ISSN: 2321-7782 (Online)
Volume 1, Issue 7, December 2013
International Journal of Advance Research in Computer Science and Management Studies
Research Paper
Available online at: www.ijarcsms.com

Study of Existing Security Enhancement for Software Agent

Sachin Upadhye[1]
Professor
Department of Computer Application
Shri Ramdeobaba College of Engg. and Management
Nagpur - India

Dr. P. G. Khot[2]
P.G.T.D. of Statistics
Rashtrasant Tukadoji Maharaj
Nagpur University
Nagpur - India

***Abstract:** Software Agent is an exciting and important new way to create complex software systems. Agents act on their own by reacting to changes and by planning their course of action. In software agent, mobile agents are moving around the network are not safe because the remote hosts that accommodate the agents can initiate all kinds of attacks and can attempt to analyze the agents' decision logic and their state and behaviour. This paper is based on existing study of software agent security. The paper is organized as, first section introduces the software agent, Next sections focus on software agent security threats Section 3 explain the countermeasures of software agent; last section explain area for further research and finally concludes the paper work.*

***Keywords:** software agent security, Security analysis of software agent, software agent security threats, countermeasures of software agent, intelligent agent.*

I. INTRODUCTION

Agents make an interesting topic of study because they draw on and integrate so many diverse disciplines of computer science, including objects and distributed object architectures, adaptive learning systems, artificial intelligence, expert systems, genetic algorithms, distributed processing, distributed algorithms, collaborative online social environments, and security. In the wider context, software agents represent an area of Artificial Intelligence (AI) combined to computer science. Distributed Artificial Intelligence (DAI) is a subfield of AI that is concerned with solving problems in distributed manner and one of its research areas is agent-based computing. Game theory has been widely used in order to understand the behavioural framework as well as decision makers for software agents. In most of the cases agents need to negotiate and collaborate with other agents in order to achieve their objectives [1].

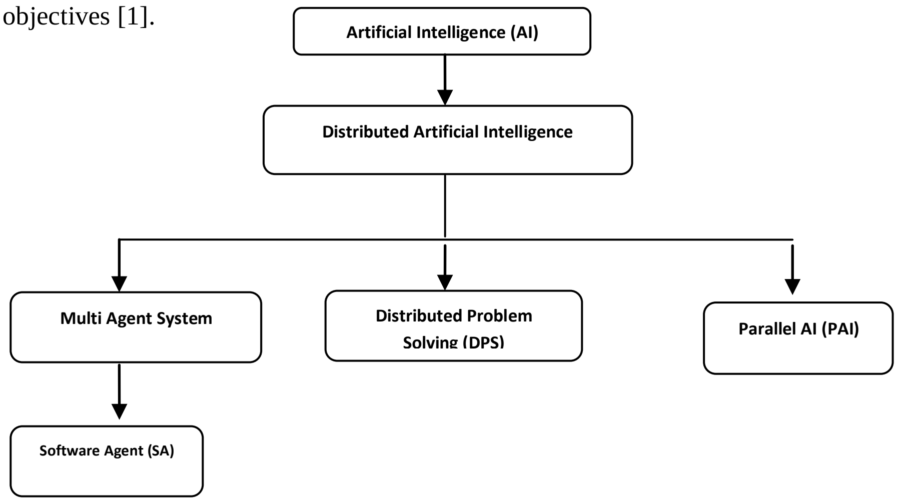
Artificial Intelligence (AI)
Distributed Artificial Intelligence
Multi Agent System
Distributed Problem Solving (DPS)
Parallel AI (PAI)
Software Agent (SA)

Figure 1: Software Agent and AI

The building of software agents suggests a new way of viewing existing technologies as tools to build software applications that dynamically interact and communicate with their immediate environment. In recent years the interest in Agent has grown tremendously, and today Agent technology is being used in a large range of important industrial application areas. These application ranges from information management through industrial process control to electronic commerce, one thing is common agents must be able to "talk" to each other to decide what action to take and how this action can be coordinated with others action. There is no commonly agreed definition of software agents. Different authors have different views regarding these. A few of the popular definitions are: "a persistent software entity dedicated to a specific purpose" [2], "computer programs that simulate a human relationship by doing something that another person could do for you" [3], "a software entity to which tasks can be delegated" [4]. Many people agree that: "software agent is a component that interacts with its environment and with other agents on a user's behalf." Based on the above statements, scholars gave a summarized statement for software agents:-"an autonomous software agent is a component that interacts with its environment and with other agents on a user's behalf."Some definitions emphasize one or another characteristic such as mobility, collaboration, intelligence or flexible user interaction [5].

