The Design and Implementation of Secure Event Manager Using SPKI/SDSI Certificate*

YoungLok Lee[1], HyungHyo Lee[2], Seungyong Lee[1], HeeMan Park[1], and BongNam Noh[1,**]

[1] Dept. of Information Security, Chonnam National University, Gwangju 500-757, Korea
{dogu, birch, hareup, bongnam}@lsrc.jnu.ac.kr
[2] Div. of Information and EC, Wonkwang University, Iksan 570-749, Korea
hlee@wonkwang.ac.kr

Abstract. In the ubiquitous computing environment new service components should be able to connect to networks at any time, and clients also should be able to use them immediately even without extra settings. Jini is one of the widely used middlewares today. Although event management is an essential component of ubiquitous middlewares, Jini is distributed without event management service. Accordingly, we design and implement the event manager based on Jini and suggest three methods in which only right event consumer can listen to the event using Access-Control Lists and SPKI/SDSI certificates. In the proposed method, our event manager controls the access of events by putting trust checking engine on Jini.

1 Introduction

We only assume a dim prospect of what the ubiquitous computing enabled future might perhaps bring and do not clearly know what is coming. For this reason, we do not know in what ways the ubiquitous computing scenario can be abused by ingenious attackers and do not know who the attackers are going to be.

The important thing is to identify which objects exactly we want to protect. The urgent object to be protected is the event, among many of those objects. There are various events ranged from low level signals generated by sensors to deduced valuable information in high level. Users in the ubiquitous computing environment should be able to adapt themselves to their current context information and high level information generated by these events. If user's event information is illegally achieved, someone can illegally generate a dossier that all the event information issued in the supermarkets, airports, bookstores, or banks are merged and also use it without user's acknowledgement. Above all, the event management is important because the illegal modification of generated events results in a wrong adaptation of users who use the event.

* This research was supported by the MIC (Ministry of Information and Communication), Korea, under the ITRC (Information Technology Research Center) support program supervised by the IITA (Institute of Information Technology Assessment).
** Corresponding author.

T. Enokido et al. (Eds.): EUC Workshops 2005, LNCS 3823, pp. 490 – 498, 2005.

Jini, the Home Network middleware helps new service components connect to Home Networks at any time and helps clients promptly use them without extra settings, and even in the case of service component upgrade, the existing client service can operate with no problems. However, there is no event service implemented in Jini. Instead, it provides JavaSpace, storage system based on object and attributes in order to use various service objects.

By modifying the JavaSpace, we develop the event manager to manage events and control the access of events in order that only right users with the authority granted by the event generator can take event.

This paper consists of as follows;

Chapter 2 shows related researches, and chapter 3 explains the design and implementation of event management system based on JavaSpace. Chapter 4 suggests three methods to control the access of events by event manager, using SPKI/SDSI certificates and ACLs(access control lists). Chapter 5, among those three methods, describes the prototype of the second method which is adaptable in actual life and compares other event manager services. Finally, chapter 6 makes a conclusion and further research works.

2 Related Researches

Among middlewares of ubiquitous computing environment are Gaia, M3-RTE, Aura[1,2,3], etc. Each has the similar structure in its event system, however, does not include the security.

One of middlewares is the Gaia of the University Illinois under active research. The event manager of Gaia[4] satisfies many of general needs in event management. The event manager distributes load among multiple machines. It creates event channel factories remotely on pre-configured machines whenever it decides that the existing event channel factories are already overloaded. The event manager is also responsible for recreating those channels if any supplier of events complains that the particular event channel no longer exists. In essence, the event manager keeps state for the channels it creates and recreates them if they crash, transparent to the event consumers. Event manager service implementation of Gaia makes use of CORBA event service as the basic event storage. Nonetheless, Gaia depends on the basic security policy in the event manager as well.

Another is the Context Observer of Carnegie Mellon University. It provides information about the physical context and report events in the physical context back to Prism and the Environment Manager. Prism and the Environment Manager are components of the Aura[3]. Context Observers in each environment may have different degrees of sophistication, depending on the sensors deployed in that environment.

3 Event Management System Based on JavaSpace

Jini[5] is a middleware in composing the home networking. The purpose of Jini is to accomplish “Network plug and Work”. Although new printer components are connected to network, Jini should be what clients immediately use it without extra setup. Also despite the upgrade of service components, it has no problem in running the existing client with no extra setting.

