A Security Evaluation Model for Multi-Agents Distributed Systems

Chunyan Ma
Department of Computer Science
California State University
San Bernardino, CA, 92407 USA
chuma@csci.csusb.edu

Arturo Concepcion
Department of Computer Science
California State University
San Bernardino, CA, 92407 USA
concep@csci.csusb.edu

Abstract

In this paper, we propose a mathematical model for evaluating the security of multi-agent distributed systems. Using the developed security risk graph to model the system's vulnerabilities, the transition time on the shortest path between two vertices can be calculated to evaluate the estimated breach time. The diameter of the system is used to represent the security measure of the entire system. Thus, the security level of different systems can be compared.

1. Introduction

Over the years computer systems have evolved from centralized computing devices supporting static application, into client-server environments that allow distributed computing due to the advances in communication technology and the occurrence of powerful workstations. A new phase of evolution is now under way in which complete mobile software agent can be sent among supporting platforms to form a large scale distributed system. A mobile agent [6] is a program that represents a user in a computer network, and is capable of migrating autonomously from node to node, to perform computation on behalf of the user. Its tasks can range from online shopping to real-time device control to distributed scientific computing. Having accomplished their goals, the agents may return to their "home site" in order to report their results to the user.

By using agent technology, we can move the code to the remote data to avoid the difficulty of moving the large volumes of data. We can also send out the agent to access the remote resources without keeping the network connection alive all the time. Despite its many practical benefits, mobile agent technology results in significant new security threats from malicious agents and hosts. A malicious agent can corrupt information on its host and in other agents as well. It is even more difficult to prevent a host from changing an agent's states or even killing it. Therefore, solving the security problems of multi-agent distributed systems is crucial. It would be especially ideal if we can have a method to evaluate how secure an agent or a host is. So that we can compare the security of different mobile-agent distributed systems by using quantitative measurements. Currently there is not much work done in this field. [1] and [2] are the few attempts to quantitatively measure the security of the mobile agent distributed systems. In this paper, we develop the security risk graph to model the system's vulnerabilities. Based on the analysis of the different security threat situations in a mobile agent system, we derive a mathematical security model for quantitatively evaluating the security of an agent-based distributed system.

This paper is organized as follows: Section 2 gives the taxonomy of security threats for an agent-based system and describes other related work. In section 3, we will present a mathematical model for security assessment of an agent-based distributed system, followed by an example to explain how this model works. The section 4 provides the conclusions and future directions.

2. Related works and the taxonomy of security threats for mobile-agent systems

Dispatching the application to other computers in the network can reduce the network communication overhead, get more resources and introduce concurrency. However, more security threats emerge along with this new technology. In this section, we are going to discuss the related works in the security evaluation of the mobile-agents systems and develop a security threats taxonomy on which we base our security model.

2.1. Related works

Presently only two papers proposed probabilistic security evaluation models for agent-based distributed system. Chan and Lyu [1] proposed the idea about coefficient of malice k_i of each host and coefficient of vulnerability of an agent ν and used them to calculate the probability of breaches on an agent when it travels around on each host as

P(breach at host i) = $1 - \exp(-\lambda_i t_i)$, where $-\lambda_i = \nu k_i$,

t_i is the amount of time during which the agent stays on host i.

The agent security E is the probability of no breach at all hosts in its itinerary, $E = \exp(-\sum \lambda iti)$.

Concepcion and Ma [2] proposed to use the probabilistic Mean Effort To Failure (METF), as the measure of the system security. The mean effort in node

j, denoted as E_j, is given by the inverse of the sum of state j's output transition rates:
$E_j = 1/ \Sigma \lambda_{jl}, l \in out(j)$, where λ_{jl} is the transition rate from node j to node l.

The transition rate is the success rate of the corresponding attacks, defined as 1/t, where t is the time needed for completing the attack. Then the mean effort to failure $METF_k$ when node k is the initial node and P_{kl} is the conditional transition probability from node k to node l, is:

$METF_k = E_k + \Sigma P_{kl} * METF_{kl}$; where $P_{kl} = \lambda_{kl} * E_k$

2.2. Taxonomy of security threats

Considering the consequence of the security breaches, the traditional taxonomy of the security threats identifies three main categories [3] as: confidentiality, integrity and availability.

- Confidentiality is violated when unauthorized principals can learn protected information.
- Integrity is infringed when unauthorized principals modify information.
- Availability is breached when the system is prevented from performing its intended function.

However, when scrutinizing the scenarios in the mobile-agent distributed systems, we identified a new type of security threat - repudiation, i.e., when one party in a communication exchange or transaction later denies that the transaction or exchange took place. Since the repudiation relates with the trust and credit history, we name this category of security threats as creditability. Therefore, now we have the forth category of the security threats in the agent-based systems considering the consequence of the security breaches. We define the creditability as following:

- Creditability is violated when principals deny having performed a particular action.

