Security and Performance in Network Component Architecture*

Dongjin Yu[1,2], Ying Li[3], Yi Zhou [3], and Zhaohui Wu[3]

[1] Zhejiang Gongshang University, 310035 Hangzhou, China
yudongjin@hz.cn
[2] Zhejiang Institute of Computing Technology, 310006 Hangzhou, China
[3] College of Computer Science, Zhejiang University, 310027 Hangzhou, China
{cnliying, zhouyi, wzh}@zju.edu.cn

Abstract. In the Internet computing environment, clients usually invoke remote components to accomplish computation, which leads to many security problems. Traditional network component architecture model focuses on business logic, and takes little consideration for security problems. This paper proposes Secure Network Component Architecture Model (SNCAM). It classifies clients according to their credibility levels on component-side. Next, it presents the main idea of this paper, which is the security domain. To reduce the overhead introduced by the security mechanism, a method called security agent is also proposed.

1 Introduction

In the Internet computing environment, it is very pervasive that clients may not implement all the services they need. When clients need certain services, they can invoke remote components to get them, leading to the network component architecture model [7]. In such a model, local programs invoke remote components to accomplish the tasks, which may be required to transmit a great amount of data between clients and remote components. If the data is transmitted on the network without encryption, malicious listeners may get and change it. Traditional network component architecture model [6] focuses on business logic, and takes little consideration for security problems when business logic is invoked. All the parameters and returned results are transmitted on the network as plain text, causing security hazard.

The Common Component Architecture (CCA) [2] provides the means to manage the complexity of large-scale scientific software systems and to move toward a "plug and play" environment for high-performance computing. In the design of CCA, the considered requirements are: Performance, Portability, Flexibility and Integration. CCA gives concerns to performance issues, but does not address the requirements of security, which is very important in distributed component computing environment.

There are two measures to ensure secure communication between computers. One is data-encryption, which is further divided into an algorithm and an encryption key.

* The work is supported in part by Natural Science Foundation of Zhejiang Province of China with grant No. Z104550, Zhejiang Provincial Science and Technology Department Key Projects with grant No. 2005C21024, of China.

The negotiation of encryption key is always made using public-key technology, such as that of the IETF "SPKI"(Simple Public-Key Infrastructure).

The second measure is authentication or access-control. Many research works [9] address this problem, but few combine the authentication process with the data-encryption process for an integrated security framework. A language called Ponder [4] defines a declarative, object-oriented language for specifying policies for the security and management of distributed systems. Ponder focuses on specifying authentication rules, but lacks the security while data is being transmitted after the authentication.

Another issue is performance. Most of the existing security architectures do not address the problem of performance reduction [3]. Actually, we can do many performance improvements under the security framework.

In response to this situation, we propose the Secure Network Component Architecture Model (SNCAM) to ensure secure communication of clients and remote components in both ways: data-encryption and access-control. We also take into consideration for performance reduction owing to security mechanisms.

SNCAM adds security mechanisms to the traditional model. It forces components and clients to negotiate encryption algorithms and exchange an encryption key in every session for future communication. It also forces the authentication of clients to ensure that clients have the right to access the services provided by components. The participants of SNCAM are clients and security domains. A security domain consists of many business components and one security component. The relationship between these participants is: 1. a client invokes a business component; 2. the business component invokes security service provided by the security component, and the result indicates whether it should give the client the opportunity to access the business service; 3. if the client is granted the right to access the business service, the business component may also invoke other business services provided by components either in the domain or outside the domain.

2 Client Classification and Security Domains

SSH [1] (Secure SHell) is a protocol for secure remote login and other secure network services over an insecure network. It divides the process of establishing a secure connection into five phases: version exchange, algorithm negotiation, key exchange, authentication and session. The version exchange phase is to negotiate the protocol and software versions that the client and the server are both compatible to. In the session phase, the communication between up-level applications of the client and the server has already begun. These two phases are irrelevant to security mechanisms. Therefore, we can simplify the 5-phase SSH protocol into 3 phases: algorithm negotiation, key exchange and authentication. These three phases are loosely coupled: algorithm negotiation is to decide which algorithms to use during encryption and authentication; key exchange is to decide the key that is used to do encryption and decryption; authentication is to verify the client's access right. To accomplish the above three tasks, we can design three separate pieces of programs, which are also called as *security procedures*.

