A Semantic Based Certification and Access Control Approach Using Security Patterns on SEAGENT

Fatih Tekbacak
Department of Computer Engineering
Izmir Institute of Technology
Urla, Izmir, Turkey 35430
Email: fatihtekbacak@iyte.edu.tr

Tugkan Tuglular
Department of Computer Engineering
Izmir Institute of Technology
Urla, Izmir, Turkey 35430
Email: tugkantuglular@iyte.edu.tr

Oguz Dikenelli
Department of Computer Engineering
Ege University
Bornova, Izmir, Turkey 35100
Email: oguz.dikenelli@ege.edu.tr

***Abstract*—In this paper, we propose a security infrastructure for communication between agents adaptable to FIPA security specifications by employing security patterns and semantic based policy descriptions. Security patterns are used as a generalized approach for generating security based services. This paper analyzes the authentication and semantic based access control among agents by using the security patterns.**

I. Introduction

Multi-agent systems(MAS), which communicate in an open environment like Internet, face safety and security problems. Therefore, MAS should have some strategies, policies and mechanisms for confidentiality, integrity, authentication, non-repudiation [1], [2], [3]. This paper proposes a security infrastructure intended for a FIPA compliant multi-agent system namely SEAGENT [4], and provides a solution by using security patterns with semantic web for policy based approaches. SEAGENT agents demanding for secure access will be utilizing *Agent Authenticator*, *Agent Certification Authority(ACA)* and *Access Controller* patterns that have been explained in Tropos methodology [5].

II. Background

SEAGENT is a P2P Java framework for writing Semantic Web enabled Multi-Agent applications [4]. The main objective of SEAGENT project is to develop an agent framework for constructing FIPA-compliant multi-agent platforms that work on semantic web environment. It's communication module supports current web based communication protocols both for intra-platform and inter-platform communication.

Agent oriented software engineering is one of the most natural ways of characterizing security issues in information systems. This approach allows developers first to model the security requirements in high-level and then incrementally apply these requirements to security mechanisms as services(or agents) in the multi-agent systems [6], [7].

This paper uses the approach that has been detailed in [7]. The authors of Tropos merged the advantages of the agent oriented programming and security patterns paradigms by applying both of them in the Tropos methodology. Secure Tropos extends the agent oriented software engineering methodology by providing a set of security-related concepts and processes to allow developers to consider security issues throughout the development stages. By using this methodology, agent oriented concepts could identify a set of security requirements needed by the system and these requirements can be transformed to a design with the use of security patterns.

Since SEAGENT is a FIPA-based multi-agent system, FIPA specifications have been followed throughout this work. First, all agents must register to Agent Management System(AMS). AMS has the responsibility to monitor the lifecycle of agents and agents must inform AMS about their platform related actions(register, deregister and so on). Second, software agents must also register their service descriptions to Directory Facilitator(DF). During this process, a masquerading agent should be prevented from registering its services or service descriptions and at the end damaging the platform. Third, there is also a communication layer called Agent Communication Channel(ACC) which transmits agent communication messages. Those messages should have confidentiality, integrity, authentication and non-repudiation properties according to FIPA security specification. This specification introduces Agent Platform Security Manager(APSM) which defines security issues of AMS and requires a PKI for registering agents. The specification of FIPA for message-based communication security uses Agent Communication Language(ACL) envelope added properties.

III. Security Patterns

Security is often an afterthought functionality in system design and implementation. The enterprise context and requirements that drive system security are often not addressed explicitly and are not incorporated into system architectures. The desired approach is to begin to address security together with the system design rather than the repair-service approach [3].

The basic idea behind patterns is to capture expert knowledge in the form of documentation with a specific structure containing proven solutions for recurring problems in a given domain. In particular, security patterns can be more useful when people responsible for systems have little or no security expertise.

In this paper, *Agent Authenticator*, *Agent Certification Authority* and *Access Controller* patterns have been examined in detail. The remaining patterns defined in [7], namely *Agency Guard* and *Sandbox* patterns are out of scope of this paper.

A. Agent Authenticator

Agents must be authenticated in the platform before they are allowed for intra- and inter-platform communication. The *Agent Authenticator* pattern determines the authentication process of agents in an agency. The authentication process is implemented by using digital signatures generated with agent's or agency's secret key.

The advantage of *Agent Authenticator* is to check the agent's identity before it involves in any communication within the agency. Authentication of the requesting agent could be verified by *Agent Authenticator*. This pattern also prevents implementation of different authentication mechanisms for different agents.

The disadvantage of this approach is that *Agent Authenticator* becomes a central point. So that when the *Agent Authenticator* crashes, the whole system would be under risk.

