AN APPROACH TO PROTECT MOBILE AGENT PRIVACY

Suat Uğurlu, Nadia Erdoğan
Istanbul Technical University, Computer Engineering Department,
Ayazaga, 34390 Istanbul, Turkey

suat@suatugurlu.com, erdogan@cs.itu.edu.tr

Özet. *Hareketli etmenlerin yakın zamanda bir çok araştırmaya konu olduğunu görmekteyiz. Önemli derecede ilgi görmesine rağmen, hareketli etmenlerin büyük ölçekli mimarilerde kullanılmaya başlanması, güvenlik mekanizmalarının yeterli derecede anlaşılıp uygulanması tamamlanmadan mümkün olmayacaktır. Hareketli etmenler yaşam süreleri boyunca bir çok risk altındadır. Etmene ait kod okunabilir, gizli verisi çalınablir, taşıdığı mesajlar değiştirilebilir. Bu bildiride güvenli bir hareketli etmen mimarisi olan SECMAP'ın hareketli etmenleri çevresinden korumak için kullandığı yöntemler incelenecektir.*

Abstract. *Mobile agents have gained a great deal of attention in research and industry in the recent past. Although mobile agents are a promising technology, the large-scale deployment of agents and the existence of hosts running agencies will not happen until proper security mechanisms are well understood and implemented. Mobile agents are plain enough for malicious parties to read and to analyze. A malicious host may read or alter the content of the agent, or analyze the accumulated information carried by the mobile agent. Another program or agent running on the same host as the agent is another source of threat for the agent. In this paper we analyze the security mechanisms of a secure mobile agent system, SECMAP, related to protecting the mobile agent code and state against malicious hosts and agents.*

1. INTRODUCTION

Mobile agents are currently a hot topic in the domain of distributed systems. Reasons are the problems more traditionally designed distributed systems, especially client/server systems, might have to handle such as work-load, the trend to open large numbers of customers, direct access to services and goods, and user mobility. Mobile agent technology can help to design innovative solutions in this domain by complementing other approaches, simply by adding mobility of code, machine based intelligence, and improved network and data-management possibilities.

A mobile agent is a program that can migrate from host to host in a network of heterogeneous computer systems and fulfill a task specified by its owner. It works autonomously and communicates with other agents and host systems. During the self-initiated migration, the agent carries its code and execution state with it.

There are mainly two security issues in mobile agent systems. The first is that the machine hosting the mobile code is under risk in case the mobile code includes dangerous code fragments for the host. The second is that the agent itself is under risk because of malicious hosts and other malicious agents. Many techniques for the first problem have been developed. The second problem is much harder than the first one and there are still unresolved security problems on the subject.

A secure framework is needed to satisfy agents' security concerns in mobile environments. Both agent code and state should be protected during its life time. In this paper we will examine the

security mechanism of SECMAP, (Secure Mobile Agent Platform) and the methods it uses to protect the agent from its environment. Unlike other agent systems, SECMAP proposes a new agent model, *the shielded agent model*, for security purposes. A shielded agent is a highly encapsulated software component that ensures complete isolation against unauthorized access of any type, especially by other agents and programs in the host memory. In addition, SECMAP uses DES algorithm to encrypt the agent code and previous state in the host disk to prevent access to the agent code while it is inactive. All forms of communication is established through SSL. Security administration is also distributed and is performed by integrated cooperation of multiple managers.

2. RELATED WORK

There are mainly two methods for the protection of an agent's code and state in hostile environments. The first one aims to block unauthorized access to the agent's code and state, while the second one tries to determine if the agent was tampered using different mechanisms. Developers and researchers have taken a variety of approaches to assure the security of mobile agent environments using one of these two methods. Hohl proposes what he refers to as Blackbox security to scramble an agent's code in such a way that no one is able to gain a complete understanding of its function. Proof carrying code requires the author of an agent to formally prove that the agent conforms to a certain security policy. By digitally signing an agent, its authenticity, origin, and integrity can be verified by the recipient. The idea behind path histories is to let a host know where a mobile agent has been executed previously. State appraisal attempts to ensure that an agent's state has not been tampered with and that the agent will not carry out any illegal actions through a state appraisal function which becomes part of the agent code. There does not seem to be a single solution to the security problems introduced and most of the solutions are inadequate in protecting agent and host data, while others that provide adequate protection but cause an unacceptable overhead to the programmer. SECMAP architecture uses shielded objects to protect the agent from other agents and programs while running in a host memory. In addition, it also encrypts the agent code and state in the host disk to provide a black box security without requiring additional coding. The security management is fully distributed.

