

An Infrastructure for Secure Interoperability of Agents

(Position Statement)

Ramesh Bharadwaj, Judith Froscher, Amit Khashnobish, James Tracy

Center for High Assurance Computer Systems

Naval Research Laboratory

Washington DC 20375

{ramesh, froscher, amith, tracy}@itd.nrl.navy.mil

Introduction

Building distributed applications is difficult. Therefore, it is hardly surprising that in spite of all the hoopla surrounding the Internet and distributed computing, truly distributed applications are few and far between. The problem seems to be with the tools available to developers of distributed applications. For example, the most widely used mechanism for distributed computation is the remote procedure call (RPC), the first implementation of which dates back to the early '80s. Typically, a remote procedure call is executed on a server on behalf of a client (the so-called "client-server" model). It is hardly surprising therefore that most distributed applications today are exclusively based on the client-server architecture. A lot can be (and has been) accomplished with this architecture, as exemplified by the World Wide Web and HTTP, a protocol that implements RPC. However, the client-server model has a number of limitations. There are problems of fault tolerance, load balancing, survivability, dynamic reconfiguration, rollover recovery, and distribution of control. Attempts in the past to break through this bottleneck have had only limited success.

More recently, there has been an emerging body of work in the area broadly known as Peer-to-Peer (P2P) distributed application frameworks. Many major organizations, both in industry and academia, have been jumping on the P2P bandwagon. However, as with other emerging

technologies, these companies and organizations are paying scant attention to security (an exception seems to be the JXTA consortium being put together by Sun Microsystems). In our opinion, rather than trying to make these systems secure as an afterthought, it would be much better if organizations think of security from the ground-up.

Why Software Agents?

It is widely acknowledged that intelligent software agents are central to the development of the capabilities required to write robust, re-configurable, and survivable distributed applications. This is because agents are an efficient, effective, and survivable means of information distribution and access. Agents are efficient because only relevant information needs to be passed along. Agents are effective because they allow local control over updates and the dissemination of data. Agents are more survivable because their control is distributed. This new technology, which includes both autonomous and mobile agents, addresses many of the challenges posed by distribution of applications and is capable of achieving the desired quality of service, most notably over unreliable, low-bandwidth communication links. However, agent technology carries with it associated security vulnerabilities. Distributed computing in general carries with it risks such as denial of service, Trojan horses, information leaks, and malicious code. Agent technology, by introducing autonomy and code mobility, may exacerbate some of these problems. In particular, a

malicious agent could do serious damage to an unprotected host, and malicious hosts could damage agents or corrupt their data. Such threats become very real in a distributed computing environment, in which a malicious intruder may be actively trying to disrupt communications.

The goal of the Secure Agents Middleware (SAM) project is to provide the required degree of trust in addition to meeting a set of achievable security requirements. Such an infrastructure is central to the successful deployment and transfer of agent technology to industry because security is a necessary prerequisite for distributed computing. To make agent-based systems economically viable, it is imperative that their development, upgrade, integration, testing, certification, and delivery be rapid and cost-effective. However, immense and profound challenges of software trustworthiness remain. Commercially available methods and tools for software development are not sufficient to meet the challenges posed by the distribution of processing functions, real-time and non-real-time integration, multi-level security, and issues characteristic of COTS products, such as malicious code, viruses, worms, and Trojan horses.

Technical Approach

The Secure Agents Middleware (SAM) and its associated Agent Creation Environment (ACE) are explicitly designed to solve the security problems described above and other related problems of agent creation and deployment. Although security is our primary concern, we also address problems of efficiency, robustness, and usability. To support usability, ACE provides agent templates and other visual aids to ease the agent creation process.

The following are highlights of the functionality provided by SAM/ACE:

- SAM provides role-based access control and management in addition to trust management.
- SAM includes functions for intrusion detection and tolerance.
- SAM is designed for *survivability* and supports Multi-Level Secure (MLS) access and authentication.
- ACE uses SADL (Secure Agent Description Language), a flexible and powerful notation in which to express the rules (i.e., the logic) of agents.
- The notation SADL and its associated user-friendly agent creation templates include a notation for specifying security and safety properties.

We plan to develop an open source compliance checker (CC) which will prove compliance of agents with policies and goals. By ensuring that security properties are satisfied and that an agent behaves as specified, we address the issue of agent **integrity**. The architecture of SAM *improves* efficiency because the flow of information between hosts is optimized. This is because our representation of information is finer grained than current architectures based on distributed objects, where information granularity is at the object level. We gain efficiency and better utilize bandwidth by a controlled exchange of information between networked hosts. Also, because our agents are *composable* and *modular*, ACE can evaluate **emergent behavior** of agent communities, which is generally not possible in the absence of a component aggregation framework. This capability enables early detection and prevention of an organized, cooperative attack on a distributed computing environment in which each agent performs some action that falls beneath the threshold of most analysis techniques, but effects serious damage as a distributed attack. Currently these types of vulnerabilities have defied analysis.