Characteristic of Software Agent

As historical relationship with the AI community and the vague notion of intelligence various characteristic for the software agents have been proposed. For instance, Woodridge and Jennings‟ weak agents [6] should be autonomous, reactive and social. A definition from Franklin & Graesser [7] lists autonomous, reactive, communicative, adaptive, mobile, flexible, goal-oriented, continuous and with some form of character or emotion. The most of the characteristic are based upon object, state and behaviour. There is various characteristics work together to make agent -oriented systems more flexible and robust to Change such as [8].

- *Autonomous* - An agent should be able to execute without the need for human interaction, although intermittent interaction may be required.
- *Adaptability*- Adaptive agents have the ability to adjust their behaviour over time in response to internal knowledge or changes in the environment around them.
- *Collaboration* - The degree to which agents communicates and works cooperatively with other agents to form multi agent systems working together on some task.
- *Social / Communicative* - An agent should have a high level of communication with other agents. The most common protocol for agent communication is the Knowledge Query and Manipulation Language (KQML).
- *Knowledgeable* - The degree to which an agent is capable of reasoning about its goals and knowledge.
- *Mobility*- Mobile agents can proactively decide to migrate to a different machine or network while maintaining persistence.
- *Reactive / Responsive* - An agent should be able to perceive its environment and react to changes in it.
- *Proactive* - proactive agents do not just react to their environment but can take active steps to change that environment according to their own desires.
- *Goal-oriented / Intentions* - These agents have an explicit internal plan of action to accomplish a goal or set of objectives.
- *Persistence / Continuous* - Persistent agents have an internal state that remains consistent over time.The degree to which the infrastructure enables agents to retain knowledge and state over extended periods of time, including robustness in the face of possible run -time failures.[8]

- *Emotion* - Agents with the ability to express human -like emotion or mood. Such agents might also have some form of anthropomorphic character or appearance.
- *Intelligence* - Agents with the ability to reason learn and adapt over time.
- *Honesty*- Agents that believe in the truthful nature of the information they pass on.

II. Software Agent Security Threats

Threats to security generally fall into three main classes: disclosure of information, denial of service, and corruption of information. There are a variety of ways to examine these classes of threats in greater detail as they apply to agent systems[10].Four threat categories are identified: threats stemming from an agent attacking an agent platform, an agent platform attacking an agent, an agent attacking another agent on the agent platform, and other entities attacking the agent system. The last category covers the cases of an agent attacking an agent on another agent platform, and of an agent platform attacking another platform, since these attacks are primarily focused on the communications capability of the platform to exploit potential vulnerabilities.[11, 12, 13]