Although event management is a necessary component of ubiquitous middleware, Jini is now being distributed with no event management service. This section describes how we design and implement the event manager which manages events, using JavaSpace[5].

3.1 Process Procedure of Event Service

Our event manager (JS-EM : Event Manager based on JavaSpace) is made by modifying JavaSpace[5]. The procedure that the event consumer takes events generated by the event producer is as follows:

1. As JS-EM itself is registered as service in the Jini LookUp service, the event consumers or the event producers can search the JS-EM and use it. An event consumer registers herself to JS-EM as a listener of the event he or she is interested in.
2. JS-EM takes the stub of event listener for the communication with the event consumer through web server and by doing this, there accomplishes the channel between event listener and JS-EM.
3. The event producers write its events to the JS-EM.
4. Events written to the channel in JS-EM are transmitted to the event consumer via previously registered event listener.

Our event manager provides a model for decoupled communication among event consumers and event producers. It allows creating and deleting channels. And through our event manager service, the context management service is able to generate the high level context information.

3.2 Modified JavaSpace

JavaSpace is found through lookup service as similar to other services of Jini and used through Proxy. JavaSpace is in charge of saving objects. In order to use JavaSpace as event manager, some of problems should be solved in advance. If the event producer writes the event to JavaSpace through write( ) operation, one of APIs of JavaSpace, the event consumer takes the event through read( ) operation.

Because, however, this JavaSapce of the general structure does not fully play a role as event manager, we add necessary interfaces into JavaSpace and modify some Classes of JavaSpace. Our event manager made by using the modified JavaSpace will be reused with the enlarged function when we implement Prototype of our Event Manager in Chapter 5.

4 Methods of Secure Event Management Using SPKI/SDSI

Ubiquitous services must appropriately adapt to the context information of the user. In need of privacy protection and proper adaptation, context information should be generated from the accurate event information and only right possessor of the authority of the event should utilize it.

The ad hoc network environment introduces fundamental new problems. One is the absence of an online server, and another is secure transient association. Authentica-

tion is one of the most interesting security problems in ad hoc networking, because much of the conventional wisdom from distributed systems does not quite carry over. For solving these problems, we use SPKI/SDSI(Simple Public Key Infrastructure/Simple Distributed Security Infrastructure) certificates.

This chapter, after summarizing the name certificate and authorization certificate structure and certificate chain discovery algorithm, describes three methods suggested by us in order to control the access to the event.

4.1 SPKI/SDSI Name Certificate and Authorization Certificate

4.1.1 Name Certificate

SPKI/SDSI Name certificate is binding between subject and local name. Local name is defined at any rate based on public key of issuer. Name certificate consists of 4-tuple[6].

<Issuer, Local Name, Subject, Validity>

The principals are public-key in SDSI. Issuers sign certificate with his or her private key. Local name, the one that issuer hopes to bind with the subject, consists of public key of issuer and more than one principal. Name in SDSI is defined only locally. The issuer controls the name space.

Subject is a principal or a name, which is the target bound to the Local Name and simultaneously receiving the certificate. If subject has a name, then that name has no global meaning but is defined only by the principal whose name space it is in. The name can only be bound to a key by a name certificate issued by the principal controlling the name space. What is assigned to subject is public key or local name composed of more than one public key or more than one local name. Validity is the period during which this certificate is valid.

4.1.2 Authorization Certificate

Authorization certificate is the one that certificate issuer gives some other subject the right to access a resource, such as reading a file. Authorization certificate is the same with the name certificate excluding the authorization-tag which grants authority to the subject and it has delegation bit. Authorization certificate consists of 5-tuple as follows[7]. This certificate can be combined with other certificates to create a chain of authorization, and it can be verified when accompanied by a valid signature.

<Issuer, Subject, Delegation bit, Authorization-tag, Validity>

4.1.3 Certificate Chain Discovery in SPKI/SDSI

Certificate Chain Discovery Algorithm"[8] is the one that searches, in his certificate cash, the name certificates and authorization certificates related to the subject in ACL transmitted from the server. Generally, clients run this algorithm.

The inputs of this algorithm are ACL(access-control list) for a protected resource and a collection of SPKI/SDSI certificates that client store in her or his certificate cash. This algorithm determines whether a given principal or set of principals, represented by their public keys, is authorized to access the protected resource.