3. SECMAP ARCHITECTURE

In a mobile agent system, agents cannot be reliably associated with end users without taking certain precautions. The approach taken by SECMAP is to treat every agent as a distinct principal and to provide protection mechanisms that isolate agents. SECMAP differs from other mobile agents systems in the abstractions it provides to address issues of agent isolation. It provides a light-weight implementation of agents; they are implemented as threads instead of processes. Each agent is an autonomous object with a unique identity. We have used Java for the implementation of the execution environment because it offers several features that ease the development process. Figure 1 shows the SECMAP architecture. The main component of the architecture is a Secure Mobile Agent Server (SMAS) that is responsible of all agent related tasks such as creation, activation, communication, and migration. The system comprises of several SMAS executing on each node which acts as a host for agents.

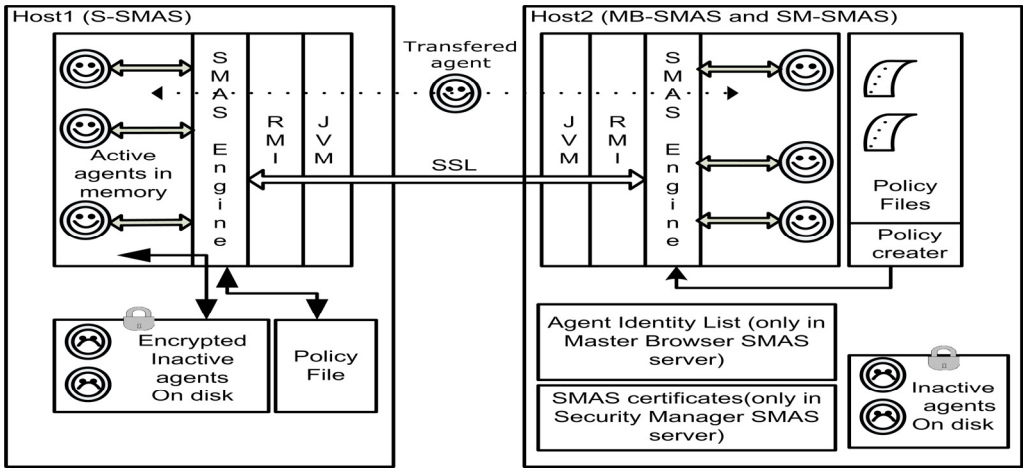

Figure 1. SECMAP architecture

A SMAS may operate in three modes according to the functionality it exhibits. It can be configured to execute in any of the three modes on a host through a user interface. A SMAS can also operate in all three modes at the same time.

Standard Mode (S-SMAS): S-SMAS provides standard agent services such as agent creation, activation, inactivation, destruction, communication, and migration. It also includes a policy engine that checks agent activity and resource utilization according to the rules that are present in a policy file, which has been received from a Security Manager SMAS. In addition, S-SMAS maintains a list of all active agents resident on the host and notifies the Master Browser SMAS anytime an agent changes state. Keeping logs of all agent activities is another important task S-SMAS carries out. Log content may be very useful in the detection of certain kinds of attacks which are difficult to catch instantly.

Master Browser Mode (MB-SMAS): When agents are mobile, location mappings change over time, therefore agent communication first requires a reference to the recipient agent to be obtained. In addition to supporting all functionalities of S-SMAS, MB-SMAS also maintains a name-location directory of all currently active agents in the system. This list consists of information that identifies the host where an agent runs and is kept up to date as information on the identities and status (active/inactive) of agents from other SMAS is received.

Security Manager Mode (SM-SMAS): In addition to supporting all functionalities of S-SMAS, SM-SMAS performs authentication of all SMAS engines, handles policy management, and maintains security information such as DES keys and certificates. Any SMAS engine in the system has to be authenticated before it can start up as a trusted server. SM-SMAS holds an IP address and key pair for each of SMAS engine that wants to be authenticated. If the supplied key and the IP address of the requesting SMAS engine is correct then it is authenticated. The authenticated SMAS engine gets a ticket from the SM-SMAS and uses this ticket when communicating with other SMAS engines. A SMAS that receives a request from another SMAS refers to SM-SMAS to verify the validity of its ticket before proceeding with the necessary actions to fulfill the request. Every SMAS, regardless of its mode, creates a private-public key pair once. Next, it creates its certificate and sends it to the SM-SMAS. Any SMAS can receive the certificate list from SM-SMAS before authentication and store it in its key store in order to use SSL communication with other SMAS engines in the system. From then on, all agent-to-agent and SMAS-to-SMAS communication is established through SSL.

