SAgent: A Security Framework for JADE*

Vandana Gunupudi
Dept of Computer Science and Engineering
University of North Texas
gunupudi@cs.unt.edu

Stephen R. Tate
Dept of Computer Science and Engineering
University of North Texas
srt@cs.unt.edu

ABSTRACT

This paper presents SAgent, a general-purpose mobile agent security framework that is designed to protect the computations of mobile agent applications in potentially hostile environments. SAgent works with the JADE (Java Agent DEvelopment) platform [6], a FIPA-compliant multi-agent environment. SAgent supports modular and mostly orthogonal development of agent protection techniques and secure agent applications, so protocols and applications can be developed independently of each other. To accomplish this, a clean conceptual framework is presented which encapsulates in several general class interfaces the common security functionality required by secure agent applications. Furthermore, implementations are provided for two secure multi-agent protocols, and we give experimental results showing the feasibility of these protections. While a few other research projects have examined protocols and techniques for protecting agents, these have been theoretical explorations. SAgent's goal is to bring these theoretical techniques into practice so that they can be experimented with and used, in the framework of a design generic enough to support both software-based and hardware-based protections. The abstractions are clean, giving a well-defined way for a new security provider to implement and experiment with new techniques for protecting mobile agents.

Categories and Subject Descriptors

K.6.5 [**Management of Computing and Information Systems**]: Security and Protection

Keywords

Privacy, Safety and Security in agent systems; Agent-mediated electronic commerce and trading agents; Frameworks, infrastructures and environments for agent systems

1. INTRODUCTION

SAgent is a general-purpose agent security framework for the JADE platform that is designed to protect the computations of mobile agent applications in potentially hostile environments. The JADE (Java Agent DEvelopment) platform [6] is a popular mobile agent platform that enables the development and deployment of multi-agent applications using Java, and conforms to FIPA standards for software agents. SAgent supports the development of secure mobile agent applications, where data is protected from rogue or compromised hosts. While other researchers have developed frameworks that provide solutions for various agent security problems, such as authentication of agents, agent replication, and voting [3], SAgent is concerned specifically with protecting the data that travels with a mobile agent. We believe that SAgent is unique in bringing into practice a software framework for protecting agents from malicious hosts with rigorous security requirements. Much of the previous work deals with securing communication or infrastructure, or providing agent authentication. Strong methods for protecting agent data have been proposed by researchers, including hardware-based techniques [9] and techniques based on clever use of cryptography and distributed protocols [2, 7], but SAgent is the first attempt at unifying these techniques under a common framework, and as far as we're aware provides the first experimental explorations of the proposed agent protections.

In SAgent, methods of protecting agent data are cleanly separated from applications using those protections, so both application and protection technique can be independently considered. A number of general security interfaces are defined, which encapsulate the common security functionality required by secure agent applications. "Security providers" can provide implementations using these interfaces, and SAgent is distributed with two different implementations of software-based protection techniques, one by Algesheimer *et al.* [2] and one by Tate and Xu [7]. While the support for these software-based techniques is clear, the SAgent framework is general enough to support hardware-based protections just as cleanly. While the current lack of hardware to provide such protection makes this impossible to experiment with now, SAgent was designed so that if the current proposals for "trusted platforms" [8] become widespread and useful, then simple support in the SAgent framework would be possible.

SAgent was designed keeping in mind a basic tenet of software engineering, that definition of proper abstractions is essential for software reusability. The key abstraction behind SAgent mirrors that of public/private key pairs in public key cryptography in which a private key is known only to the owner of the key, while the corresponding public key is available to everyone. Similarly, a mobile agent application has *public* functionality and information that it needs to perform computations at remote hosts and *private* information and functionality that is kept by the originator of the agent and not exposed to other entities. Note that while we refer to the information that travels as "public," this does not mean it is under-

*This research is supported in part by NSF award 0208640.

AAMAS'06 May 8–12 2006, Hakodate, Hokkaido, Japan.

standable to an outside viewer — encrypted data (ciphertext) can be public even if the corresponding plaintext remains unintelligible. The private information is necessary only for the final interpretation of the results and does not need to be available in any way during the agent's travels. To that end, our design separates the public and private functionality of the agent into distinct pieces.

There are two different views of these pieces in SAgent — one from the point of view of a programmer that develops protection techniques for SAgent and another from the point of view of a developer who writes applications for SAgent. The architecture of SAgent is designed so that the security provider and the application-developer can remain unaware of each other and develop protocols and applications independently of one another. Data transfer and usage in SAgent revolves around a generic, intermediate data format that is not secure but also not application-dependent. The application developer is responsible for providing translation routines that convert application-specific data into the intermediate format, and the security provider provides routines which convert this intermediate format into a secure but protocol-specific representation. Data can then be operated on in this secure format by going through methods in the generic public interface, which resolve to the appropriate protocol-specific methods. The contributions of this work include

- A clean and useful abstraction of data protection in mobile agents;
- A working platform that realizes these abstractions and provides an effective framework for both security providers and agent application developers; and
- Implementation of two software-only security solutions that are usable and practical for protecting small amounts of agent data.

