SOFTWARE – PRACTICE AND EXPERIENCE
Softw. Pract. Exper. 2011; **41**:87–101
Published online 30 August 2010 in Wiley Online Library (wileyonlinelibrary.com). DOI: 10.1002/spe.1003

A secure migration process for mobile agents

Najmus Saqib Malik, David Ko and Harry H. Cheng*,†,‡

Integration Engineering Laboratory, Department of Mechanical and Aerospace Engineering, Computer Science Graduate Group, Electrical and Computer Engineering Graduate Group, University of California, Davis, CA 95616, U.S.A.

SUMMARY

This article describes a decentralized secure migration process of mobile agents between Mobile-C agencies. Mobile-C is an IEEE Foundation for Intelligent Physical Agents (FIPA) standard compliant multi-agent platform for supporting C/C++ mobile and stationary agents. Mobile-C is specially designed for mechatronic and factory automation systems where malicious agents may cause physical damage to machinery and personnel. As a mobile agent migrates from one agency to another in an open network, the security concern of mobile agent systems should not be neglected. Security breaches can be minimized considerably if an agency only accepts mobile agents from agencies known and trusted by the system administrator. In Mobile-C, a strong authentication process is used by sender and receiver agencies to authenticate each other before agent migration. The security framework also aims to guarantee the integrity and confidentiality of the mobile agent while it is in transit. This assures that all agents within an agency framework were introduced to that framework under the supervision and permission of a trusted administrator. The Mobile-C Security protocol is inspired from the Secure Shell (SSH) protocol, which avoids a single point of failure since it does not rely on a singular remote third party for the security process. In this protocol, both agencies must authenticate each other using public key authentication, before a secure migration process. After successful authentication, an encrypted mobile agent is transferred and its integrity is verified by the receiver agency. This article describes the Mobile-C secure migration process and presents a comparison study with the SSH protocol. The performance analysis of the secure migration process is performed by comparing the turnaround time of mobile agent with and without security options in a homogeneous environment.

Received 7 May 2009; Revised 5 July 2010; Accepted 7 July 2010

KEY WORDS: mobile agent; SSH protocol; secure migration process; authentication; integrity; confidentiality; mechatronic and embedded system

1. INTRODUCTION

The agent paradigm has evolved into a useful technology to build distributed applications [1, 2]. In the agent paradigm, tasks are performed by so-called 'agents'. An agent is a piece of executable code that interacts with the environment on behalf of a person, company or a system to meet the requirements for the assigned task(s). Agents are typically stationary agents that perform their tasks while staying on the same machine. Typical examples of stationary software agents are e-mail programs, which poll mail servers for new mail messages with the authority of a human user while running on the user's desktop. A mobile agent is a software agent that is able to move

*Correspondence to: Harry H. Cheng, Integration Engineering Laboratory, Department of Mechanical and Aerospace Engineering, Computer Science Graduate Group, Electrical and Computer Engineering Graduate Group, University of California, Davis, CA 95616, U.S.A.
†E-mail: hhcheng@ucdavis.edu
‡Professor.

around a network, migrating from host to host, in order to fulfill the given tasks as shown in Figure 2. Mobile agents have been used in many applications, such as electronic-commerce [3–5], manufacturing [6–8], network management [9–11] real time control systems, and automation environments [12–15].

Mobile agents require an agency framework to host the agents and allow them to perform their tasks. In a typical multi-agent system, there are multiple agencies running on separate hosts and agents are able to freely migrate between any of the agencies. A mobile agent that arrives at an agent platform executes its task according to the privileges given by the creator of the agent and the host platform [16].

However, a malicious person may craft an agent that, if executed, may result in substantial harm to the host agency. As an agent typically runs with the same permissions as the user running the agency, it would be relatively easy for an agent to ruin the agency account by deleting files, for instance. Alternatively, a malicious agent could steal important user data, or a person may steal data directly from a legitimate agent using a man-in-the-middle attack.

Consider the following scenario, which represents a typical mobile-agent framework deployed in an industrial environment. A distributed automation system as shown in Figure 1 is used for example for the control of industrial plants. It consists of actuators, sensors connected with plants, cables and wires for transmission, data acquisition devices and controllers. These controllers are distributed over various plants that are connected through the Internet. Mobile agents from the control room or remote office can visit the plants through the network.

In such a system as depicted in Figure 1, there are a variety of ways in which an attacker might cause harm.

(1) An outside attacker may intercept a migrating agent and steal sensitive data. Migrating agents might be carrying sensitive blueprints, or other data which should be kept secret.
(2) An outside attacker may intercept a migrating agent and modify it to perform a malicious action.
(3) An outside attacker may compose its own agent and send it into the agent framework. That agent might be able to disrupt systems and cause physical harm by controlling mechanical systems in an unsafe manner.

For this article, the authors assume that malicious attackers do not have physical or remote access to the agency servers. For instance, we assume that no malicious attackers have valid user accounts on any of the agency machines. Guarding an agency system from insider attacks and attacks exploiting the computer operating system and file system is beyond the scope of this paper.

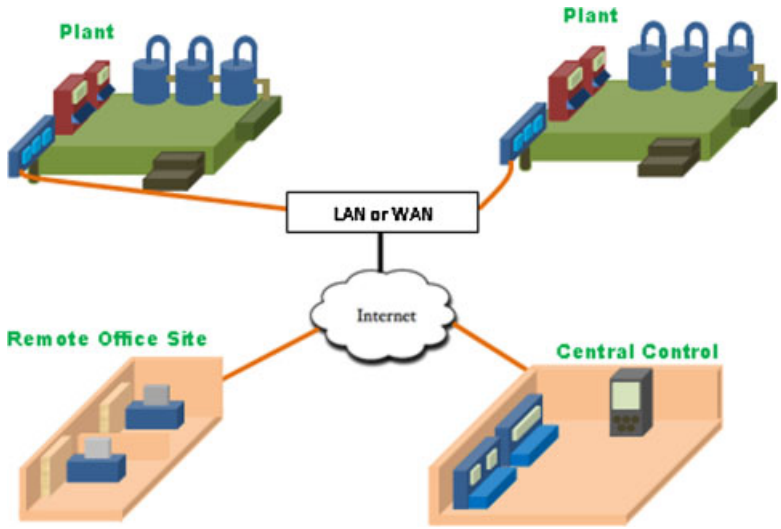

Figure 1. An example of distributed factory system.