3.1 SECMAP Agents

SECMAP requires agents to conform to a software architectural style, which is identified by a basic agent template. The agent programmer is provided a flexible development environment with an

interface for writing mobile agent applications. He determines agent behaviour according to the agent template given and is expected to write code that reflects the agent's behaviour for each of the public methods. For example, code for the OnCreate() method should specify initial actions to be carried out while the agent is being created, or code for the OnMessageArrive() method should define agent reaction to message arrival.

```
public class Main extends Agent
{
   public void OnMessageArrive() {... }
   public void OnCreate() { ... }
   public void OnActivate() {... }
   public void OnInactivate() {... }
   public void OnTransfer() {... }
   public void OnEnd() {... }
}
```

An instance of class *AgentIdentity* is defined for the agent on an initial creation. All agents in the system are referenced through their identities, which consist of three parts. The first part, "ID", a random string of 128 bytes length, is unique and, once assigned, never changes throughout the life time of the agent. The second part is the name which the agent has announced, and the third part is the address of the SMAS on which the agent is currently resident and may vary as the agent moves among hosts. *AgentIdentity* class is given below.

```
public final class AgentIdentity Implements serializable
{
   private String strAgentHostName=null;
   private String strVisibleAgentName=null;
   private byte [] ID =new byte[128]
}
```

3.2 Protecting the Agent in Host Memory (Shielded Agent Model)

SMAS provides functionalities that meet security requirements and allow the implementation of the shielded agent model. A shielded agent is a highly encapsulated software component that ensures complete isolation against unauthorized access of any type. On a request to create a new agent, SMAS instantiates a private object of its own, an instance of predefined object AgentShield, and uses it as a wrapper around the newly created agent by declaring the agent to be a private object of AgentShield object. This type of encapsulation ensures complete isolation, preventing other agents to access the agent state directly. An agent is only allowed to communicate with its environment over the SMAS engine through the methods defined in a predefined interface object, AgentInterface, which is made the private object of the agent during the creation process. The interface provides limited yet sufficient functions for the agent to communicate with SMAS. All variables of agents are declared as private and they have corresponding accessor methods.

SECMAP allows the concurrent execution of several agents on the same host and each agent runs as a separate thread in the same memory area of the host. In this mode of operation, the shielded agent model suffices to guarantee inter agent isolation and protection. Figure 2 depicts the layered structure of a shielded agent.

