Secure Computing with the Actor Paradigm

Bhavani Thuraisingham
The MITRE Corporation
202 Burlington Road
Bedford, Massachusetts 01730

Abstract

This paper describes the actor model of concurrent computation and discusses some of the issues in securing such a model.

1 Introduction

Various computing paradigms (or models of computation) have been proposed for concurrent computing systems. Notable among these are (1) the sequential process paradigm, (2) the functional paradigm, and (3) the actor paradigm. In the sequential process paradigm, sequence of transformations are performed on states which are mapping from locations to values. The transformations may depend on certain inputs and may produce certain outputs which may depend on the inputs (see for example [HOAR78]). In the functional paradigm, a function is a computational element which acts on data without the use of a store. Functional models are derivatives from the lambda-calculus based languages such as Lisp. Concurrency is exploited by evaluating arguments of a function in parallel and is being used in data flow architectures (see for example [WENG75]). In the actor paradigm, actors are computational agents which receive communication from other actors and respond to the communication in a specified manner. That is, these computational agents communicate asynchronously with each other by exchanging messages which are called tasks (see for example [AGHA86]). Actor is a more powerful model of computation than the other two as the sequential model and the functional model can be defined in terms of the actor model. It is envisaged that the next generation computing systems will be those based on massively parallel architectures and the actor model of computation appears to be an appropriate one for such systems.

While much of the previous work on secure computer systems has focussed on nonconcurrent computing systems (see, for example, [GASS88]), recently some work on the security aspects of the sequential process paradigm has been reported (see for example [FOUN90]). However, if the next generation computing systems are to be made secure, then the security issues of the models proposed for massively parallel architectures need to be examined. Since the actor model of concur-rent computation is becoming popular for such systems, we feel that it is useful to start with the actors model. Therefore, in this paper we discuss some of the issues on securing the actor model.*

The organization of this paper is as follows. In section 2, we provide an overview of the actor model as given in [AGHA86]. In section 3, our proposed model for secure computation will be discussed. Some of the complexities involved in proving that an actor system is secure will be noted in section 4. Since the actor model can be regarded as a variation of object-oriented computation, some of the related work in securing object-oriented systems will be given in section 5. The paper win be concluded in section 6.

2 Actor Model of Concurrent Computation

Although the actor model has roots in the programming language Simula [DAHL70], it was not until the work of Hewitt and Baker in 1977 [HEWI77] that research began actively on such a model for parallel architectures. Much of the concepts and ideas that we know today of the actor model have resulted from Agha's thesis on this subject. The discussion on actors given in this section has been obtained from [AGHA86].

An actor system consists of a collection of actors which are the computational agents. As stated in [AGHA86], computation in an actor system is carried

*We are not proposing the actor model to be the ideal one for concurrent computing systems. Our objective is to investigate only the security issues for the actor model.

out in response to communications sent to the system. Communications are contained in "tasks." Tasks consist of three components. A tag (which identifies the task), a target address (which is the mail address of an actor to which the communication is sent) and a communication. Communication could contain data values, expressions, and even commands. When a communication is received by an actor, new tasks and actors are created. When an actor is no longer active, it is removed from the system. Similarly, when the processing of a task is completed, it is also removed from the system.

An actor accepts a communication when it processes the task containing the communication. The tasks sent to an actor are mailed in a queue. An actor is specified as a pair containing a mail address and a behavior. The behavior is a function of the communication accepted by the actor. When an actor accepts a communication, in addition to creating new actors and tasks, it must also compute a replacement behavior. Certain actors within an actor system communicate with the outside world. These actors are called the recipients. The outside world could even be another actor system.

We will describe the essential points with an example taken from AGHA86]. The example is illustrated in figure 1.[†] When an actor machine X_n accepts the nth communication in a mail queue, it will create a new actor machine X_{n+1}, which will carry out the replacement behavior of the actor. The new actor machine will point to the cell in the mail queue in which the $n+1$st communication is placed. That is, when X_n processes the nth communication, it will determine the replacement behavior for the n+1th communication. In other words, while X_n continues to process the nth communication, X_{n+1} could start processing the $n+1$th communication. The two actor machines X_n and X_{n+1} will not affect each others behavior. Each of the actor machines may create their own tasks and actors as defined by their respective behaviors. Before X_n creates X_{n+1}, X_n may have already created some actors and tasks. Furthermore, X_n may still be in the process of creating more tasks and actors even as X_{n+1} is doing the same. Once X_n completes processing the nth communication, it will no longer process any additional communications. While processing the $n+1$th communication, X_{n+1} could create a new actor X_{n+2} and a replacement behavior for X_{n+2} so that X_{n+2} can process the $n+2$th communication received at the same mail address.

