When agents roam, who can you trust?

Joann J. Ordille
Bell Labs
Innovations for Lucent Technologies
Computing Science Research Center
600 Mountain Ave., Murray Hill, NJ 07974
joann@research.att.com

Abstract

Roaming agents threaten a server with theft of assets, system resources and reputation. Generally ignored is the reciprocal threat of server against agent. Not only might a server steal the assets, resources, and reputation of an agent, but it might actually kill the agent. The central security concern for systems that allow agents to roam is how to establish trust, and how to limit risk for both the server and the agent. The travels of the agent are key to establishing trust, and trust determines the level of risk.

Introduction

When agents roam, who can you trust? It's tempting to say no one – not the agent, not even the server who hosts the agent. Agents threaten a server with theft of assets (information, money), system resources (through excessive consumption of cpu, bandwidth, disk space), and reputation. Reputations can be lost because the actions of a misbehaving agent can be wrongly attributed to the server. The agent might also alter the server causing it to fail or misbehave. Generally ignored is the reciprocal threat of server against agent. Not only might a server steal the assets, resources, and reputation of an agent, but it might actually kill the agent. A server steals resources from an agent when it implants its own tasks into an agent for execution at other servers. If the agent pays to execute the tasks, theft of assets occurs. If the tasks cause the agent to misbehave, the agent's reputation may be ruined. Many of these threats are discussed in detail in [Chess, 1996] and [Tardo and Valente, 1996].

Although it's tempting to say, "Trust no one," the situation is really a tradeoff between risk and trust. Safe languages such as Java [Gosling and McGilton, 1994], Safe-Tcl [Borenstein, 1994], and Telescript [Tardo and Valente, 1996], provide ways for limiting the risk of a server when it hosts an agent. The execution environments for these languages control the operations that can be executed, the system resources the operations can consume, and the assets the operations can access. Safe language approaches face several challenges including proper implementation [Dean and Wallach, 1995] and proper use [Chess, 1996]. The importance of trust increases whenever the control of risk decreases.

The central security concern is how to establish trust, and how to limit risk for both the server *and* the agent. Trusting an agent depends on the identity of the agent and where the agent has been. Has the agent changed for the worse during its travels? Trusting a server depends on the identity of the server. Interestingly it also depends on where an agent has been, because the agent requires the cooperation of its current server to arrive at its next destination. Has the agent been misdirected somewhere on its path?

Once the level of trust is established, the level of risk for server and agent can also be established. It is common for safe languages to limit the risk to a server differently depending on the owner of an agent. Telescript goes a step further by making the level of risk to a server contingent on the owner of the agent and the forwarding server that delivers the agent. In this paper, we show that the level of trust, and hence risk, depends not only on the owner and the forwarding server but on all servers visited by the agent.

Few techniques for safeguarding agents from their servers exist. Agents are so vulnerable that they must only visit servers who they trust with their very selves. An agent that has nothing valuable and is itself expendable might visit any server. An agent carrying monetary assets requires a high level of trust in its server. In the future, limiting the independence or obscuring the purpose of agents may further protect agents from servers. Limiting an agent's ability to make purchases without its owner's explicit approval would reduce the financial risk to the agent. Directing an agent to process more and different information than it needs would obscure (and hence protect) the true goal of the agent.

Traveling Agents

The travels of an agent are key to establishing a server's trust in the agent. A server that hosts an agent has access to the program and data of the agent. Although it may be possible to protect the program portion of an agent with a digital signature as suggested by [Chess et al., 1995] and [Tardo and Valente, 1996], it is not possible to protect the data that agents acquire for use as they travel. Servers can alter an agent's data and send the changed agent into the network. The agent may act on the new data in undesirable ways. Because agents that carry state, i.e. data that they use in their travels, are integrally tied to the servers where they acquire that state, knowing where an agent has been can be crucial to establishing a server's trust in the agent.

The travels of an agent are key to establishing the agent's trust in a server. During execution on a server, the agent is under control of the server. When the agent chooses to leave that server to visit another, it requires the cooperation of the server in being forwarded. The agent might, for example, authenticate its intended destination only to be shipped off to a different location. The identity of the intended destination and how the agent gets to the destination are important for establishing the agent's trust in the destination server.

The different travels of an agent impact the difficulty with which trust is established for server and agent. In general, agents can travel to many servers and may or may not return to their server of origin (or *home*). Each trip from one server to another is called a *hop*. Agents that travel many hops are called *multi-hop agents*. Establishing trust for the case of multi-hop agents is difficult. There are two special cases of traveling agents where establishing trust is simpler: one-hop agents and two-hop boomerang agents. A *one-hop agent* travels from its home to one other server, and then does not travel further. Examples include Java applets that are loaded into a Netscape browser, and information retrieval agents that search one server and return results to their owners by message passing. A *two-hop boomerang agent* travels from its home to another server and back again. An example of this agent is the agent that visits a server, purchases information, and returns home with the purchase.