4.2 Secure Methods to Manage Events

This section suggests three methods to manage events and implement the prototype of the second method, by judging that it is appropriate to the current ubiquitous environment among three methods

4.2.1 ACL and Encoded Event

When the owner of the event producer first installs a sensor, it provides the sensor with ACLs in secure satisfactory solution – physical contact. The ACL is the same with security policy that an owner of the sensor provides authority to use her event to a principal. As the event producer issues events, it sends encoded events and ACLs to an event manager, and the events are saved in the channel matching event type. Since then, the event producer sends only the encoded event objects generated into the related channel, managed by event manager.

After the event consumer registers herself to the event channel, the event manager sends all listeners of the event the encoded event and the ACLs related to the event. Accordingly, the event consumer able to decode the ACLs can also decode the encoded event and takes an action for adaptation.

4.2.2 SPKI/SDSI Certificate Manager in Middleware

Clients in ubiquitous computing environment are gadgets of micro-sized and low battery and most of them with no computing power. Consequently, gadgets have limit on computing to check trust relations.

Our second method to securely manage events is that only the event consumer who has proper right for any specific event can have the right to register herself as the listener of that event to event manager. Accordingly, in order to register oneself as the event listener, the event consumer requests JS-EM to check whether itself has the right registration authority, or not by sending its possessing SPKI/SDSI name certificates, authorization certificates, and event listener objects. After the checking process is complete, if the event consumer is estimated as the authorized one, JS-EM registers it as the listener of the event, if not, JS-EM destroys the registration.

More details of this method are explained in chapter 5.

4.2.3 Event Consumer Verifying Trust Relationship

This method is most suitable for the event consumer in case of having computing ability. This method is similar to 4.2.2, but it is differ that the event consumer checks the trust relationship between subject of ACL and herself. The event consumer registers herself as an event listener on event channel in JS-EM related to the event. At first, the event producer sends ACLs to the channel that JS-EM manages. And after then, the event producer sends only plain event objects to the channel. If the event producer writes the event to the JS-EM, the JS-EM requests to verify the authority by returning the ACL related event channel to the event consumer. The event consumer inputs both of name certificate and authorization in his or her cash and just received ACL in “Certificate Chain Discovery” algorithm so as to verify that it owns the authority to bring out the event. If he finds a certificate chain, he sends the chain to the JS-EM then the JS-EM sends the related event to the event consumer.

5 Prototype Implementation and Analysis

5.1 System Implementation

Our prototype is implemented in Linux/Windows OS, JDK 1.3 and Jini 1.2 development environment. We implemented our event manager by modifying JavaSpace as described at chapter 3. In this chapter, we explain prototype in order for only right users to take the event. Our prototype checks whether the event consumer is authorized to access the event or not by communicating between event manager and certificate manager. We define and use ACLs and SPKI/SDSI certificates for this trust checking, and extend the event manager we already implemented. We make and add the LRM(listener registration manager) and SSCM(SPKI/SDSI certificate manager) into Jini.

ACL involves security policy, which the event producer delegates authority to the event consumers for receiving the event. When an event sensor is installed, owner of the sensor provides her sensor with ACLs describing event authorization policy. Since then, the owner of sensor can modify or add another ACLs if necessary. A JS-EM is registered as services in the Jini LookUp service, and the event consumers or the event producers search the JS-EM. Now they are able to use it.

The procedure that only the right event consumer takes the event generated by the event producer is as figure 1.