DOI: 10.1002/spe

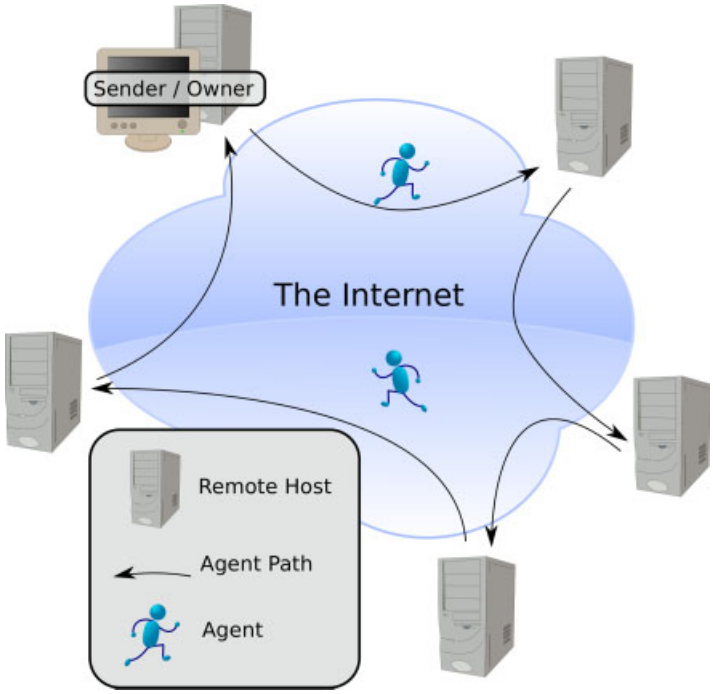

Figure 2. Mobile agent migrates from one agency to another in order to complete the task and returns back to sender.

An important goal of our security mechanism, then, is to ensure that each agent that exists on our agency framework originated inside of that framework, free from the influence from any malicious attackers. Although our agency framework runs on an open network, we need to ensure that every agent on the system is accountable to trusted system administrators who have access to the agencies.

By meeting the following requirements, an agency can deter a malicious attacker from staging any of the previously mentioned attacks.

(1) The agency can prove that the incoming agent could not be read or understood by any other entity other than itself while it was in transit.
(2) The agency can prove that the agent was not tampered with while in transit.
(3) The agency can prove that the agent came from another agency which is also trusted by the human administrator of the entire agent system.

These requirements ensure that each agent that exists on the agency framework originated from inside the agency framework. As will be described in the remainder of the article, agencies in Mobile-C satisfy this requirement [17–19].

For a decentralized agency framework, a central server may not fulfil the necessary security and performance requirements. By designing a decentralized security protocol, it eliminates the single point of failure for an agency framework, and also eliminates the need for a particular server to remain up at all times for the agency framework to function.

There exist some well-known methods for checking code for dangerous activity such as code auditing, but research has shown code auditing to be complex and difficult, if not impossible, to properly implement. Sand-boxing or otherwise severely limiting the permissions of an agent are also unacceptably prohibitive, as our agents need a high level of access in order to perform interesting tasks, such as robotic control.

This article presents a security process that fulfills the requirements of Confidentiality, Integrity, and Authentication (CIA) for the secure migration of mobile agents and ACL messages between Mobile-C agencies without the need for a central server. Mobile-C is an IEEE Foundation for

DOI: 10.1002/spe

Intelligent Physical Agents (FIPA) standard compliant multi-agent platform for supporting C/C++ mobile and stationary agents [17–19]. Mobile-C is a decentralized agency framework, in which no central server or 'main agency' is required for the agency framework to operate. Any computer in the agency network may go offline at any point and the rest of the agency framework will continue to function. In a typical Mobile-C framework, each agency is exactly like the rest, with no singular main agency which controls the rest. The design of the agency is primarily motivated by applications that require low-level hardware interface. Agents in Mobile-C are presented as an Extensible Markup Language (XML) that contains C/C++ code for easy interfacing with control programs and underlying hardware [20]. Mobile-C uses an interpretive environment to execute the agent C/C++ code known as Embedded Ch [21–23] which does not require the agent code to be compiled before it is executed. Mobile-C has been used in various applications [24], such as computational steering [25], distributed vision sensor fusion [26], and flexible robotic automation systems [27]. The security process presented in this article for Mobile-C is inspired from the Secure Shell (SSH) protocol [28]. The SSH protocol is a secure means to transmit data without the requirement of a central certificate server.

This mechanism does not consider the integrity of a mobile agent while it is executing on a current agency. We assume that the agencies are running on operating systems with virtual memory systems which prevent processes from accessing the memory space of other processes. Such attacks are beyond the scope of this paper.

The article is organized as follows: Section 2 describes the security requirements fulfilled in Mobile-C and discusses various security aspects of other mobile agent systems. Section 3 explains the Mobile-C security process in detail and Section 4 provides its comparison with SSH protocol. Section 5 shows the experiments for the performance analysis of Mobile-C with the security process. Section 6 concludes this work and finally Section 7 provides the future intentions.

2. RELATED WORK

This section presents the security requirements [29] incorporated in Mobile-C. This section also discusses the security mechanisms employed by other popular mobile agent systems for the agent migration process.

2.1. Security requirements

Following are the security requirements that are incorporated in Mobile-C for the secure migration process:

Confidentiality demands that mobile agents can only be read/understood/executed by a legitimate agency. In the context of agent systems, the primary assets of an agent are data, state, and code [16]. Confidentiality must guarantee the secrecy of the assets of an agent during the migration process.

Integrity is the property that a mobile agent has not been altered in an unauthorized manner. The success is measured based on two key factors: integrity of the mobile agent and integrity of the agent platform. The mobile agent demands that only authorized entities modify its data and code. The platform on the other hand must ensure that only authenticated agents can modify shared data.

Authentication is a process in which a receiver agency can verify that a sender of mobile agent is actually who it claims to be. Similarly, the sender is also able to verify the receiver of mobile agent. A platform must be able to hold a mobile agent responsible for its actions, performed on that host. For this reason a mobile agent must be uniquely identified and authenticated.

2.2. Security mechanism in other mobile agent systems

In Mansion [30], the user and agents are identified with a 160 bit SHA-1 hash code of their public keys. The public key corresponding to this identifier is stored in a self-signed certificate. A hand-off

DOI: 10.1002/spe

protocol is performed before the migration of a mobile agent from agency *A* to agency *B*. In this, agency *A* sends a message to a central Agent Location Service (ALS) with the identity of mobile agent and receiver agency. Agency *A* then sends the agent to agency *B*, and if agency *B* accepts the mobile agent successfully, it sends an acknowledgment to agency *A*. At the end of migration, both agencies send updated information about the migration of the mobile agent to the ALS. Both agencies can abort migration if either one does not agree for any reason. Note that for each migration of a mobile agent, both agencies must perform communication with the central service (ALS). Mansion uses the zonelib library built upon OpenSSL [31] toolkit for cryptographic functions.

Concordia [32] is a Java-based mobile agent system that provides flexible agent mobility, collaboration, and transmission. Each agent is assigned an identity that defines its access privileges to host resources. These privileges of agents can be changed dynamically by the Security Manager. Concordia uses Secure Socket Layer (SSL) [33] for secure migration of mobile agents from one agency to another.