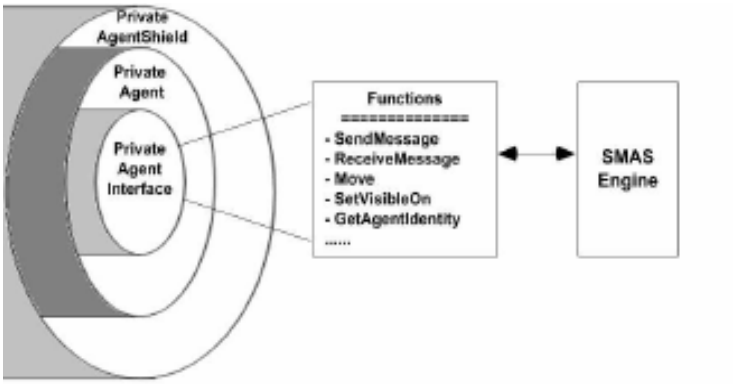

Figure 2. The shielded agent model

3.3 Protecting the Agent Code and State

As it is known, Java class files can easily be decompiled. That means a mobile agent's code written in JAVA may be clear to anyone who wants to read if no precaution is taken. Agent code in SECMAP consists of one or more class files. SECMAP holds the agent's code (its class files) and state in a single zipped file all through its life time, except when it is loaded into host memory and is running. Furthermore, this zipped file is encrypted with a DES key which is obtained from a SM-SMAS. When a local SMAS wants to activate an agent after a successful migration operation, first, it has to acquire the DES key from a SM-SMAS in order to be able to decrypt the agent's files. Thus, agent code and state are seen as a blackbox for other third parties. While an agent is running on the host memory, its code and "previous" state are kept encrypted on the host disk. When the agent becomes inactive or it issues a request to migrate, only its current state in memory is written back to the disk. In Figure 3 and Figure 4, SECMAP code fragments that encrypt and decrypt code and state files of an agent are given respectively. Java's *javax.crypto* and *javax.crypto.spec* packages are used for the necessary functions.

```
private final byte[] encrypt (byte[] plainin)
 {

   ByteArrayInputStream is=null;
   ByteArrayOutputStream os=new ByteArrayOutputStream();
   byte key[] = Key.getBytes();  // obtain the key from a MB-SMAS
   try  {
     CipherInputStream cis;
     SecretKeySpec secretKey = new SecretKeySpec(key,"DES");
     Cipher encrypt = Cipher.getInstance("DES/ECB/PKCS5Padding");
     encrypt.init(Cipher.ENCRYPT_MODE, secretKey);
  // input data is opened
  try {
     is = new ByteArrayInputStream(plainin);
  } catch(Exception err) {  System.out.println(err.toString()); }
  cis = new CipherInputStream(is, encrypt);
  // write the encrypted data
  os = new ByteArrayOutputStream();
  byte[] b = new byte[8];
  int i = cis.read(b);
  while (i != -1) { os.write(b, 0, i); i = cis.read(b); }
  cis.close();  is.close();
  } catch(Exception e){ e.printStackTrace();
  }
  return os.toByteArray();
 }
```

Figure 3. Encryption of code and state files of an agent

```
private final byte[] decrypt (byte[] encryptedin) {
   byte key[] = Key.getBytes();
   ByteArrayInputStream  is = new ByteArrayInputStream(encryptedin);
   ByteArrayOutputStream os=new ByteArrayOutputStream();
 try{
   SecretKeySpec secretKey = new SecretKeySpec(key,"DES");
   Cipher decrypt =Cipher.getInstance("DES/ECB/PKCS5Padding");
   decrypt.init(Cipher.DECRYPT_MODE, secretKey);
   CipherInputStream cis = new CipherInputStream(is, decrypt);
   byte[] b = new byte[8];
   int i = cis.read(b);
   while (i != -1) {os.write(b, 0, i); i = cis.read(b); }
     }
 catch(Exception ex){ex.printStackTrace();}
 return os.toByteArray();
 }
```

Figure 4. Decryption of code and state files of an agent

3.4. Agent Class Loader

Creation of an agent on a host, either initially or after a migration request, involves a class loader to locate and load any classes necessary for the agent to be active. Java virtual machine includes a class loader, called the primordial class loader, embedded in the virtual machine. This embedded loader is special because the virtual machine assumes that it has access to a repository of trusted classes which can be run by the VM without verification. The primordial class loader implements the default implementation of `loadClass()`. Thus, it assumes that the class name `java.lang.Object` is stored in a file with the prefix `java/lang/Object.class` somewhere in the class path. As agent code and state is held in a file that is both zipped and encrypted, SECMAP uses a customized class loader rather than the system class loader. A new class loader, `AgentClassLoader`, is defined and it's `loadClass()` method is modified to satisfy SECMAP requirements. The custom agent class loader follows the steps listed below when activated:

- Verify class name
- Check if class has already been loaded into the memory before
- Check if class is a system class
- Define class for JVM
- Specify other referenced classes, if any
- Return class to the caller

The definition of `AgentClassLoader` class is stated below and its method `loadAgentClass` that carries out the actions to load a zipped and encrypted class file is given in Figure 5.

```
final class AgentClassLoader  extends ClassLoader
{ ... }
```

```
private final Class  loadAgentClass(String strClassName) throws
ClassNotFoundException {
    Class c = null;
//return class if already loaded
    if ( (c = (Class) hashtableClasses.get(strClassName)) != null) {
      return c;
    }
//Read agent code and state  from the zipped and encrypted file.
    String strFileName = transform(strClassName);
    ZipEntry zipentry = zipfile.getEntry(strFileName);
    if (zipentry == null) {
      throw new ClassNotFoundException(strClassName);
    }
    byte[] rgb = null;
    try {
      int n = (int) zipentry.getSize();
      rgb = new byte[n];
      InputStream inputstream = zipfile.getInputStream(zipentry);
      int m = 0;
      while (m < n) {
      m += inputstream.read(rgb, m, n - m);
      }
    }
     catch (IOException ex) {
      throw new ClassNotFoundException(strClassName);
    }
 //Decrypt encrypted data
    byte[] rgb1 = decrypt(rgb);
//Define class
    c = defineClass(strClassName, rgb1, 0, rgb1.length);
    hashtableClasses.put(strClassName, c);
//Return class to caller
    return c;
  }
```

Figure 5. Method loadAgentClass of SECMAP class loader

3.5. Protecting Agent Messages

Mobile agents generally need to coordinate their activities and do so by passing messages between them. SECMAP allows for inter-agent communication in a location transparent way. Agents need only know the identifier of the agent with which they wish to communicate. For reasons of security, A SECMAP agent is only allowed to communicate with its environment over the SMAS engine through the methods defined in a predefined interface object, *AgentInterface*, which is made the private object of the agent during the creation process. The interface provides limited yet sufficient functions for the agent to communicate with SMAS.

