Security Infrastructure and Applications for Mobile Agents

Awais Shibli

Doctoral Dissertation in Computer and System Sciences Stockholm, Sweden 2010

Academic dissertation for the Degree of Doctor of Philosophy in Computer and System Sciences at Kungliga Tekniska Högskolan to be publicly defended on Thursday June 10, 2010 at 13:00 in Sal D, Forum, Isafjordsgatan 39 Kista.
Abstract

Research areas of this dissertation are security for mobile agents, for applications based on mobile agents, and for distributed network environments in which mobile agents execute. Mobile agents paradigm has captured researchers’ and industry’s interests long time ago because of its innovative capabilities and attractive applications. The ability of mobile agents to autonomously migrate from host to host, transferring their code and internal state, enables them to accomplish tasks in network and distributed environments more conveniently, robustly, and efficiently than traditional client-server applications. But, in spite of significant benefits of the mobile agent paradigm, the technology is still mainly in a research domain and so far it has not been adopted on a large scale by the industry and users. One of the reasons for that is security related issues and security concerns.

Current research in the area of mobile agents’ security is focused mainly on protection and security of agents and agents’ runtime platforms. But most of the currently available mobile agent systems do not support comprehensive security requirements for a general mobile agents paradigm. Therefore, there is a need for a complete and comprehensive security infrastructure for mobile agents, not only in the form of security services and mechanisms for agents’ runtime execution, but also as a complete set of infrastructural components, along with methodology for creation, classification, adoption, and validation of mobile agents before their deployment in real-environments. In addition, protection of mobile agents’ code and their baggage during execution is also needed. The lack of such concept, infrastructure and security solutions is hindrance for wider adoption of mobile agent systems at the time of this research.

In our research, we solve these comprehensive requirements with solutions that can be classified in two groups:

The first group is solutions for designing, implementation and deployment of a security infrastructure for mobile agents, along with methodology for secure deployment and execution of mobile agents. The proposed infrastructure for mobile agents is based on a methodology for creation, classification and validation of trusted mobile agents. It includes security architecture for publishing, discovery and adoption of mobile agents. Moreover, it provides integrated system for mobile agent deployment that supports launching, authorization and execution of mobile agents. Mobile agents execution is based on a protective approach, as compared to traditional detective or preventive methods, that not only provides code protection, but code execution and data privacy as well.

The second group is solutions for use of security infrastructure and, in particular, secure and trusted mobile agents for real-life applications. The main result in this group is the design and implementation of a network intrusion detection and prevention system based on mobile agents. The system efficiently solves several problems of existing IDS/IPS. It can detect new vulnerabilities before they are exploited by hackers, it can process and filter large volumes of log entries, it reacts to intrusions in real–time, it provides protection against unknown attacks, it supports and improves commercial IDS/IPS products, and it also efficiently handles software patches. The system not only improves use of existing popular IDS/IPS, but it also eliminates several of their core problems. In addition, it is self–protected by full encryption, both of mobile agents and their execution platforms, and therefore not vulnerable to attacks against its own components and resources.

Keywords: Mobile Agents Security, Access Control, Network Security, Trusted Mobile Agents
Acknowledgements

I am grateful to Allah Almighty for giving me the courage, determination as well as guidance in accomplishing this milestone in my life. I am really thankful for all the blessings bestowed upon me.

I would like to express my special gratitude to Professor Sead Muftic for providing me the opportunity to work with him. I could have never been able to achieve my goals without his intellectual support and enthusiasm. He has been a source of constant motivation for me and will remain in the future as well. I would also like to extend my heartfelt gratitude to Professor Arshad Ali for trusting in my abilities and providing me the opportunity of studying at KTH.

I wish to extend my utmost gratitude to everyone at Sec Lab for providing conducive environment for research and development. Finally I would like to thank my friends Ammar, Abdul Ghafoor, Mohsin, Qaiser, Ezhar and Haseeb for making my stay wonderful and memorable at Stockholm.

Thank you all!

Stockholm June 2010

Awais Shibli
I would like to dedicate my life and my work to my parents. I do not have words to express my gratitude to my parents. Throughout my life they provided me with every comfort despite of a lot of constraints. My mother was always there praying for my success. Throughout my educational career she provided me with every possible support and comfort at home so that I can study. Whenever I got de-motivated and tensed she consoled and encouraged me. My father has always been a source of inspiration for me. He encouraged me at every step of my educational career and made me believe in my abilities. In his presence I never had to bother about anything because he was always there to take care of everything. I consider myself very fortunate to have such loving and supportive parents. I know I can never thank my parents enough for all the sacrifices they made for me. I pray that may Allah Almighty give them a healthy long life so that they can see their dreams coming true. Ameen.
# Table of Contents

## PART I: INTRODUCTION

1. Overview of the Approaches, Results and Achievements 3
   1.1. Introduction 3
   1.2. Security Aspects of Mobile Agents 4
      1.2.1. Security Infrastructure for Mobile Agents 4
      1.2.2. Security of the Mobile Agent Systems 7
      1.2.3. Network Security System based on Mobile Agents 7
   1.3. Research Method 9
   1.4. Specific Contributions and Achieved Research Results 9
   1.5. Validation of Results 13
2. Related Research 15
   2.1. Overview and Analysis of the Related Work 15
      2.1.1. Security Infrastructure for Mobile Agents 15
      2.1.2. Security for Mobile Agents Systems 20
      2.1.3. Comprehensive Network Security System based on Mobile Agents 23
   2.2. Thesis Contributions and Research Problems Solved 25

## PART 2: SECURITY ARCHITECTURE FOR CREATION, CLASSIFICATION AND VALIDATION OF MOBILE AGENTS

3. MagicNET System 31
   3.1. MagicNET System 31
      3.1.1. Phases of MagicNET System 31
      3.1.2. Agent’s Life Cycle of Development and Execution 32
      3.1.3. Research Problems addressed for each Phase 33
4. Creation and Validation of Mobile Agents 37
   4.1. Creation and Validation of Trusted Mobile Agents 37
   4.2. Specification of Assurance Levels for Mobile Agents 38
   4.3. Creation and Validation Phase 39
      4.3.1. Roles for Creation and Validation of Agents 39
      4.3.2. Components for Creation and Validation of Agents 40
      4.3.3. Creation and Validation Procedures 40
5. Adoption of Trusted Mobile Agents 43
   5.1. Adoption of Trusted Mobile Agents 43
      5.1.1. Roles in the Adoption Process 43
      5.1.2. Components for Adoption of Mobile Agents 44
      5.1.3. Procedures for Adoption 44

## PART 3: SECURITY SYSTEM FOR PROTECTION OF MOBILE AGENTS

   6.1. Authorization of Mobile Agents 51
   6.2. XACML Policies for Mobile Agents 51
      6.2.1. The Structure of a Policy 52
   6.3. Mobile Agents Deployment Phase 56
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.3.1. Components in the Deployment Phase</td>
<td>56</td>
</tr>
<tr>
<td>6.3.2. Operations in The Deployment Phase</td>
<td>57</td>
</tr>
<tr>
<td>7. Protection of Mobile Agent’s Code and Baggage</td>
<td>61</td>
</tr>
<tr>
<td>7.1. Mobile Agents Protection</td>
<td>61</td>
</tr>
<tr>
<td>7.2. Mobile Agents Execution Phase</td>
<td>61</td>
</tr>
<tr>
<td>7.2.1. Components in the Execution Phase</td>
<td>62</td>
</tr>
<tr>
<td>7.2.2. Operations in The Execution Phase</td>
<td>66</td>
</tr>
<tr>
<td>7.2.3. Secure Communication between Mobile Agents</td>
<td>74</td>
</tr>
<tr>
<td><strong>PART 4:</strong> NETWORK SECURITY SYSTEM BASED ON MOBILE AGENTS</td>
<td></td>
</tr>
<tr>
<td>8. Network Security System</td>
<td>79</td>
</tr>
<tr>
<td>8.1. Network Security Problems</td>
<td>79</td>
</tr>
<tr>
<td>8.1.1. Intrusions and Damages</td>
<td>79</td>
</tr>
<tr>
<td>8.1.2. Current Solutions and TheirWeaknesses</td>
<td>80</td>
</tr>
<tr>
<td>8.2. Network Security System based on Mobile Agents</td>
<td>80</td>
</tr>
<tr>
<td>8.2.1. The Functions of The Proposed Solution</td>
<td>82</td>
</tr>
<tr>
<td>8.2.2. The Components of the System</td>
<td>83</td>
</tr>
<tr>
<td>8.2.3. Operations of the System</td>
<td>86</td>
</tr>
<tr>
<td>8.3. Conclusions</td>
<td>94</td>
</tr>
<tr>
<td><strong>PART V:</strong> CONCLUSIONS</td>
<td></td>
</tr>
<tr>
<td>9. Validation of The System Design</td>
<td>97</td>
</tr>
<tr>
<td>9.1. Validation Methodology</td>
<td>97</td>
</tr>
<tr>
<td>9.1.1. Qualitative Properties</td>
<td>98</td>
</tr>
<tr>
<td>9.1.2. Quantitative Properties</td>
<td>98</td>
</tr>
<tr>
<td>9.2. Development Results</td>
<td>100</td>
</tr>
<tr>
<td>9.2.1. Interfaces at Creation and Validation Phase</td>
<td>106</td>
</tr>
<tr>
<td>9.2.2. Interfaces for the Adoption and Deployment Phase</td>
<td>115</td>
</tr>
<tr>
<td>9.3. Validation Methodology Conclusions</td>
<td>122</td>
</tr>
<tr>
<td>10. Conclusion and Future Research</td>
<td>123</td>
</tr>
<tr>
<td>10.1. Conclusion</td>
<td>123</td>
</tr>
<tr>
<td>10.2. Future Research</td>
<td>124</td>
</tr>
<tr>
<td>References</td>
<td>125</td>
</tr>
<tr>
<td>Appendix</td>
<td></td>
</tr>
<tr>
<td>Appendix A: Abbreviations and Conventions</td>
<td>131</td>
</tr>
<tr>
<td>Appendix B: Snort</td>
<td>133</td>
</tr>
<tr>
<td>Appendix C: Web Services Descriptions at Agent Factory</td>
<td>137</td>
</tr>
<tr>
<td>Appendix D: Sample Agent Data Access Control Policy</td>
<td>141</td>
</tr>
<tr>
<td>Appendix E: List of Figures</td>
<td>143</td>
</tr>
</tbody>
</table>
Part 1 provides general introduction to thesis objectives. Chapter 1 contains overview of approaches, results, achievements and research methods. Chapter 2 provides the overview of the related work in the area.
1 Overview of the Approaches, Results and Achievements

1.1 Introduction

Mobile agents are a recent and hopefully better network computing paradigm compared with the currently used approach to remote computing, based on a client–server model. Mobile agents are self-contained software modules and data that can be launched by a human user and then, they autonomously migrate through the network. They visit remote hosts, perform there their tasks, migrate to the next host, eventually returning to the management station, where their actions were initiated.

In order to perform complex network tasks addressed in this research, mobile agents must be organized in the form of teams and their actions must be supported by various specialized components in the networking environment. Such approach represents a complex Mobile Agents System (MAS), comprising several different types of components. These components perform their functionality and, combined together, they represent the complete infrastructure for management and execution of mobile agents.

It is expected that deployment of mobile agents, as the new computing paradigm, will show several advantages compared to the current network computing principles based on client–server architectures. The most relevant for this research are:

- their mobility, therefore suitable for instantaneous local reactions to various events relevant for security, and
- their autonomous operations, therefore convenient for automated administration, distribution and synchronization of various aspects of security in distributed, heterogeneous networking environments.

Because of these two properties, it is expected that network security system based on mobile agents will be better and more effective than security technologies used in current networks. Besides basic capabilities, to travel through the networks and perform their individual tasks, agents can also perform very complex and sophisticated functions. They do that through three processes: collaboration, cloning, and learning.

An active agent in the MAS collaborates with other agents in order to accomplish its tasks. This collaboration may be with static agents, located at the hosting servers, with members of agent’s own team, or remotely with other independent agents, located at other agent servers. In case of collaboration
with remote agents they communicate using some agent communication language (ACL). Agents capable of understanding ACL exchange messages based on the standardized syntax and semantics of those messages.

Another important feature of mobile agents is that they can reproduce themselves. There is a difference between agent collaboration and agent reproduction. In case of collaboration, two or more mobile agents from the same or different platform divide certain task, execute portions assigned to them, and finally merge their individual results into a single result. However, in case of agent reproduction, an agent (parent agent) reproduces a child during its execution life cycle and assigns to it some small specific tasks to execute. Parent agent may or may not receive results from a child agent. In principle, child agent’s task is limited only to some specific small purpose.

Besides being mobile, capable to communicate and clone, agents can also learn by accumulating information, knowledge, and experience, and thus they may be “intelligent”. This process can be performed, at one extreme, through simple accumulation of information during their execution steps, all the way up to the most complex approach, using methods of artificial intelligence. Such “smart” agents are software agents with the element of “intelligence” demonstrated by their independent and autonomous creation and enforcement of various decisions. Agents may also acquire intelligence by accepting users' instructions, assertions and options for “intelligent” execution of tasks assigned to them.

Over the last decade usage of mobile agents technology was relatively slow, compared, for instance, to web or client-server computing model. At the first glance, mobile agents seem very attractive due to their characteristics, mainly mobility, i.e. the ability to autonomously move from one network server to another. Based on their autonomous operations, mobile agents can be conveniently used for automated tasks, like network management, distributed retrieval and filtering of information, distributed access to Internet databases, etc. They are suitable for execution of asynchronous client requests and for dynamic adoption to various types of mobile wireless environments [31]. However, trust and security issues significantly reduce popularity and usefulness of mobile agent technologies [32]. Therefore, research community started focusing on the security requirements and solutions for mobile agents. The most popular topic is runtime security for mobile agents, i.e. how to provide confidentiality, integrity, availability and non–repudiation of mobile agents at remote hosts.

