Journal of Zhejiang University SCIENCE
ISSN 1009-3095
http://www.zju.edu.cn/jzus
E-mail: jzus@zju.edu.cn

Instance-oriented delegation: A solution for providing security to Grid-based mobile agent middleware*

MA Tian-chi (马天驰), LI Shan-ping (李善平)

(*School of Computer Science and Technology, Zhejiang University, Hangzhou 310027, China*)

E-mail: tcma@csis.hku.hk; shan@cs.zju.edu.cn

Received Feb. 2, 2004; revision accepted May 8, 2004

Abstract: New challenges are introduced when people try to build a general-purpose mobile agent middleware in Grid environment. In this paper, an instance-oriented security mechanism is proposed to deal with possible security threats in such mobile agent systems. The current security support in Grid Security Infrastructure (GSI) requires the users to delegate their privileges to certain hosts. This host-oriented solution is insecure and inflexible towards mobile agent applications because it cannot prevent delegation abuse and control well the diffusion of damage. Our proposed solution introduces security instance, which is an encapsulation of one set of authorizations and their validity specifications with respect to the agent's specific code segments, or even the states and requests. Applications can establish and configure their security framework flexibly on the same platform, through defining instances and operations according to their own logic. Mechanisms are provided to allow users delegating their identity to these instances instead of certain hosts. By adopting this instance-oriented security mechanism, a Grid-based general-purpose MA middleware, Everest, is developed to enhance Globus Toolkit's security support for mobile agent applications.

Key words: Mobile agent, Grid, Trust model, Delegation
doi:10.1631/jzus.2005.A0405 **Document code:** A **CLC number:** TP393

INTRODUCTION

Grid computing, since its emergence, has been widely regarded as a new revolution of information technologies (Foster *et al.*, 2001). Due to the high heterogeneity and complexity in Grid environments, mobile agent (MA) has been considered as one of the promising solutions for realizing flexible and scalable Grid Computing. With the MA support, one can execute parts of its program on any networked hosts offering needed services.

In the Grid world, applications usually need to get services scattered among multiple WAN-connected sites. In order to adapt well to this heterogeneous computing environment, there is a need to build a general-purpose platform across hosts, to support agents that come from multiple applications. In this scenario, agents will access resources by general Grid services.

The goal of this work is to build a Grid-adaptive framework to tackle the security problems within general-purpose MA systems. Since security is an essential issue of MA systems, much research has been done to tackle security threats from agents to hosts, or from hosts to agents (Borselius, 2002). For example, to tackle various security issues brought from multiple organizations, the Grid Security Infrastructure (GSI) (Foster *et al.*, 1998) has established as a trust framework on which general authentication and authorization can be carried out. To ensure trust protection, when an agent is trying to move, the host currently hosting the agent will be asked to sign a further delegation to the target host that the agent wants to move onto. This is called a host-oriented delegation mechanism because the authorizations are bound onto hosts. In GSI, however, a continuous delegation or chain-delegation for an agent is not

* Project (No. 602032) supported by the Natural Science Foundation of Zhejiang Province, China

recommended. This is mainly because the host-oriented mechanism requires each host to hold the privilege of further signing delegations in the name of the original user, while, in a common circumstance, an individual host will not be trusted by the user during an agent's full trip. Once a privileged host has been cracked, it will probably maliciously delegate to its other cooperators to spread the damage across the whole network. This makes the agent behave like viruses.

To improve that, an instance-oriented security framework is then proposed in this paper. Different from the original host-oriented policy, we define security instances as an encapsulation of one set of authorizations and their respective validity specifications. User and applications can define several kinds of security instances and their possible operations, according to the application's own logic. The instances are signed by its creator and recorded into the mobile agent's delegation document. A strictly-protected trace list is adopted for validity computing and intrusion detection. These constitute the delegation document together. Besides, we adopt several firmly trusted servers in our framework, called exchange servers, to be the security checkpoints. All the user's security policies are only required to be delivered on these servers, instead of all the concerned hosts.