Agents in SECMAP are provided with a flexible communication environment where they can question the results of message send requests, wait for responses for a specified period of time, and receive messages or replies whenever it is convenient for them. The details of the messaging subsystem are not covered here since it is not the subject of this paper. The SECMAP approach in order to provide secure message exchange is to use SSL to send and receive message packets. In fact, all agent-to-agent and SMAS-to-SMAS communication is handled by SSL over RMI. Thus, transferred message packets can not be interpreted by any other party or a program sniffing the network.

4. CONCLUSIONS AND FUTURE WORK

This paper describes the methods that SECMAP uses to protect agent code and state, as well as agent messages in mobile environments. The proposed Shielded Agent Model guaranties protection of the agent against unauthorized access by other agents and programs in the host memory. In addition, SECMAP uses DES algorithm to encrypt the agent code and previous state in the host disk to prevent access to the agent code while it is inactive. Thus, the agent is a blackbox for any party other than the SMAS, which also can not read the agent code if it can not get the correct key from the SM-SMAS. All agent-to-agent and SMAS-to-SMS communication is established through SSL. Security administration is also distributed and is performed by multiple SM-SMAS integration.

As the system and agents are written in JAVA, any security hole in Java causes a hole in our architecture as well. However, we believe that Java is getting more secure. The only security weakness may be a fake SMAS execution environment in the system. As future work, we plan to improve the system with a method to detect fake SMAS engines. SECMAP architecture also keeps logs of all agent activities such as activation, inactivation, migration and messaging. By analyzing these traces, it may be possible to find out fake SMAS servers in the system.

References

(Hohl, 1997) F. Hohl. "Protecting mobile agents with blackbox security" Proc. 1997 Wksp. Mobile Agents and Security , Univ. of Maryland , Oct 1997

(Necula,Lee,1998) G. C. Necula and P. Lee. "Safe, untrusted agents using proofcarrying Code" In Giovanni Vigna, editor, Mobile Agents and Security, Number 1419 in LNCS, pages 61-91. Springer-Verlag, Berlin, 1998.

(Farmer,Guttmann, Swarup,1996) W. Farmer, J. Guttmann, and V. Swarup, "Security for mobile agents: Authentication and state appraisal", In E. Bertino, H. Kurth, G. Martella, and E. Montolivo, editors, Proc. of ESORICS 96, Number 1146 in LNCS, pages 118-130. Springer-Verlag, Berlin, 1996

(Varadharan,Foster,2003) V.Varadharan and D.Foster, "A Security Architecture for Mobile Agent Based Applications" World Wide Web:Internet and Web Information System, 6,93-122, 2003

(Wilhelm,Staaman,1998) Uwe G. Wilhelm and Sebastian Staaman "Protecting the itinerary of Mobile Agents" Laboratoire de Systemes d'Exploitation, Switzerland, 1998

(Karnik,N.M., Tripathi, A.R. , 1998) Karnik, N.M., Tripathi, A.R., Design issues in mobile-agent programming systems. IEEE Concurrency 6 (3), 52–61 , 1998

Autobiography

Suat Uğurlu received BS and MSc degrees in Control and Computer Engineering Department of Istanbul Technical University in Istanbul, Turkey in 1997 and 2001 respectively. He is still a doctoral student in Computer Engineering Department of Istanbul Technical University. His current research areas include distributed computing and execution environments, mobile agent systems, computer networks and security.

Nadia Erdoğan received BS degree in Electrical Engineering and MSc degree in Computer Science Departments of Bosphorus University in Istanbul, Turkey in 1978 and 1980 respectively. She received PhD degree in Computer Engineering Department of Istanbul Technical University in Istanbul, Turkey in 1987. She is a professor in the Computer Engineering Department of Istanbul Technical University, Istanbul, Turkey. Her current research areas include distributed computing and execution environments, mobile agent systems and parallel programming.