy-based security management system. Finally, the conclusion and future work section is presented.

II. RELATED WORK

Different aspects of security are handled using software policies at different levels and applications. Policies have been used to provide security management for sensor networks such as SecSNMP [2]. SecSNMP allows administrators to dynamically manage the security settings using policies. Settings include availability, authentication, confidentiality, integrity, non-repudiation, freshness, and survivability [2]. The second example of systems using policies to achieve security is proposed by [3] and uses security policies in a slightly different manner. This multi-agent system is a good example of great importance to us as one of the most interesting features in agent systems is their mobility and their adaptability. Basically, agents are supposed to move from an environment to another and autonomously adapt and provide/use services. Software policies are used to identify the security threat and launch the security mechanism that is needed to deal with it. Just like in our system, one issue is to identify the nature of the threats that could exist in different environments.

 ISBN: 978-1-61208-171-7

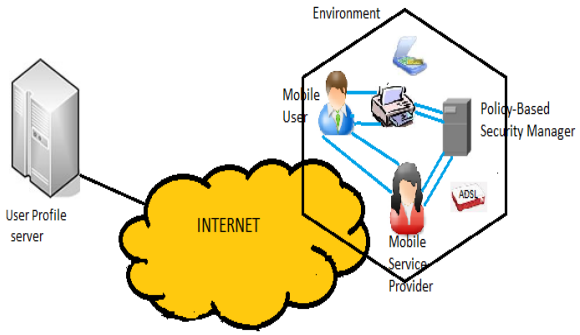

Figure 1. Security management in mobile environments

Our contribution is the design of a system, based on policies, that allows mobile users define their own security concerns and deal with them the way they want in whatever environment they are. Figure 1 shows where our policy-based security management system fits.

In the context of this paper, ensuring security involves providing the users with the tools to achieve confidentiality, data integrity, availability, and/or accountability. Achieving confidentiality means avoiding and preventing disclosure of information to unwanted parties. There are several tools that are used; the most known is encryption of the data that is stored, and the data that is being transmitted on the network. As specified in [8], using encryption can be the solution to attacks like eavesdropping. Confidentiality can also be enforced using access control as only the parties that have access to the data are allowed to access it. There are different access control methods, such as RBAC (Role Based Access Control), MAC (mandatory Access Control), and TrustAC (Trust-based Access Control) that are discussed in [9] and [10]. Access control does not only provide with confidentiality, it also enforces accountability (keeping track of the logs). However, it cannot ensure the confidentiality of the data transferred on a network. Another way to enhance security in a mobile environment is to use IPv6. The latter contains security enhancements that try to overcome the shortcomings of IPv4. For example, resistance to scanning is only possible under IPv6 addressing scheme [11]. Nevertheless, using IPv6 cannot guarantee that unwanted parties can stop regular users from accessing data or services that they are supposed to have access to. In other words, using IPv6 does not provide with availability.

From the previous discussion about the types of security that can be achieved and the tools that are used to achieve them it is noticeable that there are at least three approaches to security. The first approach is one that is meant to protect from a specific type of attacks. A good example is encryption which provides confidentiality by avoiding the dangers of eavesdropping attacks. Further, access control management provides with availability and integrity. However, if interactions take place through a network, access control mechanisms cannot provide with confidentiality. The second approach is meant to provide with security at a certain level only. For example, IPv6 provides with security at the low levels of the OSI reference model. The use of IPv6 does not provide with security at the application level. Finally, the third approach is the one that provides different types of security at different levels. Our work fits in this third category. As shown in Figure 2, service providers, as much as users, can specify any type of security at any level. Some services might require the encryption of the data being transferred, while others may emphasize on the need to use IPv6 for the transfer of the data. The combination of encryption and the use of IPv6 is therefore possible through policies.