### 1.2 Security Aspects of Mobile Agents

Our research was focused to and organized in the following three major research areas: (a) security infrastructure for management and execution of mobile agents, (b) security of mobile agent systems, and (c) network security system based on mobile agents.

#### 1.2.1 Security Infrastructure for Mobile Agents

We have extended the concept of standard network security infrastructure with new research results specific to security infrastructure for mobile agents. The principles, functions and topology of standardized infrastructure for security services were known before this research was performed, based
on, for instance, PKI, secure XML/Web federation, various secure network protocols, and other security standards. Large-scale intrusion detection and prevention systems, based on secure mobile agents and various security services in a networking environment, could best be provided by such a network security infrastructure.

We have identified the following core components of our security infrastructure for mobile agents, classified in two groups:

**1.2.1.1 Components for Deployment and Execution of Mobile Agents**

**Mobile Agents:** The most important component of the MAS is mobile agents. Mobile agents are self-contained software modules with additional configuration and security credentials and accumulated data baggage. They roam a network, moving autonomously from one server to another, perform their designated tasks, and finally, eventually, return to their agents management station.

Besides their code, mobile agents also have
- associated credentials, comprising header and trailer information, that describe the type and other attributes of an agent,
- a baggage, that agents have created and accumulated during their activities at various hosts, and finally
- routing information for traversing the network.

**Mobile Agent Servers:** The second component of our infrastructure is Mobile Agent Servers (Runtime nodes). Those are network hosts that have the capability to accept, execute and launch further mobile agents. In order to become agent server, each network server needs, first, our mobile agents platform (software), i.e. a hosting environment for mobile agents. Mobile agents migrate from one host to another and execute at remote hosts using supporting functions, provided by mobile agents platform. Second, the servers provide some local security, configuration, and control data in order to support execution of mobile agents. In other words, agents’ servers represent a complete environment for all aspects of secure execution of mobile agents.

**Policy Enforcement Point (PEP):** Agent platforms mentioned in the previous section must be tightly coupled with Policy Enforcement Points (PEP). PEP has the same role in our infrastructure as defined in [1]. “PEP is the system entity that performs access control by making decision requests and by enforcing authorization decisions”. The role of PEP at runtime nodes is to protect sensitive resources from incoming mobile agents and visiting mobile agents’ baggage from agent platform.

**Management Station:** The third component of our infrastructure is Management Station. It is used by mobile agents’ administrator to perform two types of functions with mobile agents: administrative and management functions. Administrative functions include, among others, launching agents and receiving the results of their processing. Management Station is also called agents home, since agents are launched from that station and eventually return the results there. Management functions are related to mobile agents’ applications, e.g. view logs, perform updates etc.
Security Infrastructure Servers: The fourth component of our infrastructure is various Security Infrastructure Servers. The purpose of these servers is to provide supporting data, functions, and services to mobile agents. Infrastructure servers are:

- **IDMS Servers:** they keep registration data about agents, like agent’s owner information, agent’s author information, agent’s credentials, etc.;
- **Certificate Authority (CA) Servers:** That server issue, distribute and verify X.509 certificates for mobile agents, their authors, owners, servers, and other components of the MAS.
- **Policy Decision Point (PDP Server):** is the infrastructure component responsible for all security decisions concerning authentication and authorization in the mobile agent system. PDP performs verification and certification of agent’s code, verifies trustfulness of their authors and authorization of their owners, and stores and enforces authorization and access control policies. In this context of user authentication and authorization, PDP server acts as a credential collector and asserting entity that issues SAML tickets to agents.
- **Policy Administration Point (PAP) Station:** In our infrastructure Policy Administration Point Station is the component used by mobile agents’ administrators to create various XACML policies for agents, users and agent platforms. There are different policy types that PAP manages: agent policies, describing agent authorization roles and used by PDP at a runtime environment to decide whether or not an agent is authorized to access a resource. User policies, describing users’ authorization to adopt an agent i.e. which user can adopt which agent. Platform policies, describing platforms’ authorization to read or append data to a visiting mobile agents’ baggage.

Security infrastructure for mobile agents system also provides facilities for mobile agents to collaborate with each other in order to execute security and administrative functions in the network. This means that we have also addressed security problems for agents’ collaboration. In order words, our infrastructure also provides strong security for agents’ communication messages.

1.2.1.2 Components for Creation, Validation, Publishing, Discovery, and Adoption of Mobile Agents

In addition to these general purpose infrastructure components, we have designed several additional components of the security infrastructure for mobile agents.

**Agent Factory Server:** Agent Factory Server is the infrastructure component used as an incubator and repository of trusted agents. Life cycle of agents’ development includes many steps, starting from agent creation and ending with its adoption. During this process, agents pass through a number of steps. Agent Factory acts as an intermediate server during these steps, contains intermediate and final evaluated agents, ready for adoption by the end-users.
**UDDI Server:** UDDI Server\(^1\) is based on the standard concept of UDDI servers, as specified by OASIS [2], i.e. web services publishing and discovery. In our infrastructure, UDDI Server acts as a registry containing a special category of services: agent provision web services. Any business entity (in our case Agent Factory) that wants to provide agents to the community (i.e. end users) must properly expose agents as a web service interface and correctly publish this web service interface. Users and network processes, using UDDI Server, may locate mobile agents, query their functionality, and download them into their local workstations for subsequent execution or may also deposit newly created agents.

**Key Distribution Server (KDS):** KDS in our infrastructure is used to achieve agent data baggage protection from agent platform i.e. only authorized platforms can see data contributed by itself and other platforms. KDS handles mainly functions like creation of random symmetric key and returning a symmetric key assigned to a particular runtime node during mobile agent execution.

### 1.2.2 Security of the Mobile Agent Systems

Since mobile agents roam and execute through an entire network, it is important to provide strong protection of agents and all their resources. At the same time, it is equally important to protect agent’s platforms against malicious agents.

We have applied various standard security mechanisms and services in order to protect mobile agents, their baggage, communication messages, control structures and platforms against various accidental and intentional threats. In that context, security services for secure computing and secure handling of data applied to mobile agents systems have also been addressed. Our system provides mobile agents system security services, such as confidentiality and integrity of resources, access control to resources, authentication and authorization of users and other active components, protection and non–repudiation of transactions, etc.

Therefore, our mobile agents system is very secure and all its components and resources are strongly protected. We provide protection of mobile agents’ code and data using cryptography. Our system provides also an effective solution to perform computing with the encrypted code of mobile agents. This feature provides protection against reverse engineering and allows preservation of code correctness in spite of malware, worms, or viruses.

### 1.2.3 Network Security System based on Mobile Agents

The last goal of this research and one of the major expected contributions to the current state-of-the-art of network security is the design of a network security system based on mobile agents. The innovative approach in this research was to establish such a system based on mobile agents. Precisely, new security

---

\(^1\) UDDI Server has been implemented using an open source Java project called JUDDI [3], which follows OASIS specification of UDDI for web services [2]. Moreover, it has been integrated into the WSO2 WSAS secure application server, so that Agent Registrar (publisher) can use HTTPS SSL/TLS for authentication.
system is using mobile agents to protect network resources and also to provide security services to various network applications, functions and resources.

When considering security in a networking environment based on mobile agents, we treat network security system and mobile agents system as two logically separate, but mutually related systems. For network security, mobile agents have been used to achieve effective protection of network resources. This has been achieved by designing an innovative and effective Intrusion Detection and Prevention (IDP) system using mobile agents.

Due to the two already mentioned characteristics of mobile agents (mobility and autonomous operations), we expect our new network security system to be simpler to manage and more effective in its operations than other current security solutions for network environments. There are three main functions of network security system:

**1.2.3.1 Vulnerabilities Analysis and Patches Management**

The first function of the system is to assist network and system administrators to analyze different vulnerabilities in the installed IT components of the network. For this purpose, a comprehensive vulnerability database has been created using different open access vulnerabilities databases. Afterwards mobile agents are used to analyze different hosts in the network in order to identify different vulnerabilities on them by querying our integrated vulnerability database against installed software profiles. Based on the results of vulnerability analysis, the system can apply available patches to vulnerable hosts.

**1.2.3.2 Intrusions Detection and Response**

The second function of the system is to detect malicious activities in the network with the help of mobile agents. For this purpose mobile agents provide three groups of functions: (a) analysis of large volume of data in various logs generated at remote hosts and return effective reports, (b) detection of and reaction to host-based intrusion attempts in real time, and (c) detection of and reaction in real time to distributed intrusion attempts.

**1.2.3.3 Network Security Management**

The third function of the system is to assist network administrators by keeping the network up-to-date against new potential threats. For this function a team of mobile agents is launched, which roam the network, visit different hosts and analyze and install different services or security software. For this purpose mobile agents use the following techniques: (a) connectivity and status of remote hosts are checked and reported; (b) configuration of remote hosts are checked and recorded; (c) security configuration management related tasks are applied; (d) mapping of SNORT rules and identified vulnerabilities.
1.3 Research Method

Our research method is based on positivistic philosophical assumption. According to Michael D Myers “Positivist researchers typically formulate propositions that portray the subject matter in terms of independent variables, dependent variables, and the relationships between them”. The main aspect of positivist research is formulation of research problems in terms of dependent and independent variables and relationships between them. Therefore, we have identified different variables for mobile agents’ security, from NIST’s special publication “Mobile Agent Security” [43]. It has identified four types of attacks. We refer “agent to platform” attacks as “Type I” attacks, “platform to agent” attacks as “Type II” attacks, “agent-to-agent” attacks as “Type III” attacks and “external entities to agent system” attacks as “Type IV” attacks. These threats are further explained in the context of masquerading, unauthorized access, denial of service, repudiation, alteration, copy and reply, and eavesdropping.

Now, after the identification of variables, the next step was to understand “mobile agents’ security”, especially in context of validity, scope and limits of it. We have opted to do in depth literature review of existing work to identify mobile agents security areas covered and those which are not covered. NIST’s report was published in 1998 and there have been reasonable amount of work done in the last 12 years. Based on literature survey, we have identified potential research areas not covered so far in mobile agents’ security, and therefore contribute towards negative trends for adoption of mobile-agents-based applications in the real world on a large scale. We propose innovative solutions (design and prototype implementation) for those uncovered areas and evaluate our design with respect to the variables identified in the previous paragraph.

1.4 Specific Contributions and Achieved Research Results

As conclusions, we expect our new security system to be an important contribution to an alternative, more effective, network computing paradigm based on secure mobile agents and also significant improvement of the current state–of–the–art of network security.

In this research we have solved the following problems and we have created the following contributions to the area of security of mobile agents

1.4.1 In our paper entitled “MagicNET: Security Architecture for Creation, Classification, and Validation of Trusted Mobile Agents”, authors Awais Shibli and Sead Muftic, published in The Proceedings of the 11th IEEE International Conference on Advanced Communication Technology Phoenix Park, Korea, February 15-18, 2009, we have described security architecture for creation, classification, and validation of trusted mobile agents. The Motivation behind this architecture is that most of the current research and development results, dealing with security of mobile agents, describe solutions only for usage of mobile agents. These contributions usually assume that agents possess unique and recognizable identities, cryptographic keys, assigned assurance level, and various other security parameters. But, very few research contributions describe how to create, classify, and evaluate mobile agents before their adoption and deployment. These are the issues we have addressed in the referenced
paper: how mobile agents are created, validated, tested, and classified before their deployment. Based on certain security parameters, we have established classification scheme for mobile agents structuring them into three assurance levels: low, medium, and high. The most important results are the definition of assurance levels for mobile agents and procedures for their classification into those assurance levels.

1.4.2 In our paper entitled “MagicNET: Security Architecture for Discovery and Adoption of Mobile Agents”, authors Awais Shibli and Sead Muftic, published in The Proceedings of the 1st International Workshop on Security and Communication Networks, Trondheim, Norway, May 20-22, 2009, we have extended the architecture described in our previous paper with discovery and adoption processes for mobile agents. In particular, this paper addressed the process of acquiring mobile agents by their owners. This is also another important issue. Current research in the area of security for mobile agents’ deals mainly with the runtime issues of agents’ protection. Mobile agent systems do not address precisely the process of adopting mobile agents by their owners. They assume that the agents are somehow already available for use. This assumption is acceptable for experimental or prototyping environments, but it is inadequate for the real world scenarios, where agents should be trusted and reliable, but agent creators and agent owners are separated and manage agents from mutually remote locations. Thus, the issue of agents’ adoption for use in serious, sensitive and business networks is very important, if agents are used in real–life applications. In this paper we have described an abstract architecture and procedures for secure, verifiable and authenticated discovery and adoption of mobile agents. The main contribution is that in this process agent’s code and its functionality can be verified, so that such agents can be deployed in serious applications and scenarios.

1.4.3 In our paper entitled “MagicNET: Security Architecture for Authorization of Mobile Agents”, authors Awais Shibli and Sead Muftic, published in The Proceedings of the 3rd International Conference on Internet Technologies and Applications, Wrexham, North Wales, UK, September 8-11, 2009, we have proposed security architecture for authorization of mobile agents. After mobile agents’ system deployment, controlling access to resources at remote hosts by mobile agents during their execution is a challenging problem. Current solutions use mainly methodology that detects agents’ incorrect access attempts to a particular resource. For that purpose agents’ execution logs are checked in order to identify malicious activities or misuse of resources at a particular host. As an alternative, instead of detective approach, we focused on preventive approach for the control of access by mobile agents. While detective approach may provide some protection, its primary shortcoming is that it does not provide protection of resources in advance, before access, and in fact requires post–fact manual intervention and activation of countermeasures. With our solution we provide a solution to authenticate and authorize agents at remote hosts before they execute any action.