Ara [34] is a mobile agent system that supports multiple agent languages (Tcl, C, and Java). Mobile agents can move between or stay at agencies where they use specific services provided by the agency or other agents. In Ara [35], mobile agents contain a passport that contains its identity, name, certificates, and signatures. An Ara agency is required to verify three digital signatures (agent, user and host) for authentication of mobile agents. It also provides a simple authorization of mobile agents visiting an agency. Ara uses the SSL protocol for the migration of mobile agents.

Mole [36] is a Java-based mobile agent system, in which mobile agents are uniquely identified by Ids. Each mobile agent uses badges as an identifiers for the services they can provide in a certain time period. A badge is an application generated identifier and may not necessarily have to be unique. An agent pins on a badge as long as it provides that service. As Mole is written in Java, it inherently uses the concept of the Java *Sandbox* to secure agencies from malicious intentions of mobile agent. A *sandbox* restricts the access of visiting mobile agents to system resources [37]. Mole puts restrictions on the acceptance of mobile agents from other agencies based upon their types; this would help in access control. Mole does not describe any method for a secure migration of a mobile agent from one agency to another, e.g. authentication of agency before migration, encrypted mobile agent transfer, and integrity of mobile agent during migration.

JADE [38] is a software framework implemented in Java for a multi-agent system that complies with FIPA specification. JADE uses a separate security plug-in called JADE-S [39]. It supports authentication of user and agents, rights management, encryption and signature of messages between agents within an agency. Although JADE supports migration of mobile agents among agencies, it does not provide any process for secure migration of mobile agents from one agency to another [39]. Furthermore, JADE agencies are organized in a centralized manner. Among a group of cooperating JADE agencies, known as 'containers', there must be a designated main container accessible by all the other containers for JADE to function correctly.

Tacoma [40] system supports multiple languages for mobile agents. In Tacoma, each agent contains digital certificates that are interpreted by service agents to define access permissions on the current agency. In the current version, the agents that get authorization based upon these certificates are given unrestricted access to the system. Tacoma does not provide a security mechanism for the mobile agent migration process.

JADE is one of the most popularly used Java-based mobile agent systems. However, Java-based software may consume more resources due to garbage collection, etc., and may not be suitable for operation on small embedded devices or real-time systems [41]. Furthermore, the JADE framework requires a centralized main agency to coordinate other JADE agencies. Because of this, in order for an agency system using the JADE framework to function in a persistent manner, a designated main agency must remain online and always connected. Unlike Mobile-C, which is a decentralized agency framework as described previously, the JADE main agency presents a single point of failure for the entire agency framework. If the JADE main agency goes offline, then the entire agency framework is broken.

DOI: 10.1002/spe

Ara and Tacoma support C/C++. Ara uses the Mobile Agent Computing Environment (MACE) interpreter to interpret mobile agent code. But before this interpretation, a two stage pre-compilation has to be performed that seriously degrades its performance. Tacoma does not use an interpreter. Rather, mobile agent code is compiled and the resulting binary code is executed by vm_bin (virtual machine) without any safety.

3. SECURE MIGRATION PROCESS

The migration process of mobile agents and ACL messages in Mobile-C is inspired from the SSH protocol. The security protocol was based on SSH because SSH already contains the key features required for our security process. The SSH protocol provides CIA system for the transfer of data without utilizing a central server.

The SSH protocol was chosen over the SSL protocol, which is popularly used by other agency platforms, because the SSH-style usage of a 'known-host file' was deemed to be more convenient than distributing certificates for each of the known hosts, especially for a small- to medium-sized network. This was done at the expense of system scalability, which is acceptable since Mobile-C is mostly intended to be used in industrial and embedded applications, which generally do not require as much scalability as, for instance, a pure peer-to-peer network. Furthermore, the typical Mobile-C network composed of desktops and embedded devices in an industrial setting is relatively static.

Like SSH, before performing an agent migration, each agency must authenticate each other. A successful authentication indicates that each host is who it claims to be. Authentication is performed via public-key cryptography, similar to SSH. During the authentication process, each agency keeps a private known-host file containing public keys of all other agencies with which agent migration is allowed, and its own public and private key file.

The known-host file: Modeled after the SSH 'known-hosts' file, the Mobile-C known-host file contains the host name and public key of each trusted agency in a network. The known-host file is created and supplied by a system administrator. In a secure system setup, each agency is only allowed to communicate with hosts listed in the known-host file. The security of the known-host file is considered very important and should not be neglected. Therefore unlike SSH, a Mobile-C agency provides the option to store the known-host file in an encrypted form on secondary storage. The structure of a known-host file is shown in Figure 3.

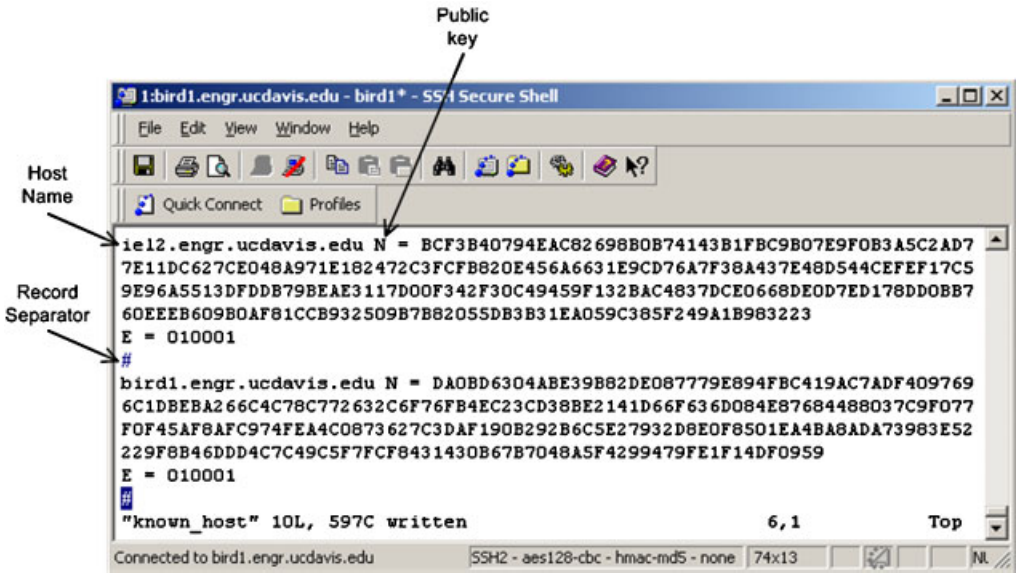

Figure 3. A sample known-host file.