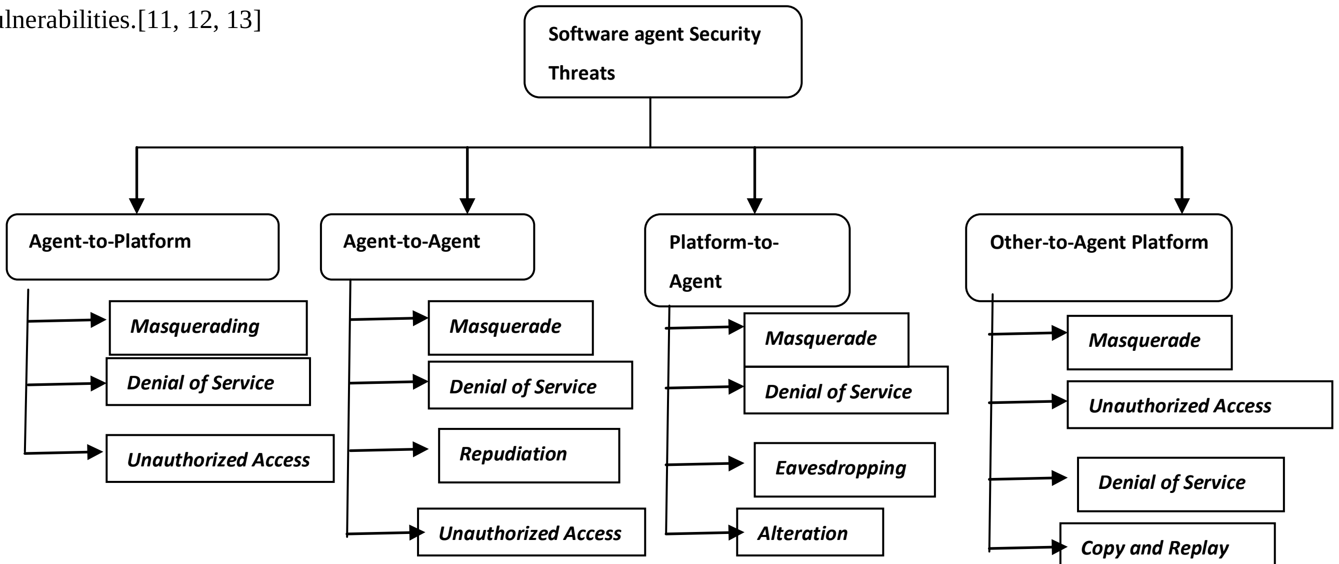

Figure 2: Software Agent Security Threats

Agent-to-Platform

The agent-to-platform category represents the set of threats in which agents exploit security weaknesses of an agent platform or launch attacks against an agent platform. This set of threats includes masquerading, denial of service and unauthorized access.

Masquerading

When an unauthorized agent claims the identity of another agent it is said to be masquerading. The masquerading agent may pose as an authorized agent in an effort to gain access to services and resources to which it is not entitled. A masquerading agent may damage the trust the legitimate agent has established in an agent community and its associated reputation.

Denial of Service

Mobile agents can launch denial of service attacks by consuming an excessive amount of the agent platform's computing resources. This denial of service attacks can be launched intentionally by running attack scripts to exploit system vulnerabilities, or unintentionally through programming errors

Unauthorized Access

Access control mechanisms are used to prevent unauthorized users or processes from accessing services and resources for which they have not been granted permission and privileges as specified by a security policy. Each agent visiting a platform must be subject to the platform's security policy.

Agent-to-Agent

The agent-to-agent category represents the set of threats in which agents exploit security weaknesses of other agents or launch attacks against other agents. This set of threats includes masquerading, unauthorized access, denial of service and repudiation.

Masquerade

Agent-to-agent communication can take place directly between two agents or may require the participation of the underlying platform and the agent services it provides. In either case, an agent may attempt to disguise its identity in an effort to deceive the agent with which it is communicating.

Denial of Service

In addition to launching denial of service attacks on an agent platform, agents can also launch denial of service attacks against other agents. Agents that are being spammed may choose to block messages from unauthorized agents, but even this task requires some processing by the agent or its communication proxy [13].

Repudiation

Repudiation occurs when an agent, participating in a transaction or communication, later claims that the transaction or communication never took place. Whether the cause for repudiation is deliberate or accidental, repudiation can lead to serious disputes that may not be easily resolved unless the proper countermeasures are in place.

Unauthorized Access

An agent can directly interfere with another agent by invoking its public methods (e.g., attempt buffer overflow, reset to initial state, etc.), or by accessing and modifying the agent's data or code. Modification of an agent‟s code is a particularly insidious form of attack, since it can radically change the agent's behaviour (e.g., turning a trusted agent into malicious one).