1.4.4 In our paper entitled “MagicNET: Security System for Development, Validation and Adoption of Mobile Agents”, authors Awais Shibli, Sead Muftic, Alessandro Giambruno and Antonio Lioy, published in the Proceedings of The 3rd IEEE International Conference on Network and System Security, Golden Coast, Australia, October 19-21, 2009, we have described security system for deployment of mobile agents. Mobile agent systems usually do not provide an extensive security methodology for an entire agent’s life cycle, from agent’s creation to its deployment and execution. In this paper we have described a secure system for deployment of mobile agents. The system provides methodology that spans
a number of phases in agent’s lifetime: it starts from agent’s creation and ends with agent’s execution. It addresses classification, validation, publishing, discovery, adoption, authentication and authorization of mobile agents.

1.4.5 In our paper entitled “MagicNET: XACML Authorization Policies for Mobile Agents”, authors Alessandro Giambruno, Awais Shibli, Sead Muftic and Antonio Lioy, published in The Proceedings of the 4th IEEE International Conference for Internet Technology and Secure Transactions, November 9-12, 2009, London, UK, we have designed the structure of authorization policies for mobile agents. In our paper entitled “MagicNET: Security Architecture for Authorization of Mobile Agents” we have identified system components and data flow model for mobile agents’ authorization. In this paper we have shown how traditional XACML polices, used for user access control in distributed environments, can be used for mobile agents’ access control. We have used such policies to manage delegation of access rights from users to agents while at the same time following the core principles of the XACML standards. We have also proposed a combination of policies that map users to their mobile agents and make access control decisions for mobile agents by evaluating complex policy sets.

1.4.6 In our paper entitled “MagicNET: Security System for Protection of Mobile Agents”, authors Awais Shibli, Imran Yousaf and Sead Muftic, published in The Proceedings of The 24th IEEE International Conference on Advanced Information Networking and Applications, Perth, Australia, 20-23 April 2010, we have addressed the problem of mobile agent protection. Once mobile agents get an authorization to execute on a remote host, the next important issue is mobile agents’ protection, which is difficult problem in the area of mobile agents’ security. There is not a single comprehensive solution, which provides complete protection of agents against malicious hosts. Existing solutions only either detect or prevent attacks on agent. Even in detective mechanisms, integrity of agent’s code/state is being checked, but there is nothing about agent’s code and baggage privacy. In this paper we have proposed an extension of our system, which not only provides code protection, but also provides code execution protection and data privacy as well. Design of the system is based on a protective technique, which provides better security compared to traditional detective or preventive methods.

1.4.7 In our paper entitled “MagicNET: Secure Communication Methodology for Mobile Agents”, authors Awais Shibli, Imran Yousaf, Kashif Dar and Sead Muftic, published in The Proceedings of The 12th IEEE International Conference on Advanced Communication Technology. Phoenix Park, Korea, February 7-10, 2010, the problem of secure communications between two mobile agents has been addressed. This is a novel contribution, because most of the current research and development results, dealing with authentication of mobile agents, describe solutions which address only agent-to-platform authentication. These solutions give full privileges to agents to execute and then communicate with other agents at the platform after initial authentication. They do not address further agent-to-agent communication security requirements. Moreover, further communications are not based on standard communication protocol, which increases the possibilities of false communications between benign and malicious agents. We have proposed additional agent-to-agent secure communication that guarantees authenticated, authorized and confidential communication between agents.
1.4.8 In our paper entitled “Intrusion Detection and Prevention System Using Secure Mobile Agents,” authors Awais Shibli and Sead Muftic, published in The Proceedings of The International Conference on Security and Cryptography, Porto, Portugal, July 2008, we have designed an application based on secure mobile agents. Although a number of mobile agent based applications are available in the market, we have in this paper described design and architecture of the Intrusion Detection and Prevention system based on secure mobile agents along with the analysis of commercial products and current research efforts in this area. The system efficiently solves several problems with existing IDS/IPS solutions: it can detect new vulnerabilities, it can process and filter large volumes of logs, it reacts to intrusions in real-time, it provides protection against unknown attacks, it supports and improves commercial IDS/IPS products by different vendors, and it also handles software patches. The system not only improves existing IDS/IPS solutions, but it also eliminates several of their core problems. In addition, it is self-protected by full encryption of both, mobile agents and their platforms, and therefore it is not vulnerable to attacks against its own components and resources.

1.4.9 In our paper entitled “Vulnerability Analysis and Patches Management using Secure Mobile Agents”, authors Pasquale Stirparo, Awais Shibli and Sead Muftic, published in The Proceedings of The 11th IEEE International Conference on Advanced Communication Technology. Phoenix Park, Korea, February 15-18, 2009, we have addressed in detail the issue of detecting existing vulnerabilities in the network using mobile agents. The main motivation behind it is there are many software applications being developed daily all over the world, but unfortunately those applications usually contain problems and vulnerabilities, because of poor programming practices or poor development strategy used by developers. Those vulnerabilities are exploited by hackers for their malicious intents. In order to eliminate this problem we have designed and developed a system for comprehensive analysis of vulnerabilities and management of patches. Our system a) autonomously collects the most current information about vulnerabilities, b) analyzes hosts in a local network for potential vulnerabilities, and finally, c) autonomously applies appropriate patches.

1.4.10 In our paper entitled “Building Secure Systems Using Mobile Agents”, authors Awais Shibli and Sead Muftic, published in The Proceedings of The 4th International Workshop on Multi-Agent Systems and Semantic Grid, Rawalpindi, Pakistan, December 16, 2006, we have extended our previous system for intrusion detection and response with security management. Motivation behind IDP system was that several emerged solutions like antivirus, firewall, spyware, and authentication mechanisms provide security to some extent, but they still face the challenge of inherent system flaws, OS bugs, and social engineering attacks. Traditional Intrusion Detection and Prevention systems have the problem of detecting and responding to intrusions in a real time. Increasing number of penetrations, Internet crimes cases, identity and financial thefts are an inefficient solution to the overall problem, because they use reactive approach in case of security breaches and apply local remedies (firewall, antivirus and intrusion detection systems) – “point solutions”. Mobile agent based strategy operates in reactive and proactive mode and provides security based on the principle of defense in depth. This strategy includes three aspects: (a) autonomously detect vulnerabilities at different hosts (in a distributed network) before an attacker can exploit them (b) protect hosts by detecting attempts of intrusions and responding to them in real time; and finally (c) perform tasks related to security management.
1.4.11 In our paper entitled “MagicNET: The Human Immune System and Network Security System”, authors Awais Shibli, Jeffy Mwakalinga and Sead Muftic, published in the IJCSNS International Journal of Computer Science and Network Security, Vol.9 No.1, January 2009, we describe extensions of our previous three papers with additional subsystems to cater pre-intrusion and post-intrusion vulnerability analysis, distributed intrusion detection and response capability, and enhanced IDP system management.

1.4.12 In our paper entitled “Security Architecture and Methodology for Authorization of Mobile Agents”, authors Awais Shibli, Alessandro Giambruno and Sead Muftic, accepted for publication in the International Journal for Internet Technology and Secured Transactions (IJITST), Vol. 5, ISSN 1748-569X (Print), ISSN: 1748 - 5703 (Online). We presented a solution for authentication and authorization of mobile agents based on RBAC and XACML policies. This solution can be applied in two steps: a) by creating security architecture for authorization of mobile agents (authorization system), and b) by specification of the structure/model for role-based XACML policies for mobile agents. These two steps combined provide a complete authorization system for mobile agents.

1.5 Validation of Results

A basic goal in designing a secure system is to not have any weakness or vulnerabilities that could be exploited by a potential attacker.

The second aspect is that all security requirements should be fulfilled. NIST’s publication on mobile agent security [43] provides the basis for mobile agents’ security research. They have identified many potential security threats and security requirements for different entities in a mobile agents’ paradigm. They had mentioned two basic entities i.e. mobile agents and agent platform. Agent platform provides execution environment for mobile agents. They have identified security requirements with respect to confidentiality, integrity, accountability, availability and anonymity for mobile agents and agent platforms. In our approach we have extended the basic agent model (shown in figure 1.1.) with additional entities, components and security services.
In our solution we have addressed all those security requirements mentioned above by using different security mechanisms. Our proposed solution is evaluated against each threat category mentioned by NIST. We have shown that all new entities and components of our system are secure and all security requirements (confidentiality, integrity, availability and accountability) are fulfilled.

Second, our solution is based on existing proven security technologies. We have used different standards/technologies/mechanism, like PKCS7, XACML, GSAXMP, Apache Rampart, Single Sign-On, SAML and FIPS-196 mutual authentication. The use of proven technologies has strengthened the security of our system.
2.1 Overview and Analysis of the Related Work

In order to focus and clearly specify the scope and boundaries of our research, in this chapter we describe the most significant problems and current related research contributions, and suggested solutions for those problems. The chapter is based entirely on an overview of large number of research papers, which are quoted in individual sections and also listed at the end of this thesis. The intention of the classification and grouping of various topics is to structure them as smaller, uniform and autonomous areas. Of course, all of them are mutually related and jointly they represent the complete picture of research activities in the area of security for mobile agents. All results are quoted directly and analyzed from the available literature.

Current activities, reported results, important issues and remaining problems of security for mobile agent systems can be structured in the following individual areas:

2.1.1 Security Infrastructure for Mobile Agents

2.1.1.1 Security Architecture for Mobile Agent Systems

Problems and Requirements:

We did not find in the literature any comprehensive infrastructure for mobile agent systems that provides

- Trust for mobile agents and associated entities (i.e. trusted authorities, servers and databases etc),
- Management of security for the infrastructural components, and
- Complete security features for mobile agents and agent platforms and for their various operations.

At the time of this dissertation, research efforts in mobile agent systems have resulted in the development of different mobile agents platforms, mainly, for purpose of E-Commerce applications. Security was not the driving force for these agent platforms. For instance, Foundation for Intelligent Physical Agents (FIPA, 2008) standardized specifications for mobile agents and agent platforms. However, they did not pay much attention to security issues concerning mobile agents and platforms. To some extent research community has addressed the issue, but still lacks global, standard security architecture for mobile agents.
Current Results/Contributions:

Yi Cheng proposed **Comprehensive Security Architecture for Mobile Agents** [4]. Different components specified by her includes a) Mobile agent: A mobile agent has structure that includes header, code, and data. Header field contains security attributes. b) **Certificate Management System (CMS)**, the purpose of CMS is to provide support for generation, distribution, storage, retrieval, verification and revocation of public key certificates. c) **State Server**, which stores the state of agent code at a particular host. **State Server** provides a way to detect any malicious act with mobile agent by agent server during its execution. Significant contribution of Yi was that she specified very detailed security attributes of mobile agents. She also introduced additional roles, in addition to agent owner, to verify the trust of mobile agents. These roles were Trust Appraiser and Privilege Authority. Trust Appraiser assigns level-of-trust to an agent based on code’s conformance to its declared functionality. Privilege Authority signs agent-owner-credentials.

Karnik proposed **Ajanta Security Architecture** [5]. The components of his architecture include agent servers, agents, agent owner, guardian object, name service, credentials, Code Base Server, Security Manager, Agent Transfer Protocol (ATP), and domain registry. **Agent owner** concept is the same as in Yi’s architecture; it is a user on whose behalf agent executes. Creator of an agent can be an application program, agent server, client program or another agent. There is a Guardian object used to handle exception conditions which an agent fails to handle. Ajanta uses name service for mapping URNs to physical locations of entities (agents, agent servers, resources, etc). Every agent carries tamperproof certificates, called credentials, assigned to it by its owner. Agent code base server provides codes for the classes required by an agent while executing at the remote host. **Resource** in Ajanta is any service or information provided to agents in a specific application. The ATP is responsible for secure transfer of agents. **Domain registry** keeps information about currently hosted agents on an agent server. Agent servers have also access control lists that keep policies to be applied to host operating system resources. Server’s Resource Registry keeps the information about the resources available to visiting servers. Ajanta Security Manager mediates agents’ accesses to system level resources. A proxy object is used in Ajanta to provide indirect reference to agents for application defined resources. **Resource Access Protocol (RAP)** is defined in terms of generic resource objects. In order to secure agents in Ajanta, they have structured agents in such a way to provide three mechanisms for their security: read-only, append-only container and selective visible objects. Ajanta agents have also appended only logs that cannot be deleted or modified.

Tanenbaum proposed a multilayered middleware system **Mansion middleware** [6]. Middleware security architecture provides secure agents communication, secure mobile agent transport, start-up and secure auditing of all changes to mobile agents. Mansion introduced the concept of hyperlinked rooms. The following are the entities that can be in the room: agents, objects, and hyperlinks. Each agent is a multithreaded process running on a host. Hyperlinks determine how the rooms in a world are connected. A world consists of one or more sections, each containing a set of rooms. Each section has one or more Section Entrance Rooms (SER). Each world has specially marked (SER) called World Entrance Daemon. Each world has its owner and world entrance policy. Agent Management Station (AMS) keeps agent’s whereabouts. Each agent in a world has a unique AgentID. Each room has a special object Room Monitor.
Object (RMO) that registers all contents in a room. RMO has an attribute sets (ASes) which contains the description of entities in a room. Each section has Distribution Policy (DP) which is set by the section administrator. DP is expressed with the help of Zone, which is a group of processes which are referred by a single, cryptographically protected name. Besides these components, Mansion middleware is implemented as a multilayered system. The lowest layer above the operating system is called Agent Operating System (AOS). It provides an interface to higher layer middleware systems with primitives for secure communication, process start up, and agent storage and migration. Application Middleware (AppMW) above AOS layer provides middleware protection and services specific to a given application. AOS provides a secure container for storing an agent and its data, called an Agent Container (AC). Tanenbaum also introduced the concept of confined room. Agent in a confined room can communicate only with the outside world with the help of a guardian agent.