DOI: 10.1002/spe

Public and private key files: Each agency contains a pair of key files (public and private) that are used for authentication process. Like in the SSH protocol, an agency's private key file plays a pivotal role in the security process. The private key file must by kept secure by the system administrator, and may also be stored in encrypted form on secondary storage.

3.1. Authentication process

Authentication refers to a process in which an agency ensures that the other agency in a conversation is in fact who it is declared to be. Before the secure transfer of a mobile agent between two agencies, they must authenticate each other.

Each agency in a network contains a list of known hosts provided by the administrator. This list provides RSA public keys of other trusted agencies in a network. Before agency *A* wants to transfer a mobile agent to agency *B*, agency *A* must verify that agency *B* contains a correct private key for the public key in agency *A*'s known-host list. In addition, agency *B* must verify that agency *A* contains a correct private key for the public key in agency *B*'s known-host list. An overview of the authentication process is shown in Figure 4. All messages between agency *A* and agency *B* are encrypted/decrypted using RSA algorithm with 1024 bit key size.

3.2. Confidentiality

ISO defines confidentiality as '*ensuring that information is accessible only to those authorized to have access*'. This means that while a mobile agent is migrating from agency *A* to agency *B*, it would not be accessible in an understandable form by any adversary.

Mobile-C uses an AES 256 bit key to encrypt the mobile agent at the sending agency and to decrypt it at receiver agency. The insurance of security of this process relies on a secure transfer of the AES 256 bit key. Public key en-/decryption is used to transfer the AES key securely. The AES key is exchanged between two agencies in the authentication process. This eliminates the further exchange of messages between two agencies for the AES key transfer. After successful authentication, the sender agency encrypts the mobile agent with the AES key and the receiver can decrypt it. According to National Institute of Standards and Technology (NIST) AES with 256 bit key size is safe to use for data encryption until 2030 [23].

The same nonce (as used in the authentication process) is used as session identifier during the transfer of both the AES key and the mobile agent. This is to avoid the replay back attack on agencies.

3.3. Integrity

Integrity is a process to ensure that the contents of a mobile agent are the same as sent by the sender agency. In other words, a successful integrity check should ensure that the agent was not tampered with while in transit from the sending agency to the receiving agency. Mobile-C uses a SHA2 hash code to check the integrity of mobile agents. Figure 5 shows a process of secure transfer of a mobile agent from agency *A* to agency *B*. At the end, it shows an integrity check in which agency *B* compares two hash codes. If the hash code does not match, agency *B* informs agency *A* through a verification code. Then, another agent migration is attempted.

3.4. Random number generation

True random number generation is always an issue and an important concern for cryptographic algorithms. The programming language C's random function has vulnerabilities and is thus not recommended for use in cryptographic applications. For this reason, Mobile-C uses HArdware Volatile Entropy Gathering and Expansion (HAVEGE) for high random number generation [42]. It is a heuristic software approach to generate empirically strong random numbers. Mobile-C uses HAVEGE to generate the nonce, challenge text, and AES 256 bit key during the mobile agent migration process.

DOI: 10.1002/spe

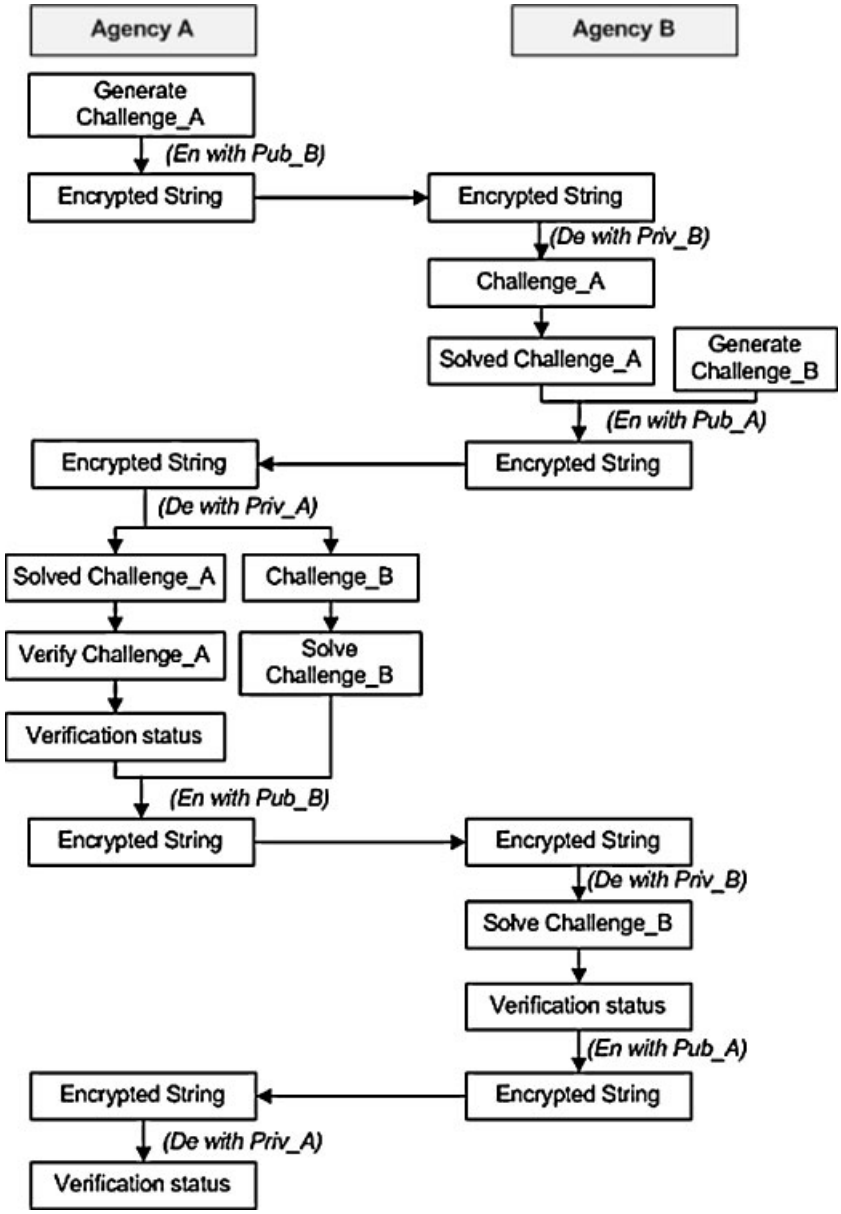

Figure 4. The authentication process for Mobile C agencies: En., Encryption; De., Decryption; Pub_B., Public key of B; and Priv_B., private key of B.

4. COMPARISON WITH SSH

In this section a comparison study with the SSH protocol is performed. Mobile-C security bases the authentication, confidentiality and integrity mechanism from SSH.