One-Hop Agents

Establishing trust is simpler for one-hop agents than for multi-hop agents, because establishing responsibility for the agent is simpler. The one-hop agent travels from its home server to its execution server. The owner of the agent is responsible for problems with the agent. The execution server, on the other hand, is responsible for executing the agent in good faith, without modification and with reliable information. Ultimately, both the agent's owner and the execution server are responsible to the user for the agent's results[1].

There are several ways an execution server can establish a level of trust with the owner of the agent. In a basic system, the owner is the agent's home server or an individual authorized by that server. In this case, the execution server must evaluate the level of trust to be placed in the home server. In a more general system, the program and data of the agent can be digitally signed by the agent's owner, and authenticated by the execution server. In this case, the execution server can directly evaluate the level of trust to be placed in the owner of the agent.

Since the agent requires the mediation of the home server in enforcing its policy regarding trust, it is best for the agent's owner to specify the policy so the home server can enforce it directly. The policy might associate a level of trust with particular servers or with groups of servers administered by particular organizations. Alternatively, the policy might delegate the determination of trust to a completely trusted external authority. The home server authenticates the execution server for the agent and establishes the appropriate level of trust for the server based on the policy. Assuming that

[1] For simplicity, we assume in this paper that the user of the agent is either a trusted colleague of the agent's owner (or, in fact, the owner) or a user on the execution server who has requested the services of the agent. In either case, the user has a natural ally – either the agent's owner (for the trusted colleague) or the execution server (for the server's user). The user may supply additional assets, such as information, processing goals or monetary resources, to the agent through the ally. The user should be able to strengthen the trust and risk policies of the ally to reflect the user's concern for her/his assets. The user will also add concerns about the level of trust in the results of the agent. For simplicity, we bundle the trust and risk concerns of the user with the concerns of the ally, and do not consider them further in this paper. In the future, we will explicitly consider the role of the user in establishing levels of trust and risk.

the home server is completely trustworthy, the one-hop agent's level of trust in its execution server is determined solely by the trustworthiness of the execution server.

The level of trust helps establish what each party is willing to risk, as well as whether the results of the agent will be believed. Each party determines how to manage the risk involved in the execution of the agent. The server and the agent may each specify asset and resource requirements to the other in advance. Each party can then determine if it is willing to risk the requirements of the other. If the risk for either party is too high, the agent does not begin or continue executing on the execution server.

Similar to establishing the agent's trust policy, it is best for the agent's owner to specify the risk policy to the home server so the server can enforce it directly. For example, an information retrieval agent that carries a document to match for similarity with other documents on the execution server risks the document that it carries. The risk policy must specify the level of trust required in a server before the document can be risked. As another example, an information retrieval agent that returns information to its owner reveals the information needs of its owner to the execution server. The policy must specify the level of trust required in a server before the agent is willing to execute on that server. If the agent does not trust the server with the assets it will carry or its code resources, or does not trust the server to watch its execution, then it does not hop.

Two-Hop Boomerang Agents

Establishing trust for two-hop boomerang agents is similar to establishing trust for one-hop agents. Two-hop boomerang agents return to execute at home after running on the execution server. The two cases are similar, because the players are the same; each involves the owner of the agent, and the home and execution servers. The two cases differ, because the the home server is now at risk. The execution server does not incur any new risks with two-hop boomerang agents. While the one-hop agent can communicate results to its owner through messages, the two-hop boomerang agent carries its results home to its owner. In either case, the execution server risks theft of its assets and the agent risks miscommunication of its results.

Knowing the limited number of players before the agent executes is what makes the security of these two special cases simpler to manage than general multi-hop agents.

Multi-Hop Agents

When we consider multi-hop agents, we find ourselves in a different situation. Perhaps the best way to summarize our situation is to say: "The friend [of a friend of a friend ...] of your friend is not necessarily your friend." It is unreasonable to expect widely traveled agents to know all their destinations in advance, because they may be searching for new information and learning as they travel. Trusting an agent or server to behave properly is different from trusting either one to know who can be trusted.

Agents can determine levels of trust and risk in servers by looking ahead during their travels. At each hop, the agent chooses a destination that satisfies the agent's requirements. In the cases of one-hop and two-hop boomerang agents, the agent's owner specifies trust and risk policies for the home server to enforce. In the case of multi-hop agents, these policies must be carried from server to server for the execution servers to enforce at each hop in the agent's path. Agents must only travel to execution servers they trust to enforce these policies.

Execution servers, on the other, must look back at the history of the agent's travels to determine the appropriate levels of trust and risk. Each execution server can have a different level of trust in the servers along an agent's path, so it will not be possible to construct a composite level of trust as the agent travels that is useful to each new server visited. Each new server will need to determine the history of the agent in an efficient and trusted way in order to set its own levels of trust and risk. The new server can form its own composite level of trust by taking the minimum of its levels of trust for the agent and all the servers on the agent's path.