The basic elements in a mobile-agent distributed system are the agents and the hosts. The hosts along with the underlying networks are the basic environment for the agents to execute. When considering the relationships between the actors in the mobile-agent systems, there are four main categories identified [4]: threats occurred when an agent attacks an agent platform; when an agent attacks against other agents; when an agent platform attacks an agent, and when an agent attacks the underlying network. Based on this structural characteristic of the agent-based systems, we can partition the security threats into two broader categories. As shown in Figure 1, security threats can be towards the host or towards the agent. They both have two subcategories as shown:

- the security breaches towards the hosts:
  - the malicious agent against agent platform
  - against the underlying network
- the security breaches towards the agents:
  - the malicious host against agent
  - agent against other agents

For each subcategory of the security threats, we have identified the possible security requirements when considering the consequence of the security breaches. As shown in Figure 1, the agent-against-network category will not violate the creditability security threat. Because if an agent has used the network to transfer information through the network connections, there is no possibility to deny that it has used the service.

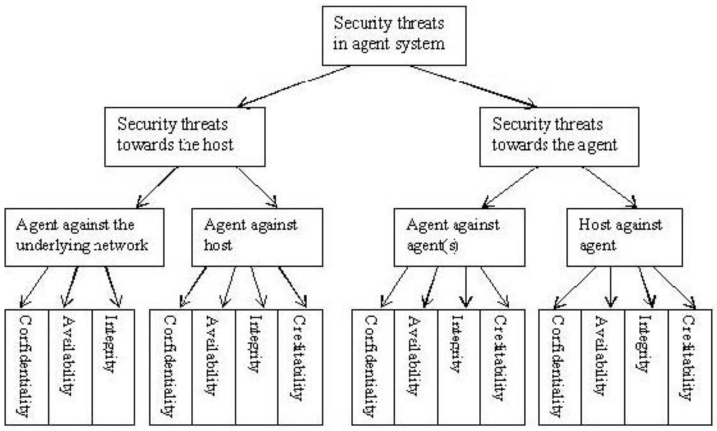

Security threats in agent system
Security threats towards the host
Security threats towards the agent
Agent against the underlying network
Agent against host
Agent against agent(s)
Host against agent
Confidentiality
Availability
Integrity
Creditability

Figure 1. Taxonomy of the security threats the agent-based systems.

3. A mathematic model for agent-based distributed system security threat evaluation

In the model proposed by Chan and Lyu [1,] they did not consider the security threats of agent-against-host, agent-against-agent and agent-against-network scenarios. Also the concepts of the coefficient of malice and vulnerability in Chan and Lyu [1] are not well defined. It is not clear on how to obtain the coefficient of malice and vulnerability. While Concepcion and Ma [2] have considered all the security threat types, but the Mean Effort To security Failure calculation is too complicated. In this section, we will introduce the security risk graph. Based on the security risk graph and the threats taxonomy, we will set up a mathematical model to evaluate the security of the mobile-agent systems.

3.1. Introduction of the security risk graph

A security risk graph consists of a set of vertices and edges.

Definition 1. A vertex of a security risk graph is an agent or a host in the agent-based distributed system.

Definition 2. An edge in a security risk graph is an arc from vertex X to vertex Y, represented as (X, Y).

An edge starts from vertex X and ends at vertex Y in the security risk graph means that a method exists for X to launch attack to Y.

Definition 3. A security threat of type r exists from vertex X to vertex Y means that there exists a method for vertex X to perform a type r attack to vertex Y.

For each type of security threat, there is an average access time associated. We call it the transition time of a specific type of security threat.

Definition 4. Transition time is the time needed for a specific type of security threat r to succeed from one vertex to the next vertex.

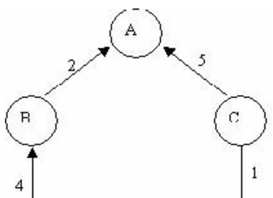
Here the transition time can be obtained from the statistical estimation of agent's and host's profiles based on the analysis of the agents' and hosts' behavior and of their interaction with each other. By observing the system, we can get the transition time indicating how hard for one vertex to perform one particular attack to another vertex and assign that value to the same kind of attack identified from the system we want to analyze.

Definition 5. The weight of each edge is the transition time of each edge.

Definition 6. A directed path in a security risk graph is a sequence of vertices along with the edges in between of them such that, for any adjacent pair of edges e_i and e_j, the ending vertex of e_i is the starting vertex of e_j. In this paper we call a directed path as a path.

Definition 7. A security risk graph of an agent-based distributed system is a directed and weighted graph G(V,E, W) where V is a set of vertices, W is the set of weights and E is a set of edges between the vertices, $E = \{w(u,v) \mid u, v$ □ V, w □ $W, u \neq v\}$.