Four exciting goals can be achieved using the proposed security framework:

1. The problem of the abuse of delegation document can be avoided.

2. The system's flexibility is enhanced by the instance-oriented model. All restrictions and verifications are bound onto the instances-oriented delegations which can be transferred everywhere under the direct authorization of the original user. This makes the delegation a standard document-centered flexible model. And the dynamic resource reservation can also enhance the flexibility of the agent.

3. Enables easy delivery and updating of user's security policies. Once a policy needs to be updated, only a few of the exchange servers need to be flushed to achieve consistence.

4. Compatible for heterogeneous applications. The implementation details of components (agent functionalities, accessing protocols) are encapsulated into the instance.

THE EVEREST ARCHITECTURE

The core modules implemented in Everest are shown in Fig.1. We follow the "gatekeeper-backend servers" architecture. The system depends on Globus Toolkit 2.2. We define each gatekeeper as a simulated Grid site (host), and select some of them to act as the exchange servers.

In the Everest system, the code maintenance and migration are all processed on the gatekeeper of each host, while the job manager and the backend server need only deal with the mobile agent like normal jobs. During migration of the agent, the preliminary procedures are handled by the migration daemon on the gatekeeper. An execution daemon on each backend server will then start to save the code's current execution states and transfer them to the target backend server directly without passing the two gatekeepers.

The migration daemon is in charge of the code migration (i.e., handover operation). It will also send delegation-updating requests to the exchange server, when a delegation document is found falling into invalidation.

The exchange server is composed of a code daemon and several code maintainers. The code daemon is in charge of obtaining request from hosts. Then it forwards those requests to the corresponding code maintainer. Each agent has a code maintainer in the exchange server, to maintain and monitor its current states. When a code asks for a delegation document renewal, it is the maintainer who will attend to these requests. For other requests like resource reservation, the maintainer will ask the code daemon to contact the corresponding resource provider, and transfer the requests to it. Besides, the code maintainer should have enough knowledge to carry out a client side validation check on the request delegation documents.

There are two modules covering the resource provider's standard service interface. The reservation service module has the privilege of signing a resource instance delegation document. Actually, the real reservation operation will be done under the standard service interface, while the reservation service takes care only of those businesses concerned with delegation document. Another module is the security guard, which is used to deal with the resource requests. The security guard will first make a server side validation