4.1. Authentication method

In the SSH protocol, there is a clear distinction between the SSH client and the server. However, in Mobile-C, each agency is a peer. That is, all agencies are treated at the same level, e.g. an agency can behave like a client (e.g. sending mobile agents) and at the same time as a server (receiving mobile agents). Thus in Mobile-C, each agency uses only public key authentication to authenticate

DOI: 10.1002/spe

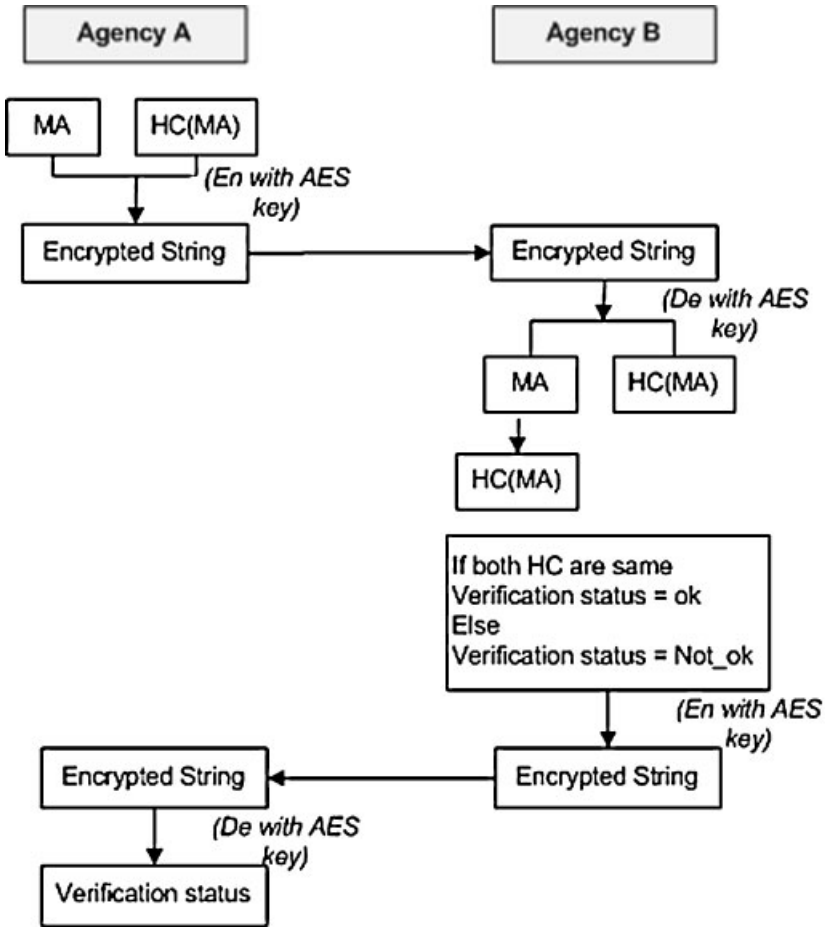

Figure 5. Secure transfer and integrity process of mobile agent in Mobile-C HC—Hash code. AES key is already exchanged in the Piggy back authentication process.

the other agency, unlike SSH which may use a variety of methods such as password-based and RhostsRSA authentication. In addition each agency encrypts an MD5 digest that is sent to the other peer, which is used in unencrypted form in SSH. Thus, any MD5 weakness can become vulnerable for the protocol that uses it, as long as the MD5 digest is sent in unencrypted form. In a security advisory report (No. 961509) [43], Microsoft Security Research and Defense has mentioned the weakness of MD5 and recommended not to use MD5 further in any application.

The Mobile-C authentication process provides much less cipher text to a passive attacker to find possible plaintext. This is because every message which is encrypted and sent to the other agency in the authentication process is different in length and structure as shown in Figure 6.

In SSH, brute force attacks can be prevented by using public key authentication provided that other forms of authentications, e.g. password authentication are disabled. This means that brute force attacks cannot break the Mobile-C authentication process, since Mobile-C uses the public key authentication process to authenticate agencies.

4.2. Private key

The SSH protocol uses a program called *agent* that caches keys (which are stored in encrypted form on the secondary storage) into main memory. The main SSH program communicates with *agent* when requested for key-related operations such as decryption and signing. The purpose of storing the private key in the encrypted form is to minimize the attack in which an adversary tries to steal the private key from the secondary storage such as the hard disk. Thus when the SSH

DOI: 10.1002/spe

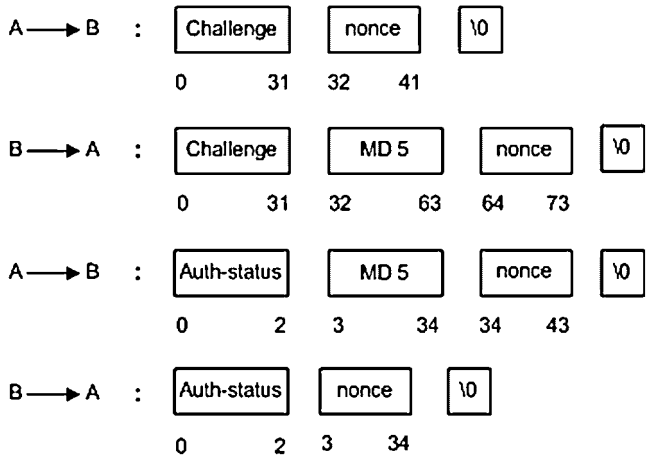

Figure 6. Messages passed between Mobile-C agencies *A* and *B* for authentication. Every message is encrypted with the receiver's public key.

starts, this key is decrypted by the *agent* and stored in main memory from where this attack is difficult to perform.

In Mobile-C, each agency has private and public keys. Mobile-C provides an option to store the private key file in encrypted form using AES (256 bit key) algorithm or in plaintext on the hard disk. When an agency starts, it checks whether the private key file is encrypted or not. If the private key file is encrypted and passphrase is not provided within the agency source code, then it prompts for a passphrase from the administrator. After that, the user does not need to enter passphrase each time during the decryption process. Unlike the SSH, there is no separate program (*agent*) for this process. Thus, it eliminates communication between the main program and *agent* each time there is a need to perform the decryption process.

4.3. Known-host file

SSH performs authentication using a known-host file at the client side and authorization file on the server. A known-host file contains names and public keys of all hosts in a network, whereas the authorization file also includes authorizations for clients/users. The first step in the authentication process is the file lookup. If the file lookup is successful for a connecting peer, then the authentication process proceeds. Otherwise it is terminated or it prompts the user for permission to add a public key of the connecting peer in the file. These files are contained by clients and servers, provided by the administrator. Unlike the argument that the SSH provides to encrypt private key files contained by each entity in a network, SSH stores the known-host (client side) and authorization (server side) files in plain text in the secondary storage.