[†]Permission to reproduce figure 1 will be requested from the author of [AGHA86] and MIT Press.

The key issues in the actor model is to exploit concurrency, but at the same time encourage cooperative computing. As a result, the actor model is being proposed for not only systems such as operating systems, distributed systems, and parallel processing systems, but also for cooperative and collaborative computing applications.

3 Towards a Multilevel Secure Actor System

We are concerned with developing a model for concurrent computation in a multilevel environment with actors as the underlying computation agents. The first question that must be answered is what are the entities of classification? That is, should they be actors, tasks, behaviors, and communications. The next question is how should computation proceed in such a model so that there is no information flow from a higher level to a lower level? In this section we propose a model for secure computation based on the actors paradigm.

The entities of classification in the proposed model are the actors themselves, among others (such as tasks, behaviors, communications, and mail addresses). That is, whenever an actor is created, it is assigned a security level. An actor is a pair consisting of a mail address and a behavior. That is, an actor is created by another actor by first creating a mail address and then assigning a behavior to the address. the security level of an actor is also specified by the creator. An actor whose security level L may create actors at a level which dominates L. If an actor $A1$ at level $L1$ creates an actor $A2$ at level $L2$ where $L2 > L1$, then the address of $L2$ is visible to $L1$. This means that any actor at level $L(L1 < L < L2)$ may send communications to $A2$. $A2$ will not be able to send any communications to the actors at level $L*$ $(L* < L2)$. We define a multilevel secure actor system (MLS/AS) to be a system of actors in which each actor is assigned a security level and the actors in the system send tasks in such a way that there is no information flow from a higher level to a lower level. Similarly, a multilevel actor model is an actor model for a multilevel environment.

Consider the example discussed in section 2. Suppose an actor X_n at level L processes the nth communication in its mail queue. This communication must have been sent by an actor at level L or below. X_n may create new actors at a level which dominates L, it may create additional tasks, and also creates an actor X_{n+1} and specifies a replacement behavior for

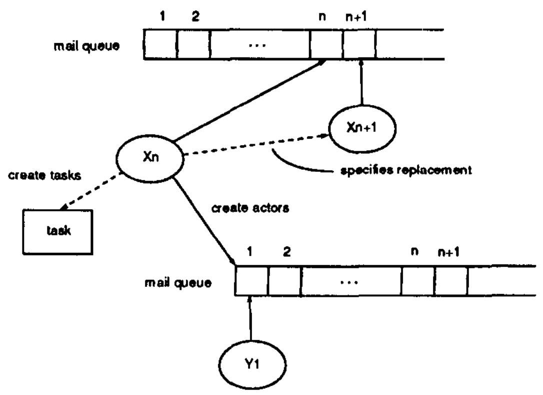

Figure 1: An abstract representation of transition

X_{n+1}. X_{n+1} will process the $n+1$th communication received. The question is should X_n and X_{n+1} be at the same level or could the level of $X+1$ dominate the level of X_n. Since X_n and X_{n+1} share the same mail address, whenever an actor sends a communication to this mail address it is reasonable to assume that the security level of the actor is the same as the first actor to be assigned to such an address. Therefore, in our proposed model, X_n and X_{n+1} are at the same level L. The essential points are illustrated in figure 2.

Next we formalize the notions discussed in the previous paragraphs. In particular, we define tasks, actors, and behaviors for a multilevel environment.

Suppose an actor A at level L creates a task t. Then t is a triple (i,m,k) where i is a tag, m is a mail address to which the task is being sent (i.e., the target address), and k is a communication. The task t has a security level and is equal to L. That is, we assume that any information that is created by an actor at level L must be classified at level L also. In our model, tags, mail addresses, and communications also have security levels. The security level of k and i are also L. However, if the creation of the task t resulted from some other task (possibly sent by a lower level actor) received by A, then information about that task may be embedded into it. The security level of the mail address m is dominated by L. This is because an actor can create actors at a higher level. Since m is visible to A, m may have been created by a lower level actor, in which case m is assigned the level of its creator. That is, an actor at level L can have a mail address at a lower level.

The set of all possible tasks T is defined by

$$T = I \times M \times K \qquad (1)$$

where I is the set of all possible tags, M is the set of all possible mail addresses, and K is the set of all possible communications.