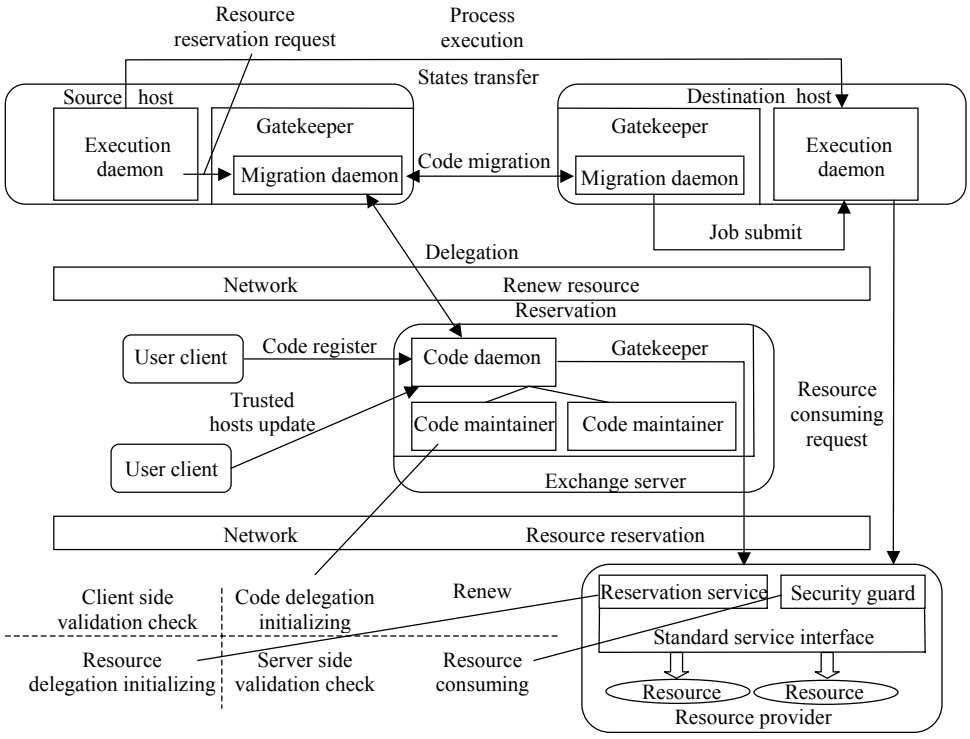

Fig.1 Core modules of the Everest mobile agent platform

check on the coming delegation document, and then establish a connection between the remote agents and the standard service interface.

INSTANCE-ORIENTED DELEGATION MODEL

Why instance-oriented?

A delegation (also called impersonation) is a statement, which includes one or a set of authorizations from the user or policy publisher. The original delegation document is very simple. It records and proves the following statement:

A authorizes *B* to behave in the name of *A*.

i.e. by showing this delegation document, *B* will have the privilege to do whatever *A* is allowed to do. This delegation designates a kind of universal deputy. However, it does not allow a user to specify detailed privileges to his deputy. To deal with this, an extended delegation is proposed on which the detailed privileges are recorded there. It makes the following statement:

A authorizes *B* to perform *S* in name of *A*.

However, the above statement is not enough either because in many circumstances, the given privileges should be restricted under one or some conditions. A very familiar example is the TTL of the delegation. Beyond the TTL, a delegation should be regarded as invalid. This information must be recorded in the delegation document, or else the host may mistrust an invalid delegation that is out-of-time.

To deal with this, the delegation document is further extended, by adding conditions like the following:

A authorizes *B* to perform *S* in name of *A*, under the condition *U*.

This delegation document is detailed enough, yet at the same time it also introduces much inflexibility. In fact, the condition *U* just mentioned should not be bound to the destination host *B*, but it should be bound to the target which is being performed. We will find that the target host *B* can be regarded as a part of the condition, if it must be specified.

To further improve the flexibility, a new type of delegation document is proposed, in which the item describing the destination host has been abolished. Instead, another item is introduced describing the content(s) to be performed on. This makes the following statement:

A authorizes instance (S, U) to be performed in the name of A, where S will be valid only under condition U.

It can be learned from the statement that instance (S, U) records the detailed privilege(s) and their validity specifications. The condition U is bound on S. This delegation can be used on whatever host. Of course, user can specify a target host B by adding B to the condition set.

The delegation making the statement just mentioned above is called an instance-oriented delegation. The goal of the instance-oriented delegation is mainly for the purpose of introducing a convenience for the security designers (there are three kinds of security designer: framework security designer, component security designer and policy security designer), and providing a more flexible and stable protocol to implement those kinds of security polices.

Advantages

The proposed model can be regarded as a mutation of the object-oriented model. Following such software architecture will reap at least two benefits: implementation independency and reuse-ability. The implementation independency enables the framework security designer to build his framework only by considering the public operations and properties provided by those security instances. He need not care about the detailed implementation of a certain instance. From the policy security designer's point of view, he need not care about the complex trust relationship among hosts. What he needs to pay attention to is only how to give security solutions for specific operations in specific instances. The reuse-ability is based on the component security designer's point view. He can derive them from the existing classes/instances and produce new instances with a few codes added only. Also, he needs only to care about the pre-defined interface when designing a security instance, to make their productions achieve reuse-ability for other security frameworks.