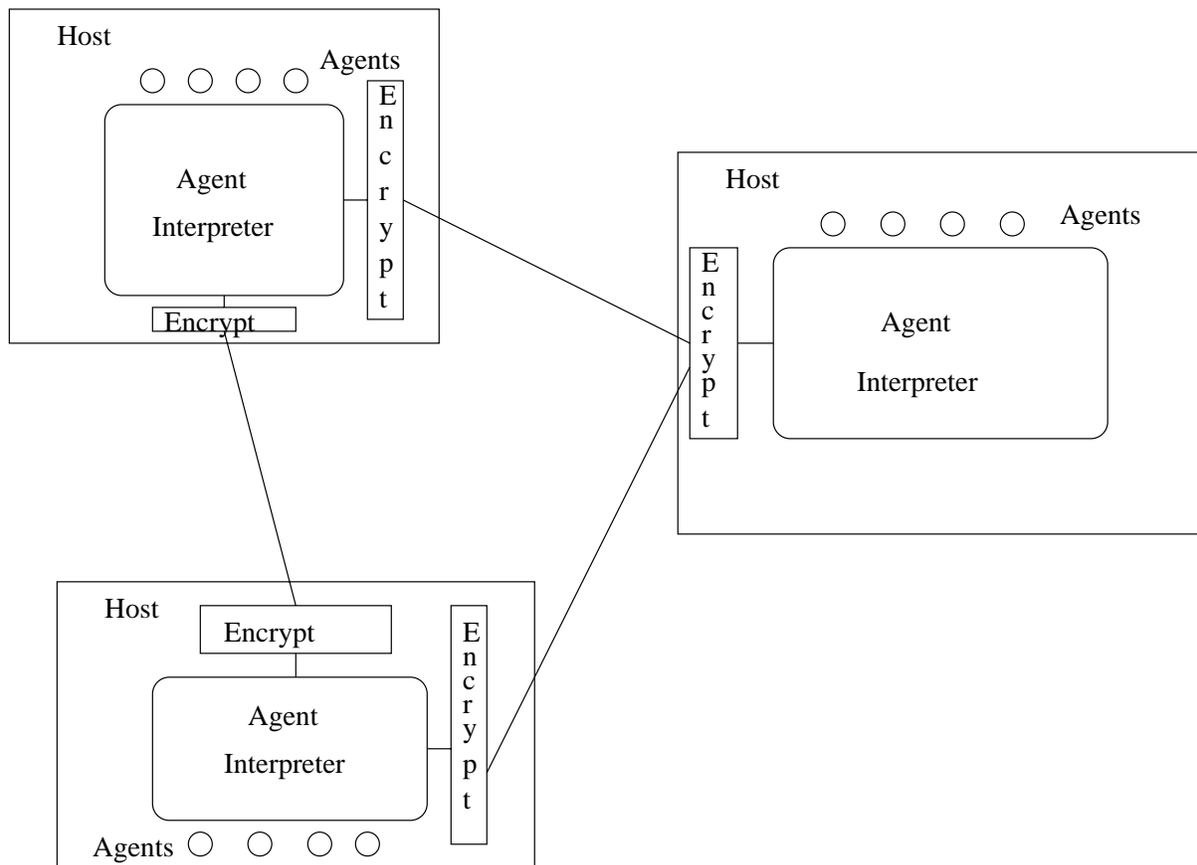


Figure 1

Figure 1 shows the architecture of SAM. Agents are distributed over one or more *Hosts*, each of which runs one or more *Agent Interpreters* (AIs), that execute agents in compliance with a set of *Security Policies*. Agents are created using special-purpose templates in ACE (not shown), and are translated into SADL. Agents may be created on any host. Agent Interpreters communicate among themselves using a lightweight protocol such as XML/SOAP, over secure channels, with strong encryption using a public key infrastructure (PKI). SOAP is particularly appealing because it can support both HTTP as well as SMTP protocols for transporting XML data and metadata. Also, because SOAP is a

lightweight protocol, its overhead is not as high as the overhead of other inter-object protocols such as CORBA IIOP. Hosts will initially run a COTS operating system such as Solaris or Windows XP, but will eventually transition to a trusted operating system such as secure Linux (a product of NSA) or secure Solaris, or alternately use NAI's DTE (Domain Type Enforcement). We will also investigate the use of other secure COTS components such as the secure Java Virtual Machine and other secure interpreters, as well as secure protocols for using the public key infrastructure to distribute keys among interpreters and for authentication of agents.