The set of all possible actors is given by

$$ACT = M \times B \qquad (2)$$

where M is the set of all possible mail addresses and B is the set of all possible behaviors. Each actor A in ACT is a pair which consists of a mail address and a behavior. the security level L of A must dominate the levels of its address and behavior. This is because the actor who creates A assigns a behavior to the mail address created for A.

Let b be the behavior of an actor A at mail address m, which processes a task with tag t and communication k. The behavior is a function which is defined as follows:

$$b(k,m,t) = (T*, ACT*, A*) \qquad (3)$$

where $T* = *(p_1, p_2, \cdots, p_n)$ is a set of tasks created, and $ACT* = (A_1, A_2, \cdots, A_m)$ is a set of actors created, and $A*$ is an actor which shares the same mail address as A.

The following conditions hold:
(i) The tag t of the task processed is a prefix of all tags of the tasks created. That is:

$$\forall i(1 < i < n \Rightarrow m_i \in M \exists k_i K t'_i \in I(p_i = (t.t'_i, m_i, k_i))) \qquad (4)$$

Furthermore, the level of p_i is the same as that of A.
(ii) The tag t of the task processed is a prefix of all mail addresses of the actors created. That is:

$$\forall i(1 < i < m \Rightarrow bi \in B \exists t'_i \in I(A_i = (t.t'_i, b_i))) \qquad (5)$$

Furthermore, level of A_i must dominate the level of A.

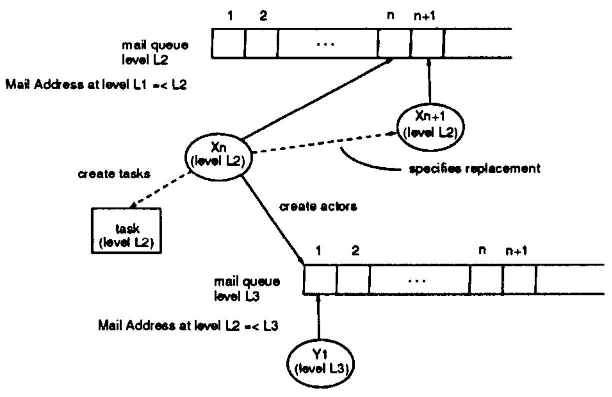

Figure 2: An abstract representation of transition in a multilevel secure actor system

(iii) Let $I*$ be the set of tags of newly created tasks and $M*$ be the set of mail addresses of newly created actors. then no element of $I * UM*$ is the prefix of any other element of the same set.

(iv) There is always a replacement behavior b'. That is:

$$\exists b' \in B(A* = (m, b')). \tag{6}$$

Furthermore, the levels of A and $A*$ are the same.

4 A Note on Configurations and Transitions

At any instant, an actor system is defined by its configuration. A configuration of such an actor system is described by the actors and tasks it contains. To define configurations, we first define a local states function. A local states function F is a function whose domain is $M*$ and its range is B where $M*$ is a finite set of mail addresses and B is the set of all possible behaviors. That is, a local states function defines the actors of the systems by assigning behaviors to mail addresses. A configuration is a pair $(F, T*)$ where F is the local states function and $T*$ is a finite subset of the tasks T such that (i) no task in $T*$ has a tag which is a prefix of either another tag of a task or of a mail address in the domain of F and (ii) no mail address in the domain of F is the prefix of either another mail address in the domain of F or of a tag of a task in $T*$. These restrictions are necessary to ensure that for a given configuration, there exist transitions with unprocessed tasks. This way, an actor system can evolve.

The evolution of an actor system is defined by the initial configuration and transitions between the configurations. One initial configuration consists of a set of actors and tasks that are created initially. The transitions in an actor system are quite different from a sequential possibly non deterministic model. While in a nondeterministic sequential process a unique transition does occur, as stated in [AGHA86], in concurrent systems such as actors, many transition paths with different viewpoints may be consistent representations of the actual evolution.

Because of the complexities involved in the actor system, could the usual techniques that have been used to prove that a system is secure be applied for such systems? Usually it is shown that the initial state of the system is secure and that state transitions maintain the security properties. As stated earlier, the transitions in a concurrent system are not straightforward and therefore the traditional approach to proving that a system is secure may not be sufficient. Research needs to be carried out in order to determine ways of proving the security of concurrent processing systems.