The most remarkable characteristic of the delegation is its mobility. For conventional host-oriented delegation, everything will be reestablished on its moving. For example, host A delegates host B to perform instance S, host B wants to further transfer this delegation to host C. For safety considerations, host B must ask host A that whether it is feasible to authorize host C to have the privileges. Without this query, host C will not be regarded as a safe deputy. This makes the verification more complex. Besides, suppose A has approved the further delegation, then B will rewrite a new delegation of $B \rightarrow C$, and append A's approval behind the delegation context. We can imagine that if the delegation is asked to be moved further and further, the procedures will become very complex. An instance-oriented delegation will greatly simplify the procedure, as it does not need to bind on a target host. The instance is free to be moved everywhere except that its moving territory is restricted to the conditions of the delegation document. During the moving, the structure of the instance's description needs not to be changed. The only thing a host may be asked to do is to append a trace node behind the delegation context, to record what it has done with the corresponding security instance.

Contractual history

Besides the instance-oriented model, a new content is proposed to be inserted into the delegation document, and is called contractual history. Contractual history is adopted to be the proof of past operations.

In the real world, people use contracts to record and prove some agreements that had been established, and then to ensure they are executed correctly. In the proposed security framework, a contract is adopted to record and prove some experienced procedures or operations, and protect them from being denied. Usually, a real-life contract should include signatures of the people concerned. Similarly, all the hosts with direct relation to the to-be-proved procedure or operation must put a digital sign on the contract, to make a trustable assurance. Contractual history is a record of past operations as described by a list of contracts.

The contractual introduces several functions and benefits:

1. A third party can check and testify to the ex-

istence of past businesses by verifying the contracts. No one can deny the content of the contract, because all the concerned hosts have signed on it.

2. It is strictly required that all the concern hosts who had signed on the contract should reserve a copy of the signed contract. Besides, it is not necessarily required but highly recommended that a third party reserves a copy also. This is to ensure that when one or some signatories (not all) try to deny the past, another contract holder can stand out to impeach them for that.

3. Contracts are chain-linked in the contractual history, and therefore they can be checked together. For example, contract 1 records that an instance has been moved from A to B ($A \rightarrow B$); contract 2 records $B \rightarrow C$ and contract 3 records $C \rightarrow D$. The three contracts can be collected together to prove that everything is ok. If only contract 1 and contract 3 are provided, the system can rapidly find out that one or some contracts have been disguised, by comparing and matching them.

The contract is designed to prove a certain operation. It will not be changed along with the variety of hosts. This makes it naturally simple to be bound onto an instance-oriented delegation.

Exchange server

A Step-To-Live (STL) restriction can be introduced into an agent instance. The STL restriction is recorded into the agent's validity specification, in order to avoid unlimited damage diffusion which is caused by a compromised agent migrating itself again and again. If the times an agent has migrated exceed the value of STL, it will fall into invalidation. To continue the trip, the agent must look for an exchange server to apply for a Renew. The exchange servers are some hosts selected from the network that are firmly trusted by the user. They will act as privileged proxies to help user to carry out the verifications and further authorizations. The exchange server mechanism is a trade-off; it can hold the user's privileges within a small region while providing a better performance comparing with a purely centralized architecture. Besides, the user needs only to publish his security policies to these exchange servers, instead of to all the concerned hosts.

Using exchange server can also avoid overfull authorization. An agent may have to access many remote resources during its whole trip. However, the user cannot give all the needed privileges to an agent, at its launch time. This is mainly because an overfull authorization will cause the agent's potential damage to grow larger. Furthermore, one can hardly predict all the behaviors of his agent during the trip. The exchange server allows the user to apply for resource reservation dynamically. It will contact the target resource and process with all the authorizations on behalf of the agent. Therefore, a normal host does not need to care about the security verification and authorization, as well as the application's specific logic. This gives the platform large compatibility.

DELEGATION PROFILE

Terms

According to the above introduction, an instance will be moved, renewed and consumed during its trip. We call these operations. To get a feasible security policy, how an instance is operated should be first taken into consideration. Usually, in each operation, there will be an Initiator and an Acceptor. In common MA systems, the Initiator is always defined as the resource requestor, or the provider of the agent. An Initiator is composed of the agent (code section plus current execution states) and site currently hosting it. An Initiator may have motivations of sending fake or malicious resource requests, to filch some data or even crash the resource provider's system. An Acceptor refers to a resource provider or the host an agent is trying to move onto. Generally, Acceptors are supposed to have a firm trust relationship with the code's original provider or launcher. Initiators will be under suspicion of whether or not they have modified the agent to make it unable to completely enforce the mission specified by the user, and whether they have any malicious intentions. There may be some attackers who would like to masquerade as an Acceptor. It should also be considered that both the Initiators and the Acceptors could have the motivation of denying one or some historical operations.