The design of the agent authentication model in SEAGENT by using SEAGENT Plan Editor is shown in Figure 1. *SupplyPrivateInformation* behaviour takes the owner policy and key pair of the agent. Outcome of this behaviour is passed to *SupplyPrivateKey* action. Private key of the agent is used for creating its digital signature for authentication issues in *SupplyDigitalSignature* action. *AuthenticationManager* action is the connection point of *AgentAuthenticator* with *AgentCertificationAuthority* to obtain the related request parameters by the issued certificate of the agent. These parameters could be subject, issuer of the certificate, validity time of the certificate to validate the digital signature and apply the authentication rules to decide agent's authentication for the communication. *AgentAuthenticator* lastly takes its external provision as RequestParameters from *AgentCertificationAuthority* and passes these parameters to *AuthenticationManager* by provision inheritance. So the Agent Authentication mechanism halts by the decision of this behaviour's planning activities and the outcome of *AuthenticationManager* causes the authentication decision for the requestor agent.

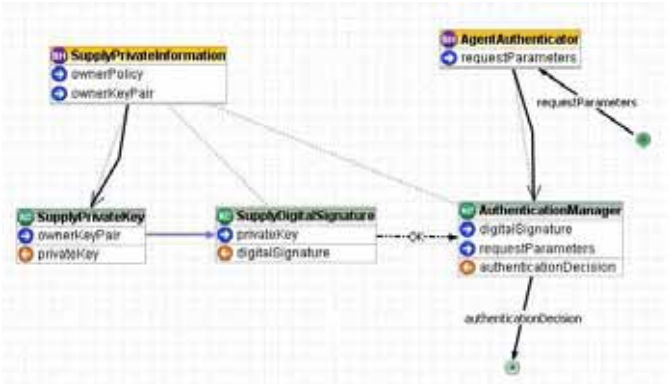

Fig. 1. Agent Authenticator Plan in SEAGENT (General View)

B. Agent Certification Authority

In a trusted environment, every agent is required to possess a certificate which includes a public key. The corresponding private key is stored by the agent in a secure manner. These agents verify their identities by generating digital signatures using their own private keys.

The advantage of *Agent Certification Authority* is the ability to verify a requestor agent. So that the indicated public key is proven to be really used for the communication. This pattern helps to design an appropriate signature verification mechanism to satisfy identity and authentication requirements for a specific domain or situation [3]. The disadvantage of this pattern could be scalability problem when a lot of agents want to request for certification.

C. Access Controller

This is a pattern that restricts agent access to resources. Agents with various privileges can exist in an agency. Agents requesting for a resource could be denied or accepted according to the requested action. Agents in the agency could access the resources(or other agents) according to the response of *Access Controller*. These responses have been sent to the agents with the indicated privileges. If there is an agent's resource requirement instead of access to an agent, *Resource Manager* behaves as a helper service to the *Access Controller* and accesses the related agents' resources.

The advantage of *Access Controller* is the usage of different policies for different actions. In Figure 2, the ACA gives *Agent2* to *send-to* privilege for communicating with Agent1. These privilege types could be obtained from FIPA-ACL based message envelope by different SecurityObjects. If the *Agent2* tries to take *send-to* privilege, the SecurityObjects for different agents could determine the acceptance or denial of the message with inform or refuse communicative acts in FIPA. There is a disadvantage of this pattern that the crash of the *Access Controller* makes the access protocol unusable in agency.

```
:envelope (
  :destination(...)
  :return-address (...)
  :Security(
     : type authorize
     : permission message
     : policy send-to
     : user Agent1)
  :confidentiality high
  :integrity high)
:message
(inform
  :sender    SCA @seagent
  :receiver Agent2@seagent
  :ontology ....
  :content .....)
```

Fig. 2. FIPA ACL Message Example with Security Access Information

IV. Software Agent Certification

The proposed approach attempts to employ security patterns for model driven design with reusable code and suggests utilization of semantic data on certification and policy based agent access models. It also defines a message extension for a new element that describes a form of the message security.

Essential security services explained in [8] are presented in layers.

ACA is a security wrapper for the system that dominates the protocol steps and supplies them to the agents. During Agent Certification Authority [5] process time, ACA enables both sender and receiver agents to negotiate security parameters and then on agents will communicate using the negotiated values. This negotiation also helps to decouple the multi agent system from selected security approach.