Platform-to-Agent

The platform-to-agent category represents the set of threats in which platforms compromise the security of agents. This set of threats includes masquerading, denial of service, eavesdropping, and alteration.

Masquerade

One agent platform can masquerade as another platform in an effort to deceive a mobile agent as to its true destination. An agent platform masquerading as a trusted third party may be able to lure unsuspecting agents to the platform and extract sensitive information from these agents. The masquerading platform can harm both the visiting agent and the platform whose identity it has assumed. An agent that masquerades as another agent can harm other agents only through the messages they exchange and the actions they take as a result of these messages [14], [15]

Denial of Service

When an agent arrives at an agent platform, it expects the platform to execute the agent's requests faithfully, provide fair allocation of resources, and abide by quality of service agreements. Agents on other platforms waiting for the results of a non-responsive agent on a malicious platform must be careful to avoid becoming deadlocked. An agent can also become live locked if a malicious platform, or programming error, creates a situation in which some critical stage of the agent's task is unable to finish because more work is continuously created for it to do.

Eavesdropping

The classical eavesdropping threat involves the interception and monitoring of secret communications. The threat of eavesdropping, however, is further exacerbated in mobile agent systems because the agent platform can not only monitor communications, but also can monitor every instruction executed by the agent, all the unencrypted or public data it brings to the platform, and all the subsequent data generated on the platform.

Alteration

When an agent arrives at an agent platform it is exposing its code, state, and data to the platform. Since an agent may visit several platforms under various security domains throughout its lifetime, mechanisms must be in place to ensure the integrity of the agent's code, state, and data. Agent platforms can also tamper with agent communications. The security risks resulting from an agent moving from its home platform to another is referred to as the "single-hop" problem, while the security risks resulting from an agent visiting several platforms is referred to as the "multi-hop" problem.[10]

Other-to-Agent Platform

The other-to-agent platform category represents the set of threats in which external entities, including agents and agent platforms, threaten the security of an agent platform. This set of threats includes masquerading, denial of service, unauthorized access, and copy and replay.

Masquerade

Agents can request platform services both remotely and locally. An agent on a remote platform can masquerade as another agent and request services and resources for which it is not authorized.

Unauthorized Access

Remote users, processes, and agents may request resources for which they are not authorized. Remote access to the platform and the host machine itself must be carefully protected, since conventional attack scripts freely available on the Internet can be used to subvert the operating system and directly gain control of all resources.

Denial of Service

Agent platform services can be accessed both remotely and locally. The agent services offered by the platform and inter-platform communications can be disrupted by common denial of service attacks. Agent platforms are also susceptible to all the conventional denial of service attacks aimed at the underlying operating system or communication protocols.

Copy and Replay

Every time a mobile agent moves from one platform to another it increases its exposure to security threats. A party that intercepts an agent, or agent message, in transit can attempt to copy the agent, or agent message, and clone or retransmit it. The interceptor may copy and replay an agent message or a complete agent.

III. COUNTERMEASURES OF SOFTWARE AGENT

There are two main broad categories of software agent hardware based solution and software based solution

Hardware Based Solution

In this case agent execution is only possible if a special peripheral exists. The agent test the existence of the peripheral on start or while executing the agent In this scheme the agent can be tampered if we delete (jump) the test of peripheral from the agent code [16].

Trusted Processing environment

This approach is proposed by Wilhelm and consists in the total execution of the agent inside a special peripheral called TPE (Trusted Processing Environment). The agent communicates with the visited site through a secure logical interface and migrates from one TPE to another in encrypted asymmetric form. [17]

Smart Card

The idea consists in the segmentation of the agent; some segments will be encrypted with the public key of the card. While executing the agent the site transmits the encrypted segment to the card where it will be decrypted and executed inside it [18].