The structure of the delegation document

An instance-oriented delegation can have one or multiple instances. Each instance-oriented delegation document is composed of two parts (as shown in

Fig.2).

The first part is instance details. For each instance, there must be an instance detail recorded. An instance detail records the instance's properties, operating rules, and specifications about its validity. The instance detail can only be filled and modified on the initialization of the instance's delegation or in renew operations. Otherwise, it is read-only. Details about the renew operations will be discussed in Section 4.3.

Both the consumer and the holder of the delegation document should declare their approval of the contents recorded in the instance detail. At the bottom of the instance detail, the signature of the delegation's original creator or last modifier (maybe an exchange server) on the instance detail should be attached, in order to prove the detail's validity. And the original creator's certificate list should also be attached on it, to prove the validity of its signature (this is not always necessary, because the certificate list can be retrieved from another place).

The second part is a trace list, which is an array including records of the contractual history of the corresponding instances. Each item of the array is called a trace node, in which a single operation is recorded. An instance-oriented delegation document can only have one trace list, in which trace nodes for all its instances are linked together.

A single trace node is divided into two sections:

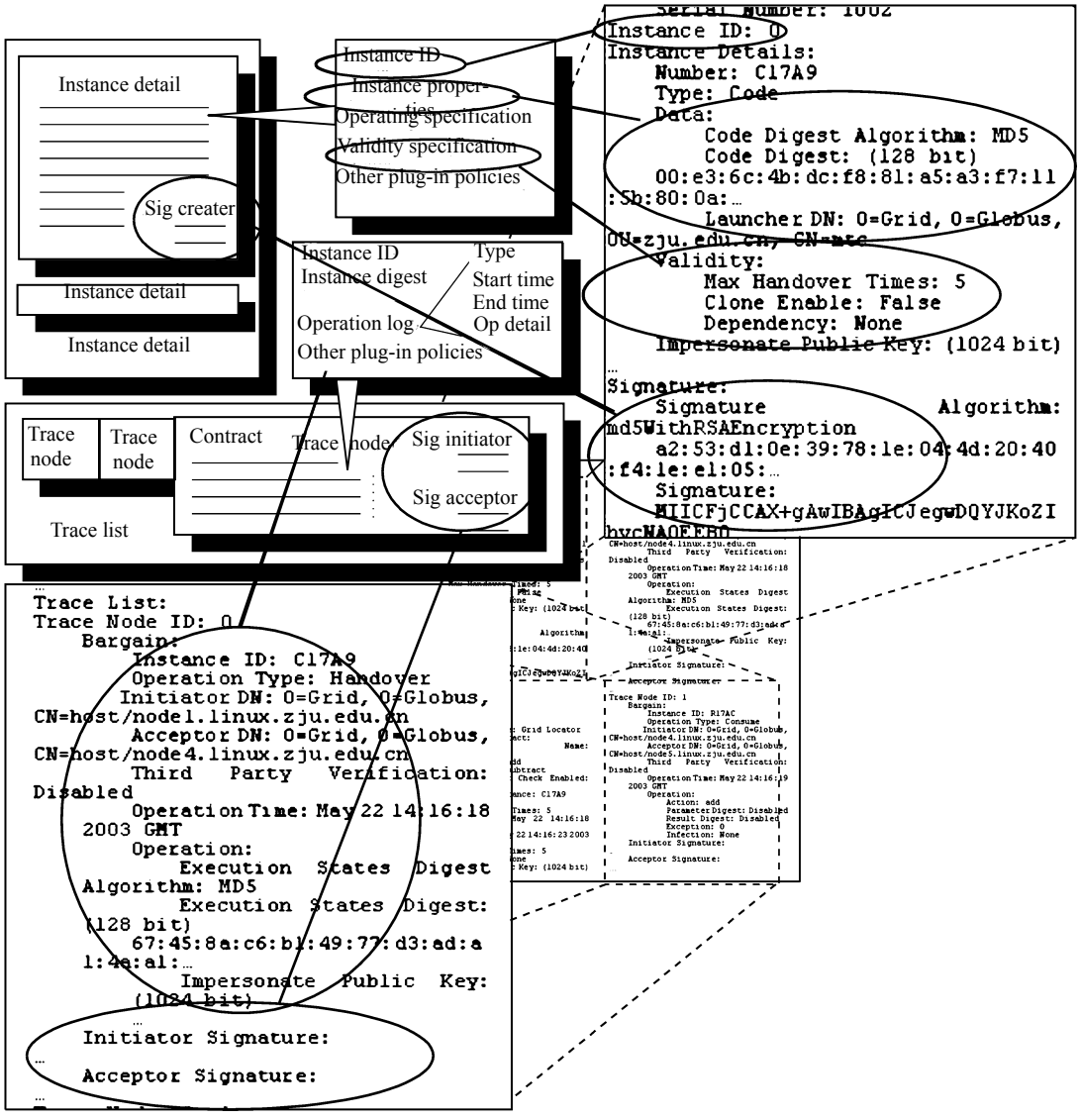

Fig.2 Instance-oriented delegation profile

contract content and a signature pair. In the contract content, first the instance detail's content's digest or a global instance ID will be recorded to prove that the contract is related to exactly the corresponding instance. Here multiple instances' ID or digest can be put together for a batch operation. After the digest or global instance ID, there should be an operation log, in which the detailed information on the corresponding operation should be recorded. The operation log can include the start and end time of the operation, its returning result, and exceptions. The signature pair is the second section of the trace node. It is comprised of both the Initiator and the Acceptor's signatures on the contract, to show their approval of the contract's contents. Both the Initiator and the Acceptor will hold a copy of the signed trace node. It can be shown as evidence when the opposing party tries in the future to deny one or more historical operations.

