BOND SYSTEM SECURITY AND ACCESS CONTROL MODEL

RUIBING HAO, KYUNGKOO JUN, AND DAN C. MARINESCU
(hao, junkk, dcm@cs.purdue.edu)
Computer Sciences Department, Purdue University, West Lafayette, IN, 47907, USA

ABSTRACT

Bond is a message-oriented middleware for network computing on a grid of autonomous nodes. In this paper we overview the basic architecture of the Bond system and introduce the security model implemented in the Bond system. User programs as well as data are subject to the security constraints imposed by the local operating system. Bond objects are subject to the access control and the authentication model discussed in this paper. Secure Bond objects are allowed to create their own security agents to implement access control.

KEYWORDS

Middleware, Security, Access Control

INTRODUCTION TO BOND SECURITY

Bond is a message-oriented middleware for network computing. When completed, the Bond environment will consist of a distributed object infrastructure, a set of servers, a set of agents, and a set of user objects. A survey of the Bond architecture is presented in [1-4] and reviewed briefly here.

Servers are permanent, their life-time is rather long and provide system-wide services e.g., Directory Server(DS), Authentication and Software Distribution(ASD) Server, and so on. Agents are started upon request, work in conjecture with one or more objects, and disappear after completing their function. Examples of agents are Scheduling Agents and Security Agents[1].

Each Bond object is uniquely identified by a Bond Id. Each Bond executable consists of a local directory, a main thread of control, and a messaging thread with two mailboxes: an ***in*** box and an ***out*** box. Once an object is created it is registered with the local directory. Objects located at different sites communicate with one another as shown in Figure 1.

To locate a remote object **B**, object **A** sends a (**find**) message to the Directory Server, which in turn multicasts the request to all the members of the Directory Server Virtual Network (DSVN). Eventually one of the Directory Servers locates object **B**. Object **B** then sends back the information necessary to create the Bond Shadow. The *shadow* of an object is a light-weight communication abstraction which supports unidirectional communication. From this instance on, all messages from **A** to **B** are sent to the shadow of **B** which acts as a local proxy for the remote object. A Bond communication act consists of three steps: (1) Send the message to the local Bond shadow, which will perform format conversion and security checks, and then deposit the message into the ***out*** box of the local messaging thread at the sender's site; (2) Transport the message from the ***out*** box of the sender to the ***in*** box of the receiver; (3) The local messaging thread at the destination will get the message in the ***in*** box of the receiver, parse and deliver the message to its destination object.

Each Bond object provides a set of services. These services are available to other objects by means of message patterns called Bond *sub-protocols*[4]. For example, an object reveals its properties and allows other objects to modify them by means of the *property access sub-protocol*. Each sub-protocol is implemented by exchanging KQML messages. KQML messages, called performatives, allow to encode basic abstractions like asking, replying, achieving, subscribing or notifying[5]. So the most suitable approach to implement access control of Bond objects is to define and check the rights to use every type of KQML message. Each KQML message in Bond consists of a *performative* field, a *content* field and one or more *parameter* fields.

Bond uses a two-prong approach to system security. Bond objects are subject to the security model discussed in this section while each Bond executable running on a host relies on the security features of the local system including password, access rights, quotas, etc.

Bond is an object-oriented system, therefore we decided to support object level security [6][7]. We opted for a uniform model; the security model does not differentiate among system and user objects. We decided to allow each object to define and implement its own security, we only enforce a uniform interface to the security functions. An object may define its own security agent to implement its security policy. A group of objects may share a security agent. The mechanisms used to enforce security cover authentication and access control policies. The creator of a Bond object has the option to enforce security. Only a secure Bond object may request authentication and access control.