Software Based Solution

The purpose of this class of approaches is to provide security of agents only with software mechanisms. It reduces costs as well as provides maintainable products. They are broadly categories as two

- protecting the agent platform
- protecting agents

Protecting the Agent Platform

Securing Agent platform developed techniques aimed at mobile code and mobile agent security have for the mostpart evolved along. More recently developed techniques aimed at mobile code and mobile agent security have for the most part evolved along these traditional lines. Techniques devised for protecting the agent platform include the following:

- Software-Based Fault Isolation,
- Safe Code Interpretation,
- Signed Code,
- State Appraisal,
- Path Histories, and
- Proof Carrying Code.

Software-Based Fault Isolation

Software-Based Fault Isolation [20], as its name implies, is a method of isolating application modules into distinct fault domains enforced by software. The technique allows untrusted programs written in an unsafe language, such as C, to be executed safely within the single virtual address space of an application. Untrusted machine interpretable code modules are transformed so that all memory accesses are confined to code and data segments within their fault domain. Access to system resources can also be controlled through a unique identifier associated with each domain. The technique, commonly referred to as sandboxing, is highly efficient compared with using hardware page tables to maintain separate address spaces for modules, when the modules communicate frequently among fault domains.

Safe Code Interpretation

Agent systems are often developed using an interpreted script or programming language. The main motivation for doing this is to support agent platforms on heterogeneous computer systems. Moreover, the higher conceptual level of abstraction provided by an interpretative environment can facilitate the development of the agent’s code [21]. Security is enforced through a variety of mechanisms. Static type checking in the form of byte code verification is used to check the safety of downloaded code. Some dynamic checking is also performed during runtime. A distinct name space is maintained for untrusted downloaded code, and linking of references between modules in different name spaces is restricted to public methods. A security manager mediates all accesses to system resources, serving in effect as a reference monitor. In addition, Java inherently supports code mobility, dynamic code downloading, digitally signed code, remote method invocation, object serialization, platform heterogeneity, and other features that make it an ideal foundation for agent development. There are many agent systems based on Java, including Aglets [22], Mole [23], Ajanta, and Voyager.

Signed Code

A fundamental technique for protecting an agent system is signing code or other objects with a digital signature. A digital signature serves as a means of confirming the authenticity of an object, its origin, and its integrity. Code signing involves public key cryptography, which relies on a pair of keys associated with an entity. One key is kept private by the entity and the other is made publicly available [24].

State Appraisal

The goal of State Appraisal is to ensure that an agent has not been somehow subverted due to alterations of its state information. The success of the technique relies on the extent to which harmful alterations to an agent's state can be predicted, and countermeasures, in the form of appraisal functions, can be prepared before using the agent. Appraisal functions are used to determine what privileges to grant an agent, based both on conditional factors and whether identified state invariants hold. A similar approach was suggested by Jansen and associates an X.509 [ISO9594-8] certificate to the agent. In which certificate the creator of the agent define the actions that his agent can do and for which he is responsible. In his approach, Jansen defines the attribute certificate in which the creator of an agent defines the behavior of his agent and policy certificate in which the site defines his security policy in front of visitor agents [24, 25].

Path Histories

The basic idea behind Path Histories [27] is to maintain an authentic table record of the prior platforms visited by an agent, so that a newly visited platform can determine whether to process the agent and what resource constraints to apply. Computing a path history requires each agent platform to add a signed entry to the path, indicating its identity and the identity of the next platform to be visited, and to supply the complete path history to the next platform.

Proof Carrying Code

The approach taken by Proof Carrying Code [28] obligates the code producer (e.g., the author of an agent) to formally prove that the program possesses safety properties previously stipulated by the code consumer (e.g., security policy of the agent platform). Proof Carrying Code is a prevention technique, while code signing is an authenticity and identification technique used to deter, but not prevent the execution of unsafe code

Protecting Agents

The agent-to-agent category represents the set of threats in which agents exploit security weaknesses of other agents or launch attacks against other agents. This set of threats includes masquerading, unauthorized access, denial of service and repudiation. Some more general purpose techniques for protecting an agent include the following:

- Partial Result Encapsulation,
- Mutual Itinerary Recording,
- Execution Tracing,
- Environmental Key Generation,
- Computing with Encrypted Functions, and