General operations

All instances have a common operation–handover. In general, this operation can be performed by all instances, by transferring the instance delegation from one host to another. In a handover operation, an instance's states will not be changed. Only its current location is changed. Once a delegation migration is carried out between hosts, there must be one or more handover operations performed.

To prevent unlimited diffusion of potential damage, the max times of performing the handover operation should be specified. It should be considered as an issue when an instance's validity is being measured. Once an instance reaches the max handover times, it will be regarded as no longer valid and thus no operations can be performed on it. In this circumstance, to make a further delegation, the instance should request for a renew operation. The exchange server will act as a privileged modifier of the delegation documents. It is always authorized by a user, and can update and recreate the instances issued by that user. However, it has no privilege to update other instances in the same delegation document. Usually, a renew operation will remove some obsolete trace nodes and make the delegation shorter. On the other hand, the recorded times of the handover operations is reduced and become lower than the specified max times. Thus the delegation can resume as valid again.

EXPERIMENTS AND ANALYSIS

The Everest platform was installed onto the super cluster Gideon 300. There are totally 300 PCs connected through Giga-byte LAN. Each was equipped with one PIV 2.0 GHz Intel processor with 512 MB memory, 40 GB hard disk space. RedHat 8.0 was installed on those PCs. The Globus Toolkit 2.2 installed in each PC acted as both gatekeeper and backend server. We adopted linux-fork as the simple job manager because it was required to specify a job manager in GT2 GRAM. (It can be fork, PBS or other supported job managers. Here the fork was chosen because it was the simplest one.)

We have built an application of agent-based market. Agent travels among hosts to exchange information for them. Each host has an information pool, which follows the Monitoring and Discovery Service (MDS) and can be regarded as a standard information service. Host puts what it can provide and what it required into the pool. Agent retrieves the host's requests and moves to other hosts to discover the needed information. After it has got the required information, the agent will move back to the original host and deliver the information to the pool.