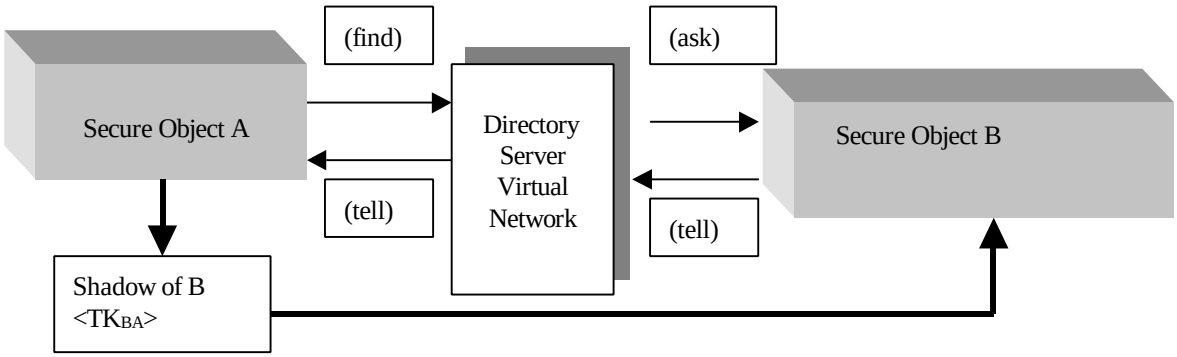

FIGURE 1. ACCESS CONTROL

To implement the access control model discussed in the next section we need to expand the Bond shadow to contain the access control object, called *Bond ticket*, for the destination object and the *public key* of the destination object.

ACCESS CONTROL

Bond is a Message Oriented Middleware. Therefore we decided to embed access control mechanism into the Bond communication fabric. Bond objects act in response to messages and perform functions according to pre-defined message patterns called *sub-protocols.* The access control mechanism is based upon access rights granted by an object by means of a *Bond ticket* and enforced by a *Bond Security Agent*.

The access control mechanism requires an extension of the Bond shadow objects. The access control ticket is generated at the time when the Bond shadow is created. The ticket is signed by the issuer to prevent possible malicious modification at the shadow side.

Recall that a Bond object **A** sends messages for object **B** to the shadow of **B**, which in turn delivers the messages to **B** (Figure 1). We extend the shadow to contain a copy of the *Bond ticket*.

If the shadow has a copy of the ticket granted by **B,** it can filter messages and deliver to **B** only those which conform to the access rights in the granted ticket. A *Bond ticket* defines the access rights as the ability to send KQML message with specific sub-protocol, performative, content and parameters. A Bond object can grant the access to its services at four levels. The first allows the use of specific sub-protocol, the second restricts the performative, the third restricts the contents of the performative and the fourth further restricts the parameters.

For example the ticket granted by **B** to **A** may have the following format:

subprotocol	performative	contents	parameter
PropertyAccess	ask-one	get	-
PropertyAccess	achieve	set	alpha
AgentControl	achieve	start-agent	WorkSpace

In this example **A** can only send to **B** the following types of KQML messages:

1. any (**ask-one**) message in PropertyAccess sub-protocol, this means **B** allows **A** to get the value of any property of itself;
2. (**achieve set alpha**); this message allows **A** to exercise the property access sub-protocol, and only can set the value of property **alpha** of **B**.
3. (**achive start-agent WorkSpace**); this message allows **A** to exercise the AgentControl sub-protocol, but can only start **WorkSpace** on **B**.

The scheme described above works well if the grantor of a ticket trusts the grantee. Filtering messages at the source guarantees that no unwanted messages cross the network. But we need to provide also a scheme that goes beyond the trust relationship and enforces the access control scheme. Such a scheme is based upon *security agents.* A *security agent* is one capable of generating and storing Bond tickets for one or more Bond objects and enforcing the access control. Instead of sending a message to the destination object its shadow will send the message to the *security agent* which acts as a proxy at the receiving end and enforces the access control specified by the ticket. Now the shadow of the *security agent* of **B** is the local proxy rather than the shadow of **B**. Messages are filtered both at the source by the shadow object and at the destination by the security agent.