- Obfuscated Code (Time Limited Blackbox)
- *Cryptographic Function*

Partial Result Encapsulation

Often the amount of information gathered by an agent is rather small in comparison to the size of the encryption keys involved and the resulting cipher text. A solution called sliding encryption [30] allows small amounts of data to be encrypted and yield efficient sized results. Another method for agent to encapsulate result information is to use Partial Result Authentication Codes (PRAC) [31], which are cryptographic checksums formed using secret key cryptography (i.e., message authentication codes). This technique requires the agent and its originator to maintain or incrementally generate a list of secret keys used in the PRAC computation. The PRAC technique has a number of limitations. The most serious occurs when a malicious platform retains copies of the original keys or key generating functions of an agent.

Mutual Itinerary Recording

One interesting variation of Path Histories is a general scheme for allowing an agent's itinerary to be recorded and tracked by another cooperating agent and vice versa [30], in a mutually supportive arrangement. The scheme can be generalized to more than two cooperating agents. For some applications it is also possible for one of the agents to remain static at the home platform. Because the path records are maintained at the agent level, this technique can be incorporated into any appropriate application. Some drawbacks mentioned include the cost of setting up the authenticated channel and the inability of the peer to determine which of the two platforms is responsible if the agent is killed.

Execution Tracing

In this approach every site visited by the agent generates a trace of the agent execution .This trace contains every code line, every variable and every external variable read by the agent. Before the migration of the agent, they calculate the hash of the trace (with a hash function like SHA), sign that hash and associate the result to the agent. A light version of this solution consists in the agent code split in white segments and black segments. The trace is calculated only on the black segments where the agent interacts with the visited platform. A trace is composed of a sequence of statement identifiers and platform signature information. The approach has a number of drawbacks, the most obvious being the size and number of logs to be retained, and the fact that the detection process is triggered occasionally, based on suspicious results or other factors. Other more subtle problems identified include the lack of accommodating multi-threaded agents and dynamic optimization techniques. [32]

Environmental Key Generation

Environmental Key Generation [33] describes a scheme for allowing an agent to take predefined action when some environmental condition is true. The approach canters on constructing agents in such a way that upon encountering an environmental condition (e.g., string match in search), a key is generated, which is used to unlock some executable code cryptographically. The environmental condition is hidden through either a one- way hash or public key encryption of the environmental trigger.

Computing with Encrypted Functions

The goal of Computing with Encrypting Functions [33] is to determine a method whereby mobile code can safely compute cryptographic primitives, such as a digital signature, even though the code is executed in untrusted computing environments and operates autonomously without interactions with the home platform.

Obfuscated Code

Hohl [34] gives a detailed overview of the threats stemming from an agent encountering a malicious host as motivation for Blackbox Security. The strategy behind this technique is simple, scramble the code in such a way that no one is able to gain a complete understanding of its function (i.e., specification and data), or to modify the resulting code without detection. A serious problem with the general technique is that there is no known algorithm or approach for providing Blackbox protection.

Cryptographic Function

In the approach proposed by Sander and Tshudin [35], an agent A implements a function that will be encrypted by an algorithm E to obtain E (f). E (f) conceals details of A and will be implemented by a program P(E(f)) that is considered as a new agent B. The agent B migrates to the next site where it will be executed on a value x, thus we have P(E(f)) (x) that will return to the source site that execute the decryption algorithm E- 1(P(E(f))(x)). For a homomorphism function f the result is f(x).

The area of mobile agent security is still in a somewhat immature state. There are a number of applications for agents where conventional and recently introduced security techniques should prove adequate, until further progress can be made. Following are the topic for Future research in Software agent.

IV. Area For Further Research

Mobile Agent Security Framework

Agent system implementations incorporate appropriate security techniques; often little regard is given to interoperability among agent systems. What is needed is an overall framework that integrates compatible techniques into an effective security model and provides an umbrella under which interoperability can exist.