Model and architecture of an agent system is proposed by IBM Research Division in Zurich research laboratory, named “A Security Model for AGLETS” [7]. They have proposed security model and then architecture based on the model. Aglet has five major elements: aglets, contexts, messages, aglet transfer protocol, and aglets API (A-API). Aglets security architecture has two components: policy database and the preferences of aglet owner. Their model implements security policies. They have identified manufacturer, owner, master and authority principals, as roles. The context master policy database defines security policies for aglet context under control. Agent Owner establishes a set of security preferences for contexts that aglet might visit. The model defines eight principals: Aglets, AgletManufacturer, AgletOwner, Context, ContextManufacturer, ContextMaster, Domain and DomainAuthority. Aglet is an agent program that has its independent thread. The author of this aglet program is called AgletManufacturer. The individual that has legal responsibility of aglet’s behaviour is supposed to be AgletOwner. Context is the interpreter that executes aglets. Author of a context program is called ContextManufacturer, controlled by ContextMaster. A group of contexts owned by same authority is called domain and its administrator is known as DomainAuthority.

Adam Pridgen in “A Secure Modular Mobile Agent System (SMASH)” proposed layered security architecture [8]. Above OS layer SMASH has un-trusted layer. When an agent arrives to a platform, it is checked for integrity. Upon successful completion, it is moved to an un-trusted layer. Afterwards, agent moves to authorization layer, where it is authenticated and then placed into Trusted Containment Zone (TCZ). Agent interacts with Security Manager from the TCZ. Security Manager has three sub modules: Task Manager, Resource Manager and Agent Staging Area. Task Manager decides which services agents will use. Resource manager sets-up proxies, so that agent can access resources. SMASH has an area called Blackboard, a memory-constrained FIFO queue is available to any agent for read-write access.

Varadharajan and Foster proposed security architecture for agents called Vishnu [9]. Their agent platform comprises the following logical components: Agent Server Base (AgB), Agent Host (AgH), Agent (Ag), Security Management Component (SMC), and Security Management Authority (SMA). Agents can migrate from one agent base to another and agent base runs on every host in the network that can be a potential host for agents during migration. Each agent base has a trusted security management component called SMC. The SMC provides mechanisms for generation and validation of agents. Moreover, SMC also maintains security policy information enforced on agents, list of commonly used certificates and
name servers. The group of agents under the same policy is called a domain. Each domain has a security authority referred to as Security Management Authority (SMA). The role of an SMA is to interact with SMCs in the domain for establishment and maintenance of security policies within the domain and also to interact with SMAs domain for inter-domain communication scenarios. SMA is mainly used to validate agents originating from other domains.

**Analysis and Comparison with our Results:**

Analysis of the above research results shows that there has been reasonable amount of research work done over the years related to security architecture for mobile agents. We have observed that emphasis is more towards runtime security aspects of mobile agents i.e. identification and roles of different runtime components that are necessary for mobile agents security, architecture of mobile agents platforms and identification of entities those are directly consumers or producers of mobile agents actions. In addition to this, we found that research work explicitly specify mechanisms for agents’ registration after arrival at agent platform, along with the specifications of agents’ mobility models between different agent platforms. Agents’ discovery and agents’ services discovery model are key aspects of existing work. This analysis shows that research and development emphasis is more towards security architecture and components during execution of mobile agents. There has been very little effort to make mobile agents creation process secure and trusted. We have addressed this problem in our infrastructure. We provide a system for creation, classification and validation of trusted mobile agents, before their execution in the network.

### 2.1.1.2 Authorization Systems for Mobile Agents

**Problems and Requirements:**

One of the important problems of security for mobile agents is authorization of agents. Appropriate access control should be applied to mobile agents at remote hosts, i.e. how agents can be authenticated and then authorized to perform different operations. There are a number of solutions for this problem, all solving it at the architectural level. It means identification of high–level system components and their mutual interactions in order to provide infrastructure support for mobile agents’ authentication and authorization. The key component that was missing at the time of this research was the delegation of access rights from human users to mobile agents. Mobile agents perform their functions on behalf of human users. Therefore, the responsibility of mobile agents’ actions relies on their owner. Indirectly, it means that mobile agents will have the same access rights as their owners for a particular system/application. In the current research results we did not find any comprehensive authorization architecture that identifies components, methodology and policies for mobile agents all together.

**Current Results/Contributions:**

Navarro et al. [10] presented authorization framework for mobile agents based on XML standards. Their framework supports role-based access control policies with discretionary delegation of authorizations. They have identified infrastructural components used in their authorization framework. A significant
amount of their work is categorizing authorization into two types: a) access level authorization and b) management level authorization (e.g. “the right to delegate access level authorization”).

Zhang [11] presented a framework for mobile agents’ policies to protect data and resources carried by mobile agents in runtime environments using trusted computing technologies. The notable feature of their work is “mobile policy”, which is different from the traditional policy approach. Local host enforces only local security policy, which protects only local resources, while “mobile policy” is created by the originator of the mobile code in order to protect sensitive information carried by the code itself. Their framework provides mobile policy specification and definition, as well as high level implementation architecture using Java.


Aglets system [7] implemented flexible security policies for mobile agents and for infrastructural components by introducing the notion of roles (manufacturer, owner, master and authority) for principals. Aglets system provides definition of policy language based on security concepts mentioned in their security model. Policy language comprises named groups, composite principals, and hierarchical resources with associated permissions. Their sample security policy consists of a set of named privileges and a mapping from principals to privileges. In addition, they provided the concept of black lists, which blocks malicious aglets or contexts.

Mubarak, et al. [12] describes semantically rich security policies for mobile agents. According to the authors, policy enforcement consists of policy storage area, policy distributor, policy implementer, and monitor and policy specifier. They structured policies for mobile agents into three different types: authorization policies, obligation policies, and refrain policies. These policies address issues associated with agents’ security, like encryption, access control, and authorization. Although they have categorized policies, they did not specify the structure of a policy.

Tripathi [5] presented a framework for building policy-based autonomous distributed agent systems. Policies are used for recovery and configuration mechanisms in the event–based distributed systems. Policies are defined using ECA model (event, condition and action) and they are expressed using XML. In their framework distributed mobile agents are used to monitor the environment for events that may violate security policies. As soon as agents get an event that matches with one of the pre–defined conditions, they apply corresponding action. Their framework has an entity called System Management Agent (SMA), responsible for creation and configuration of agents according to system-level configuration, specified by the administrators.
Analysis and Comparison with our Results:

The analysis of the above research results shows that there is diversity of solutions related to authorization system of mobile agents. Different research groups and different mobile agent systems provide solutions for authorization of mobile agents. However, they do not follow any particular standard for authorization of mobile agents. They define their authorization systems for mobile agents in a proprietary form. Due to this, they are not able to process external mobile agents nor can they get authorization decision for their mobile agents from external standard components, like Policy Decision Point (PDP). We have overcome this problem by introducing an authorization system for mobile agents that uses standardized role-based XACML policies. Hence these policies can be processed by any external PDP that follows XACML standard. Role-based XACML policies also give us leverage to combine user access rights with mobile agents’ functional features under a single policy set. In order words, it binds user and his/her access rights with his/her mobile agents.

2.1.2 Security for Mobile Agents Systems

2.1.2.1 Protection of Mobile Agents’ Code

Problems and Requirements:

Mobile agent’s code must be protected for integrity, confidentiality and availability. Integrity means that the code is not illegally or accidentally modified or extended with malicious code. It is necessary for mobile agent to perform only the tasks which it is supposed to perform. Unauthorized change of mobile agent’s code results in an unauthorized or malicious activity by a mobile agent, which is not acceptable and can result in a security violation. Similarly, availability of mobile agent’s code is directly related to availability of mobile agents for their tasks. If agent’s code’s changed during migration or at the start of execution at a remote platform, then we can expect any type of malicious activity from an agent. It is very difficult to achieve confidentiality of agent code against an agent platform. Mobile agents execute at agent platforms, hence platform can easily read mobile agent’s code and can illegally modify it. Class files of agents can easily be decompiled to produce agent’s source code, called reverse engineering. There should be a solution to prevent reverse engineering of mobile agents’ code in order to achieve confidentiality of agent’s code.

Current Results/Contributions:

Ajanta [5] is Java based mobile agent system which provides a good solution for code security. Ajanta uses read-only containers and appends them in order to provide mobile agent state security. In order to secure agents, Ajanta has structured agents in order to provide three mechanisms for their security. Agents can have read-only, append-only container and selective visible objects. Ajanta agents also have appended only logs, which can only be appended and cannot be deleted or modified. Ajanta has not provided any explicit mechanism for payload security. Every agent carries a tamperproof certificates, called credentials, assigned to it by its owner. Agent code base server provides code for the classes required by an agent while executing at remote hosts. But this solution does not provide code security; it
only covers securing the state of code. Moreover, it does not incorporate a mechanism where a mobile agent platform next in path has the authorization to see previous platform’s data.

MARISM-A [20] provides security for agents’ code by following the itinerary and encrypting the code with public key of next mobile agent platform in the path, thus making the code unreadable for any other mobile agent platform, except the next in the path. The problem with this solution is the performance issues while communicating with the next mobile agent platform in the path to perform mutual authentication and encryption with its keys. MARISM-A has classified agents’ architecture into five types. Among them is Mobile Agent with Explicit Non-recursive Itinerary agents category which uses verification of agent’s hash in order to avoid attacks against agent’s integrity.

Mole [21] Mobile Agent System provides the concept of the “Black Box” security mechanism for mobile agents. In this mechanism it converts plain mobile agent into a Black Box Agent and this agent has the security features which can protect confidentiality of its code and data. Mole assumes that a mechanism to convert plain agent into Black Box Agent is available, which means that Mole itself does not provide any explicit mechanism to protect code of the mobile agents.

Xinwen Zhang [11] describes how remote policy enforcement can be used for runtime protection of agent’s code. His paper provides good mechanisms for restricting unauthorized users from executing the code, which in turn provides good security for code of mobile agents. They have shown how JAAS can be used to enforce access control restrictions over data. However, the downfall is that system uses ‘Trusted Computing Devices’ and depends on ‘Trusted Runtime Environment (TRE)’. Therefore, in order this system to work, TRE should be present.

Mobile Architecture proposed by Varadharajan and Foster [9] roughly covers mobile agent’s code security. The architecture indicates an overall idea how mobile agents should incorporate security into them. Their architecture provides mechanisms for protecting code from malicious mobile agent platforms by encrypting agents using symmetric key. The key is protected by the public key of the receiving platform with an assumption that the next platform is trusted.

Palo Bellavista, Antonio Corradi et.al indicated in their paper that mobile agent’s code protection is the problem that does not have universal solution [13]. Also, literature survey indicates that only prevention or detection techniques can be used. There is no such method that can 100% protect agent’s code against malicious host/platform. Agent needs to utilize services provided by the host platform, hence it cannot be 100% self-sufficient. Once agent migrates to another host, it stores the code temporarily on that host machine. During this time period, there is a possibility that the agent’s code can be illegally modified or tampered. Alternative solution is that an agent is self sufficient and brings complete logic and comprehensive logical resources with it. However, such agent systems would have significant transmission cost, because of the larger size of agent’s byte code.
Prevention Techniques

Sander proposed a prevention technique that does not allow agents to migrate to un-trusted hosts [14]. In this case agents assumed to be benign, as only trusted parties are permitted to setup an agent platform and to host agents’ execution.

The second prevention technique can be the use of special tamper-proof hardware devices proposed by Zachary [15]. This approach avoids unauthorised modifications of agent’s code by executing agents in a physically sealed environment. Smart cards can be an example of such hardware. An agent accepts input, processes it, and produces output. However, this approach has several serious drawbacks, such as resources environment constraints, cost, and usability, etc.

Another prevention technique is to obfuscate agent code and data [16] in order to make it difficult for malicious environments to analyze the code. However, this technique has several serious limitations: a) the difficulty to identify in advance an agent’s lifetime that minimizes the possibility to successfully inspect agent data and code, b) the need for global time clock to check expiration time [13].

Detection Techniques

Current detection techniques are mainly based on cryptographic approaches to encapsulate results of agents’ computation at each visited platform, so that integrity of results can later be verified. These results can be partially or fully encapsulated. In the case of partial encapsulation, Trusted Third Parties (TTP) are needed. The roles of TTPs are to track agents’ execution at each intermediate step by recording partial computational results [17].

Execution tracing [18] is another method used for detecting unauthorized modifications of agents’ code/state. It is based on non-modifiable traces of agents’ computations created at each intermediate agent platform/host. Signed hash of the trace is sent from an execution node to the sender’s node. Received traces can be compared with computation results upon agent’s return in case of any doubt about malicious behavior by some host.

Replication of agents [19] and then sending them through different paths is also used to detect malicious behavior of agents. Results are compared with the same type of agents, but coming through different paths. We can also expect graceful degradation of results in case of attacks on agents.

Analysis and Comparison with our Results:

In general, the problem with the above solutions for mobile agents’ protection is that they do not provide foolproof mechanism for mobile agents’ code protection. Detective techniques allow detecting malicious behaviour of mobile agent’s code, while preventive techniques prevent problems. However, we need a complete solution that does not allow malicious host to alter mobile agent’s code or to spoil mobile agents’ execution. Therefore, our solution is based on a protective approach. We provide complete encryption of mobile agent’s code and its baggage. We do not assume that the next host security
mechanisms are trusted. Hence, we do not send agent’s code to the next host in a clear form. Mobile agent’s code is decrypted on the fly when it is already inside the JVM. Hence, we completely avoid any possibility of alteration of mobile code. It is important to note there is difference between next host and our Software running on next host. We do not trust next host security mechanism but our software running on next host that receives agents is trusted using methodology explained in [80]. How it is trusted and used in our system is explained in section 7.2.1.2.