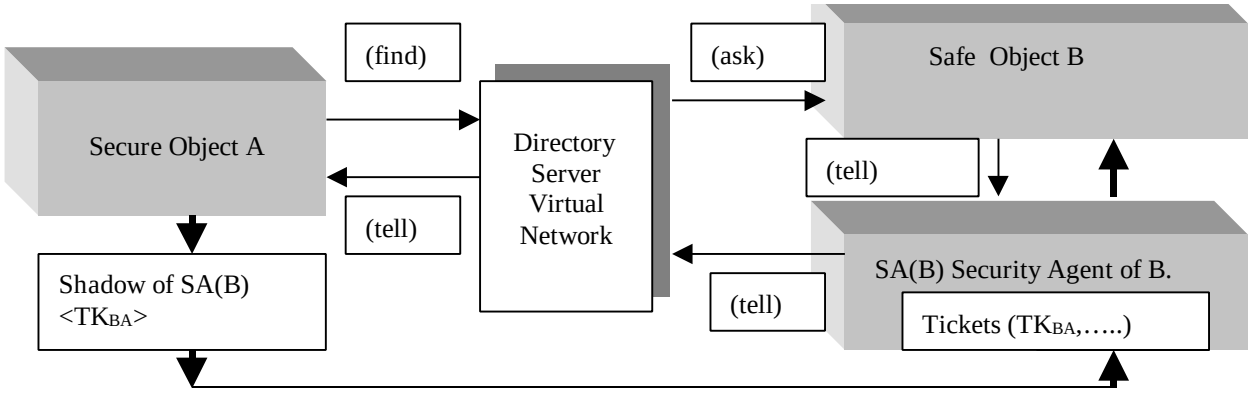

FIGURE 2. ACCESS CONTROL FOR A SAFE BOND OBJECT **B**

Figures 1 and 2 illustrate the generation of the access control tickets and the communication path for both trusted and enforced access control. The scheme which enforces the access control adds additional overhead and is less performance but more secure.

AUTHENTICATION

There are two types of authentication in Bond: the first is the authentication of Bond users when they sign on, the second is the mutual authentication between secure Bond objects. Both of them are done by the Authentication and Software Distribution (ASD) server.

ASD server is responsible for: (a) Bond code version control; (b) Creation and maintenance of Bond user accounts; (c) Creation and maintenance of Bond security keys; (d) Authentication.

User can download the latest version of Bond code from the Bond HTTP server which coordinates with the ASD server. The downloaded code is stamped with the IP address of the requester, the public key of ASD server and the Bond Version Number. The ASD public key is shared among all Bond users who start Bond sessions from the same host. To start a Bond session a user follows the procedure illustrated in Figure 3.

Authentication of Bond users is carried out in the following way. The user starts a workspace that has its unique Bond Id (BID) and inherits the Bond Version Number, the ASD public key from the downloaded Bond code. Then the user provides the Bond User Name, the Bond Password and specifies if a secure workspace is desired. The workspace sends as plain text the Bond User Name, the IP address of the host, and the security flag (it is set when a secure workspace is desired) as well as the Bond Version Number, the Bond Id of the workspace, the password and a new generated session key encrypted with the downloaded ASD public key. The ASD server decrypts the encrypted content, uses the Bond User Name as index to verify the Bond Password. It also compares the Bond Version number and informs the user about the need to download and update the code of Bond resident if outdated. If a secure workspace is desired then a pair of public and private keys are generated for the workspace by the ASD server and stored in the BondKey object indexed by the Bond Id. After a successful login a secure workspace gets the public/private key pair encrypted with the session key it sends to ASD server.