Mobile Agent Security Design Tool

Mobile agent application developers currently face a number of obstacles before they can efficiently design and develop large-scale mobile agent systems. Such as the lack of advanced development and modelling tools, the lack of mature agent standards, and the difficulty in optimizing performance under varying computational and communication loads. The selection of security mechanisms has a direct impact on agent migration, autonomy, disconnected operation, network latency, performance, and agent messaging. Mobile agent security design tools can help agent system developers determine the effects of employing various security mechanisms and make better decisions about functionality and performance tradeoffs.

Mobile Agent Security Autonomy

This topic refers to protection mechanisms for preserving the anonymity of a user on whose behalf an agent operates. Most of the work on this topic has been done in the context of messaging systems. Anonymity of not only an initiating agent, but also of any recipient agent or intermediaries may apply,

V. Conclusion

Security is a very large issue that is not easy to solve in the general case, and contrary to critics, based the analysis of mobile agent security in different environments, here show the main security problems as well as security threats in details. One of the most important things of enforcing security is to find the between the restriction and empowerment of mobile agents. In this paper, we have surveyed various security threats and there solution of agents and have thrown some light on the delicate areas that needs to be paid more attention to promote growth in optimistic direction.

References

1. Chira. 2003. Software Agents, IDIMS Report
2. D. C. Smith, A. Cypher and J. Spohrer (1994) “Programming Agents without a programming language” Communications of the ACM 37 (7) pp 55-67.
3. P. C. Janca (1995) “Pragmatic Application of Information Agents: BIS Strategic Decisions.
4. T. Selker (1994) “A Teaching Agent that learns” Communications of the ACM 37 (7) pp 92-99. D. C
5. Upadhye-Sakarkar “A Survey of Software Agent and Ontology” ©2010 International Journal of Computer Applications (0975 – 8887) Volume 1 – No. 7
6. M. Woodridge and N. Jennings, “Intelligent Agents: Theory and Practice”, The Knowledge Engineering R eview, 10(2):114-152, June 1995.