2.1.3 Comprehensive Network Security System based on Mobile Agents

2.1.3.1 Intrusion Detection and Prevention Systems using Mobile Agents

Problems and Requirements:

Intrusion detection and recently also intrusion prevention systems are one of the most popular approaches to networks security today. Although they provide some degree of protection, various penetrations, intrusions, and espionage reports indicate many serious problems and weaknesses of those systems. The major reasons are the lack of efficiency, limited flexibility, and vulnerabilities to direct attacks. At the same time they generate high number of false positives, even with high maintenance costs. Finally, the worst is that they have limited response capability. Usually, IDSs require significant processing by security administrators, they are very difficult to analyze, and they usually do not detect problems and provide response in real time. Manual responses and the lack of coupling of Automatic Intrusion Response System (IRS) with IDSs is another major hindrance towards full efficiency of IDSs today.

Moreover, IDSs lack mechanisms to detect vulnerabilities before they are exploited and to apply patches before attacker can exploit them. Different research organizations and vendors have their own vulnerabilities databases. These databases are scattered and do not provide comprehensive list of all vulnerabilities that exist in current software components.

Current Results/Contributions:

An architecture for intrusion detection using autonomous agents (AAFID [22]) proposed at Purdue in 1998 is an agent based hierarchal architecture for IDS. It decomposes the traditional IDS into lightweight autonomous cooperating agents, which can be easily reconfigured. Autonomous agents used in the AAFID project, using static and special purpose agent’s platform, are used to dynamically reconfigure IDS components. The other property worth noticing is that AAFID is based on a hierarchal architecture, which is vulnerable to direct attacks. If any of the internal nodes is compromised, the whole branch is disabled. Secondly, the transfer of large logs across the hierarchy also overloads network traffic.

ADVPP project [23] approached detection of vulnerability exploitations in privileged programs by monitoring operational audit trails. Their work is based on the assumption that privileged programs are more likely to exploit vulnerabilities. They have introduced the concept of Program Policy Specification Language based on a simple predicate logic and regular expressions.
“The Java Agents for Meta-learning (JAM)” project [24] deals with the concept of meta-learning for distributed data mining, using intelligent agents. It has two components: local fraud detection agent, that learns how to detect fraud and provides intrusion detection capabilities, and a secure integrated meta-learning system, that combines the collective knowledge acquired by individual local agents. Data mining, like neural networks and other single-point learning applications, does not enable knowledge sharing between agents. Meta-learning approach tries to reduce this limitation by integrating a number of remote agents.

The Information-technology Promotion Agency [25] (IPA) in Japan has developed an IDS called the Intrusion Detection Agent System (IDA). The IDA is a multi-host based IDS. Instead of analyzing all of the users’ activities, IDA works by watching specific events that may relate to intrusions, IDA gathers and analyzes the information and decides whether or not an intrusion has occurred. IDA system relies on mobile agents to trace intruders between various hosts involved in an intrusion and to gather information.

Intrusion Prevention System Design (IPSD) [27] presents an idea of integrating the isolation function of firewalls with the detection capability of IDS. Combination of both provides a new concept of intrusion prevention system. Both firewall and IDS will use the merits of each other in order to provide tightly coupled solution that can react to network changes in a more effective manner.

Commercial and Open Source Products

In this section we review some commercial and open source IDS/IPS products. There are many other IDS/IPS products, but they are not as advanced as the reviewed products.

SNORT is an open source cross-platform lightweight network intrusion detection tool used for network traffic monitoring in order to detect suspicious network activities. It has rules-based logging in order to perform content pattern matching and detects a variety of attacks and probes, such as buffer overflows, stealth port scans, CGI attacks, etc. However, its rules database must be updated regularly in order to protect against new threats [54].

Cisco provides an extensive set of security features in their different security products, such as Defeat Distributed Denial-of-Service Attacks, Cisco Intrusion Prevention System (IPS) sensors, etc. However, common to all solutions and products is that Cisco is still using common security solutions to protect networks. Those security solutions fail to provide adequate level of protection, because of ever increasing security incidents. The main reason is still using the traditional signature–based approach for many products [55].

nCircle provides security risk and compliance management solutions. They have a number of products. Major deficiency of their products is that they are still using existing static methods to provide protection of network resources and do not provide preventive and automatic response capability with their solutions [56].
Reflex Security’s Intrusion Prevention™ solutions provide end-to-end enterprise network protection. Reflex IPS applies packet inspection with signature, anomaly and rate-based algorithms in order to inspect and control network traffic flows. This detection methodology is already proved to produce either high rate of false positives or false negatives and thus does not provide effective and efficient secure protection against ever-increasing threats [58].

Nessus™ is a vulnerability scanner that provides a couple of good features, like efficient discovery of vulnerabilities, network configuration and auditing, asset profiling, etc. However, the major problem with Nessus is that it requires significant involvement of security administrators [57].

Analysis and Comparison with our Results:

The analysis of the current situation and essential causes of the current problems indicates that there are essentially two main reasons for those problems today: a) Humans (system and network operators) are slow to detect vulnerabilities, process logs and react to intrusions in real time. They have no time to follow discovery of new vulnerabilities due to diversified locations and structure of their announcements, and they are slow to react to on-going attacks in real time, since most of the time they are even not aware of those attacks. b) Software is always produced by humans using a manual process, which is prone to errors and vulnerabilities.

The system we proposed uses an innovative approach to eliminate the first essential problem: secure mobile agents – i.e., active entities that can migrate from one network node to another by transferring their code and by eventually also preserving their reached execution state. The approach comprises the concept, the set of components and an effective architectural solution for building secure network systems using mobile agents, The system performs the following four functions in the network: (a) autonomous detection of vulnerabilities on different host (in a distributed network) before an attacker can exploit them, (b) monitoring, retrieval and installation of patches, (c) protection of hosts by detecting attempts of intrusions and responding to them in real time, (d) tasks related to security management.

Network protection and prevention of intrusions show best effects when they are incorporated into the system architecture, instead of being added on later. Our system is based exactly on such approach: the architecture, components and all protocols of the system are using secure mobile agents. Those agents monitor the network, react timely and more accurately to various intrusion attempts, and thus eliminate or greatly reduce vulnerabilities.

2.2 Thesis Contributions and Research Problems Solved

Based on the literature survey we have found that in the last decade significant research has been conducted on mobile agents’ security. Although this area is very popular, there are still many open research problems that no one has addressed so far. We have identified those problems in the form of a six research questions. Next, based on those research questions, we have also identified security requirements. Finally, we solved, the identified security requirements using our security infrastructure for mobile agents. This infrastructure not only provides security architecture and security mechanisms for
agents’ runtime execution, but also infrastructural components along with methodology for creation, classification, adoption, and validation of mobile agents, before their actual deployment in a real environment. In particular, we solved protection of mobile agent’s code and its baggage during execution.

Our infrastructure provides innovative solutions for the following research questions: a) How to create trusted mobile agents, b) How to adopt trusted mobile agents in a secure manner, c) How to establish trust in mobile agents’ code during their deployment, d) How to protect mobile agents and their baggage during execution, e) How to establish secure communication between two mobile agents during execution, and finally, f) As a proof of concept, show how mobile agents can be used for network security.

Our unique solutions are the following:

a) Creation of trusted mobile agents. We have found through in-depth analysis of literature that there are significant research contributions that address the issue of trust for mobile agents. However, they are mainly addressing trust when agents are already launched in the network and in the execution phase. Trust is based on reputation of an agent, i.e. how agent is performing, its behavior, and its interactions with other agents. Alternative trust is being established based on agent’s credentials, but in all cases trust is established only during agents’ execution. However, we suggest a methodology which establishes trust in mobile agent’s code even before its initialization. It means that agent’s owner should have verifiable information about the creator of the agent, verifiable information about the entity who has checked the agent’s code and certified its correctness, and information about the entity that assigned appropriate roles to an agent. This verifiable information helps agent’s owner to establish trust in mobile agent’s code, so that later he/she can use an agent with confidence that it will not perform any malicious activity in the network. To the best of our knowledge, this is a unique contribution. We do not know of any paper that addressed and solved this issue.

b) Adoption of trusted mobile agents in a secure manner. We have proposed a new concept for adoption of mobile agents by end-users before their initialization. We have identified a secure mechanism for mobile agents’ adoption. The concept is that a trusted authority assigns a role to an agent based on its functionality. As a next step, agent code is being published, so that potential end-users can adopts specific agent based on their task requirements and then use them. In FIPA-compliant agent systems there is a concept of an agent publishing its services and those services then can be discovered in an agent platform. In such systems there is an entity called Directory Facilitator (DF). DF is used by agents to register their services (in order to offer them to other agents) and also to search for services offered by other agents. DF must have an instance at each agent platform. It keeps information of available agents’ services at the platform. Different agents can discover the services of other agents and then use them. However, in our innovative concept, agent’s code, before its initialization, is published. Publishing of agents is at a global level. Anyone around the globe can search for required agents, can adopt them, and finally, can use them. In addition to the DF concept, we found in the literature the idea of integration of agent systems with Web services [81], which allows mobile agent systems to extend their functions by exploitation of Web services for access to external generic systems. However, in our case, we are using secure Web services to distribute mobile agents’ code, which is a novel contribution in
a mobile agents’ paradigm. The significance of the proposed approach can be realized from the perspective of an end-user who is a layman and does not have any idea about mobile agents. He/she is only concerned to get a service. We provided secure adoption mechanism for this purpose, so that agents can fulfill their requirements.

c) Establishing trust in mobile agents’ code during deployment. In the previous two research problems, the issue was to establish trust in a mobile agents’ code from the perspective of an agent owner. Agent owner wants to convince himself that agents were being created by a trusted person. Moreover, trusted authorities have evaluated agents’ code, certified their correctness and, finally, published them. Next, agent owner wants to use adopted agents in an execution environment. In order to establish trust in mobile agents’ code during execution and to establish binding between agents and their owners, it is required to delegate trust from an agent owner to the adopted agent. Our solution not only authenticates agent’s owner credentials, but also agent’s code (before its initialization). As the next step, trusted authority creates XACML/RBAC access control policies for mobile agents and their owners. Our policies’ structure effectively binds users with their agents. Moreover, we have introduced Single Sign-On (SSO) protocol for mobile agents, which was never used before, as reported in the literature. SSO provides a lot of flexibility for mobile agents, as they are not required to carry with them large agent’s owner credentials and they are not required to authenticate themselves at each runtime node. It is important to highlight that there are many systems in the literature that address mobile agents’ authentication and authorization using different approaches, but, as far as reported in the literature, no one has ever combined the benefits of SAML, XACML, RABC and SSO technologies for the purpose of mobile agents’ authentication and authorization. Moreover, delegation of agent’s trust from its creator to it owner and then eventually to other mobile agents was never reported before in the literature.

d) Protection of mobile agents during execution. In the case of mobile agents’ protection, we discussed in section 2.1.2, that there are only preventive or detective methodologies/solutions available to address mobile agents’ security problems. However, our major contribution is a protective approach, as our agents are decrypted only when they are already loaded in the JVM. Agents are transferred encrypted and also temporarily stored at remote hosts in the encrypted form.

The second and the most significant contribution in this area are protection of agent’s baggage and control of access to that baggage. The best available approach for agent’s baggage protection is to protect it with the public key of its owner. This is very restrictive approach, since only agent’s owner can see contributions from specific platforms. However, with our proposed solution, during mobile agents’ execution, different agent platforms can share their contributions with other platforms in addition to the agent’s owner. Our solution maintains confidentiality and integrity of agent’s baggage when they migrate between different platforms.

e) Establishing secure communications between mobile agents during execution. The problem of secure communication between two mobile agents residing on two different agent platforms is already solved by many research groups. However, the analysis shows that most of available agent systems use an approach in which agent platform is authenticated to another agent platform before mobile agents’ communication. After mutual authentication, mobile agents communication starts using agent platform
message services. Therefore, indirectly, trust is being established between agent platforms hosting communicating agents. Our solution is unique and advanced in a sense that we not only authenticate agent platforms, but also mobile agents and their owners. Mobile agents possess a ticket as a result of the SSO protocol. Tickets are being verified at the start of a communication between two agents. Therefore, in our case, trust is being established through agent platforms, mobile agents themselves, and indirectly, through their owners.

f) Using mobile agents for network security. The concept of using mobile agents for IDS is not a new. NIST suggested in 1999 to use mobile agents for intrusion detection and response system [82]. There are many publications about using mobile agents for IDS. However, we have used mobile agents in a holistic manner in order to address the issue of intrusions for a complete network. We proposed new multifaceted approach that provides layered protection. Mobile agents are being used at every stage of an attack timeline, thus holistically solving the problem of intrusions in a network. As far as reported in the literature, we do not know of any solution that uses mobile agents at multiple levels for protection of a complete network.
PART TWO

SECURITY ARCHITECTURE FOR CREATION, CLASSIFICATION AND VALIDATION OF MOBILE AGENTS

Part 2 explains the security architecture for the creation classification and validation of mobile agents, so that end users can trust mobile agents and can adopt mobile agents with full confidence that agents will execute in a correct way and will perform only their legitimate role. Chapter 3 introduces the MagicNET system. It gives overview of the complete system. Chapters 4 to 7 explains different phases of MagicNET system, along with specific problem address at each phase. Chapter 4 explains the architecture and methodology for creation and validation of trusted mobile agent. Chapter 5 explains methodology for adoption of trusted mobile agents.
3.1 MagicNET System

MagicNET, which stands for *Mobile Agents Intelligent Community Network*, is a system designed and developed in the Network Security Lab at ICT/KTH. MagicNET provides all infrastructural and functional components needed for research and development of secure mobile agents: support for building secure and trusted mobile agents, agent’s repository (agents’ store), mobile agent servers (platforms for their execution), management station, and all necessary security servers (IDMS, CMS server, Local CA Server, UDDI Server etc).

