Secure Agent Data Protection for E-Commerce Applications

Sheng-Uei Guan
Brunel University, UK

INTRODUCTION

One hindrance to the widespread adoption of mobile agent technology (Johansen et al., 2002) is the lack of security. SAFER, or Secure Agent Fabrication, Evolution, and Roaming, is a mobile agent framework that is specially designed for the purpose of electronic commerce (Zhu, Guan, Yang, & Ko, 2000; Guan & Yang, 1999, 2003; Yang & Guan, 2000). By building strong and efficient security mechanisms, SAFER aims to provide a trustworthy framework for mobile agents. While such an agent transport protocol provides for the secure roaming of agents, there are other areas related to security to be addressed.

Agent integrity is one such area crucial to the success of agent technology. The integrity protection for agent code is relatively straightforward. A more complex code integrity scheme to handle code-on-demand is also proposed in Wang et al. (2002). Agent data, however, is dynamic in nature and will change as the agent roams from host to host. Despite the various attempts in the literature (Chionh, Guan, & Yang, 2001), there is no satisfactory solution to the problem so far. Some of the common weaknesses of the current schemes are vulnerabilities to revisit attack and illegal modification (deletion/insertion) of agent data.

DESCRIPTION OF SADIS

SADIS has been designed based on the following assumptions:

1. Entities including agents, agent butlers, and hosts should have globally unique identification number (IDs).
2. Each agent butler and host should have a digital certificate that is issued by a trusted CA. These entities will be able to use the private key of its certificate to perform digital signatures and encryption.
3. While the host may be malicious, the execution environment of mobile agents should be secure and the execution integrity of the agent can be maintained.
4. Entities involved are respecting and cooperating with the SADIS protocol.

Key Seed Negotiation Protocol

The proposed key seed negotiation protocol defines the process for key seed negotiation, as well as session key and data encryption key derivation. When an agent first leaves the butler, the butler generates a random initial key seed, encrypts it with the destination host's public key, and deposits it into the agent before sending the agent to the destination host. It should be noted that agent transmission is protected by the agent transport protocol (Guan & Yang, 2002), thereby protecting the system from being compromised by malicious hosts.

The key seed negotiation process is based on the Diffie-Hellman (DH) key exchange protocol (Schneier, 1996) with a variation. The agent will first generate a private DH parameter *a* and its corresponding public parameter *x*. The value *x*, together with the ID of the destination host, will be encrypted using a communication session key and sent to the agent butler.

The agent butler will decrypt the message using the same communication session key (to be discussed later). It too will generate its own DH private parameter *b* and its corresponding public parameter *y*. With the private parameter *b* and the public parameter *x* from the agent, the butler can derive the new key seed and use it for communications with the agent in the new host. Instead of sending the public parameter *y* to the agent as in normal DH key exchange, the agent butler will encrypt the value *y*, host ID, agent ID, and current timestamp with the destination host's public key to get message *M*. Message *M* will be sent to the agent after encrypting with the communication session key.

$$M = E(y + \text{host ID} + \text{agent ID} + \text{timestamp}, H_{pubKey})$$

At the same time, the agent butler updates the agent's itinerary and stores the information locally. This effectively protects the agent's actual itinerary against any hacking attempts related to itinerary, thereby protecting against the data deletion attack.

When the agent receives the double-encrypted DH public parameter *y*, it can decrypt with the communication session key. Since the decrypted result *M* is parameter *y* and some other information encrypted with the destination host's public key, the current host will not be able to find out the value of *y* and thus find out the new key seed to be used when the

agent reaches the destination host. It should be noted that this does not prevent the host from replacing *M* with its own version *M'* with the same host ID, agent ID, and timestamp, but different *y*. The inclusion of host ID, agent ID inside *M* can render such attack useless against SADIS. A detailed discussion on this attack can be found in the security analysis section of this article.

Subsequently, the agent will store *M* into its data segment and requests the current host to send itself to the destination host using the agent transport protocol (Guan & Yang, 2002).

Upon arriving at the destination host, the agent will be activated. Before it resumes normal operation, the agent will request the new host to decrypt message *M*. If the host is the right destination host, it will be able to use the private key to decrypt message *M* and thus obtain the DH public parameter *y*. As a result, the decryption of message *M* not only completes the key seed negotiation process, but also serves as a means to authenticate the destination host. Once the message *M* is decrypted, the host will verify that the agent ID in the decrypted message matches the incoming agent, and the host ID in the decrypted message matches that of the current host. In this way, the host can ensure that it is decrypting for a legitimate agent instead of some bogus agent. If the IDs in the decrypted messages match, the decrypted value of *y* is returned to the agent.

With the plain value of *y*, the agent can derive the key seed by using its previously generated private parameter *a*. With the new key seed derived, the key seed negotiation process is completed. The agent can resume normal operation in the new host.

Whenever the agent or the butler needs to communicate with each other, the sender will first derive a communication session key using the key seed and use this communication session key to encrypt the message. The receiver can make use of the same formula to derive the communication session key from the same key seed to decrypt the message.

The communication session key K_{CSK} is derived using the formula below:

$$K_{CSK} = \text{Hash}(\text{key\_seed} + \text{host ID} + \text{seqNo})$$

The sequence number is a running number that starts with 1 for each agent roaming session, and is reset to 1 whenever the agent reaches a new host. Each message communicated will therefore be encrypted using a different key. As this means that the butler and agent will not be able to communicate if messages are lost without detection, SADIS makes use of TCP/IP as a communication mechanism. Once the communication is re-established after a send failure, the sender will resend the previous message (encrypted using the same communication session key). The agent and the butler can therefore synchronize on communication session key calculations.

The agent encrypts host information with a data encryption key K_{DEK}. The data encryption key is derived as follows:

$$K_{DEK} = \text{Hash}(\text{key\_seed} + \text{hostID})$$

The details on encryption will be discussed in the next section.

Data Integrity Protection Protocol

The key seed negotiation protocol lays the necessary foundation for integrity protection by establishing a session-based key seed between the agent and its butler. Digital certificates also help protect the agent data integrity.

Our data integrity protection protocol comprises two parts: chained signature generation and data integrity verification. Chained signature generation is performed before the agent leaves the current host. The agent gathers data provided by the current host d_i and construct D_i as follows:

$$D_i = E(d_i + ID_{host} + ID_{agent} + \text{timestamp}, k_{DEK})$$

or

$$D_i = d_i + ID_{host} + ID_{agent} + \text{timestamp}$$

The inclusion of host ID, agent ID, and timestamp is to protect the data from possible replay attack, especially when the information is not encrypted with the data encryption key, thereby creating an unambiguous memorandum between the agent and the host. The construction of D_i also gives the flexibility to encrypt the data or keep it plain. After constructing D_i, the agent will request the host to perform a signature on the following:

$$c_i = \text{Sig}(D_i + c_{i-1} + ID_{host} + ID_{agent} + \text{timestamp}, k_{priv})$$

where c_0 is the digital signature on the agent code by its butler.

One design focus of SADIS is not only to detect data integrity compromise, but more importantly to identify malicious hosts. To achieve malicious host identification, it is an obligation for all hosts to verify the incoming agent's data integrity before activating the agent for execution. In the event of data integrity verification failure, the previous host will be identified as the malicious host.

Data integrity verification includes the verification of all the previous signatures. The verification of signature c_0 ensures agent code integrity; the verification of c_i ensures data provided by host h_i is intact. If any signature failed the verification, the agent is considered compromised.

While the process to verify all data integrity may seem to incur too much overhead and also seem somewhat redun-