Establishing the level of a server's trust in an agent is a difficult and interesting topic for further research, because the two most straightforward techniques for determining the history of an agent involve considerable risk or cost. Both techniques require servers to annotate an agent with its path. In the first technique, each new server, S_i, adds itself to the annotation of the agent's path, and supplies a signed copy of that annotation to the next server, S_{i+1}, in the path. S_{i+1} authenticates the annotation. It then determines if it trusts every previous server in the path (1) to forward the agent properly and (2) to authenticate its immediate predecessor in the path properly. In general, a server may not wish to incur the risk of trusting the authentication procedures of all its predecessors. In the second technique, a server, S_i, that forwards an agent to another server, S_{i+1}, signs an annotation for the forwarding hop $S_i \rightarrow S_{i+1}$. S_i supplies the list of signed annotations for each hop in the agent's path to S_{i+1}. S_{i+1} authenticates the path of the agent by authenticating each signed hop annotation. As the path grows, the cost of authenticating the digital signatures becomes prohibitive, but each server is relieved of the need to trust the authentication procedures of its predecessors. Similar path analysis issues arise in the problem of policy routing in computer networks [Breslau and Estrin 1991]. Policy routing differs, because it is possible to specify the route of a message in advance and the intermediate servers do not execute the message.

Simplifying cases, in addition to two-hop boomerang agents, may exist for multi-hop agents. If there are practical situations where the execution servers can be completely trusted, the security analysis reduces to establishing trust for the agent. The appeal of this scenario has motivated much of the safe language work and the proxy server idea suggested by [Berghoff et. al, 1996] and [Lingnau and Drobnik, 1995]. In situations where the agent can be completely trusted, it may not matter where the agent has been. This occurs if the agent does not acquire state as it travels but delivers a self-contained service to each server it visits. The agent propagates through the network more efficiently by hopping from server to server, or by replicating itself at each server visited. The agent can carry a digital signature to indicate that it is being delivered unaltered to the server. There may even be beneficial environments where neither the agent nor the server need to be trusted, i.e. beneficial trustless environments. For example, traveling games and other forms of entertainment can be executed from volatile memory on servers with user interfaces, e.g. TV monitors, but no other resources or assets. Misbehaving agents are simply erased from memory. Services provide agents for free, or require an inexpensive monthly fee and simple authentication of the user's subscription to the service. Users cancel the service if agents are generally unsatisfactory. Detailed work on these special cases will illuminate the general problem of multi-hop agents.

Summary

The central security concern for systems where agents roam is how to establish trust, and how to limit risk for both the server and the agent. The travels of the agent are key to establishing trust, and trust determines the reasonable level of risk. Security for agents with constrained travel itineraries is simpler to manage than security for general multi-hop agents. One-hop agents and two-hop boomerang agents engage a limited number of players during their travels and the players are known in advance, making it more straightforward to establish trust between the parties. Techniques for managing the risk in these cases exist and are still evolving.

For agents to roam freely, it must be possible for the agent and the execution servers to analyze the trustworthiness of servers along the agent's path. Agents can do their analysis by looking forward and evaluating new servers for inclusion in their path. Standardized policy specifications will help to automate the roles of servers in enforcing trust and risk policies for agents. Execution servers must look backwards to evaluate the history of the agent's travels before determining their levels of trust and risk. Algorithms for evaluating that history are an exciting area for future work.

References

Jürgen Berghoff, Oswald Drobnik, Anselm Lingnau, and Christian Mönch. Agent-based Configuration Management of Distributed Applications. *First IEEE International Conference on Configurable Distributed Systems*, Annapolis, 1996.

Nathaniel Borenstein. Email With A Mind of Its Own: The Safe-TCL Language for Enabled Mail. *IFIP WG 6.5 Conference on Upper Layer Protocols, Architectures, and Applications (ULPAA)*, Barcelona, 1994.

Lee Breslau and Deborah Estrin. Design and Evaluation of Inter-domain Policy Routing Protocols. *Internetworking: Research and Experience*, Vol. 2, December 1991, pp. 177-198.

David Chess. Security Considerations in Agent-Based Systems. *First IEEE Conference on Emerging Technologies and Applications in Communications (etaCOM)*, May 1996.

David Chess, Benjamin Grosof, Colin Harrison, David Levine, Colin Parris, and Gene Tsudik. Itinerant Agents for Mobile Computing. *IEEE Personal Communications Magazine*, October 1995, pp. 34-49.

Drew Dean and Dan Wallach. Security Flaws in the HotJava Browser. Computer Science Technical Report 501-95, Princeton University, November 1995.

Anselm Lingnau and Oswald Drobnik. An Infrastructure for Mobile Agents: Requirements and Architecture. *Thirteenth Distributed Information Systems Workshop*, Orlando, September 1995.

James Gosling and Henry McGilton. The Java Language Environment: A White Paper. Sun Microsystems, 1994.

Joseph Tardo and Luis Valente. Mobile Agent Security and Telescript. *Forty-First IEEE Computer Society Conference (COMPCON)*, 1996.