3.1.1 Phases of MagicNET System

We have conceptually structured the use of the overall system into four functional phases: a) *Trusted Mobile Agents Creation and Validation Phase*: when agents are being created, validated and appraised; b) *Mobile Agents Acquisition Phase*, when agents are published and adopted; c) *Mobile Agents Deployment Phase*, when agents are retrieved, users are authenticated and XACML polices are created for specific local domains, and d) *Mobile Agents Execution Phase*, which contains runtime components (physical network) for agents. Agents traverse the network and perform their tasks during execution phase. Figure 3.1 shows different phases and information flow. Information objects in these phases are in fact mobile agents. Mobile agents’ development starts in the first phase i.e. “trusted mobile agents’ creation and validation phase”. The outputs of the first phase serve as inputs for the next phase, and so on. Each phase provides input to the subsequent phase. Finally, mobile agents execute in a network during “mobile agents execution phase”.

![Figure 3.1 MagicNET System Phases](image-url)
We have identified in chapters 4-7 different roles and components in each phase. All these components and roles in the system possess public key certificates issued by Local Certification Authority (LCA), which is the component of PKI infrastructure. We will explain each component and its role in upcoming chapters, in the context of specific solutions.

3.1.2 Agent’s Life Cycle of Development and Execution

Existing literature addresses only agent’s execution life cycle, which has four stages: agent launching, agent hosting, agent traversing, and agent return. It specifically means that all research results have focused on how to secure mobile agents during execution. There has been less emphasis on mobile agent’s development life cycle. Our infrastructure clearly addresses security requirements during agent’s development and agent’s execution life cycle.

In our infrastructure we have proposed comprehensive agent’s development and execution life cycle, which comprises eleven stages: agent creation, agent trust appraisal, agent privileges assignment, agent publication, agent retrieval, agent owner authentication, agent code authentication, agent launching, agent hosting and execution, agent transfer and, finally, agent return. These eleven stages are spread over four phases mentioned above. Each phase includes different stages and thus solves specific security requirement of a particular phase. The next section explains different security requirements and motivation of security issues related to mobile agents’ security infrastructure in each specific phase. Those requirements are then addressed in subsequent chapters, where security procedures corresponding to each phase have been discussed in detail. Figure 3.2 shows different stages in sequence starting from agent creation.

a) Trusted Mobile Agents Creation and Validation Phase
   i. Agent Creation
   ii. Agent Trust Appraisal
   iii. Agent Privileges Assignment

b) Mobile Agents Adoption Phase
   i. Agent Publication
   ii. Agent Retrieval

c) Mobile Agents Deployment Phase
   i. Agent Owner Authentication
   ii. Agent Code Authentication
   iii. Agent Launching

d) Mobile Agents Execution Phase
   i. Agent Hosting and Execution
   ii. Agent Transfer
   iii. Agent Return
3.1.3 Research Problems addressed for Each Phase

This section covers motivations for different research problems, related to mobile agents’ security, being address in four phases.

3.1.3.1 Trusted Mobile Agents Creation and Validation Phase

Most research papers and practical mobile agent systems, described in the literature at the time of this research, treat mainly deployment of mobile agents. That is, they describe how agents are launched, how they traverse the network, perform their functions, create and accumulate baggage, and report results. Current security issues and solutions are also addressing aspects relevant only for deployment of mobile agents. In most of the papers, agents’ functions, identities, security parameters, authorizations, assurance levels, etc. are assumed as being already available at the time when agents are adopted or launched.

However, important issues during creation of mobile agents, like their appraisal, validation, calibration, evaluation, adoption, and customization for deployment environments are rarely treated in the literature. Those are all very important aspects that must be solved before agents are launched. Therefore, they are mandatory prerequisites for all other, operational security aspects of mobile agents. In this phase we refer to these issues as assurance and trust methodology, since the main goal of such schemes and processes is to create reliable agents that can be trusted and whose trust can be verified. We have identified infrastructural roles, components and procedures for assurance and trust methodology.
3.1.3.2 Mobile Agents Adoption Phase

Current research solutions have adequately solved security issues for mobile agents’ runtime security, but in spite of that we still do not see wider adoption of mobile agents’ technology. In our opinion, one of the reasons is inadequate attention has been given to the approach and methodology for adoption of mobile agents. Current examples of mobile agent systems are either being used in limited, closed environments or they provide good mobile agents security, but they are limited to experimental laboratories. One significant assumption in most of the current mobile agent systems is agent’s author (a person who writes agent’s code) and agent’s owner (a user who uses agents for accomplishment of specific tasks) are the same person. Under that assumption, implicit conclusion is agent’s owner has access to the code of the required agent. This assumption is one of the major constraints for the wider adoption of mobile agents. Mobile agent’s author and mobile agent’s owner are usually two different persons, sometimes even located at two different continents. In this case, the relationship of trust between agent’s owner and agent’s author is very important to establish.

In this phase we extend our infrastructure with components for publishing, discovery and adoption of mobile agents. The proposed extension provides the complete set of components and procedures for secure discovery and subsequent adoption of mobile agents.

3.1.3.3 Mobile Agents Deployment Phase

Most of the currently available mobile agent systems treat authentication of agents as a redundant issue, because mobile agent systems are mainly used either in experimental environments where agent’s author, owner and agent’s execution environment are administered by the same person or in closed environments, where mobile agent’s code is written specifically for a particular mobile agent system and no other platform has ability to execute the agent. Therefore, it is implicitly assumed that incoming mobile agent belongs to the local mobile agent system. Otherwise, if it is a foreign agent, then agent system rejects it due to its inability to process alien mobile agent code. However, authorization of incoming mobile agents is an important problem in the area of mobile agents’ security. Appropriate access control should be applied to mobile agents at remote hosts, i.e. agents must be authenticated and then authorized before performing their operations. There are a number of solutions for this problem, in which all can be solved at the architectural level. Precisely, it means identification of high–level system components and their mutual interactions in order to provide infrastructure support for mobile agents’ authentication and authorization. The key aspect that is missing is the delegation of access rights from human users to mobile agents. Mobile agents perform their functions on behalf of human users. Therefore, the responsibility for mobile agents’ actions relies on their owner. Indirectly, it means that mobile agents will have the same privileges as their owners for a particular system/application.

Traditionally, access control for users in some security domain is provided by using Access Control Lists (ACLs), Role-based Access Control (RBAC) or Attribute-based Access Control (ABAC) for a wide variety of applications. Extensible Access Control Mark-up Language (XACML) is usually used for
policy specification together with enforcement of access rights for various resources (objects) and users (subjects). XACML policies have effectively solved user access control issues and are now widely used in real-life systems. Based on the success of XACML for user access control, a number of researchers suggested to apply role-based XACML policies to mobile agents by identifying infrastructure components of their authorization systems. However, the reported results did not address the key issue that is user delegation of access rights to a mobile agent in the system.

In this phase we have provided a security architecture for authorization of mobile agents and mechanisms for user and agent code authentication, along with a binding mechanism between a user and his/her adopted agents. We have also specified the structure of role-based XACML policies for mobile agents.

3.1.3.4 Mobile Agents Execution Phase

Over the last ten years, after NIST Special Publication on “Mobile Agent Security” [43], significant research has been done on mobile agent security. Research community has addressed all security threats, identified by Jansen [43], in the mobile agents’ paradigm by proposing new innovative solutions. Among them, platform-to-agent security threats are the most challenging ones. A number of solutions that have been discussed or presented mostly based on detective or preventive methodologies, which means mechanism to prevent or detect attacks against integrity and secrecy of mobile agents.

According to advocates of detective and preventive methodologies “there is no universal and general solution to the problem of agent protection [13].” Since a mobile agent ultimately has to execute on mobile agent’s platform, the platform has full access to agent’s code and data. Agent platform might be malicious and try to execute the code in a manner, in which, it is not authorized to do so. An agent platform can change agent’s state, code or agent route during its execution at the platform. So, if a malicious platform obtains a mobile agent’s code, it can even change the purpose of a mobile agent, for which it has been launched. This problem resides in the area of code protection, but in addition to this, we have mobile agent data protection problem as well. If an agent platform is not trusted and has malicious intent and has access to mobile agent, then it can alter data/payload carried by the mobile agent or can steal/illegally read the data/payload of mobile agent. Even if the mobile agent platform is trusted and has rights to execute mobile agent, still it can access data carried by mobile agent, contributed by previous mobile agent host platform in the travel path. This can be against security policy in some environments, where one platform should not be able to access data of another platform or same platform should be able to access data of some other selected platforms.

This phase specifically provides mechanism for the protection of mobile agents and its baggage. In addition to simple agent’s baggage protection, we provide layered agent’s baggage protection approach, in which different platforms’ contributions/processing results can be hidden from subsequent hosts in the agent’s route.
Figure 3.3 shows complete MagicNET system different components and roles. This figure shows only the high level system component interaction. We will cover in detail each phase components interactions in subsequent chapters. Each rectangle in above figure shows the components and roles of each phase mentioned in previous section. There are certain entities that overlap between different phases e.g. Agent Factory is the part of first three phases, however for the clarity we have shown it in only first phase.
4.1 Creation and Validation of Trusted Mobile Agents

In this chapter we refer to important issues during creation of mobile agents, like their appraisal, validation, calibration, evaluation, adoption, and customization as assurance and trust methodology. The main goal of these schemes and processes is to create reliable agents that can be trusted and whose trust can be verified. As one of the results of our research, we have designed and provided detailed specification of the trust methodology and infrastructure for mobile agents, i.e. components, procedures, roles and tools that can be used to create, validate and evaluate secure, reliable and trusted agents for their use in specific local environments. There are three main steps of our trust methodology:

**Creation of mobile agents**: This is the process of creation of agents’ code and specification of their functions and services. This aspect is related to the classical security issue for any software, namely how to create accurate and reliable software modules. For this task we introduce an IDE (Integrated Development Environment), we call it “Agents Factory”, for creation of secure mobile agents. It is based on a simple, interactive, and user–friendly tools and methodology for specification of agents’ functions and services. The main feature of our system is that it creates agents that are inherently correct, secure, and reliable software modules.

**Specification of classification parameters**: Those are parameters assigned to an agent at the time of its creation. Each agent must have, first, its unique, functional, and globally recognizable identity. Then, it needs precise specification of its functional services, specification of the type and scope of its authorization, eventually certain constraints, certain protection requirements, etc. In our methodology based on values of those parameters, agents are classified into one of the three assurance levels: low, medium, and high, initially by their creators.

**Validation procedures**: Once agents are created and classified into one of the assurance levels, it is very important to validate their classification by independent parties and, as the result of that process, confirm classification of agents. In our methodology this function is performed by independent validation authorities.
4.2 Specification of Assurance Levels for Mobile Agents

The main aspects of trust handled by our infrastructure are the methodology and procedures for specification, classification and validation of assurance levels for mobile agents. These aspects are known in the literature as “distributed trust management” and their description may be the following: “

.Distributed trust management involves proving that an agent has the ability to access some service/resource solely by verifying that its credentials comply with the security policy of the requested service” [8]. This definition refers to two key aspects: agents’ credentials and security policy. Based on values of those credentials, we specify three assurance levels for mobile agents: low, medium and high. This approach is equivalent to assurance levels for PKI certification policies [9].

In another approach, trust in mobile agents is treated as a belief in their honesty, competence, reliability and availability [10] during their use. According to [11] “

.trust (or symmetrically, distrust) is a particular level of the subjective probability with which an agent will perform a particular action, both before it can monitor such action (or independently of his capacity to monitor it) and in a context in which it affects its own action”. Based on this alternative approach, trust is established and modified dynamically, during use and deployment of agents.

We have adopted the first approach. We have established three assurance levels for mobile agents based on their characteristics and properties. Therefore, in our approach, agents’ creators decide and declare desired (or claimed) assurance level for their agents. This claimed assurance level must be validated and confirmed by independent evaluators. After that, such agents may be adopted for deployment. During deployment the results of their performance contributes to their accuracy, confidence and reliability, which we treat as trust in mobile agents. We have identified the following properties and attributes of agents used for evaluation of assurance levels:

<table>
<thead>
<tr>
<th>Properties and Attributes</th>
<th>Assurance Level</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Low</td>
</tr>
<tr>
<td>Creator’s Signature</td>
<td>No</td>
</tr>
<tr>
<td>Owner’s Signature</td>
<td>No</td>
</tr>
<tr>
<td>Appraiser’s Signature</td>
<td>No</td>
</tr>
<tr>
<td>Privileged Authority’s Signature</td>
<td>No</td>
</tr>
<tr>
<td>Agent Registrar’s Signature</td>
<td>No</td>
</tr>
<tr>
<td>Code Encryption</td>
<td>No</td>
</tr>
<tr>
<td>RSA Key Size</td>
<td>512 bits</td>
</tr>
<tr>
<td>Baggage Encryption</td>
<td>No</td>
</tr>
<tr>
<td>XML based Task Specification</td>
<td>No</td>
</tr>
<tr>
<td>Role specification</td>
<td>No</td>
</tr>
</tbody>
</table>

Table 4.1: Agent Assurance Level

Signatures: The first attribute is agent creator’s signature. If agent’s code does not have its creator’s signature, then agent is at the low assurance level. On the other hand, if agent’s code has signature and signature can be verified, then it is at medium assurance level. This verification is done by verifying that the signature attached to the code is valid, i.e. the hash value computed over the code matches the value
that results from decrypting the signature field using the public key of the agent’s creator. Agent’s creator must have its certificate issued by a CA. Finally, high assurance level means that the certificate and also the entire certificate chain of the issuer of Agent Creator certificate can also be verified.

Signatures of Agent Owner, Privilege Authority, Appraiser and Agent Registrar (these roles explained in section 5.1.1), contribute in the same ways to assurance level as Agent Creator.

**Code Encryption:** If agent’s code is not encrypted, then it is at the low assurance level. If agent’s code is encrypted by using symmetric key based on 3DES with 168 bits key size, then it is at the medium level. Finally, if agent’s code is encrypted using AES algorithm for encryption, with 256 bits key, then it is on the high assurance level.

**RSA Key Sizes:** Agent author, appraiser and owner use certificates to distribute their public keys. If keys for RSA algorithm are 512 bits, then it is low assurance level. If keys are 1024 bits or 2048 bits, then it is medium or high assurance level, accordingly.