Totally 20 nodes on the Gideon cluster were selected to participate in this experiment. One of them was appointed to act as the agent's launcher, another one was appointed to act as exchange server. Eighteen nodes remained to be the general hosts. To standardize the experiment, we predefined the travel path for the agent. The agent will be launched out, and travels as a loop from host1 to host18. On each time of the authorization and renew operation, the security sub-system will be active and a time counter will start to run. After each time the security-related operation is done, the time counter will calculate and print the time used in this operation. When the agent is about to leave a host, the platform will report the current size of its delegation document. By these, we can get the time and memory overhead used for security operations.

We compared the overheads between the instance-oriented and host-oriented delegation schemes. The Step-To-Live parameter was set to 3 (that is, will renew 5 times while traveling the first loop). Below are the data result and figures.

Fig.3 compares two scenarios. It can be con-

cluded that after 11 hosts traveling, the host-oriented scenario begins to cost more time than the instance-oriented one.

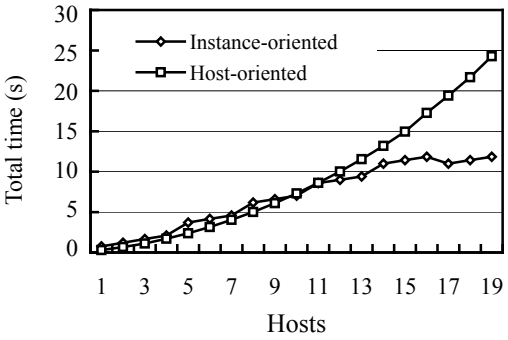

Fig.3 Total time overhead after each operation

Fig.4 compares the memory cost for each delegation document. Unlike the host-oriented scenario, the instance-oriented delegation's size will not keep increasing, because the delegation will be curtailed on the renew time. One may argue that when the Step-To-Live or resource accessing increases, the delegation document's size will grow, so the overhead will be larger because of the time for transferring over to the network. However, since the delegation's current size is only 8 K, even if a big STL is selected and the delegation suffers from poor network transfer, it will not cost too much time. And the application can adjust the STL to a shorter value if the network bandwidth is low. Finally, even if the performance is a little lower than that of the host-oriented scheme, the instance-oriented scheme is still recommended because it is worthy to pay a bit of performance to leverage the security assurance.

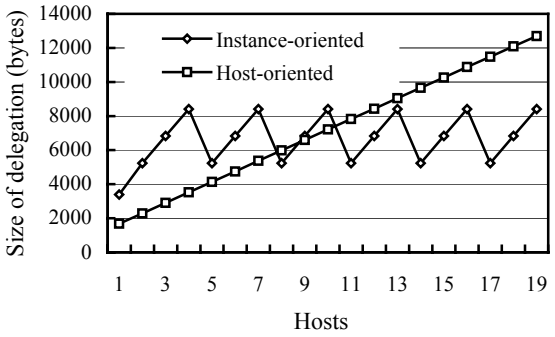

Fig.4 Space overhead after each operation

RELATED WORK

Table 1 gives a brief comparison of the main existing methods against security threats within the MA system.

Table 1 Comparison of existing solutions

Methodology	Criterions				Remarks
	Compatibility	Implementation	Verification	Update	
Trust management (Wong and Sycara, 1999)	Good	Simple	Rigorous	Normal	Depends on other's honesty
Vote (Schneider, 1997)	Normal	Simple	Normal	Normal	Depends on other machines, inefficient
Hardware protection (Wilhelm *et al*., 1999)	Limited	Hard	Normal	Hard	Efficient, but expensive & less scalable
Cryptography execution (Sander and Tschudin, 1998)	Limited	Hard	Rigorous	Hard	Not scalable
Execution states analyzing (Chander *et al*., 2001)	Limited	Normal	Normal	Normal	Need knowledge about the application's logic
Cryptography tracing (Vigna, 1997)	Normal	Normal	Normal	Hard	Counter denying. Indirect attacks possible
Proof carrying code (Necula and Lee, 1996)	Limited	Normal	Normal	Hard	Special for untrusted code. Policies embedded in the compilers
Appraisal (Farmer *et al*., 1996)	Limited	Normal	Rigorous	Normal	Less compactable, depending on code's logic
Host defense (Jansen, 2001)	Good	Normal	Rigorous	Hard	Local defense, code regardless. But depends on the policy's maturity, sometimes inefficient