In Mobile-C, each agency uses the known-host file for initial authentication of other agencies. A Mobile-C agency provides an option to its user to save the known-host file in plaintext or in the encrypted form on the secondary storage. This is not hard coded because of the tradeoff between sizes of the known-host file and available memory on agency. If the known-host file is large and the Mobile-C agency has less memory available, for example, in the case of old embedded chips, the plaintext option would be preferable.

4.4. Confidentiality and integrity

The SSH protocol uses AES 256 bit key for encryption and decryption. For integrity SSH calculates the SHA2 digest of message which is compared at the receiver side. Mobile-C uses the same algorithm for confidentiality and integrity of a mobile agent during the migration process. It calculates the SHA2 digest when encrypting and sending a mobile agent to the receiver. The receiver will verify the SHA2 digest after decrypting the mobile agent.

DOI: 10.1002/spe

5. PERFORMANCE ANALYSIS

This section provides details about the experimental setup for performance analysis of a mobile agent migration in Mobile-C with our security protocol. The performance experiment analyzes the turnaround time for a mobile agent. The turnaround time [44] can be defined as the time between submitting a job and receiving its response when finished. For a mobile agent, the turnaround time can be defined as:

The time it takes for a mobile agent to complete a given task by performing on the local machine or migrating to one or more agencies in a network.

The performance of mobile agent migration is important for critical applications in a control network, e.g. in a temperature control system. Obviously, the turnaround time of a mobile agent depends on the network traffic as well as the number of agents handled by the agencies since they affect the agent mobility. The benchmark tests that are used in the performance analysis of Mobile-C were originally proposed by the JADE research group to evaluate the performance of agent migration in JADE [38, 45].

Setup for experiments: These experiments simulate the relay race concept by using mobile agents. Each machine has a Mobile-C agency running on it. There are two types of mobile agents used in this experiment: *Trigger agents* and *Runner agents*. Initially, *Trigger agents* are located on the trigger agency and *Runner agents* on other agencies. When all the *Runner agents* are loaded in their respective agencies, the trigger agents are executed on the trigger agency. The *Trigger agents* migrate to agency no. 1 to signal the *Runner agents*. After receiving the signal, these *Runner agents* migrate to the second agency to signal local *Runner agents* and so on. Figure 7 shows a similar scenario in which mobile agent *2* running on agency *bird2* waits for the signal from mobile agent *1* that is sent by agency *rabbit* to agency *bird2*. After mobile agent *2* gets the signal it migrates to agency *phoenix*. This process continues until mobile agent *4* migrates to *rabbit*.

A homogeneous environment is used for these experiments that consist of four machines, each with Intel Pentium 4 processor 3.2 GHz and 512 MB of RAM. All four machines are running the Gentoo/Linux operating system. These machines are connected through a switch with a 100 Mbits/s transmission rate. Each value of turnaround time is an average of 5 runs. In these experiments the turnaround of mobile agent is calculated with and without security operations. Below, two cases for these experiments are discussed.

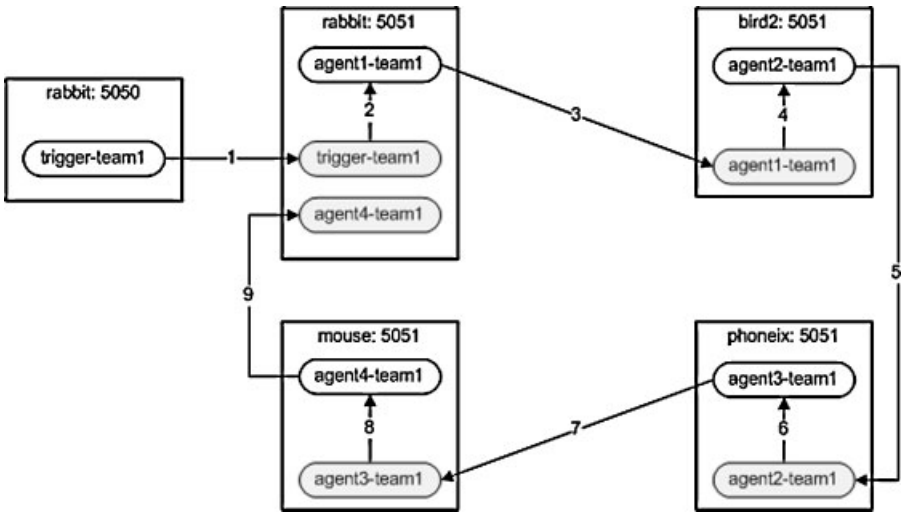

Figure 7. Relay Race simulation using mobile agents—Basic Setup—message 1, 3, 5, 7, and 9 are agent migrations and 2, 4, 6, and 8 are trigger messages.

DOI: 10.1002/spe

5.1. *Case 1*: *One agent team*

This case consists of one agent-team that migrates among different agencies. The number of runner agents in the agent-team is the same as the number of agencies. The number of agencies included in this case are two, three, and four. In the beginning, the runner agents are loaded by each agency, where they wait for the other agent to arrive. This state is called 'stand by position'. As soon as the trigger agent arrives at the first agency, the runner agent on that agency will move to the second agency. The team completes one lap when the runner agent on the last agency arrives at the first agency.

5.2. *Case 2*: *Multiple agent team*

In this case multiple agent teams migrate among four agencies. The number of agent teams are from 1 to 10. Each team consists of four runner agents and a trigger agent. The runner agents are in standby position at the beginning. The trigger agent of the respective agent team will be sent to the first agency to signal their runner agent to start migration to the next agency. Each team owns a separate trigger agent that may arrive at the first agency at different times and signal their corresponding runner agent. A potential problem here is that the runner agents on the first agency may leave the agency at different times. In order to make sure that the first runner agent of each team leaves the first agency at approximately the same time, the trigger agent will not send signals to their respective runner agent until all trigger agents arrive at the first agency. The synchronization support in Mobile-C is used to achieved the coordination among the trigger agents using a 'barrier'. This barrier allows the trigger agents to wait on the first agency until all the trigger agents arrive. In this case, each team completes one lap that is the runner agent on the last agency (in our case its fourth) arrives at the first agency.