**Baggage Encryption:** If agent does not have a capability to encrypt the baggage at each platform, then it is at the low assurance level. If agent can perform encryption of the baggage using symmetric key shared with all servers, then it is at medium assurance level. Finally, if agent’s baggage is encrypted using server’s private key as a first step and then enveloped as a second step using agent owner’s public key, then it is at the high assurance level. Handling of symmetric key between servers is covered in Chapter 6.

**Task Specification:** Agent performs a particular activity on a remote machine. If agent does not provide description of its functionality in its header, then it is at low assurance level. If agent provides description of its functionality, then it is at medium assurance level. Finally, if agent provides XML based description of its functionality and task being executed on remote host, signed by Trust Appraiser, then agent is at high assurance level. XML representation of tasks allows remote servers to identifies agents’ activity on the server and provide some basic idea about the functionality agents will perform at the server. XML format provides simplicity, openness, extensibility, self-description, and can easily processed by machines, in our case agent servers (agent platforms).

**Role Specification:** Privilege Authority assigns different roles to agents. If that is the case, then agent is at high assurance level. If agent does not have any role assigned to it, then it is at low assurance level. Mobile agents’ access and authorization levels are being determined on the basis of agent’s role. Agent without any role will be allowed to execute only in restricted environments.

### 4.3 Creation and Validation Phase

The first phase of the MagicNET system is creation and validation of trusted mobile agents, which is based on the concept of assurance levels, mentioned in the previous section.
4.3.1 Roles for Creation and Validation of Agents

In the first phase there are three roles: Agent Creator (AC), Agent Trust Appraiser (ATA), and Privilege Authority (PA). *Agent Creator* is a programmer who writes agents’ code.

*Agent Trust Appraiser* is a trusted authority who evaluates agents’ code in terms of their functionality and correctness, and then assigns a trust level to agents.

*Privilege Authority* is a trusted authority that assigns different privileges to agents by assigning different roles to them. All these three roles possess three public key certificates (digitalSignature, keyEnchipherment, digitalSignature&Nonrepudiation) issued by LCA.

4.3.2 Components for Creation and Validation of Agents

Agent Factory Server (AF)

Agent Factory Server (AF) is the only component in creation and validation of agents. AF is used as an incubator and repository for trusted agents. In our implementation, AF is RMI-based server and it provides different methods for clients to interact with it. AC, ATA and PA are the clients of AF during agents creation and validation phase. AF possesses two public key certificates (keyEnchipherment, digitalSignature&Nonrepudiation) issued by LCA.

Life cycle of agents’ development includes many steps, starting from agent’s creation and ending up with its adoption. During this process, agents pass through a number of steps (described in the next section). AF acts as an intermediate server during these steps. AF uses local database to save intermediate forms of agents. AF database schema is very simple and it contains one table “AgentFactoryTable” which has the following attributes:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type</th>
<th>Null Allowed</th>
<th>Unique</th>
</tr>
</thead>
<tbody>
<tr>
<td>Serial No (auto generated)</td>
<td>Integer</td>
<td>NO</td>
<td>YES</td>
</tr>
<tr>
<td>CreationDate</td>
<td>Date</td>
<td>NO</td>
<td>NO</td>
</tr>
<tr>
<td>AgentCreatorContribution</td>
<td>Blob</td>
<td>NO</td>
<td>YES</td>
</tr>
<tr>
<td>AppraisalDate</td>
<td>Date</td>
<td>YES</td>
<td>NO</td>
</tr>
<tr>
<td>AgentAppraiserContribution</td>
<td>Blob</td>
<td>YES</td>
<td>YES</td>
</tr>
<tr>
<td>PrivilegeAssignmentDate</td>
<td>Date</td>
<td>YES</td>
<td>NO</td>
</tr>
<tr>
<td>PrivilegeAuthorityContribution</td>
<td>Blob</td>
<td>YES</td>
<td>YES</td>
</tr>
<tr>
<td>AgentRole</td>
<td>String</td>
<td>YES</td>
<td>NO</td>
</tr>
<tr>
<td>AgentTaskSpecification</td>
<td>String</td>
<td>YES</td>
<td>NO</td>
</tr>
<tr>
<td>AgentRegistrarSignal</td>
<td>Boolean</td>
<td>YES</td>
<td>NO</td>
</tr>
</tbody>
</table>

Table 4.2: Agent Factory Server Database Schema

4.3.3 Creation and Validation Procedures

There are three main procedures in this phase: agent creation, agent trust appraisal, and agent privileges assignment. These operations must be executed sequentially. However, there is one operation that is common between all phases of MagicNET system. This operation is Entities Authentication, explained in section 5.2.3.3. In our infrastructure all interactions (client-server, user-server, and server-server) among all entities are protected by use of the strong authentication protocol, based on the FIPS 196 standard [28].
4.3.3.1 Agent Creation

During agent creation, Agent Creator (AC) first writes agent’s code, creates hash of agent’s code (from now on denoted by $\alpha_c$), and then signs it with her private key $SK_{AC}$, producing a signature $S_{AC}$. AC then creates the “signed agent”, a package containing $\alpha_c$ (agent’s code), $S_{AC}$, $PK_{AC}$ (public key certificate) and makes it available at the Agent Factory (AF). The package produced is called PKCS7$_{AC}$ and it complies with the PKCS7 standard format [29].

Now AC mutually authenticates with AF using the protocol mentioned in the previous section. After successful authentication, AC stores PKCS7$_{AC}$ to the AF. Agents created using this process will follow all requirements for “high” assurance level, as explained in section 4.2. AF adds a new row to AgentFactoryTable, adds current date as agent’s creationDate and adds PKCS7$_{AC}$ as AgentCreatorContribution by executing the following SQL statement.

```
INSERT INTO AgentFactoryTable (CreationDate, AgentCreatorContribution) VALUES (<CurrentDate>, <PKCS7AC>);
```

4.3.3.2 Agent Trust Appraisal

Agent Trust Appraiser (ATA) first authenticates to AF using the protocol mentioned in section 5.2.3.3. As the next step ATA queries AF for agent validation requests. AF queries AgentFactoryTable in the database and fetches all rows that still have null value in the AgentAppraiserContribution column. This indicates that the corresponding package is pending for validation. AF creates a list of PKCS7$_{AC}$ objects and returns it to ATA.

```
List<PKCS7> agentPendingForAppraisal = new LinkedList<PKCS7>();
```

For every member of this list, ATA verifies integrity of agent’s code by using the $S_{AC}$ and then performs a series of software tests and subsequently assigns agent’s tasks specification $(t_1, t_2, ..., t_n)$ to the agent. ATA then signs PKCS7$_{AC}$ along with all her contributions, thus creating a signature $S_{AA}$. As the result of this process, a package PKCS7$_{ATA}$ is created, containing $(t_1, t_2, ..., t_n), PKCS7_{AC}, S_{ATA}, PK_{ATA}$. ATA submits this package back to the AF.

4.3.3.3 Assignment of Privileges to Agents

The same as to AC and ATA, Privilege Authority (PA) also mutually authenticates with AF. Then, PA queries AF for agent privilege assignment requests. AF queries AgentFactoryTable in the database and fetches all rows that still have null value in the PrivilegeAuthorityContribution column. This shows that the corresponding package is still pending for assignment of privileges. AF creates a list of PKCS7$_{ATA}$ objects and returns it to PA.

```
List<PKCS7> agentPendingForPrivileges = new LinkedList<PKCS7>();
```
For every member of this list, PA verifies integrity of the package using the $S_{AA}$. Then, PA performs a series of tests (i.e. she verifies that the tasks specification matches with agent’s core functionality) and assigns agent’s credentials, like group membership, or role ($r_i$). Agent role$^2$ is expressed as an URI, which offers a hierarchical organization and simplifies agent’s categorization, role management, and policy definition. A role for, as an example, a vulnerabilities info agent, could be expressed as follows:


The same as ATA, PA signs PKCS7$_{ATA}$ along with all her contributions and thus creates a signature $S_{PA}$. Finally, PA creates a package PKCS7$_{PA}$ containing $r_i$, PKCS7$_{ATA}$, $S_{PA}$, PK$_{PA}$ and deposits it back to the AF.

Figure 4.1 shows the three entities i.e. AC, ATA, and PA and their interactions with AF. Agent Creator (AC) may be multiple instances, while ATA and PA must have one instance for each domain.

---

2Hierarchical roles are supported by standard XACML Profile for Role Based Access Control (RBAC) [30, p.13], which is used for policy definition (see section 5.2).
Adoption of Trusted Mobile Agents

5.1 Adoption of Trusted Mobile Agents

In this chapter we extend our established infrastructure with components for publishing, discovery and adoption of mobile agents. The proposed extension provides the complete set of components and procedures for secure discovery and subsequent adoption of mobile agents. For this purpose we use the principles of secure web services. The second phase of the MagicNET system is adoption of trusted mobile agents.

5.1.1 Roles in the Adoption Process

In this phase there are two roles: Agent Registrar (AR) and Agent Owner (AO). AR registers different agents at the UDDI server using methodology of web services. AR also possesses three public key certificates (digitalSignature, keyEnchipherment, and digitalSignature&Nonrepudiation) issued by LCA. After being registered, the services of mobile agents can be discovered, again based on the Web services methodology. There are three ways to perform this process: (a) searching sequentially mobile agents Registry, (b) using an index to the Registry, or (c) as a peer-to-peer system [12]. We have used the first approach, i.e. the registry approach, since it represents more authoritative, centrally controlled information storage. Our AR acts as the registry owner and therefore has complete control of the Registry in order to register services of mobile agents. This means that service providers, in our case Agents Creators, do not have permission to publish their own service descriptions. AR acts as an intermediate trusted authority that performs this task. We have selected this solution in order to enforce separation of roles and thus higher objectivity and third–party verification of agents’ services before they are published in the registry. Moreover agent creators cannot directly register services of their agents to as first their agents are required to be evaluated by trusted parties, in our case ATA and PA.

The second role in the adoption process is Agent Owner (AO). This is the person who manages mobile agents in their execution environment. That person adopts various agents, based on his/her task requirements, and launches the adopted agents into his/her local network for execution. Based on the local policy, AO can be application administrator, network administrator, or system administrator. We will discuss AO role in the deployment phase in section 6.3.2. Here we are more concerned with the role that AO plays in the agents’ adoption phase, i.e. in the context of Web services as agents’ owner and the
requester entity. AO performs all these function from its local machine, we call it Management Station (MS). MS is equipped with software modules that facilitate adoption process during this phase, while in execution phase provides additional services, explained in section 8.2.2.1.

5.1.2 Components for Adoption of Mobile Agents

5.1.2.1 Agent Factory Server (AF)

The functionality of the AF server (mentioned in the section 4.3.2) is extended to support this phase too. AF acts as the main starting point for the adoption phase (described in section 5.2.3). It exposes web services built using secure application server WSO2 WSAS [33] and protected by Rampart [34]. WSO2 WSAS has been fully integrated into our system, so that web service deployment and configuration can be performed directly through the system graphical interface. Hence AF has two interfaces, a RMI interface and a web services interface.

5.1.2.2 UDDI Server

UDDI Server\(^4\) is based on the standard concept of UDDI servers, as specified by OASIS [37], i.e. web services publishing and discovery. In our system, UDDI Server acts as a registry containing a special category of services: agent provision web services. Any business entity (i.e. AF) that wants to provide agents to the community (i.e. end-users) must properly expose a web service interface and correctly publish this web service interface (in our case by means of another entity, i.e. Agent Registrar). WSO2 WSAS, JUDDI server and the JUDDI database have been fully integrated into the system, so that all routine procedures and configuration updates can be performed directly from the system graphical interface.

5.1.3 Procedures for Adoption

This phase comprises two operations. The first one is agent publication and as the result of this operation is that end-users have visibility of agents available for adoption. The second is agent retrieval, it provides mechanism for end-users to query and adopt particular agent that corresponds to their needs.

5.1.3.1 Agent Publication

AR, a person using Agent Registration station, is responsible for publishing of agents in a public repository, so they can be adopted by users. In that process AR first obtains the list of agents and then agents for publishing. This is a two step process, each step is secured and the overall process executes in mutually authenticated and authorized manner between AR and the AF. The information exchanged (mobile agents) maintains its integrity, data confidentiality, and non-repudiation. Both AR and AF own X.509 certificates issued by the LCA. Secure connection is established between AR and AF, based on

---

\(^3\) Rampart is the Apache Axis2 security module responsible for web service security based on standard WSS specifications [35].

\(^4\) UDDI Server has been implemented using an open source Java project called JUDDI [36], which follows OASIS specification of UDDI for web services [37]. Moreover, it has been integrated into the WSO2 WSAS secure application server, so that Agent Registrar (publisher) can use HTTPS SSL/TLS for authentication.
strong authentication protocol compliant to FIPS 196 standard [28] (explained in section 5.2.3.3). At the end of this process, entities own each other’s certificates. Entities exchanged digitalSignatureCert and keyEnciphermentCert.

AR retrieves the list of available agents from the AF. AR fetches then agent’s package from AF, verifies package’s S_pA, reads tasks/role specification, and finally publishes it, by making it available at the Universal Description, Discovery, and Integration (UDDI) server.

UDDI server exposes two different service interfaces located at the same server: publish and inquiry interfaces. Both services require authentication of the server. Publication is more sensitive operation than inquiry, so UDDI server can be accessed only via HTTPS/SSL using server-side certificate. Therefore, AR authenticates UDDI server using its certificate and subsequently she can authenticate herself, by sending her security credentials. During publication operation, client authentication has been implemented using customized version of the JUDDI authenticator that supports a certificate-based authentication, while the original version supports only simple username/password authentication.