Figure 2 shows an example of a security risk graph. A, B, C, D and E are the vertices and the edges are labeled by the security threat types. Note that only when a path between an attacker and a target exists, is there a possibility that a security breach can occur. For instance, as illustrated in Figure 2, B cannot gain access to E through any path. So B cannot be a potential attacker to E. By formalizing this intuitive idea, we can get the following corollary.



1) X can read files from Y (intercept);
2) X can guess Y's password;
3) X can get hold of Y's CPU cycle;
4) Y has no password;
5) Y uses a program owned by X.

Figure 2. An example of a security risk graph with edges labeled by security threats.

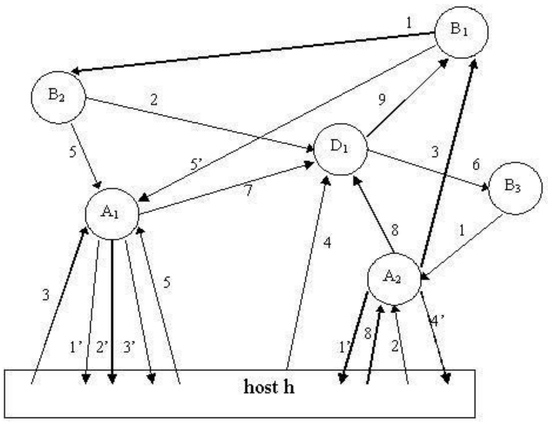
Corollary 1. A security threat exists from one vertex X to another vertex Y whenever there is a path that starts from X and ends at Y.

Proof: Let us consider about the reciprocal statement of this Corollary. That is, "If there is no path starts from vertex X and ends at Y, there exists no security threat from vertex X to Y". Since we know this reciprocal statement holds and the proof is trivial, we can deduce that Corollary 1 is true.

3.2. Security risks analysis

Based on the taxonomy of the agent-based system vulnerabilities, we can deal with the security situation for a host and an agent separately.

Combine the security risks related with agent against host and agent against the underlying network, we can analyze the security risks a host needs to consider. As shown in the example in Figure 3, the security risks labeled in the numerical values and a prime are the security breaches to the host.



1) X can guess Y's password;
2) X can eavesdrop Y's communication with others;
3) X has write access to Y (alteration);
4) X can masquerade as another platform to Y;
5) Y uses a program owned by X;
6) X can repudiate the result from Y;
7) X can copy and replay Y's information;
8) Y has no password;
9) X can deny the service to Y;
1') X can read files from Y;
2') X can write files to Y;
3') X can get hold of Y's CPU cycle;
4') X can get hold of network resources;
5') X can masquerade as another agent Y to the platform.

Figure 3. Security risk graph example for agent-based system.

Thus when we analyze the security risks of a host, we can discard all the edges from the host to agents and some of the edges between agents that do not count for attacking the host and the underlying network.

We find that the masquerade is a tricky type of security threat because both the agent and the host can masquerade as another one. When vertex X masquerades as another vertex Y to a third vertex Z, is it an attack to Y, or Z or both? An example can be seen in Figure 3. In this example, agent B_1 can masquerade as agent A_1 to the host. We need to decide if this attack is toward agent A_1 only, the host only or both.

Definition 8. We call vertex X equals vertex Y if vertex X's behavior looks the same as vertex Y's behavior, denoted as X = Y.

Definition 9. Masquerade is the act of imitating the behavior of vertex X to vertex Y under false pretense, denoted as X(Y).

From Definition 8 and Definition 9, we know when vertex X masquerades as Y, its behavior looks the same as Y's behavior, that is X(Y) = Y.

Definition 10. The behavior of B as seen by C is denoted as $B \gamma C$.

Unlike other kinds of security threats, masquerade is a very special type of security threat because it has the following characteristics.

Masquerade Transition Law: If Entity A can masquerade as Entity B to Entity C, then that is an attack from Entity A to Entity C.

Proof: To C, Entity B is as itself, so $B \gamma C = B$.

When under masquerading, Entity A acts as B, so $A \gamma C = A(B) \gamma C = B \gamma C$, because $A(B) = B$.

So we have $B \gamma C = A \gamma C = B$.

Because A behaves itself to Entity C under the name of B, to obtain the privileges of C, so that is an attack from A to C.■

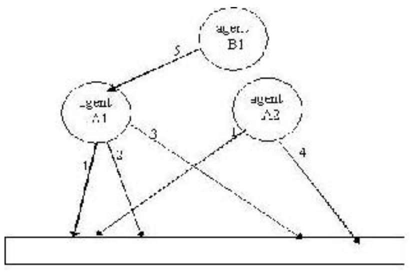
Saying that masquerade is special is because other types of security threats do not necessarily have this feature. For example, If vertex A can intercept the information of vertex C, vertex B can also intercept the information of C, then A is not necessarily definitely able to intercept the information of C.