5.3. *Analysis of performance test results*

Figure 8 shows the turnaround time for one agent team with two, three, and four agencies. Figure 9 shows the turnaround time for different number of agent team that complete one lap around four agencies. From the results it is observed that as the number of agencies increases the turnaround time of mobile agent also increases linearly. It can be observed from both figures that the turnaround time for mobile agents with the security process is approximately nine times more than without the security. This is because of the exchange of extra messages for the authentication process and processing time of these messages both at the local agencies and on network queues. Mobile agent migration, with no security, requires only two messages. First, a mobile agent is wrapped up into an http packet and sent to the receiver agency, and second is the http response that is sent by the receiver agency to the sender agency. With security option, however, seven messages are

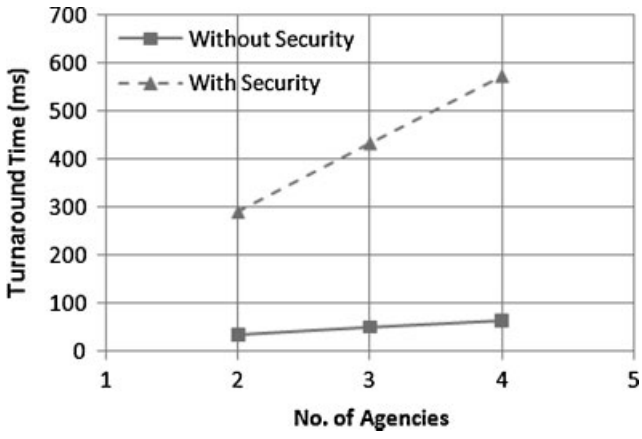

Figure 8. Turnaround time of one agent-team with different number of agencies—with and without security.

DOI: 10.1002/spe

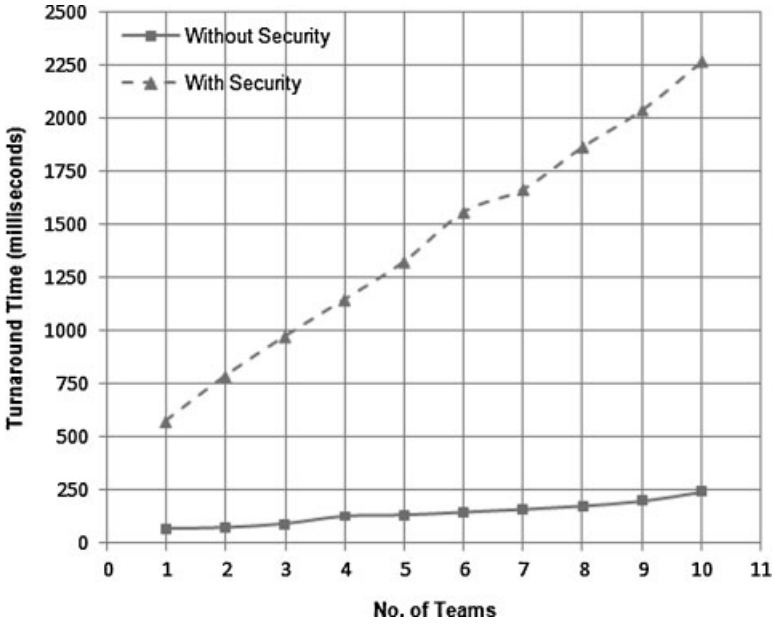

Figure 9. Turnaround time for different number of agent-teams with four agencies—with and without security.

exchanged between the sender and receiver agencies; Six messages are shown in Figures 4 and 5 and the seventh is the http response. The receiver agency sends the http response to the sender because the sender has sent the entire encrypted http packet. Thus when the receiver decrypts the message, it verifies the validity of the http packet. Based upon this, the receiver sends the http response to the sender agency.

For one migration of a message, there are four authentication messages that are en-/decrypted by the sender and receiver agencies, respectively. The size of these four messages are shown in Figure 6. The en-/decryption time of these four messages using RSA algorithm with 1024 bit key size is 30 ms. Similarly, the en-/decryption time of a mobile agent (used in these experiment) is 54 ms.

6. CONCLUSION

This paper describes a decentralized secure migration process for mobile agents between Mobile-C agencies inspired from SSH. By modeling the Mobile-C security protocol after SSH, Mobile-C is able to perform secure migrations using convenient known-host files, which is suitable for small- to medium-sized networks. By designing a decentralized process, we eliminate all single points of failure. Emphasis is given to the authentication process of agencies prior to migration of mobile agent or ACL messages. This ensures that Mobile-C agencies only receive and execute mobile agents from trusted agencies. An AES key is exchanged during the authentication process. After successful authentication, this key is used to encrypt and decrypt mobile agents at the sender and receiver sides, respectively, ensuring the confidentiality of an agent. A SHA2 hash code is used for integrity checking of mobile agents during migration process. This mechanism ensures confidentiality and integrity of mobile agents during the migration process, authentication of agencies and avoids replay attacks.

The experimental results show that the turnaround time of the mobile agent is increased about nine times with security option, but is still within acceptable range for most embedded soft real-time systems.

DOI: 10.1002/spe

7. FUTURE WORK

This paper describes a secure migration process for Mobile-C between the agencies under the direct control of a system administrator. In the future work, the emphasis will be on the following:

(1) Address scalability issues, such as a method of adding new agencies in a known-host file. An obvious drawback to using separate known-host files on each host is that the files should be synchronized across a number of hosts, and adding a new host requires modifying the known-host file on every host. For a relatively small network consisting of dozens of machines this is not much of an issue, but it becomes an exponential issue as the number of hosts increases. While our primary concern for this paper is secure operation, scalability is also important for general multipurpose agent systems.
(2) Human accountability for agent actions. An important concept to maintain a secure network is to have at least one user accountable for each agent. If an agent damages a system, whether intentionally or accidentally, actions should be logged in an unforgeable manner and a human should be held accountable for an agent's actions.
(3) Access privileges of mobile agents would provide limitations to its ability to execute code and use system resources. A subset of access privileges can be defined in general but more confined access privileges depend upon the underlying application.