Context information can also be included in policies in order to enforce security. In fact, by knowing some key context information, one can design specific policies that would enforce, for example, access control [12]. Instead of requiring a simple username and password combination, a service might require some additional confidential information only known by the user and the service. Or, the service could require that the transaction take place in an encrypted way. Another requirement would be asking a trusted third party to certify the identity of the user. These actions enforce integrity, availability and confidentiality. It is based on such real life examples that we built our policy model by integrating context information within policy conditions.

III. POLICIES IN SECURITY

Before getting into the details of how policies help achieve confidentiality, data integrity, availability, and accountability, we present first our policy model and its structure.

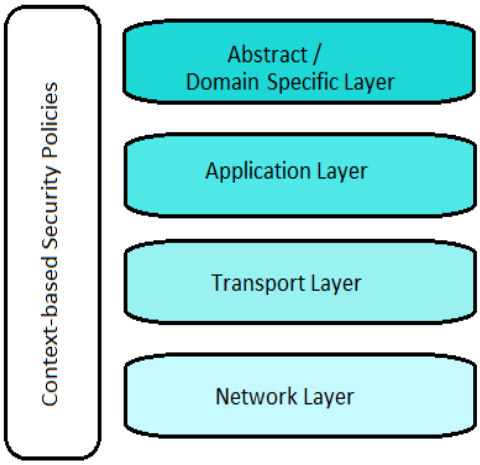

Figure 2. Security at different levels

A. *Types of Policies*

In fact, in our system there are two types of software policies. Authorization policies, as defined in [4], are rules

that are usually enforced in access control systems. In our case *authorization* policies are rules defined by the service providers to determine whether an action is authorized if a certain set of conditions is fulfilled. On the other hand, *obligation* policies are defined either by the security system or by users. They refer to actions that are to be enforced when a set of predefined conditions is fulfilled. Also, the obligation conditions are triggered by a change in the context in opposition to authorization policies that are only triggered by incoming external requests (from service clients). An incoming request is itself considered as a change in the context. To make things clear, an example of an obligation policy would be one that obliges all service providers to request authorization from the security system to perform a certain action whenever they receive a request.

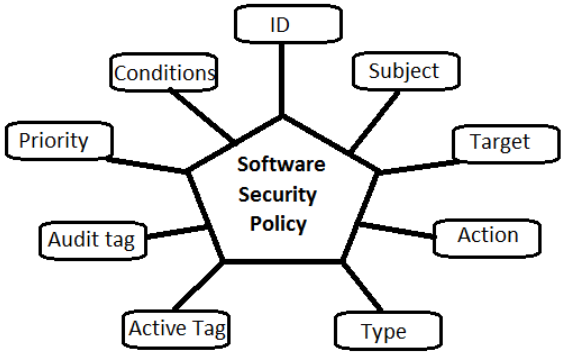

Figure 3. Structure of the policy

B. *Structure of Policies*

Several policy specification languages exist in the literature. We opted for ponder as a basis to model our policies because it is appropriate for quickly changing environments. This is due to the way policies are represented. In fact, policies could be represented using XML which facilitates the editing, modification and use of the policies [5] [7]. However, even though we were inspired by Ponder [7], while designing our policies, the most important concern was to enable service providers to express the business rules that they work with and context information. In Figure 3 we present the structure of the policies that we designed.

The first attribute of a policy is the *policy ID*. It is a number unique to every policy. In fact, this number is the only policy attribute that is assigned by the security system and not the policy owner. Assigning an ID helps in the operations of search. The next attribute of a policy in our system is the *type*. As specified previously, our system handles two major types of policies namely: obligation policies and authorization policies. The type of policies is very important when it comes to handling requests and notifications (changes in the context). In the case of requests, only authorization policies are used, while in the case of a notification, only obligation policies are used. Policies are either, system policies that are set by the system administrator, service policies that are set by services when they register to the security system, or mobile users' policies that are also set by the users when they enter the visited environment. Mobility is in fact the major reason behind the choice of a policy based security management system as it allows mobile users and to carry with them their policies.