Corollary 2. If a vertex A can masquerade as another vertex B to the third vertex C, then this is a security risk from A to B, also a security risk from A to C.

Proof: First to prove this is a security risk from A to B is trivial. Because A masquerade as B, whatever A does has affected B's reputation. So it is an indirect security risk from A to B.

Also by using the masquerade transition law, we know it is a security risk from A to C.■

Take Figure 3 as an example, agent B_1 can masquerade as agent A_1 to the host. So that is a security risk to agent A_1 and to the host as well. By using Corollary 2, we can leave agent B_1 in the graph for the analysis of the security risks for the host. Because we discover an agent could masquerade as another agent to the platform. We regard it belongs to the security risks a host needs to face, also as a security attack form between agents. Then we can isolate the security risks the host will face, see Figure 4.

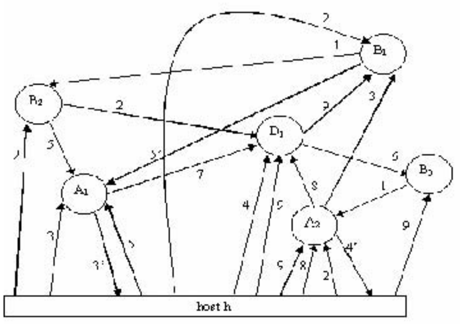


1) X can read files from Y (intercept);
2) X can write files to Y (alteration);
3) X can get hold of Y’s CPU cycle;
4) X can get hold of network resources;
5) X can masquerade as another agent Y to the platform.

Figure 4. Security risk graph analyzed the security risks of the host h.

By far, we have used an example to illustrate how to analyze the security risks a host will face in the system. Our basic idea is to eliminate all the unnecessary edges and vertices to make all the connections to this host stand out.

But for analyzing the security of the agents, using Figure 3 as an example, we cannot just discard all the edges in Figure 4. Because one agent can take all host h’s CPU cycles, so as to deny the service to other agents. For instance, as shown in Figure 5, A_1 can get hold of host’s CPU cycle, thus launch denial of service attack to every agent running on host h. Note that in Figure 3, we don’t have the edges of type 9 from host h to each agent. By the process of analyzing the security “Towards the agent” scenarios, we found that we should add those edges. In fact, we can generalize this observation into the following theorem for analyzing the security risk graph of the agent-based system.



1) X can guess Y’s password;
2) X can eavesdrop Y’s communication with others;
3) X has write access to Y(alteration);
4) X can masquerade as another platform to Y;

5) Y uses a program owned by X;
6) X can repudiate the result from Y;
7) X can copy and replay Y's information;
8) Y has no password;
9) X can deny the service to Y;
3') X can get hold of Y's CPU cycle;
4') X can get hold of network resources;
5') X can masquerade as another agent Y to the platform.

Figure 5. Security risk graph for agents on the platform

Theorem 1. If an agent A on host i can take all of host i's CPU cycles, it can in turn launch denial of service attack to all of the agents running on this host except for A itself. If there are more than two agents on the same host i can take all of i's CPU cycles, the first one launches the attack can succeed the attack.

Proof: Since all of the agents running on a host need to utilize CPU cycles for its performance, if the CPU is hold completely by one agent, then the other agents cannot function correctly. If the first agent can get hold of all CPU cycles successfully, all the other agents running on host i can not even function correctly. They would have no chance to succeed in taking hold of all CPU cycles any more.

Due to the fact that an agent or a host in the agent-based distributed systems can launch many different types of attacks, we face a problem of how to analyze them. For example, as shown in Figure 6, when agent B_2 is the attacker and agent B_1 is the target, we have an edge loop back to its ancestor, like $host_h A_1$. In Figure 4 and Figure 5, agent A_1 can perform 3 different attacks to host h (in Figure 4) and host h can launch four different attacks to agent A_2 (in Figure 5). Which one should we choose for calculation? In responding to these problems, we have developed the following theorems to handle them.

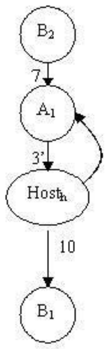


Figure 6. Security risk graph for B_2 as the starting point and B_1 the target

Theorem 2. If there are several edges with the same direction from one vertex to the next in the security risk graph, the edge with the smallest transition time will be chosen in the calculation.

Proof: Without losing generality, as seen in the security risk graph (Figure 7), suppose A is an intruder and B is a target. There are several edges from A to B, AB_1, AB_2, ..., Ab_n. Each edge has a transition time t_1, ..., t_n associated with it. Suppose $t_i = \min\{ t_1, ... t_n \}$. Since the intruders do not know the whole topology of the security risk graph. They only know the attacks that can be directly applied in a single step. So A has the options t_1, t_2, …, t_n to attack B. From the empirical results obtained from Jonsson and Olovsson [5], the intruder A would always try to perform the attack takes the least time that is t_i.■

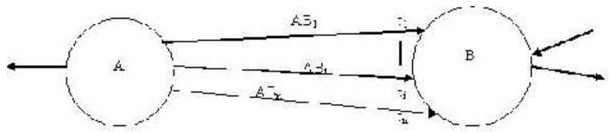


Figure 7. Security risk graph for A as the intruder and B as the target.