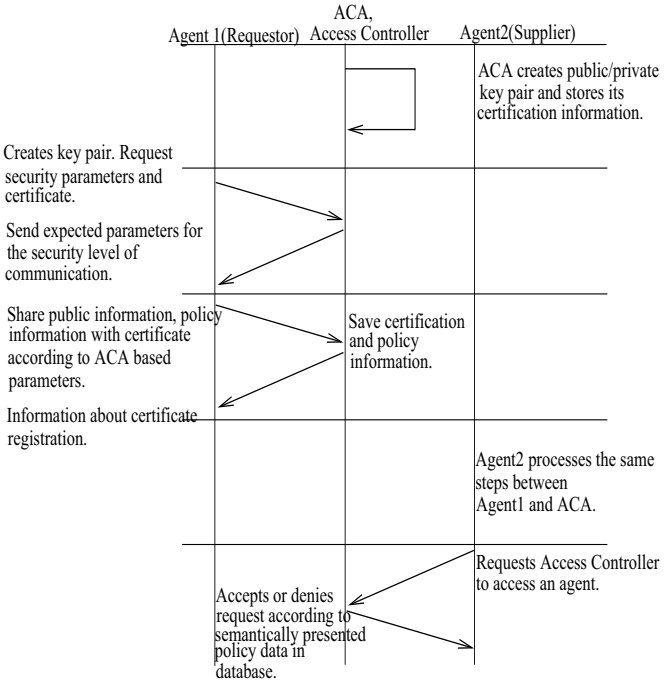

Fig. 3. Steps of Agent Certification Mechanism

The software agent certification steps have been illustrated in Figure 3. First, ACA creates public/private key pair and stores its certification information(issuer, subject, owner, validity, public key with key algorithm information, signature information, default certificate policies and policy mappings). The storage process is based on XML data for syntactic data compatibility and used by all agents. By default, authentication across agents will be accomplished by the same algorithm as the intra-platform communication. This authentication mechanism will be based on ACA as mandatory authentication and access control approach between agents. Because the access information and related certification information for supplier agents have to be supplied to the requestor agents by ACA.

If ACA accepts the requests of the *Agent1*, it sends expected parameters for the security level of communication. ACA accepts the certificate and registration information with policy data from *Agent1* and ACA saves certification information in XML and policy information in OWL(Web Ontology Language) format to its database. Then it informs the requestor agent. The semantic data for policy information aims to collect the policies in a tractable way by the Access Controller. ACA shall contractually require that the subscriber indicates acceptance or rejection of the certificate following its issuance, in accordance with procedures established by the ACA.

Agent1 creates its key pair and stores them as explained in the first step of ACA. Then this agent requests security parameters and certificate from ACA. All private keys and other security related data have to be available to their owner only. Data may not be accessible to other agents (even the agent platform). Every agent has to keep its private data secured but the platform based public information with certificate could be shared between agents according to ACA based parameters. So ACA will send certification information to the agents if the requesting side has the right to communicate in the platform. The certificate policies for agents are initialized in ACA by the security engineer of the system.

Agent1 prepares certificate and requests to register it with semantic policy information. Then the communication of *Agent1* with ACA ends for the certification and semantic data exchange.

The access information have been stored as semantic information shown in Section V. This access information is sent and received with a SecurityObject. The SecurityObject can be included as user-defined slot into the envelope (e.g. X-Security) as used in [8], or, if standardized by FIPA, as an optional slot (e.g. Security). Furthermore, the slot containing the SecurityObject can contain a set of SecurityObjects, for different security attributes applied to a message. The approach told in this paper as adding Security slot to the message Envelope.

ACA accepts or denies this request according to semantically presented data in its database. Information message from ACA to *Agent2* with security slot for access information in a FIPA-ACL message example similar to [9] is shown in Figure 2.

V. SEMANTIC BASED AGENT ACCESS CONTROL MECHANISM

After defining agent certification process details of *Access Controller* mechanism have been explained in this section. First, access control policy includes a set of rules that associate some credentials to use capability of a right. So that the issues with the specified credentials could supply the capability. Credential is any property associated with an entity. When the entity is suitable to the policy rules in the system, the required action for this entity could be populated [10].

All agents have to digitally sign all service requests for AMS, DF and other agents. As there is a public/private key pair for each of the agent, the agent can be thought as accountable by the platform. So that when an agent wants to register to the platform, its credentials have to be checked by *Access Controller* of which decision is based on security policies that have been defined in the platform. These policies could be defined in two levels: platform level and agent level [11]. Platform level policies control the requests for AMS and DF of the platform. Agent level access policies specify who can access the services of the specified agent. A simple message that assigns a right to an agent is shown in Figure 2.