The next attribute in every policy is the *subject*. This entity is extremely important as it is the one that has the ability to enforce the policy's action. After that, comes the *target* which is the entity on which the policy's action is enforced. The *action* of the policy is also an attribute of the policy that we defined. In most cases the action is a call for a method that belongs to the target. This is another point that makes this system usable as the service provider does not need to change anything in its own configuration. It only needs to provide this system with policies containing the actual method calls that it uses.

The *priority* of the policy is an important attribute and plays a major role in the system's behavior. As a matter of fact, it is only by using the priority attribute that we can solve the problem of having two or more conflicting policies. The *audit* and the *active* tags are two other policy attributes. The audit allows the system to keep track of triggered policies and the context of its triggering. Using this information, the system enforces accountability as a main security aspect provided by this system. The active tag specifies if a policy is active or not, so that it is taken into consideration when evaluating policies or not.

Finally, one of the most important attributes of the policy is the set of conditions. There was a need for a condition set that could be easily modified and that could allow for expressing conditions in a simple manner. Two decisions have been taken: the first one concerns the use of first order logic which allows combining a set of conditions using AND, OR, and NOT. The second decision concerns the values contained in the conditions. In order to be able to deal with all possible comparisons, three comparison operators were used namely: equal, greater, less. The structure of the condition set is shown in Figure 4.

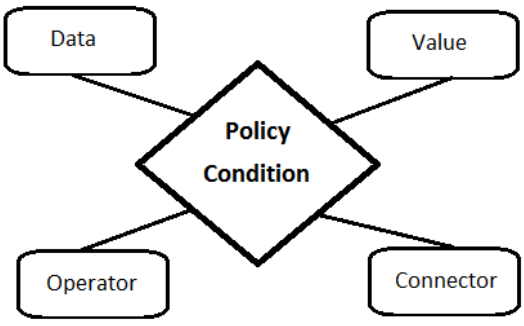

Figure 4. Condition set structure

 ISBN: 978-1-61208-171-7

C. *Context-driven Policies*

From the structure of the policies described above it is clear that the context information that will be included in the policies will be part of the condition set. In our case we consider that context information is time, location, and user's identity that refers to his profile. In the example of policy shown in Figure 5, the context information included is time that is represented by the year and the hour, the location where we have the choice between two locations, and the role of the user which needs to be provided by the user profile server.

```
</policy>
<policy id="1">
  <type>"Authorization"</type>
  <subject>"printer agent"</subject>
  <target>"printer_EP"</target>
  <action>"print document()"</action>
  <priority>"5"</priority>
  <audit>"yes"</audit>
  <active>"yes"</active>
  <conditions>
    <condition>year less 2013 AND</condition>
    <condition>year greater 2009 AND</condition>
    <condition>hour greater 1 AND</condition>
    <condition>location equal lab11 OR</condition>
    <condition>location equal lab7 AND</condition>
    <condition>doc_type equal pdf AND</condition>
    <condition>doc_size greater 10000 AND</condition>
    <condition>Role equal Graduate_Student AND</condition>
    <condition>Encryption_key greater 64</condition>
  </conditions>
</policy>
```

Figure 5. Policy example

IV. POLICY BASED SECURITY SYSTEM

A. *Policy Management Component*

Even though the applications that use policy-based management systems might seem different, the architecture of the policy management component remains the same. As explained in [13], the policy management component is mainly composed of 3 entities namely the PDP (Policy Decision Point), the PEP (Policy Enforcement Point), and the PIB (Policy Information Base). The role of the PDP is to take the decision on whether to allow an action or not based on the request's details and the policies available in the PIB. The PIB is a database that contains all the policies. Once an action has been selected, the PDP sends a message to the PEP that is responsible of enforcing the action on the target. In the next section we show how this core system has been integrated to our security system. The implementation of the PDP, PEP, and PIB are specific to our system as we have defined our own policy structure.