Theorem 3. If there is an edge from one vertex that goes back to its ancestor, then this edge would not be counted in calculation.

Proof: From the attacker's point of view, he would choose a route that takes the least time. Reflected in the security risk graph, the attacker's goal is to choose the branch that takes the least time.

Case I. Edge from one vertex goes back to its parent.

Without losing generality, suppose there is one branch in the security risk graph that has an edge from one vertex B_l goes back to its parent A_i, as circled part in Figure 8.

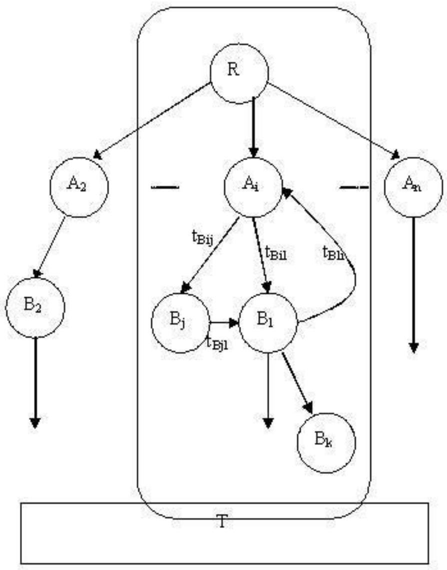


Figure 8. Security risk graph for edge from one vertex goes back to its parent case.

Then the time taken by each of all the other branches is just the sum of all the edges in that branch. From the security risk graph, we can see that if the time taken from R go all the way down to T is t (t is chosen by selecting the smallest value among different routes). But if we loop back at B_l,

for route A_i - B_l, total time = $t + t_{Bli} + t_{Bil}$, where $t_{Bli} + t_{Bil} > 0$.

So total time > t.

for route A_i - B_j - B_l, total time = $t + t_{Bij} + t_{Bjl} + t_{Bli}$, where $t_{Bij} + t_{Bjl} + t_{Bli} > 0$.

So total time > t.

That means whatever the routes in between vertex A_i and B_l, total time $> t + t_{Bli} > t$. So we discard the edges that loop back to the parent.

Case II. Edge from one node goes back to its ancestor.

Similar to case I, the time taken by the branch edge from one vertex n_i goes back to its ancestor is greater than the time taken from R go all the way through A_i, n_i down to T, as in Figure 9. If t is the time taken to loop back from n_j to A_i, $t > 0$.

So we discard the edges that loop back to the ancestor of this vertex n_i in calculation.■

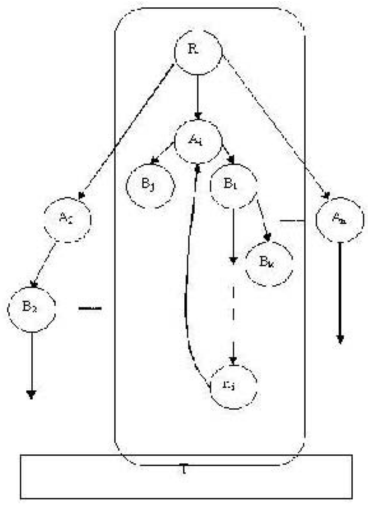


Figure 9. Security risk graph for edge from one vertex goes back to its ancestor case.

3.3. A mathematical security model for security evaluation

After we have modeled the system's security risks using the security risk graph, based on the analysis of the security threat types, we developed Corollary 2 and Theorem 1 to identify some special kinds of attacks and add all the necessary edges. Then the developed Theorem 2 and Theorem 3 are used to simplify the generated security risk graph so that we can have a cleaner graph for calculation. In this section, we will introduce a security model for evaluating the security of agent-based distributed systems using shortest path.

Definition 11. A shortest path from vertex u to vertex v is defined as any path with weight $\delta(u, v) = \min\{w(P) \mid P(u \sim v)\}$, where $P(u \sim v)$ is the set of paths from vertex u to vertex v, and w(P) is the set of weights of each path in $P(u \sim v)$.

Definition 12. Let P be a path containing vertices $v_1, v_2, \ldots v_n$, and $w(v_i, v_j)$ be the weight on the edge connecting v_i to v_j, then the length of path P is defined as

$$|P| = \sum^{n-1} w(v_i, v_{i+1}).$$

Because the time on the shortest path describes the least time the attacker will need to break into the target. If the attacker does not know the topology of the whole system, the time needed to break into the target will be definitely more than or equals to the time calculated from the shortest path. So the shortest path is a suitable measure for the system administrators to evaluate the system's security level.