REFERENCES

1. Wooldridge M, Ciancarini P. Agent-oriented software engineering: the state of the art. *First International Workshop*, *AOSE 2000 on Agent-oriented Software Engineering*. Springer: Secaucus, NJ, U.S.A., 2001; 1–28.
2. Jennings NR. An agent-based approach for building complex software systems—Why agent-oriented approaches are well suited for developing complex, distributed systems. *Communications of the ACM* 2001; **44**(4):35–41.
3. Wang Y, Wong DS, Wang H. Employ a mobile agent for making a payment. *Mobile Information Systems* 2008; **4**(1):51–68.
4. Chen H, Lam P, Chan H, Dillon T, Cao J, Lee R. Business-to-consumer mobile agent-based internet commerce system (MAGICS). *IEEE Transactions on Systems*, *Man*, *and Cybernetics*, *Part C*: *Applications and Reviews* 2007; 1174–1189.
5. Chou P, Chen K, Wu M. Use of an intelligent agent for an E-commerce bargaining system. *Knowledge-Based Intelligent Information and Engineering Systems*. Springer: Berlin, 2006; 300–309.
6. Di Stefano A, Santoro C. A3M: An agent architecture for automated manufacturing. *Software*: *Practice and Experience* 2008, **39**(2):137–162.
7. Cheng C, Wang C. Outsourcer selection and order tracking in a supply chain by mobile agents. *Computers and Industrial Engineering* 2008; **55**(2):406–422.
8. Huang C-Y, Cheng K, Holt A. An integrated manufacturing network management framework by using mobile agent. *The International Journal of Advanced Manufacturing Technology* 2007; **32**(7–8):822–833.
9. Gavalas D, Tsekouras G, Anagnostopoulos C. A mobile agent platform for distributed network and systems management. *Journal of Systems and Software* 2009; **82**(2):355–371.
10. Al-Kasassbeh M, Adda M. Analysis of mobile agents in network fault management. *Journal of Network and Computer Applications* 2008; **31**(4):699–711.
11. Verma V, Joshi R, Xie B, Agrawal D. Combating the bloated state problem in mobile agents based network monitoring applications. *Computer Networks* 2008; **52**(17):3218–3228.
12. Luder A, Klostermeyer A, Peschke J, Bratoukhine A, Sauter T. Distributed automation: PABADIS versus HMS. *IEEE Transactions on Industrial Informatics* 2005; **1**(1):31–38.
13. Plösch R, Weinreich R. An agent-based environment for remote diagnosis, supervision and control. *Internet Applications* 1999; **1749**:447–469.
14. Lange DB, Oshima M. Seven reasons for mobile agents. *Communications of the ACM* 1999; **42**(3):88–89.
15. Milojici D. Mobile agent applications. *IEEE Concurrency* 1999; **7**(3):80–90.
16. Varadharajan V. Security enhanced mobile agents. *Proceedings of the Seventh ACM Conference on Computer and Communications Security*, Athens, Greece, 2008; 200–209.
17. Mobile-C: A Multi-agent Platform for Mobile C/C++ Code. Available at: http://www.mobilec.org [24 June 2010].
18. Chen B, Cheng HH, Palen J. Mobile-C: A mobile agent platform for mobile C/C++ agents. *Software—Practice and Experience* 2006; **36**(15):1711–1733.
19. Chou Y-C, Ko D, Cheng HH. An embeddable mobile agent platform supporting runtime code mobility, interaction and coordination of mobile agents and host systems. *Information and Software Technology* 2010; **52**(2):185–196.
20. Chen B, Linz DD, Cheng HH. XML-based agent communication, migration and computation in mobile agent systems. *Journal of Systems and Software* 2008; **81**(8):1364–1376.

DOI: 10.1002/spe

21. Cheng HH. *C for Engineers and Scientists*: *An Interpretive Approach*. McGraw-Hill: New York, 2009. Available at: http://iel.ucdavis.edu/cfores [January 2009].
22. *Embedded Ch*. SoftIntegration, Inc. Available at: http://www.softintegration.com/products/sdk/embedded_ch/ [15 April 2009].
23. Embedded Ch: A C/C++ Embeddable Interpretor. Available at: http://www.softintegration.com/products/sdk/embedch/ [November 2008].
24. Mobile Agent-based Applications. Available at: http://www.mobilec.org/applications.php [December 2008].
25. Chou Y-C, Ko D, Cheng HH. Mobile agent-based computational steering for distributed applications. *Concurrency and Computation*: *Practice and Experience* 2009; **21**:2377–2399.
26. Nestinger S, Cheng HH. Mobile agent approach to distributed vision sensor fusion. *IEEE Robotics and Automation Magazine*, in press.
27. Nestinger S, Chen B, Cheng HH. A mobile agent-based framework for flexible automation systems. *IEEE/ASME Transactions on Mechatronics*, DOI: 10.1109/TMECH.2009.2036169.
28. Barrett D, Silverman R, Byrnes R. *SSH*, *the Secure Shell*: *the Definitive Guide*. O'Reilly Media, Inc.: U.S.A., 2005.
29. Jansen W, Karygiannis T. Mobile Agent Security. *NIST Special Publication 800-19*, 1999.
30. van't Noordende G, Brazier F, Tanenbaum A. Security in a mobile agent system. *2004 IEEE First Symposium on Multi-Agent Security and Survivability*, Philadelphia, 2004; 35–45.
31. OpenSSL: The Open Source Toolkit for SSL/TLS. Available at: http://www.openssl.org [24 June 2010].
32. Wong D, Paciorek N, Walsh T, DiCelie J, Young M, Peet B. Concordia: An infrastructure for collaborating mobile agents. *Proceedings of the First International Workshop on Mobile Agents* (*MA'97*) (*Lecture Notes in Computer Science*, vol. 1219). Springer: Berlin/Heidelberg, 1997; 86–97.
33. Weaver AC. Secure sockets layer. *IEEE Computer* 2006; **39**(4):88–90.
34. Peine H. Application and programming experience with the ara mobile agent system. *Software—Practice and Experience* 2002; **32**(6):515–541.
35. Peine H. Security concepts and implementation in the Ara mobile agent system. *Seventh IEEE International Workshops on Enabling Technologies*: *Infrastructure for Collaborative Enterprises*, *1998* (*WET ICE'98*) *Proceedings*, Stanford, CA, 1998; 236–242.
36. Baumann J, Hohl F, Rothermel K, Strasser M, Theilmann W. MOLE: A mobile agent system. *Software—Practice and Experience* 2002; **32**(6):575–603.
37. Gosling J. How computer security works—the java sandbox. *Scientific American* 1998; **279**(4):78–81.
38. Bellifemine F, Caire G, Poggi A, Rimassa G. JADE: A software framework for developing multi-agent applications. lessons learned. *Information and Software Technology* 2008; **50**(1–2):10–21.
39. Endsuleit R, Calmet J. A security analysis on JADE(-S) V. 3.2. *Proceedings of NordSec*, Tartu, Estonia, 2005.
40. Johansen D, Lauvset K, Van Renesse, Schneider F, Sudmann N, Jacobsen K. A tacoma retrospective. *Software—Practice and Experience* 2002; **32**(6):605–619.
41. Czajkowski G, Von Eicken T. JRes: A resource accounting interface for Java. *Proceedings of the 13th ACM SIGPLAN Conference on Object-oriented Programming*, *Systems*, *Languages*, *and Applications*. ACM: New York, 1998; 21–35.
42. Seznec A, Sendrier N. HAVEGE: A user-level software heuristic for generating empirically strong random numbers. *ACM Transactions on Modeling and Computer Simulation* (*TOMACS*) 2003; **13**(4):346.
43. Microsoft Security Advisory (961509). Available at: http://www.microsoft.com/technet/security/advisory/961509.mspx [8 January 2009].
44. PC Magazine. Available at: http://www.pcmag.com [26 March 2009].
45. Chmiel K, Gawineck M, Kaczmarek P, Szymczak M, Paprzycki M. Efficiency of JADE agent platform. *Scientific Programming* 2005; **13**(2):159–172.