B. *Policy-Based Security System Architecture*

Figure 6 shows that the policy-based security management system is composed of three major components: the security engine, the repositories, and the policy enforcement point. All the components of the system take their data from the repositories. The system interacts

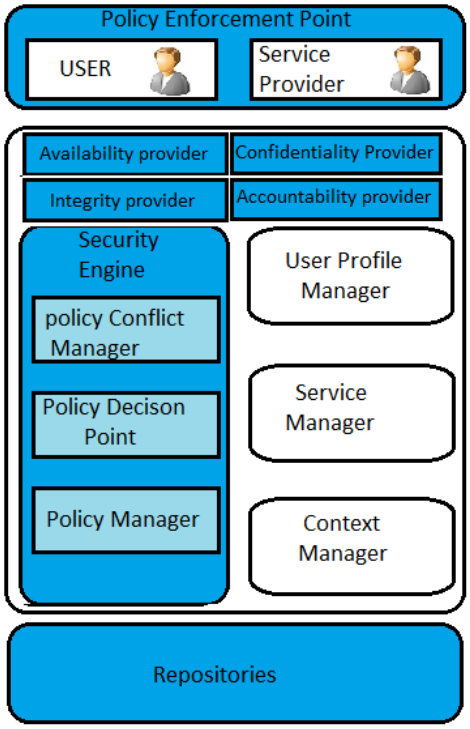

Figure 6. Policy-based security system architecture

with users and service providers through wrapper entities that are the PEPs in our case.

The *Repositories* component contains all the data repositories. First, there is the entity repository that contains all the information about the entities, such as the locations known to our system, the users, the set of activities, etc… Then, there is the context repository; it contains all context information that is of use to our system such as the time (year, month, day, hour) that is provided by our system itself, and other context information that is provided by the context aware platform implemented in our research lab [6]. Also, there is the actions log that contains a log of every policy that has been triggered, the necessary information to help provide with accountability such as the identity of the requester, whether it is an obligation policy or an authorization one, and the subject and targets of the policy. Another repository is the requests repository; it contains all the requests that have been sent to our system. It also allows the system administrator to keep track of the identity of the requesters and hold them accountable in case of problem. Finally, the last repository is the policy repository. It contains all the policies being used in our system. This means that it contains both obligation policies and authorization policies. An important note is that we have managed to keep the same format for both types of policies.

The *Security Engine* is the component where all policy manipulations are done. It contains the policy manager that is responsible for reading the policies from the policy repository and organizing them in such a way to be used by

 ISBN: 978-1-61208-171-7

the two other components, namely the policy decision point and the policy conflict manager. The policy manager is the only component that accesses the PIB. Therefore, it is also responsible for updating the policy set when new users and new service providers register with the system. The policy conflict manager sorts the list of policies in increasing order of priority. Therefore, even though many policies may be triggered by the same request only the last one to be triggered will be taken into consideration due to the fact that it bears the highest priority. The policy conflict manager will go through all the policies that are relevant to a certain event. Whenever it finds a policy that needs to be triggered (when there is a match with the set of conditions) it keeps it in memory. Therefore, if there is another one that needs to be triggered it will erase the first one that was kept in memory. Finally, as the conflict manager had ordered all policies by priority and starts from the lowest priority up, the last policy, available in memory, is the one that will be triggered. Finally, the last component is the policy decision point. This is the most important and critical component of the system as it is responsible for evaluating the policies and deciding whether a policy's action is to be triggered or not. The policy decision point is triggered either by an incoming request that is external to the system, or by an internal event that is a notification from the context manager of a change in the environment's context.

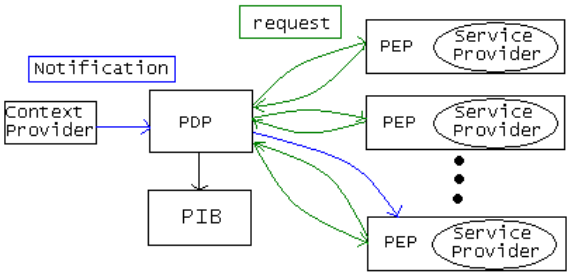

Figure 7. Notification / Request triggering of the PDP

In the first case, an incoming request, the policy decision point goes through the authorization policies specific to the target of the request and triggers the action of the policies specific to that target. In the case of a notification from the context manager, it loads all obligation policies and checks if policy conditions are satisfied for its action to be triggered.