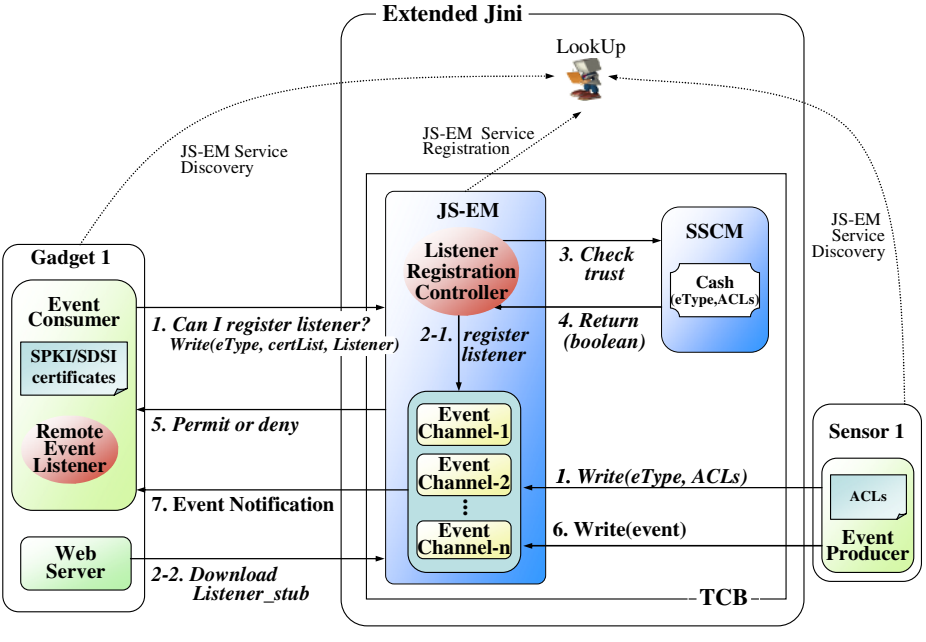

Fig. 1. Secure event management scenario in extended Jini

1. When installing, the event producer, sensor1, sends both ACLs and event type that she owns to the SSCM. Speaking in more detail, when sensor1 boots, sensor1 sends the trust verification information, a type of class objects containing ACLs and event types, to JS-EM. As SSCM was already registered as a listener to receive the trust verification information, JS-EM just notifies this information to the SSCM. Since JS-EM and SSCM are in TCB(trusted computing base) relationship, SSCM securely stores pair of the event type and ACLs into its cash. On the other hand if an event consumer, gadget1, wants to listen to event type E1, it will sends a chain of its certificate list, an event type E1, and own listener for that event to JS-EM.
2. LRC of JS-EM takes a role of registering listener of event type for the interim period and locks event objects of the event type not to be taken away for that time. JS-EM takes the stub of the event listener for the communication with the event consumer through web server and by doing this, they accomplish the channel between event listener and JS-EM. And then, LRC relays the event type and certificates to SSCM within TCB and requests to check whether it has the proper authority.
3. SSCM checks the authority of the event consumer by using the certificate chain discovery algorithm.
4. SSCM returns a boolean value as the result to the event consumer
5. If the returned Boolean value from the SSCM is true, LRC remains the listener as it is and sends a message of registration permission to the event consumer. If false, it sends a message saying “denied” and annul the registered listener.
6. Now sensor 1 issues an event object and writes it on JS-EM.
7. JS-EM notifies the events to the listeners of the event objects written by 6.

5.2 Structure of the Components in Our Prototype

When installing sensor, the owner of the sensor or the domain administrator saves ACLs by using secure ways, such as physical contact. When a sensor is activated, it takes procedures to send own ACLs and event type to SSCM via the event manager. The simple explanation of the components consisting of our prototype is as follows.

5.2.1 ACLs

The structure of ACLs also is composed of 5-tuple like authorization certificates. The example specified below is our ACL’s structure.

$$< K_{domain\_admin}, K_{gadget1}, 0, \text{“permit”}, ((05\text{-}06\text{-}01, 05\text{-}08\text{-}20) \wedge (13{:}50{:}10, 16{:}20{:}35)) >$$

This ACL means that gadget1 can take the sensor’s events during the valid expiration date. To send this ACL to SSCM, the event producers use the API function write( ) of JS-EM.

JS-EM.write(eventType, ACL-List, LeaseTime)

5.2.2 SPKI/SDSI Certificate Manager.

SSCM(SPKI/SDSI Certificate Manager) scrutinizes whether legitimate authorization link ranged from a principal of SPKI/SDSI certificates presented by the event consumer to the subject in ACLs, connecting to the event type stored in own cache is

existed. This test is achieved by using the certificate chain discovery algorithm. Then, SSCM returns a boolean value to the LRC.