As this is a short paper, many details are omitted from this presentation and the full version of this paper is available as a technical report [4]. The SAgent software is freely available under the LGPL license [1].

2. SECURITY MODEL

Agent applications are decomposed into protected and unprotected computations, with only the sensitive data and computation being run in the protected environment. For instance, a shopping agent may browse through a store using unprotected computations, but could use SAgent to protect the portions of the agent which make purchasing decisions. The protected portion could include very sensitive information, such as payment authorization values, credit card numbers, or electronic cash tokens. The protected agent computation is modeled as a 3-input, 2-output function, as shown in Figure 1. The three inputs are the current agent state, the input from the host, and an input provided by any unprotected computations performed by the agent on that host. The two outputs are an updated agent state and an output that is provided to the current host. Specific security providers guarantee slightly different security properties, but in general the agent state is protected so that it is unintelligible to an outside observer (either an attacker or a malicious host), and the host's input is protected so that it can only be used in a manner consistent with the agent functionality, which can be determined or negotiated as a contract between the originator and the host.

3. CORE INTERFACES OF SAGENT

SAgent uses sound principles of object-oriented design and achieves abstraction by specification of several general *interfaces*, which are extended by security providers to supply protocol-specific implementations. The design is similar to some well-designed cryptography libraries, such as Sun's Java Cryptography Architecture, in which generic "encryption" and "decryption" interfaces are defined, and these are bound to specific algorithms and implementations when the cryptography objects are allocated. The core of our design, therefore, includes secure private and public interfaces, along with a secure data interface, which we describe in this section.

3.1 SecureFnPrivate Interface

The `SecureFnPrivate` interface encapsulates the private functionality in our mobile agent model, defining the methods for interacting with the agent that are restricted to the agent originator. The originator creates an agent for a particular application using a constructor given by a particular security provider, with the result being a `SecureFnPrivate` object. Through this interface, the originator sets the initial state of the agent, and retrieves the "public part" (a `SecureFnPublic` object) which is then sent out in the mobile agents for this application. Each mobile agent then visits various remote hosts, utilizing its public functionality to perform computations on behalf of the originator. Note that the agent can perform both unprotected and protected computations at remote hosts, and good design would suggest that only the sensitive parts of the agent application be performed within the SAgent framework. The `SecureFnPrivate` interface also defines the functionality that allows the originator to decode the state of the agent after the agent returns home. After successful decryption of the agent state, the state of the agent is converted from a secure data format to a generic object, understood by the application. Thus, it can be seen that the `SecureFnPrivate` interface provides exactly those methods that allow the originator to perform operations that no remote host should be allowed to do.

3.2 SecureFnPublic Interface

The `SecureFnPublic` interface encapsulates the functionality required by the agent to perform protected computations at remote hosts. The originator initializes the agent with the public functionality it requires to perform computations at a host. Upon successful contact with the host (through the host agent), the agent supplies its public part to the host and asks the host for its input. Specifically, this step involves reading in the host's input in a generic form and converting it into an object of type `SecureFnData`, where the specifics of this transformation depend on the particular protocol which implements this interface. Only the encoded input is returned from the host to the agent, which is what enforces that the host data is used only as specified by the particular agent functionality. Similarly, if the agent has unprotected computation results that need to be included in the protected computation, then it encodes its input using the *encodeAgentInput* method. Once the agent obtains the host and agent inputs in a secure format, it calls the *evaluate* method to perform the protected computation. This method updates the agent state and returns a `SecureFnData` object representing the output to be provided to this host, which can be converted to plaintext using the *decodeHostOutput* method. This evaluation obviously depends on the specifics of the agent application, but the evaluation is protected in a uniform way by the particular security provider. The key to all of the mobile agent computations is that computations are only performed on `SecureFnData` objects, and the privacy and/or integrity of these operations is guaranteed by the security provider and the security protocol. Any protected information stored in the agent state can be operated on through the *evaluate* method, but can not be decoded or decrypted except