The policy enforcement point component has necessary access rights to perform the action that is specified within a policy. The user or the service provider provides all method calls that are necessary to perform actions stipulated in its policies at registration phase.

Another part of the system contains the availability provider, the integrity provider, the accountability provider and the confidentiality provider. This part is abstract. In fact, it shows the different security services that are provided by the system. Its different components are achieved through the combination of the work of both the policy management engine and the context management engine. Every service provider / user registered in our system provides its own set of policies. These policies reflect the level of security that is aimed by the service provider. For example, the condition set of the policies provided could include context information, such as the time, location, identity, role that the requester must provide. In addition, the type of authentication required could be specified in the policy set. For instance, is it only a system authentication that is needed, or a service authentication, or both. Our system also allows for the service provider to request some other type of access control that is not defined in it. An example would be requiring a digital signature from a third party. All these access control methods do provide the users of the system with Integrity, Availability, and Privacy [12]. Finally, the fact of keeping a log of all requests and policies that are triggered certainly enforces Accountability.

The sequence diagram in Figure 8 gives a better idea of how the different components of the system interact. Once the user issues a request to the service provider, its wrapper entity (PEP) intercepts it and sends it to the policy decision point. After the policies are loaded by the policy manager, the policy decision point checks which ones will be triggered. In the case where context information is needed, a request is sent to the context management entity. After the conflict is resolved, the appropriate policy is enforced on the target (service provider).

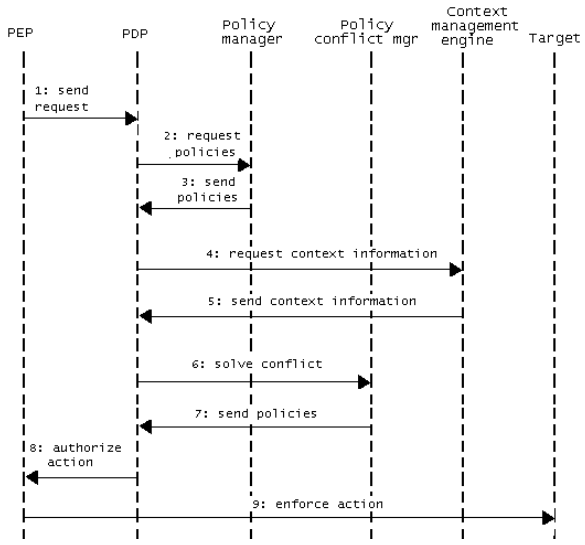

Figure 8. Request handling sequence diagram

The last components that are shown in the architecture, the user profile manager, service provider manager and the context manager, are outside our system. Figure 9 shows how our system fits within the big picture of the project being conducted in our research laboratory related to context aware platform to Support Mobile Users with Personalized Services [6].

 ISBN: 978-1-61208-171-7

V. MyCampus Service Provisioning

The scenario presented in this paper takes place in Al Akhawayn University's campus. One location in the campus is the computing lab that allows students to have access to printing, scanning, and internet connection services.

A student S1 gets into the location and requests some services. The first service that he requests is printing a document. Only registered users have access to the services offered within the environment. The registration step consists of providing the system with the information that the user wants to share, and most importantly providing the system with the user's policies.