5.2.3 Listener Registration Controller

LRC(listener registration controller) is needed to register only the listeners who have the authorized rights, so that only registered event consumers can listen to the events for given specific event types. Once LRC receives the event type, SPKI/SDSI certificates, and listener from a event consumer, it registers the event listener and requests the SSCM to check whether the event listener can take event objects of the event type or not. If LRC receives a true value, it keeps the listener registered. If not, it returns denial message to them and destroys the listener. The algorithm LRC performs is as Figure 2.

```
Algorithm: Listener Registration( ET, NCL, ARCL, LS )

Input:  ET,          /*  Event Type  */
        NCL,         /*  Name Certificate List  */
        ARCL,        /*  Authorization Certificate List  */
        LS           /*  Listener of one event type  */
Method:
   begin
      Boolean result;                                   /* permit or deny */
      result = SSCM.checkTrust(ET, NCL, ARCL);  /*request SSCM to check*/
      registerListener(LS);                      /*  Listener registration  */
      eventLock(ET);                  /*  Locking object of the EventType  */
          if( result == TRUE )                  /*  if she has right authority  */
               sendMessage( "Permitted Listener" ); /*  send Permit Message  */
          else                                  /* if she has not authority  */
               begin
                  sendMessage( "Denied Listener" );     /*send Deny Message */
                  unregisterListener(LS);       /* revoke the Listener registered */
               end
      eventUnlock(ET);                /* Unlocking object of the EventType  */
   end
```

Fig. 2. Listener Registration Algorithm

5.3 System Analysis

While Gaia event manager service uses basic event storage of CORBA, our JS-EM uses the modified JavaSpace. Gaia has not controlled the access to events. However, our service can do it by using ACLs and SPKI/SDSI certificates. Event Type is fixed in Gaia, but it is not fixed in our event manager. Both Gaia and ours allow multi event channels and the fact that those event channels are able to be distributed where multi-computing machines are common.

The table 1 is the comparison of our event manager service and other event service.

Table 1. The comparison of our event service and Gaia

	Gaia's event service	Our JS-EM
Basic event storage	CORBA	modified JavaSpace
Event access control	Not yet	Yes
Multi event channel	Supported	Supported
Event Type	Fixed	Unfixed
Loading channel	On multi machine	On multi machine

6 Conclusions and Further Work

By using JavaSpace, we implement event management system among middleware components suitable for ubiquitous computing. We also suggest three methods in order to manage events safely, and design and implement the second one among those three, because now there is a computing power limitation to gadgets or sensors.

For the purpose of controlling events, we expanded JINI by adding event manager and certificate manager, which are not existed in JINI. This is to provide events only to the event consumer with right authority by interactions between event manager and certificate manager. This paper focuses on where the streamline of event should be controlled rather than the performance of event manager

The common language for expressing security policy such as ACLs is important because it can be moved through networks. In order to solve this problem, policy-decision-point(PDP) module can be implemented by using XML. We also will implement PDP in SSCM who understands XACML(extensible access control markup language) rule context for event producers.

References

1. Manuel Román, Christopher K. Hess, Renato Cerqueira, Anand Ranganathan, Roy H. Campbell, and Klara Nahrstedt: Gaia:A Middleware Infrastructure to Enable Active Spaces. In IEEE Pervasive Computing(Oct-Dec 2002) 74-83
2. A. Rakotonirainy, J.Indulska, W.W Loke, A.Aaslavsky: Middleware for Reactive Components: An Integrated Use of Context, Roles, and Event Based Coordination, Lecture Notes In Computer Science, Vol. 2218 . Proceedings of the IFIP/ACM International Conference on Distributed Systems Platforms, Heidelberg(2001) 77-98
3. J. P. Sousa and D. Garlan: Aura:an architectural framework for user mobility in ubiquitous computing environments. In Proceedings of the 3rd Working IEEE/IFIP Conference on Software Architecture, Montreal, Canada(August 2000)
4. B.Borthakur: Distributed and Persistend Event System For Active Spaces. In: Master Thesis in Computer Science, Urbana-Champaign: University of Illinois at Urbana-Champaign(2002)
5. Philip Bishop and Nigel Warren: JavaSpaces IN PRACTICE, Addison-Wesley(2003)
6. Andrew J. Maywah: An Implementation of a Secure Web Clent Using SPKI/SDSI Certificates. Master of thesis, M.I.T, EECS(May 2000)
7. Carl M. Ellison, Bill Frantz, Butler Lampson, Ron Rivest, Brian M. Thomas, Taut Ylonen: SPKI Certificate Theory. RFC2693(September 1999).
8. Dwaine Clarke, Jean-Emile Elien, Carl Ellison, Matt Fredette, Alexander Morcos, and Ronald L. Rivest: Certificate Chain Discovery in SPKI/SDSI. Journal of Computer Security. volume 9, Issue 4(December 2000) 285-322