The following algorithm based on the Dijkstra's algorithm can be used to find the security measures. In the next section, we are going to give an illustrative example to show how this method works.

```
Security risks estimation algorithm:
Input: Weighted graph G, source, destination (G is the simplified graph using
Theorems 2 and 3)
 Output: Transition time from source to destination
Temp: temporary tree structure to hold the nodes and edges as we go through
graph G
1. add source, Transition time (source) = 0 to Temp
2. while (destination $\notin$ Temp)
   find edge (u, v), where:
     a. u $\in$ Temp;
       b. v $\notin$ Temp;
       c. minimize the transition time over all (u, v) satisfies a and b.
```

The resulted transition time = transition time(u) + w(u, v), where w(u, v) is the weight of (u, v).

Actually, Dijkstra algorithm can find the shortest path to every vertex to which the source vertex has a connection besides the target vertex. It has the same time complexity as the one needed for just finding the shortest path to the target.

The shortest minimum length path between any two vertices represents the weakest security point and the longest shortest path describes the ultimate time the attacker needs to break the whole system at most.

Definition 13. The diameter of a security risk graph is the length of the longest shortest path between any two vertices.

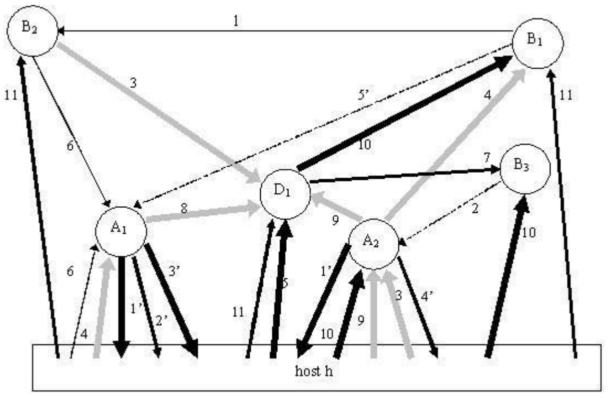
The diameter can be used to represent the security level of the whole system because it is the least time the attacker needs to break into the whole system. Thus we can use the diameters to compare the security of different system. If after reconfiguration, the diameter of the whole system increases, we can say that the whole system's security increases because the time needed to break into the hardest point of the system increases.

3.4. Illustrative example

Now let us use an example to illustrate how this approach works. We still use the example in Figure 3. Figure 10 shows that the edges are assigned different thicknesses to represent their weight and also to characterize the difficulty of the breaches: the thicker the edge, the easier the breach. For the convenience of calculation, we use one week as the unit of attack times. For instance, one day is approximately 0.2 week. Table 1 lists the transition time, its corresponding time in weeks and the graph representation in the security risk graph for the identified transition time of the attacks.

Table 1. Transition time, its corresponding time in weeks, transition rate and graph representation.

Transition time	Transition Time in weeks	Line type in the security risk graph
Quasi-instantaneous	0.0002	⇨
one hour	0.02	➡
one day	0.2	→
one week	1	→
one month	5	⇢
one year	50	⇢



1) X can guess Y’s password in one week;
2) X can guess Y’s password in one month;
3) X can eavesdrop Y’s communication with others (quasi-instantaneous);
4) X has write access to Y (alteration) (quasi-instantaneous);
5) X can masquerade as another platform to Y (one hour);
6) Y uses a program owned by X once in a year (one year);
7) X can repudiate the result from Y in one day (one day);

8) X can copy and replay Y's information (quasi-instantaneous);
9) Y has no password (quasi-instantaneous);
10) X can deny the service to Y in one hour;
11) X can deny the service to Y in one day;
1') X can read files from Y in one hour;
2') X can write files to Y in one day;
3') X can get hold of Y's CPU cycle in one hour;
4') X can get hold of network resources in one day;
5') X can masquerade as another agent Y to the platform in one month.

Figure 10. Security risk graph example with weight demonstrated in different line type.

To illustrate how the security evaluation algorithm works, we take B_2 as the attacker, A_2 as the target from Figure 10 and generate the Markov graph as in Figure 11.

By using Theorem 3, we can eliminate the edges B_1B_2, to get Figure 12. Also for transition time of edge A_1Host_h we can choose the one that gives the smallest transition time based on Theorem 2.