In the case the system does not find any policy that matches the user's request then the default policy, which does not allow any operation, is triggered In order to avoid any type of conflict with user policies, the default one bears the smallest priority

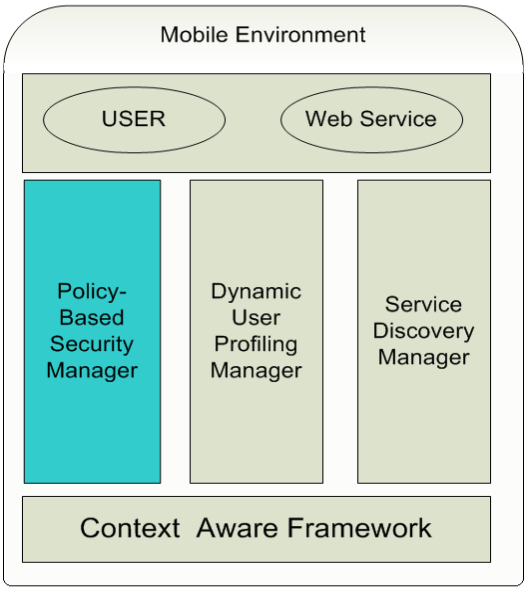

Figure 9. Context aware service provisioning in mobile environments

Another important system policy is the one that obliges the service provider to go through the security system in such a way that no request bypasses the security system. This policy bears the highest priority.

A sample of printing service policies is shown in Figure 10. The printing service wrapper receives the user request in the format shown in Figure 11. It forwards it to the policy decision point. The PDP requires from the policy manager the list of all authorization policies. A linked list of all policies which are present in the PIB is created. After using our conflict management technique, the ordered set of authorization policies is sent to the policy decision point. In terms of implementation, a simple sorting algorithm is used and all the objects of the linked list are sorted by priority. The PDP, then, before being able to compare the elements of the request and those of the policies, makes use of an XML parser to extract all elements of the request and those of the policy being checked. If we observe the list of authorization policies in Figure 10 we notice that there are two authorization policies from the printer service provider. The first policy in the list will be dismissed because its target is not the printer agent. The second policy will be considered and its condition set will be checked against the specifications of the request. The first condition will be satisfied because the system will use its context provider and know that the year is 2011 which is less than 2012 and greater than 2009. Then it will check the next conditions and find out that they hold because the document type is PDF, the size is greater than 10000, the location is lab7, and finally we assume that the request has been sent after 6PM. Therefore, as no more policy conditions are to be checked, the policy will be kept in memory and its action not yet triggered.

```
<policies>
    <policy id="4">
        <type> "obligation" </type>
        <subject> "grade server enforcer" </subject>
        <target> "grade server agent" </target>
        <action> "shut_dowwn()" </action>
        <priority> "7" </priority>
        <audit> "yes" </audit>
        <active> "yes" </active>
        <conditions>
            <condition> year greater 2009 AND </condition>
            <condition> hour equal 22 </condition>
        </conditions>
    </policy>
    <policy id="1">
        <type> "authorization" </type>
        <subject> "printer enforcer" </subject>
        <target> "printer agent" </target>
        <action> "print_document()" </action>
        <priority> "5" </priority>
        <audit> "yes" </audit>
        <active> "yes" </active>
        <conditions>
            <condition> year less 2012 AND </condition>
            <condition> year greater 2009 AND </condition>
            <condition> place equal lab11 OR </condition>
            <condition> place equal lab7 AND </condition>
            <condition> doc_type equal pdf AND </condition>
            <condition> doc_size greater 10000 AND </condition>
            <condition> AccessControl.Approval equal t_approval
                                                    </condition>
        </conditions>
    </policy>
    <policy id="2">
        <type> "obligation" </type>
        <subject> "printer enforcer" </subject>
        <target> "printer agent" </target>
        <action> "sytem.authenticate()" </action>
        <priority> "8" </priority>
        <audit> "no" </audit>
        <active> "yes" </active>
        <conditions>
            <condition> year greater 2010 AND </condition>
            <condition> hour equal 1 </condition>
        </conditions>
    </policy>
```

Figure 10. Set of policies in the system

```
<request id="1">
    <subject> "printer enforcer" </subject>
    <target> "printer agent" </target>
    <action> "print_document()" </action>
    <specifications>
        <specification> doc_type pdf </specification>
        <specification> doc_size 10004 </specification>
        <specification> location lab7 </specification>
        <specification> Role Graduate_student </specification>
        <specification> AccessControl.approval t_approval
                                              </specification>
    </specifications>
</request>
```

Figure 11. Request sent by the user

The system then enforces the triggered policy via the PEP, therefore, the document can be printed. Next is the insertion in the log of a header stipulating that the policy's action has been triggered. It is done because the audit tag in the policy is set to yes. Checking the system log allows identifying the perpetrators, or the conditions under which the felony was perpetrated.