5 Related Work

Although security issues for the concurrent computational models such as actors are yet to be investigated, the work that has been done so far on object-oriented database system security is somewhat related. Much of the work on object-oriented database systems security (see, for example, KEEF88, THUR89a, MILL89, THUR90) assume a passive model of objects. That is, the objects contain data values and subjects, which are the active entities such as processes, send messages to objects to execute certain methods and retrieve or update the values. The earliest work on an active model of objects was proposed in [THUR89b]. This model incorporated security into the active model proposed in [ROSZ89]. A more detailed investigation of security for such a model was described in [JAJO90]. However, concurrent execution and cooperation was not a consideration in these active models.

The main difference between the active models proposed in the object-oriented database security work and the actor model proposed here is that the objective of the actor model is to exploit concurrent computations as well as ensure cooperation. The active object-oriented models do not create new objects. It is assumed that the objects already exist and messages are sent in order to retrieve and update values. The messages are intercepted by a trusted filter. In the actor model, new actors are created when communication is received in order to exploit concurrent problem solving.

6 Conclusion

In this position paper, we first described the essential points of the actor model of concurrent computation. As stated earlier, the actor model is particularly useful for concurrent and cooperative problem solving applications. Next we proposed a secure model for concurrent computation which is based on the actor paradigm.

Much remains to be done before an MLS/AS can be developed. First of all, we did not consider all of the constructs of the actor model in our discussions. That is, only a very small subset of the constructs were considered. In order to develop a useful MLS/AS, the security issues for the complete actor model must be investigated. Also, our approach is one way to securing the actor model. Different alternatives need to be explored before one can be selected. Even with the model that we have proposed here, we need to prove that there is no information flow from a higher level to a lower level. As stated in section 4, the issues involved may be quite different to those for sequential processes.

Since the actor model is being proposed for a variety of systems including massively parallel architectures and cooperative computing applications, we envisage that a MLS/AS could be used for multilevel parallel processing and cooperative computing applications. We also envisage that the actor model could be used for implementing role-based security policies. The work described in this paper is just the first step towards developing an MLS/AS.

Acknowledgements

I thank Jonathan Millen and Arnon Rosenthal for their comments on this paper.

References

[AGHA86] Agha, G., ACTORS: 1986, *A Model of Concurrent Computation in Distributed Systems*, M.I.T. Press, Cambridge, MA.

[DAHL70] Dahl, 0. et al., 1970, Simula Common Base Language, Technical Report S-22, Norwegian Computing Center.

[FOUN90] *Proceedings of the Third Computer Security Foundations Workshop*, June 1990.

[GASS88] Gasser, M., *Building Secure Systems*, 1988, Van Nostrand, New York.

[HEWI77] Hewitt C. and H. Baker, 1977, "Laws for Communicating Parallel Processes," *IFIP Conference Proceedings.*

[HOAR78] Hoare, A. 1978, "Communicating Sequential Processes," *Communications of the ACM*, Vol. 21, no. 8.

[JAJO90] Jajodia S., and B. Kogan, 1990, "Integrating an Object-Oriented Data Model With Multilevel Security," *Proceedings of the IEEE Symposium on Security and Privacy*, Oakland, CA.

[KEEF88] Keefe, T., W. T. Tsai, and B. M. Thuraisingham, October 1988, "A Security Policy for Object-Oriented DBMS," *Proceedings of the 11th NCS Conference.*

[MILL89] Millen, J. and T. Lunt, 1989, "Security for Knowledge Base Management Systems," Technical Report, MTR 686, The MITRE Corporation, Bedford, MA.

[ROZE89] Rozenshtein, D. and N. Minsky, 1989, "A Law-Governed Object-Oriented System," *Journal of Object-Oriented Programming*, Vol. 2, no. 2, March/April.

[THUR89] Thuraisingham, B. M., October 1989, "Mandatory Security in Object-Oriented Database Management Systems," *Proceedings of the ACM Conference on Object-Oriented Programming Systems, Languages and Applications (OOPSLA)*, New Orleans, LA.

[THUR89b] Thuraisingham, B. M., and F. Chase, 1989, "An Object-Oriented Approach to Developing Secure Software Systems," *CIPHER* (IEEE).

[THUR90] Thuraisingham, B. M., March/April 1990, "Security in Object-Oriented Database Systems," *Journal of Object-Oriented Programming*, Vol. 2, no. 6.

[WENG75] Weng, K., 1975, Stream-Oriented Computation in Data Flow Schemas, TM 68, MIT Laboratory for Computer Science.