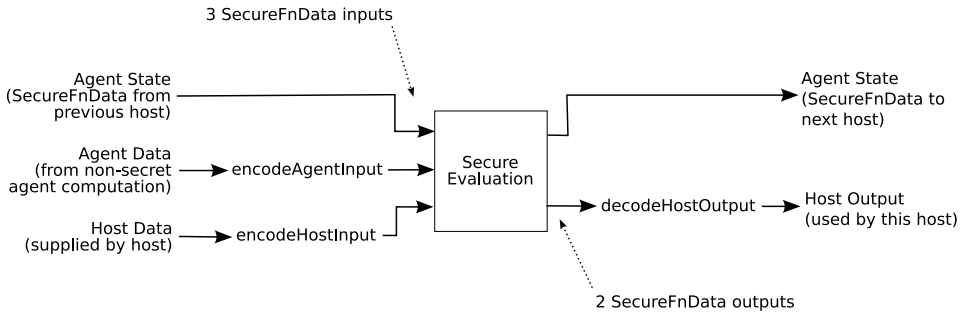

Figure 1: Agent State Update in SAgent

through the `SecureFnPrivate` object, which only the originator possesses.

3.3 SecureFnData Interface

SAgent is designed to protect the security of the mobile agent data. When data are operated upon within SAgent, the data are in a secure format. Therefore, the key idea with SAgent is that only *protected* data objects should be allowed within the protected execution. The `SecureFnData` type is simply a generic, opaque type to represent protected data, and does not provide any methods for operating on this data — all operations are performed using either `SecureFnPublic` or `SecureFnPrivate` methods, including conversion to/from insecure data representations from/to `SecureFnData` objects. Different applications have different data requirements, so SAgent defines an application-independent intermediate format that can be used within SAgent. This is just a binary representation of the data, and makes it simple for application-developers and security providers to develop their products independently of one another. The security provider just deals with bits which he encrypts, decrypts, and operates on as binary data, and is unaware of the actual form or meaning of the data in any application that may use his protocol. Once protected as a `SecureFnData` object, the actual encrypted representation of the data remains opaque to the user of the protocol. The application-developer, on the other hand, provides conversion routines between generic binary data and application-specific data.

4. EXPERIMENTS AND EFFICIENCY

In this section we present experimental results from running the applications described earlier under both protocols provided with SAgent. Results of an extensive set of experiments on the protocols themselves, showing the how complexity grows with input size and how parameters such as cryptographic algorithms and key sizes affect efficiency are available in [5]. The running times from our experiments are shown in the following table, where times are cumulative times given in seconds. The experiments were performed on a cluster of seven 2GHz Pentium IV machines with one host designated as the originator and the rest designated as remote hosts.

Application	Protocol		
	Insecure	ACCK	TX
Maxbid	0.11	0.53	12.24
BuyAgent	0.12	1.95	69.2

Note that in all cases except the more involved BuyAgent application running under the TX protocol, the times are just a few seconds per host visited, which is quite a low overhead for the rigorous security guarantees that these protocols provide.

5. CONCLUSIONS

We have developed a comprehensive security framework called SAgent for the JADE multiagent platform. Our framework allows the development of secure protocols and mobile agent applications, and can be used to provide both confidentiality and integrity protections for agent data. Use of generic, application-independent interfaces and data formats within SAgent allows programmers to create applications and protocols with ease and transparency. We also provide the first software implementations of the ACCK [2] and the TX [7] mobile agent privacy protocols, with experimental results showing the feasibility of these software-only protections within our framework. We leave as future work the inclusion of support for hardware-based protections in SAgent, but believe that such an implementation would be straight-forward in a system that provides verifiable Java environments through hardware-based attestation.

6. REFERENCES

[1] SAgent software available at:. http://www.cops.csci.unt.edu/sagent/.

[2] J. Algesheimer, C. Cachin, J. Camenisch, and G. Karjoth. Cryptographic security for mobile code. In *Proc. of the IEEE Symposium on Security and Privacy*, pages 2–11, 2001.

[3] C. Bryce. A security framework for a mobile agent system. In *Proc. of the 6th European Symposium on Research in Computer Security*, volume 1895 of *Lecture Notes In Computer Science*, pages 273 – 290, 2000.

[4] V. Gunupudi and S. Tate. SAgent: A security framework for JADE. Under review — available as CoPS Lab Technical Report 2005-01.

[5] V. Gunupudi, S. Tate, and K. Xu. Experimental evaluation of security protocols in SAgent. Under review — available as CoPS Lab Technical Report 2006-01.

[6] JADE, documentation and resources. available at: http://jade.tilab.com/.

[7] S. R. Tate and K. Xu. Mobile agent security through multi-agent cryptographic protocols. In *Proc. of the 4th International Conference on Internet Computing (IC 2003)*, pages 462–468, 2003.

[8] Trusted Computing Group. Web site. http://www.trustedcomputinggroup.org.

[9] B. Yee. A sanctuary for mobile agents. *Secure Internet Programming, Lecture Notes in Computer Science*, 1603:261–273, 1999.