From the viewpoint of a server-side component, different clients may have different credibility. We classify these clients into 4 classes according to their credibility in the ascending order, resulting in four credibility levels: A-SEC, B-SEC, C-SEC, and D-SEC. For example, clients whose credibility levels are B-SEC are more credible than clients of A-SEC. When a client passes a security procedure, its credibility level goes up. In SSH, the exchanged key is temporarily valid. When a certain period of time expires, the key becomes invalid. The client and the server then have to re-exchange the key (rekey). The clients' credibility levels can go up and down as well. For example, a client whose credibility level is C-SEC or D-SEC can migrate back to B-SEC.

The Implementation of security procedures is the implementation and organization of the program that accomplishes the tasks of algorithm negotiation, key exchange, and authentication. Security procedures, which reside on component servers, read information about a client's credibility level from storage and may change it.

In many situations, the implementations of security procedures are similar. Hence it is reasonable to combine components that have the same implementation of security procedures into a domain, and configure a security component that implements these security procedures. The security component controls and authorizes clients to access services provided by components in that domain. In this way, other components in that domain need not implement security procedures. Instead, they can use the security service provided by the security component. All the components in one security domain can be classified into two categories: business and security. The business components accomplish business logic and the security component is responsible for controlling access.

3 Security Agent

A Security Agent method provides an express way for direct communication between distrusted clients and components. The discussion below regards clients as components, and we call components that provide services as object components.

The direct trust relationship between components is a 2-tuple $DT(C_i,C_j)$, which means that component C_i and C_j can communicate directly with each other without any security procedures. The interface trust relationship between components is a 2-tuple $IT(C_i,C_j)$, which means that C_i and C_j cannot communicate directly with each other without any security procedures. But they can avoid that by communicating with the third party components.

When there is no direct trust relationship but interface trust relationship between client components and object components, it is certain that we can find a sequence of components which begins in a client component and ends in an object component, where every two adjacent elements are directly trusted. This sequence is named as the security agent sequence. Every element in it is named as the security agent.

4 Experiments and Results

We take the size of a message as a parameter of a task, and set up two environments, one using security agents, and the other not. We set the size of message in turn as: 4K, 16K, 64K, 128K, and 512K. The clients' credibility levels are fixed to B-SEC. The obtained data is shown in Table 1, with the format of "without agent/with agent":

Table 1. Performance of security agents

message size	client 1	client 2	client 3
4K	7.4/2.9	8.1/3.6	10.4/4.1
16K	11.6/10.5	14.1/14.3	15.3/14.8
64K	34.3/43.7	47.6/51.9	50.9/59.1
128K	56.2/85.3	68.3/102.4	88.2/120.7
512K	196.3/304.8	237.9/395.6	279.8/463.7

The above data is the time of transmitting messages of the corresponding size from a client to a server for 40 times, measured in seconds. In Table 1 we can see that for light-weight tasks, using security agents is better. But for heavy-weight tasks, it is better to carry out security procedures and then communicate with the server directly.

References

1. T. Ylönen, T. Kivinen, M. Saarinen, T. Rinne, and S. Lehtinen. "SSH Protocol Architecture", Internet Engineering Task Force, work in progress, (2002).
2. D.E. Bernholdt, W.R. Elwasif, James A. Kohl, and Thomas G. W. Epperly. A component architecture for high-performance computing. In Proceedings of the Workshop on Performance Optimization via High-Level Languages (POHLL-02), New York, NY, June 2002.
3. R.L. Rivest, B.Lampson. SDSI - A Simple Distributed Security Infrastructure. Working document from http://theory.lcs.mit.edu/cis/sdsi.html
4. N. Damianou, N. Dulay, E. Lupu, M. Sloman, Ponder: A Language for specifying Management and Security Policies for Distributed Systems, Imperial College Research Report DoC2001, January,2001.
5. Lacoste, G. (1997) SEMPER: A Security Framework for the Global Electronic Marketplace. IIR London.
6. Li Yang, Zhou Yi, Wu ZhaoHui. Abstract Software Architecture Model based on Network Component. Journal of Zhejiang University, Engineering Science, 2004, 38(11):1402-1407.
7. A. Thomas, Enterprise JavaBeans: Server Component Model for Java, White Paper, Dec. 1997, http://www.javasoft.com/products/ejb/.
8. M. S. Sloman, J. Magee, “An Architecture for Managing Distributed Systems”, Proceedings of the Fourth IEEE InternationalWorkshop on Future Trends of Distributed Computing Systems, pp. 40-46, IEEE Computer Society Press, September, 1993.
9. Lupu, E. C. and M. S. Sloman (1997b). Towards a Role Based Framework for Distributed Systems Management. Journal of Network and Systems Management 5(1): 5-30.