7. S. Franklin, A. Graesser, "Is it an Agent, or just a Program?: A Taxonomy for Autonomous Agents", University of Memphis, Proceedings of the Third International Workshop on Agent Theories, Architectures, and Languages, Springer-Verlag,1996
8. Sakarkar-Shelke ―A New classification Scheme for Autonomous Software Agent‖, IAMA'09,2009, IEEE Int. Conf. , India
9. upadhye-khot, "Optimize Security solution for mobile agent security: A Review" International Journal Of Engineering And Computer Science, ISSN:2319-7242 Volume 2 Issue 1, Jan 2013 Page No. 322-329
10. W. Jansen and T. Karygiannis, "Mobile Agent Security", NIST Special Publication 800-19-, 2000.National Institute of Standards and Technology.
11. A. Fuggetta, G.P. Picco, and G. Vigna,"Understanding Code Mobility," IEEE Transactions onSoftware Engineering, 24(5), May 1998http://www.cs.ucsb.edu/~vigna/listpub.html
12. "Agent Management," FIPA 1997 Specification, part1, version 2.0, Foundation for Intelligent Physical Agents, October 1998. http://www.fipa.ord/spec/fipa97/fipa97.html
13. "Mobile Agent System Interoperability Facilities Specification," Object Management Group (OMG) Technical Committee (TC) Document /97-10-05,November 1997.
14. P. Dadhich, Dr. K. Dutta, and Prof.(Dr.) M.C. Govil, "Security Issues in Mobile Agents", International Journal of Computer Applications(0975-8887), Volume 11-No.4, December 2010.
15. D.M. Chess, "Security issues in mobile code systems. In : mobile agents and security", Editor Vigna, vol. LNCS1419. Springer-Verlag 1998.
16. Fritz Hohl, "An approach to solve the problem of malicious hosts",http:// www.informatik.uni-stuttgart.de
17. http://lsewww.epfl.ch/~wilhelm/thesis/
18. AntonioMaña, Ernesto Pimentel, "An efficient software protection scheme", University of Málaga, Spain, 2002.
19. R. Wahbe, S. Lucco, T. Anderson, "Efficient Software-Based Fault Isolation," Proceedings of the Fourteenth ACM Symposium on Operating Systems Principles, ACM SIGOPS Operating Systems Review, pp. 203-216, December 1993.http://www.cs.duke.edu/~chase/vmsem/readings.html
20. John K. Ousterhout, "Scripting: Higher-Level Programming for the 21st Century," IEEE Computer, March 1998, pp. 23-30
21. James Gosling and Henry McGilton, "The Java Language Environment: A White Paper," Sun Microsystems, May 1996. <URL: http://java.sun.cm/docs/white/langenv/>
22. Markus Straßer, Joachim Baumann, Fritz Hohl, "Mole - A Java Based Mobile Agent System," in M. Mühlhäuser (ed.), Special Issues in Object Oriented Programming, Verlag, 1997, pp. 301-308.
23. Joseph Tardo and Luis Valente, "Mobile Agent Security and Telescript," Proceedings of IEEE COMPCON '96, Santa Clara, California, pp. 58 63, February 1996, IEEE Computer Society Press.
24. Wayne Jansen, Determining Privileges of Mobile Agents, Proceedings of the Computer Security Applications Conference, December 2001.
25. Wayne Jansen, A Privilege Management Scheme for Mobile Agent Systems, First International Workshop on Security of Mobile Multi agent Systems, Autonomous Agents Conference, May 2001. Wayne Jansen.
26. Joann J. Ordille, "When Agents Roam, Who Can You Trust?" Proceedings of the First Conference on Emerging Technologies and Applications in Communications, Portland, Oregon, May 1996.
27. G. Necula and P. Lee, "Safe Kernel Extensions Without Run-Time Checking," Proceedings of the 2nd Symposium on Operating System Design and Implementation (OSDI '96), Seattle, Washington, October 1996, pp. 229-243.
28. A. Young and M. Yung, "Sliding Encryption: A Cryptographic Tool for Mobile Agents," Proceedings of the 4th International Workshop on Fast Software Encryption, FSE '97, January 1997.
29. Bennet S. Yee, "A Sanctuary for Mobile Agents," Technical Report CS97-537, University of California in San Diego, April 28, 1997.
30. http://www-cse.ucsd.edu/users/bsy/index.html
31. Giovanni Vigna, "Protecting Mobile Agents Through Tracing," Proceedings of the 3rd ECOOP Workshop on Mobile Object Systems, , Finland, June 1997.http://www.cs.ucsb.edu/~vigna/listpub.html
32. James Riordan and Bruce Schneier, "Environmental Key Generation Towards Clueless Agents," G. Vinga(Ed.), Mobile Agents and Security, Springer-Verlag, Lecture Notes in Computer Science No. 1419, 1998.
33. Fritz Hohl, "Time Limited Blackbox Security: Protecting Mobile Agents From Malicious Hosts," G. Vinga (Ed.), Mobile Agents and Security, pp. 92-113, Springer-Verlag, Lecture Notes in Computer Science No. 1419, 1998.
34. Thomas Sander and Christian Tschudin, "Protecting Mobile Agents Against Malicious Hosts," in G. Vinga (Ed.), Mobile Agents and Security, Springer-Verlag, Lecture Notes in Computer Science No. 1419, 1998. http//www.icsi.berkeley.edu/~tschudin/

AUTHOR(S) PROFILE

Sachin Upadhye, received MCA degree from G.H. Raisoni Institute of Information Technology, Rashtrasant Tukdoji Maharaj Nagpur University, Nagpur, and pursuing PH.D (Computer Science) from Rashtrasant Tukdoji Maharaj Nagpur University, Nagpur.

Presently he is working as an Assistant professor in Computer Application Department, Ramdeobaba College of Engg. And Management, Nagpur.

Dr. P. G. Khot, is a Professor since 1999 at the Post Graduate Teaching Department of Statistics of the Rashtrasant Tukdoji Maharaj Nagpur University, Nagpur. He is an active researcher with over 40 publications in national and international journals of repute. He has delivered invited talks and has been on the organizing boards of many national and international conferences and Journals.