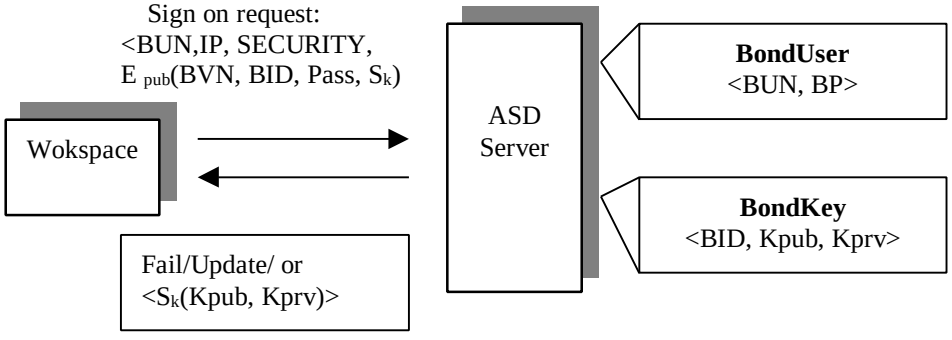

FIGURE 3. STARTING A BOND SESSION AND SIGN ON

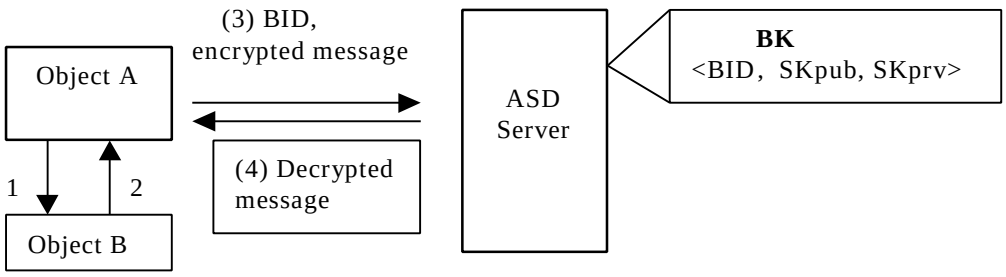

FIGURE 4. OBJECT AUTHENTICATION

Another function of the ASD server is to support object authentication. The basic idea for object authentication is illustrated in Figure 4. The presumption of object authentication is that both objects involved are secure objects, which means each object has a pair of public and private keys. Assume that both **A** and **B** are secure Bond objects. To authenticate object **B**, **A** generates a random message and asks **B** to encrypt it using its own private key (step 1) and send the encrypted message back, together with its own Bond Id (step 2). Then **A** sends the encrypted message to the authentication server for decryption (step 3) and also compares the message decrypted by the ASD server (step 4) with the original random string it had generated. If they are the same, **A** can make sure the object it's talking with is real **B**.

IMPLEMENTATION ISSUES

Each Bond object can choose its own security policy, it can vary from no security check to very rigorous security check. But interface of security check method if it is presented must be uniform. We define Java abstract classes for all the security functions e.g. security check and access ticket generation [8]. Each object can make its own decision to implement these security abstract classes or not. If implemented, it can also override some methods by its own implementation. Bond also provides default security class implementations which user can choose if they do not want to implement their own. This design leaves the decision right of security to each object instead of using a uniform security policy in whole system.

We defined some abstract security classes in Bond to fulfill the requirements of what we had discussed in above sections [6]. For example, abstract security class **Cipher** is a class that finishes the function of Encryption and Decryption. User objects can subclass **Cipher** to have their own implementation of Encryption and Decryption. Bond implements two sub-classes from the abstract security class **Cipher**: **bondSymEncrypCipher** and **bondAsymEncrypCipher**. **bondSymEncrypCipher** is a Bond class for symmetric encryption and decryption using a single key algorithm. This single key can be a password, or a session key in DES. **bondAsymEncrypCipher** is a Bond class for asymmetric public key encryption and decryption using public/private key pairs.

CONCLUSIONS

Security is an important concern in any network computing environment. Object-oriented network computing environments like Bond are generally more complex than traditional client-server systems, and the security issues are more subtle. In this paper we discussed Bond's view of security in an object-oriented network computing environment and presented the security model used by Bond for access control. In this model, each Bond object can choose its own security policy, it can vary from no security check to very rigorous security check. Secure Bond objects are allowed to create their own security agents to enforce access control. These functions are supported by a set of uniquely defined abstract security classes.

Currently we are investigating a new method of authentication based on Certification and Delegation, which can carry out authentication in an elegant and flexible way.

REFERENCES

[1] L. Bölöni, K.K. Jun, M. Sirbu and D.C. Marinescu, Seamless Metacomputing with Bond, Purdue University, CSD-TR #98-010.

[2] L. Bölöni, Bond Objects -- a white paper, Department of Computer Sciences, Purdue University, CSD-TR #98-002.

[3] L. Bölöni, K.K. Jun, T. Daniels and D.C. Marinescu, Message patterns in the Bond Distributed Object System, Purdue University, CSD-TR #98-004.

[4] L. Bölöni, R.B. Hao, K.K. Jun and D. C. Marinescu, Bond Sub-protocols, 1998, (in preparation).

[5] T. Finin, et al. Specification of the KQML Agent-Communication Language, DARPA Knowledge Sharing Initiative draft, June 1993.

[6] R. B. Hao, L. Bölöni, D.C. Marinescu, Bond System Security and Access Control Models, Purdue University, CSD-TR #98-019.

[7] B. Fairthorne, OMG White Paper on Security, OMG Security Working Group, April 1994.

[8] J. B. Knudsen, Java Cryptography, O'Reilly, 1998.