Requirements for Secure Mobile Agents

Security is a fundamental concern of SAM. By building security from the ground-up into SAM, we gain efficiency by identifying and dealing with potential bottlenecks early, i.e., at the design stage. SAM provides an efficient architecture **and** ensures security by eliminating unnecessary and/or insecure communication among agents and interpreters. Our classification of requirements for secure mobile agents is from "Security for Mobile Agents: Issues and Requirements," by William N. Farmer, Joshua D. Guttman, and Vipin Swarup, of The MITRE Corporation, Bedford, MA. The NRL SAM project addresses the following security requirements:

- The author and sender of an agent must be authenticated. In SAM, code distribution is distinct from agent mobility. Consequently, the issue of code tampering by possibly compromised hosts is addressed. This is in contrast to other mobile-agent based systems, such as Dartmouth's D'Agents, which do not make this distinction. In D'Agents, both the code as well as the data move *together* between hosts. Moreover, this movement is over an unsecure channel and without certificates or signatures. Therefore, a compromised host has the ability to tamper with the agent without being detected.
- The correctness of an agent's code must be checked.
- Interpreters must ensure that agent privacy is maintained during transmission.
- Authentication and authorization: Interpreters must protect themselves against malicious agents by first authenticating the agent and checking that its proposed activities are authorized.

- Agents must be created in a language that supports the development of safe programs. We use SADL, a language that can ensure agent safety. All analyzed and verified SADL programs are guaranteed to have no unbounded loops, violations of array index bounds, etc. This will make attacks such as Denial of Service (DOS) and malicious code propagation much harder in the SAM environment.
 - A sender must have control over an agent's flexibility; e.g., restrict or increase an agent's authorization in particular situations.
 - An interpreter must ensure that an agent is in a safe state. Because a migrating agent can become malicious, each agent must be equipped with an appropriate state appraisal function to be used each time an interpreter starts an agent. This will ensure that an agent will perform as required and has not been tampered with in a malicious way. Agent creators will be provided with appropriate static analysis tools that will ensure that the state appraisal function satisfies key safety and security properties.
 - A sender must have control over which interpreters have the authority to execute an agent.
- Currently, protecting agents from malicious hosts is an area of ongoing research. Therefore, in our initial implementation, we shall assume a degree of trust among the hosts. This is reasonable in a large organization such as the Department of Defense where it may be assumed that other policing methods and techniques for intrusion detection and tolerance will identify and sift out casual intruders and eavesdroppers. We plan to address the more general problem of agent protection in our future research.

SAM Architecture

Figure 2 shows the architecture of SAM. One of the unique features of this architecture is that we harness the power and flexibility afforded by agent technology to our advantage, thereby ameliorating the associated security and safety vulnerabilities. We accomplish this by introducing a special class of agents called *security agents* to police other classes of agents (called *secure agents*) such as application agents developed to support a distributed SIGINT system. Security agents protect a system against Information Operations (IO) attacks by implementing key security features

such as encryption, authorization, policy enforcement, virus checking, survivability, and intrusion detection. Since security agents have more privileges than other agents, we need higher assurance during development and deployment to ensure the safe and secure behavior of security agents. As outlined previously, we achieve this with a three-pronged approach: (1) We specify agents in SADL - a language for high assurance. (2) We use the compliance checker to establish formally the compliance of agent behavior with the local security policies. (3) We implement a security architecture for monitoring and coordinating agents' activities.