VI. CONCLUSION

Throughout this paper we have shown that policies represent an efficient way to provide with security at different levels for the following reasons:

- Policies allow for mobility because a user can take with him a set of policies wherever he goes.
- Policies allow for adaptability, as the user does not need to adapt to any environment, only the policies he provides manage his interactions.
- Policies allow users to specify the security tools/mechanisms that they want to use.
- Policies allow users to incorporate context information

Currently, we are investigating the use of Personal Area Network (PAN) as the entity that will represent a user with his profile, preferences, and a set of policies. The PAN is then going to compose/decompose with existing networks in smooth and ambient manner as the user moves from one location to another by means of policies.

REFERENCES

[1] M. Ouanaim, H. Harroud, A. Berrado, and M. Boulmalf, "Dynamic user profiling approach for services discovery in mobile environments", Proceedings of the 6th International Wireless Communications and Mobile Computing Conference ACM New York, NY, USA, 2010, pp. 550-554, doi>10.1145/1815396.1815523

[2] Q. Wang and T. Zhang, "Sec-SNMP: Policy-Based Security Managementfor Sensor Networks", in *Proc. International Conferenceon Security and Cryptography (*SECRYPT*)*, 2008, pp. 222-226

[3] K. Boudaoud, Z. Guessoum, C. McCathieNevile, and P. Dubois, "Policy-based security management using a multi-agent system", *HPOVUA'2001*, Berlin, Germany, June 2001

[4] C.A. Ardagna, E. Damiani, S. De Capitani di Vimercati, and P. Samarati, "Towards privacy-enhanced authorization policies and languages", in: Proc. of the 19th Annual IFIP WG 11.3 Working Conference on Data and Applications Security, Storrs, CA, USA, August 2005, pp. 16-27.

[5] G. Tonti, J. M. Bradshaw, R. Jeffers, R. Montanari, N. Suri, and A. Uszok, "Semantic web languages for policy representation and reasoning: A comparioson of KAoS, Rei, and Ponder", in Proc. International Semantic Web Conference, 2003, pp. 419-437.

[6] Y. Bouzid, H. Harroud, A. Berrado, and M. Boulmalf, "Context-Aware Platform to Support Mobile Users with Personalized Services". in Proc. WINSYS, 2009, pp. 153-158.

[7] N. Damianou, N. Dulay, E. Lupu, and M. Sloman,"The Ponder policy specification language", in Proc. POLICY, 2001, pp.18-38

[8] C. Brookson, "GSM (and PCN) security and encryption", http://www.brookson.com/gsm/gsmdoc.htm, 1994 <retrieved: 10, 2011>

[9] R. Sandhu, and P. Samarati, "Authentication, Access Control, and Audit", ACM Computing Surveys, 1996, Vol. 28, No. 1, pp. 241-243.

[10] F. Almen´arez, A. Mar´ın, C. Campo, and C. Garc´ıa, "PTM: A Pervasive Trust Management Model for Dynamic Open Environments", in FirstWorkshop on Pervasive Security, Privacy and Trust PSPT'04 in conjunction with Mobiquitous 2004.

[11] M. H. Warfield, "Security Implications of IPv6", Internet Security Systems, 2003.

[12] K. Wrona, and L. Gomez, "Context-aware security and secure context-awareness in ubiquitous computing environments", XXI Autumn Meeting of Polish Information Processing Society Conference Proceedings, 2005, pp. 255-265.

[13] R. Yavatkar, D. Pendarakis, and R. Guerin, "A framework for policy based admission control", Informational RFC (RFC 2753), January 2000, pp. 1-20.

 ISBN: 978-1-61208-171-7