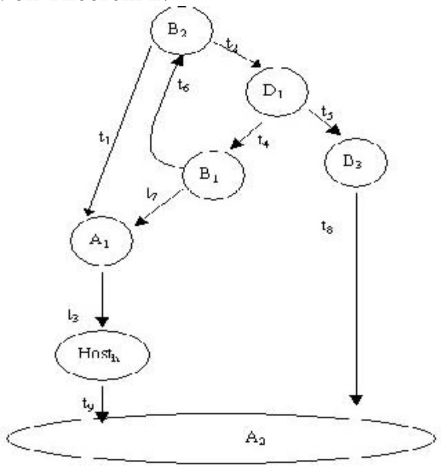


Figure 11. Markov graph for B_2 as the attacker and A_2 as the target.

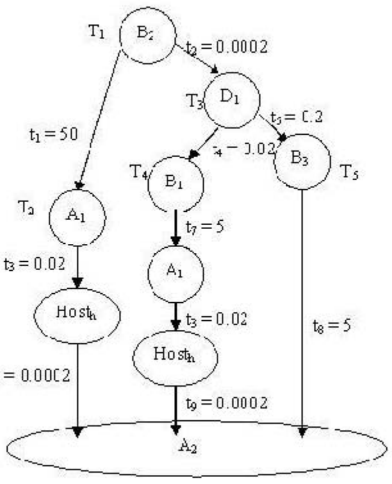


Figure 12. Simplified Markov graph for figure 11 by using Theorem 2 and Theorem 3.

Following the algorithm, the example in Figure 12 works as below:

Take the source vertex B_2 and put it in Temp. Temp = $\{B_2\}$

Since B_2 connects to A_1 and D_1, we mark A_1 and D_1 as candidates.

Compare $| B_2A_1 | = 50$ and $| B_2D_1 | = 0.0002$. Because $| B_2D_1 |$ is smaller, we take D_1 into Temp. Now Temp = $\{B_2, D_1\}$ and we also get the shortest path between B_2 and D_1 is B_2D_1 with

$| B_2D_1 | = 0.0002$.

Since D_1 is in Temp, now we need to consider the vertices connected to D_1: B_1 and B_3. After we mark them, our candidates are A_1, B_1 and B_3.

Compare:

$| B_2D_1B_3 | = 0.0002 + 0.2 = 0.2002$

$| B_2D_1B_1 | = 0.0002 + 0.02 = 0.0202$

$| B_2A_1 | = 50$

Because $| B_2D_1B_1 |$ is smaller, we take B_1 into Temp. Now Temp = $\{B_2, D_1, B_1\}$ and the shortest path between B_2 and B_1 is $B_2D_1B_1$ with $| B_2D_1B_1 | = 0.0202$.

Now that B_1 is in Temp, we need to consider the vertices connected to B_1: A_1. After we mark it, our candidates are A_1, and B_3.

Compare:

$| B_2D_1B_1A_1 | = 0.0002 + 0.02 + 5 = 5.0202$

$| B_2D_1B_3 | = 0.0002 + 0.2 = 0.2002$

$| B_2A_1 | = 50$

Because $| B_2D_1B_3 |$ is smaller, we take B_3 into Temp. Now Temp = $\{B_2, D_1, B_1, B_3\}$ and the shortest path between B_2 and B_3 is $B_2D_1B_3$ with $| B_2D_1B_3 | = 0.2002$.

Now that B_3 is in Temp, we need to consider the vertices connected to B_3: A_2. After we mark it, our candidates are A_2, and A_1.

Compare:

$| B_2D_1B_1A_1 | = 0.0002 + 0.02 + 5 = 5.0202$

| $B_2D_1B_3A_2$ | = 0.0002 + 0.2 + 5 = 5.2002

| B_2A_1 | = 50

Because | $B_2D_1B_1A_1$ | is smaller, we take A_3 into Temp. Now Temp = {B_2, D_1, B_1, B_3, A_1} and the shortest path between B_2

and A_1 is $B_2D_1B_1A_1$ with | $B_2D_1B_1A_1$ | = 5.0202.

Since A_1 is in Temp, now we need to consider thevertices connected to A_1: $host_h$. After we mark it, our candidates are A_2, and $host_h$.

10.Compare:

| $B_2D_1B_3A_2$ | = 0.0002 + 0.2 + 5 = 5.2002

| $B_2A_1host_h$ | = 50 + 0.02 = 50.02

| $B_2D_1B_1A_1host_h$| = 0.0002 + 0.02 + 5 + 0.02 = 5.0402

Because | $B_2D_1B_1A_1host_h$| is smaller, we take $host_h$ into Temp. Now Temp = {B_2, D_1, B_1, B_3, A_1, $host_h$} and the shortest path between B_2 and $host_h$ is $B_2D_1B_1A_1host_h$ with

| $B_2D_1B_1A_1host_h$ | = 5.0402.

11.Since $host_h$ is in Temp, now we need to consider the vertices connected to $host_h$: A_2. After we mark it, our candidates are A_2.