When the certification steps as explained in Section IV have been constructed, AMS controls the validity of the certificate by the default certificate authority in the platform. For organization wide certification, certification path could be chained and the verification step could be processed by the help of *Agent Certification Authority*. If the certificate is valid, AMS restores agent's policies that have to be checked during the communication with AMS. All of these policies are in *Access Controller*'s database by default. AMS and *Access Controller* always communicates with each other to inform the changing policies in the autonomous environment. So when *Access Controller* has been crashed for a short time, AMS based policy rules could still be applied by AMS with its internal policy engine. AMS and DF have a list of conditions that an agent must satisfy in order to contact a particular agent or use a particular service. While AMS and DF have to know the access privileges of agents in [11], *Access Controller* mechanism have to access and know their rights. So that the protection from the threat could be applied in a central place.

Agents could register their services in open or private ways. In open way, the only DF based policies have to be applied for the agents' access to the service. In private way, *Access Controller* communicates with DF and access control mechanism have been processed for the owner of the service. For the verification of rights, a service agent expects all the credentials from the requestor agent at the time of the request in order to use its services. The service agent will check its internal knowledge base and ask for *Access Controller* to give permission to the requestor agent. According to the allowance of the semantic policy information, request could be answered in a positive way. Otherwise, request would be denied until the requestor agent has the suitable credentials to call this service.

VI. Conclusion

In this paper, security patterns that have been used in multi-agent systems have been considered. Also policy based access control has been determined that each entity is able to specify and process policy by help of *Access Controller* or by itself for security and privacy. In future, the specifications of policies are planned to be fully constructed in declarative manner and the ACL based messages to be considered in a semantic manner. With the help of policy based semantic language, the distributed policy management could be supplied better by inter-platform communication by the platforms that use same ontologies.

Future work will be based on the new version of SEAGENT's role based agent mechanisms. Role based agents would have their own role based access policies for *Access Controller*. Then these agents could have been prioritized according to their goals in the platform [12].

References

[1] Tomás Vlcek and Jan Zach, "Secure fipa compliant agent architecture draft," in *HoloMAS*, 2003, pp. 167–178.

[2] William Stallings, *Cryptography and Network Security Principles and Practices, Fourth Edition*, Prentice Hall, 2005.

[3] Markus Schumacher, Eduardo Fernandez-Buglioni, Duane Hybertson, Frank Buschmann, and Peter Sommerlad, *Security Patterns: Integrating Security and Systems Engineering*, John Wiley and Sons Ltd, 2006.

[4] Oguz Dikenelli, Riza Cenk Erdur, Ozgur Gumus, Erdem Eser Ekinci, Onder Gurcan, Geylani Kardas, Inanc Seylan, and Ali Murat Tiryaki, "Seagent: A platform for developing semantic web based multi agent systems. autonomous agents and multi-agent systems," in *AAMAS*, 2005, pp. 1271–1272.

[5] Haralambos Mouratidis, Michael Weiss, and Paolo Giorgini, "Modeling secure systems using an agent-oriented approach and security patterns," *International Journal of Software Engineering and Knowledge Engineering*, vol. 16, no. 3, pp. 471, 2006.

[6] Haralambos Mouratidis, Paolo Giorgini, and Gordon Manson, "When security meets software engineering: a case of modelling secure information systems," *Inf. Syst.*, vol. 30, no. 8, pp. 609–629, 2005.

[7] Haralambos Mouratidis, Michael Weiss, and Paolo Giorgini, "Security patterns meet agent oriented software engineering: A complementary solution for developing secure information systems," in *ER*, 2005, pp. 225–240.

[8] Petr Novák, Milan Rollo, Jirí Hodík, and Tomás Vlcek, "Communication security in multi-agent systems," in *CEEMAS*, 2003, pp. 454–463.

[9] Stefan Poslad and Monique Calisti, "Towards improved trust and security in fipa agent platforms," in *3rd Workshop on Deception, Fraud and Trust In Agent Societies*, 2000.

[10] Lalana Kagal, Tim Finin, and Anupam Joshi, "A policy-based approach to security for the semantic web," in *Proc. 2nd Int'l Semantic Web Conf. (ISWC 2003)*. 2003, p. 402418, Springer-Verlag.

[11] Lalana Kagal, Tim Finin, and Anupam Joshi, "Developing secure agent systems using delegation based trust management," in *In Security of Mobile MultiAgent Systems (SEMAS 02) held at Autonomous Agents and MultiAgent Systems (AAMAS 02)*, 2002.

[12] Salvatore Vitabile, Giovanni Milici, S. Scolaro, Filippo Sorbello, and Giovanni Pilato, "A mas security framework implementing reputation based policies and owners access control," Los Alamitos, CA, USA, 2006, vol. 2, pp. 746–752, IEEE Computer Society.