CONCLUSION AND FUTURE WORK

In this paper, a new instance-oriented security framework is proposed to provide the Grid-based MA system a more flexible and stable solution for its security problems. There are several advantages of the instance-oriented delegation model. As discussed, this allows applications to configure their own security policy by instantiating the proposed delegation model and embedding their preferred security policies into it, which makes it compatible for heterogeneous platforms, thus to be more appropriate for Grid environment.

The instance-oriented delegation document solves successfully some problems in the original host-oriented delegation document. It prevents the hosts from abusing the delegation document to damage the resource provider, and prevents the system from suffering unlimited diffusion of damages, which is caused by some potential crisis that will never be checked out in conventional trust systems. This makes the delegation model appropriate for MA systems. And then, a general solution is given on using the instance-oriented delegation document to a build security framework in MA systems.

References

Borselius, N., 2002. Mobile agent security. *Electronics & Communication Engineering Journal*, **14**(5): 211-218.

Chander, A., Mitchell, J.C., Shin, I., 2001. Mobile Code Security by Java Bytecode Instrumentation. Proceedings of the DARPA Information Survivability Conference & Exposition, DISCEX-II 2001, Anaheim, CA.

Farmer, W.M., Guttman, J.D., Swarup, V., 1996. Security for Mobile Agents: Authentication and State Appraisal. Proceedings of the 4th European Symposium on Research in Computer Security (ESORICS), Springer-Verlag, p.118-130.

Foster, I., Kesselman, C., Tsudik, G.., Tuecke, S., 1998. A Security Architecture for Computational Grids. Proc. 5th ACM Conference on Computer and Communications Security Conference, ACM Press, NY, p.83-92.

Foster, I., Kesselman, C., Tuecke, S., 2001. The anatomy of the grid: enabling scalable virtual organizations. *International Journal of High Performance Computing Applications*, **15**(3):200-222.

Jansen, W., 2001. A Privilege Management Scheme for Mobile Agents. Workshop on Security of Mobile Multi-Agent Systems: Proceedings of the 5th International Conference on Autonomous Agents.

Necula, G.., Lee, P., 1996. Safe Kernel Extensions Without Run-Time Checking. Proceedings of the 2nd Symposium on Operating System Design and Implementation (OSDI'96), Seattle, p.229-243.

Sander, T., Tschudin, C.F., 1998. Protecting Mobile Agents Against Malicious Hosts. *In*: Vigna, G.(Ed.), Mobile Agents and Security. Springer-Verlag, p.44-60.

Schneider, F.B., 1997. Towards Fault-Tolerant and Secure Agentry. Proceedings of 11th International Workshop on Distributed Algorithms, Saarbrucken, Germany.

Vigna, G., 1997. Protecting Mobile Agents Through Tracing. Proceedings of the 3rd ECOOP Workshop on Mobile Object Systems, Jyvälskylä, Finland.

Wilhelm, U.G.., Staamann, S., Buttyàn, L., 1999. Introducing Trusted Third Parties to the Mobile Agent Paradigm. *In*: Vitek, J., Jensen, C.(Eds.), Secure Internet Programming: Security Issues for Mobile and Distributed Objects. Springer-Verlag, p.471-491.

Wong, H.C., Sycara, K., 1999. Adding Security and Trust to Multi-Agent Systems. Proceedings of Autonomous Agents'99 (Workshop on Deception, Fraud and Trust in Agent Societies), Seattle, Washington, p.149-161.