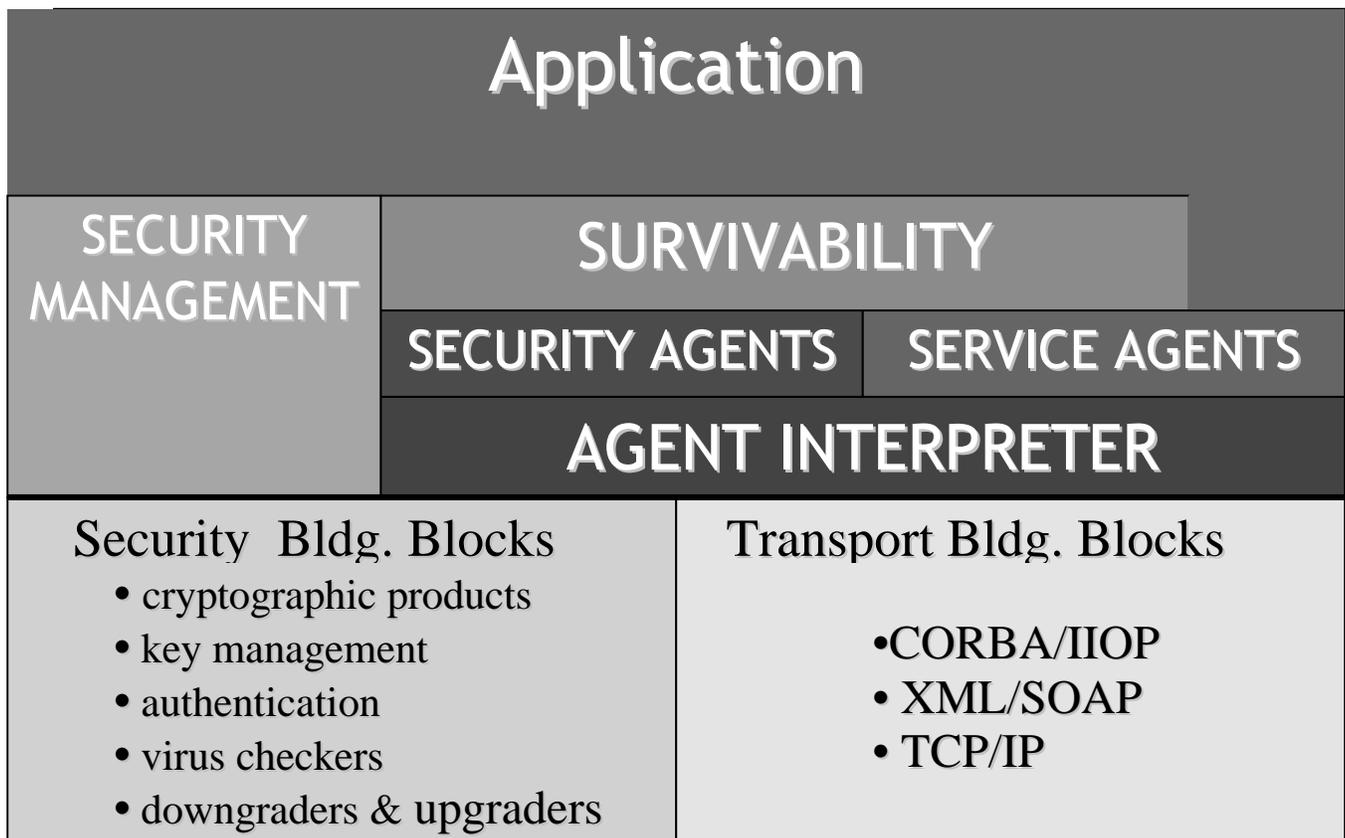


Figure 2

In the initial phase of this project we shall assume the following:

- All agent interpreters will run agents correctly
- Hosts will run all agents to completion
- Hosts will transfer agents as requested
- An agent's code and data cannot be kept private and will be readable by all agent interpreters
- Agents do not carry secret keys
- Agent-to-agent communication cannot be kept private from agent interpreters.

We will address these important technical issues in later phases of the project.

Project Goals

In this project, we address the following technical issues:

- Ensuring consistency of agent behavior
- Design and implementation of SADL:
 - Making SADL specifications composable, consistent, safe, and secure.
 - Proving application properties of SADL specifications.
- Responsibilities of Security Agents:
 - Authorization agents
 - Crypto assist agents
 - Policy enforcement agents
 - Secure agents monitoring
 - Raising exceptions
 - Establishing trust in these privileged agents
- Application-specific security agents:
 - Intrusion detection
 - Application monitoring
 - Survivability (adaptability)
 - Infrastructure monitoring
- Development of a "common I/O Picture" for secure agents:
 - Making sure security agents enforce a consistent security policy

- Secure, safe mobility of agent code.

Operational Payoff

The goal of the NRL secure agents project is to develop enabling technology that will provide the necessary security infrastructure to deploy and protect time and mission-critical applications on a distributed computing platform. Our intention is to create a *robust* and *survivable* information grid that will be capable of resisting threats and surviving attacks. One of the criteria on which this technology will be judged is that critical information is conveyed to principals in a manner that is secure, safe, timely, and reliable. No malicious agencies or other threats should be able to compromise the integrity or timeliness of delivery of this information.

Acknowledgements

This project was funded by the Office of Naval Research. The authors wish to thank Connie Heitmeyer, Cathy Meadows, and John McLean for many useful discussions pertaining to the NRL Secure Agents project, and Eric Tressler for his very useful comments on previous drafts of this manuscript. The authors also thank Connie Heitmeyer for using her PowerPoint skills in composing the architectural drawing of Figure 2.