12.Compare:

| $B_2D_1B_3A_2$ | = 0.0002 + 0.2 + 5 = 5.2002

|$B_2A_1host_hA_2$ | = 50 + 0.02 + 0.0002 = 50.0202

| $B_2D_1B_1A_1host_hA_2$ | = 0.0002 + 0.02 + 5 +0.02 + 0.0002

= 5.0404

Because | $B_2D_1B_1A_1host_hA_2$ | is smaller, we take A_2 into Temp. Now Temp = {B_2, D_1, B_1, B_3, A_1, $host_h$, A_2} and the shortest path between B_2 and A_2 is $B_2D_1B_1A_1host_hA_2$ with

| $B_2D_1B_1A_1host_hA_2$ | = 5.0404.

By using the above algorithm in all these steps, we have found all the shortest paths starting from agent B_2, and ending to every other vertex. We illustrate those shortest paths in Figure 13.

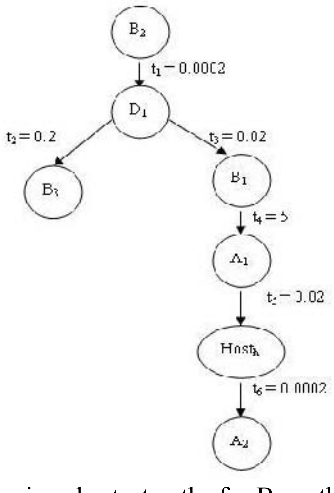


Figure 13. Graph showing shortest paths for B_2 as the initial vertex.

Following the method in this example, starting from each vertex, we can calculate the breach time to every other vertex respectively. The result of calculation is shown in Table 2. The breach time is represented in time duration as number of weeks.

Table 2. Breach time results (in number of weeks) calculated by using the proposed method.

End Start	A_1	A_2	B_1	B_2	B_3	D_1	Host$_h$
A_1	---	0.0202	0.0202	0.22	0.04	0.0002	0.02
A_2	0.0202	---	0.0002	1.0002	0.04	0.0002	0.02
B_1	5	5.04	---	1	1.2002	1.0002	5.02
B_2	5.0202	5.0404	0.0202	---	0.2002	0.0002	5.0402
B_3	5.2002	5	5.0002	5.4	---	5.0002	5.2
D_1	5.02	5.0406	0.02	1.02	0.2	---	5.04
Host$_h$	0.0002	0.0002	0.0004	0.2	0.02	0.0004	---

In the above example, since $| B_3A_2host_hB_2 | = 5.4$ is the longest shortest path for all the vertices. The diameter for this example is 5.4 weeks. If after reconfiguration, the diameter of the whole system increases to 6 weeks, we can say that the whole system's security increases because it needs more time to break into the hardest point of the system.

4. Conclusions and future directions

In this research, we have developed a security evaluation model using the shortest path to evaluate the security levels of the agent-based distributed systems by giving a mathematical measure to tell how secure a system is. By using this model, the system's administrator can not only evaluate the approximate breach success time between any two vertices, but also can evaluate the whole system's security risk. Thus we can have a way to compare the security between different systems. This model can be used to monitor the security evolution of the agents and hosts running in the system dynamically. It can also help the system administrators to manage the system's security and performance. The system administrators can evaluate the effectiveness of different configurations by comparing the values obtained from these different configurations.

By monitoring the system's risks, we can get the profile of the transition time of each type of security risks. We plan to use some probabilistic model to process the empirical data obtained from the observation.

Also, it would be desirable to apply some probabilistic method to the time value from the calculation so that it describes the security measure more accurately.

We plan to apply these models in Spider III, the multi-agent distributed system developed in CSUSB to study its feasibility.

5. Acknowledgements

The support of the National Science Foundation under the award 9810708, the Information Resources and Technology Division of California State University, San Bernardino are gratefully acknowledged.

6. References

[Chan 1997] A. Chan, M. Lyu, "Security Modeling and Evaluation for Mobile Code Paradigm", In proceedings of the Asian Computing Science Conference, 1997, pp. 371 - 371.

[Concepcion 2003] A. Concepcion, C. Ma, "A Probabilistic Security Model for Multi-Agent Distributed Systems", In proceedings of the 6th International Conference on Business Information Systems, June 2003.

[Humphries 2000] J. Humphries, "Secure mobile agent for network vulnerability scanning", In proceedings of the 2000 IEEE Workshop on Information Assurance and Security proceedings, United States Military Academy, West Point, NY, June 2000, pp. 6-7.

[Jansen 1999] W. Jansen, T. Karygiannis, "Mobile agent security", NIST Special Publication, October 1999, pp. 800-19.

[Jonsson 1999] E. Jonsson, T. Olovsson, "A Quantitative Model of the Security Intrusion Process Based on Attacker Behavior", IEEE Transactions on Software Engineering, April 1999.

[Karnik 1998] N. Karnik, A. Tripathi, "Design issues in mobile agent programming systems", Department of Computer Science, University of Minnesota, June 1998.