The limitations of the JUDDI architecture prevented implementation of challenge/response protocols. It only allows unilateral interaction protocol. Therefore, we have adopted one-way authentication protocol from X.509 authentication standard proposed by ISO, called ISO/IEC 9798-3 mechanism I [38]. It is based on timestamps to avoid random number exchange and to limit the protocol to exchange of a single message. AR authenticates itself to UDDI server using this protocol. AR possesses a public key certificate (PK) and is the originator of the message. AR must first generate a timestamp (t_AR), then concatenates it with the unique identifier of the UDDI server (its distinguished name), signs the resulting concatenation with his private key and then sends the following message to the UDDI server:

\[
AR \rightarrow UDDI : PK_AR, t_AR, DN_UDDI, S_AR(t_AR, DN_UDDI)
\]

UDDI server verifies the signature and can also verify the certificate of AR from LCA. After successful verification UDDI server returns authentication token to AR. It is used later by AR to perform different functions related to services publishing.

### 5.1.3.2 Retrieval of Agents

A user e.g. security administrator, or later, after agent adoption, called Agent Owner (AO) who wants to adopt an agent to perform some activity, queries the UDDI Server to discover a proper agent with required functionalities. For that, user queries agent roles, which are considered equivalent to web services, according to the aforementioned UDDI standard\(^5\). He/she obtains the list of agent providers using web services. As soon as user finds proper agent provider attributes (web service location and other technical details), he/she can retrieve the selected agent using secure web services. At this instance, binding information including web service URL has also been received at the user machine. So, web service client is initialized at the user machine and may be immediately launched.

---

\(^5\)We have adopted UDDI4J [39], an open source Java class library that provides an API to interact with a standard UDDI, to publish and query agent from UDDI Server.
Web service client connects to AF in order to retrieve the agent. We have configured WSO2 WSAS application server along with Rampart [34] (the Apache Axis2 security module responsible for web service security based on standard WSS specifications [35]) at the AF. We handle requirements of mutual authentication, authorization, integrity, data confidentiality, and non-repudiation for web services using Rampart. In order to provide authenticity of clients along with integrity and confidentiality of SOAP messages at the application level, we have configured Rampart with “Sign and encrypt - X509 Authentication” option in order to properly manage certificates of each involved endpoint (AF and AO as web service clients). Server authentication (Agent Factory) is provided by the traditional HTTPS/SSL protocol, also supported by this secure server. In addition, we used \(<\text{sp: Basic256Sha256/>}\) algorithm available in WSS suite\(^6\) instead of the default algorithm \(<\text{sp: Basic256/>}\) which is based on SHA1 algorithm, whose security has been already partially compromised [40] (see Table 5.1).

<table>
<thead>
<tr>
<th>Algorithm Suite</th>
<th>Digest</th>
<th>Encryption</th>
<th>Symmetric Key Wrap</th>
<th>Asymmetric key wrap</th>
<th>Minimum symm key length</th>
</tr>
</thead>
<tbody>
<tr>
<td>Basic256</td>
<td>Sha1</td>
<td>Aes256</td>
<td>KwAes256</td>
<td>KwRsaOaep</td>
<td>256</td>
</tr>
<tr>
<td>Basic256Sha256</td>
<td>Sha256</td>
<td>Aes256</td>
<td>KwAes256</td>
<td>KwRsaOaep</td>
<td>256</td>
</tr>
</tbody>
</table>

**Table 5.1 - WSS Algorithms Suite**

User finally receives the packaged agent (content of PKCS7 PA) at his machine via secured channel mentioned above. The package contains agent’s code \(\alpha_c\), agent role \(r_1\) and the credential pairs: \((S_{AC}, PK_{AC}), (S_{PA}, PK_{PA}), \) and \((S_{ATA}, PK_{ATA})\). User verifies signatures of trusted authorities (AC, TA, PA). If verification is successful, mobile agent is ready for adoption by the user. Hence, the user will be called Agent Owner (AO), i.e. mobile agents execute in the network on behalf of their owners. It is important to note that agent so far has not yet been initialized.

### 5.1.3.3 Authentication of Entities

In our security infrastructure all interactions between all entities are protected by the use of customized implementation of the standard FIPS 196 protocol [28]. We use Agent Registrar (AR) and Agent Factory (AF) interaction, explained in section 5.2.3.1, as an example of the authentication procedure. In this particular case, integrity, data confidentiality and non-repudiation of the information exchanged between mobile agents is also maintained. Both AR and AF own X.509 certificates issued by LCA.

Mutual authentication procedure comprises the following seven steps:

i. The initiator (AR), equipped with its public key certificate \((PK_{AR})\), makes an authentication request to AF, with which it wants mutually authenticate, defined as follows:

\[
AR \rightarrow AF : \text{hello, } PK_{AR}
\]

ii. The responder (AF) determines if it will continue or stop authentication exchange process. It verifies \(PK_{AR}\) by contacting LCA and then \(DN_{AR}\) by contacting IDMS. If both verifications are successful, AF then generates a random number as a challenge \((r_{nd_{AF}})\) and keeps it for subsequent steps. AF creates and sends to AR a challenge token \(Token_{AFi}\):

---

\(^6\) Algorithm Suite: Defines what algorithms should be used when cryptographic operations are involved and the minimum and maximum key sizes to be used.
Once it receives **TokenAF1**, the initiator **AR** generates a random number challenge \( \text{rnd}_{\text{AR}} \) and keeps it for subsequent steps. **AR** creates and sends to **AF** an authentication token, **TokenAR**, by concatenating data and the generated digital signature:

\[
\text{TokenAR} : \text{rnd}_{\text{AR}} \| \text{rnd}_{\text{AF}} \| S_{\text{AR}}(\text{rnd}_{\text{AR}} \| \text{rnd}_{\text{AF}})
\]

When **TokenAR** is received the responder **AF** verifies that the value of \( \text{rnd}_{\text{AF}} \) is the same as the value previously kept. Then it verifies the \( S_{\text{AR}} \) in **TokenAR** using \( PK_{\text{AR}} \) obtained in the first step.

Next **AF** creates an authentication token **TokenAF2**, by concatenating data and generating digital signature:

\[
\text{TokenAF2} : \text{rnd}_{\text{AF}} \| \text{rnd}_{\text{AR}} \| S_{\text{AF}}(\text{rnd}_{\text{AF}} \| \text{rnd}_{\text{AR}})
\]

**TokenAF2** is sent together with the responder's certificate \( PK_{\text{AF}} \).

Once it receives the **AF**'s message, the initiator **AR** verifies that the value of the \( \text{rnd}_{\text{AR}} \) is the same as the value previously retained. Then, it verifies that the value of \( \text{rnd}_{\text{AF}} \) is the same as the \( \text{rnd}_{\text{AF}} \) retrieved from the **TokenAF1**.

**AR** verifies the responder's certificate \( PK_{\text{AF}} \) and using it verifies the responder's signature of the **TokenAF2**. If both checks are successful, authentication process is complete.
Figure 5.2: Components and Steps in The Process of Adoption of Trusted Mobile Agents
PART THREE

SECURITY SYSTEM FOR PROTECTION OF MOBILE AGENTS

Part 3 describes in detail security issues for deployment and execution of mobile agents. Chapter 6 describes security architecture for authorization of mobile agents along with the structure of XACML policies for mobile agents, which is the core of our authorization system. Chapter 7 focuses on the protection of mobile agents and hosts during execution of mobile agents.
6 Security Architecture for Authorization of Mobile Agents

6.1 Authorization of Mobile Agents

In this chapter we present a solution based on RBAC and XACML policies for mobile agents. RBAC and XACML polices are described in the next section. RBAC approach, so far used only for users, provides a lot of flexibility to maintain access control for large distributed systems. Mobile agents already have a role, based on the functions they perform, assigned to them by PA (mentioned in section 4.3.3.3). This solution can be applied in two steps: a) by creating security architecture for authorization of mobile agents (authorization system), and b) by specification of the structure/model for role-based XACML policies for mobile agents. As the first step, we present an internal model of role-based XACML polices for mobile agents and we show how to provide delegation of access control rights from users to mobile agents. Our composite structure of access policies maps users to their adopted agents and allows Policy Decision Points (PDP) (explained in section 6.3.1.3) to make access control decision for a particular resource by considering both, user and its agent, at a same time. As the second step, we have identified all architectural components of our solution along with the operations for enforcement of authorization of mobile agents during execution.

6.2 XACML Policies for Mobile Agents

The key aspect for use of XACML policies in the context of mobile agents is the mapping between users, who want to become agents’ owners, and mobile agents. Then, instead of applying policies to agents only by means of a role-based access control, there is first the need to authorize users in order to adopt agents. Then, agents act as active resources and later, in the execution environment, they are treated as subjects. Therefore, it follows that we need two different types of authorization policies:

- User authorization policies; and
- Mobile agent authorization policies.

This can be achieved by the adoption of the XACML 2.0 standard [1], which natively includes the functionality of splitting policies in groups with not necessarily uniform topologies and structures. As shown in Figure 6.1, each main policy set is contained inside a distinct <xacml:PolicySet> element.
We have earlier related mobile agents to the concept of their “role”, based on the main goal (functionality) of the agent itself. Adoption of a technology natively based on the concept of roles, such as RBAC [41], has been proven to be advantageous, especially in this context [30]. Roles are expressed as URI [42] in order to enhance role flexibility, guarantee their uniqueness, and simplify their management. This fits perfectly into the RBAC hierarchical model [41], which has also been used in our system to create policies. Hierarchical model has been supported by XACML from the very beginning (version 1.0). Our solution complies with the XACML/RBAC profile [30]. Our solution adopts RBAC role concept both for mobile agents and users, so that users are also grouped into roles and policies are expressed accordingly.

In addition to these two policies, there is also the third type of policies, which is used to specify agent platform authorization to access mobile agent’s data baggage. This type of policies is “Agent Data Authorization Policies”. As mentioned above, agents act as active resources, and in the execution environment, they are treated as subjects, because agents access different resources at an execution node on user’s behalf. However, if we go one step further, then during agent’s life cycle, agent visits a number of hosts and each host contributes its data to mobile agent’s baggage. Mobile agent carries this data to each node and eventually brings back results to its owner. It is important to apply appropriate access controls to agent’s baggage as well. Hence, in this context, agent’s baggage is treated as a resource and visiting agent platforms act as subjects. Our “Agent Data Authorization Policies” specifically address this issue. We have mentioned “Agent Data Authorization Policies” separately, because they are not part of the above policy set and hence they are evaluated separately by the PDP server. These are not RBAC policies, but rather simple XACML policies.

6.2.1 The Structure of a Policy

Overall, mobile agent authorization policy is structured as a root element <xacml:PolicySet>, shown in Figure 6.1, comprising two macro sets of policies:

- a set regulating adoption of mobile agents by human users (user authorization policies); and
- a set regulating mobile agent resource access control (mobile agent authorization policies).

The former is characterized by the identifier <&AgentAdoptionPolicies;>, the latter is characterized by the identifier <&AgentAccessCPolicies;>. The adopted root element’s policy combining algorithm
is the standard deny-overrides, whose resulting combined effect is permit-only if both macro policy sets’ results are permit. Such policy combination guarantees that access to a resource is granted to a mobile agent only if a proper policy rule allows such access and if mobile agent, which is acting on behalf of a user (the adopter), is properly authorized. Conventionally, root element is identified by the entity <&AgentAuthPolicy;> followed by an alphanumerical identifier, unique for the local PDP. <Target> element of this root policy set is empty\(^7\), because all the contained rules will be filtered differently, depending on the macro set that they belong to.

Table 6.1 contains the full description of the entities and namespaces described above and it is used from this point on. That is a short form of the attribute values in the form of XML entities, as well as the namespace definitions, both used to ease and shorten the realistic policies definition.

<table>
<thead>
<tr>
<th>Entity Name</th>
<th>Entity Value (URI)</th>
<th>Std.</th>
</tr>
</thead>
<tbody>
<tr>
<td>xacml</td>
<td>urn:oasis:names:tc:xacml:1.0:</td>
<td>XACML</td>
</tr>
<tr>
<td>xml</td>
<td><a href="http://www.w3.org/2001/XMLSchema#">http://www.w3.org/2001/XMLSchema#</a></td>
<td></td>
</tr>
<tr>
<td>rule-combine</td>
<td>urn:oasis:names:tc:xacml:1.0:rule-combining-algorithm:</td>
<td>XACML</td>
</tr>
<tr>
<td>policy-combine</td>
<td>urn:oasis:names:tc:xacml:1.0:policy-combining-algorithm:</td>
<td>XACML</td>
</tr>
<tr>
<td>function</td>
<td>urn:oasis:names:tc:xacml:1.0:function:</td>
<td>XACML</td>
</tr>
<tr>
<td>subject-category</td>
<td>subject-category &quot;urn:oasis:names:tc:xacml:1.0:subject-category:</td>
<td>XACML</td>
</tr>
<tr>
<td>subject</td>
<td>urn:oasis:names:tc:xacml:1.0:subject:</td>
<td>XACML</td>
</tr>
<tr>
<td>resource</td>
<td>urn:oasis:names:tc:xacml:1.0:resource:</td>
<td>XACML</td>
</tr>
<tr>
<td>action</td>
<td>urn:oasis:names:tc:xacml:1.0:action:</td>
<td>XACML</td>
</tr>
<tr>
<td>environment</td>
<td>urn:oasis:names:tc:xacml:1.0:environment:</td>
<td>XACML</td>
</tr>
<tr>
<td>role</td>
<td>urn:oasis:names:tc:xacml:2.0:subject:role</td>
<td>RBAC</td>
</tr>
<tr>
<td>UserRole</td>
<td>urn:magicnet:names:AgentAuthPolicy:1.0:UserRole-values:</td>
<td>-</td>
</tr>
<tr>
<td>CategoryRole</td>
<td>urn:magicnet:names:AgentAuthPolicy:1.0:CategoryRole-values:</td>
<td>-</td>
</tr>
<tr>
<td>CategoryAttribute</td>
<td>urn:magicnet:names:AgentAuthPolicy:1.0:Attribute:SubjectCategory</td>
<td>-</td>
</tr>
<tr>
<td>AgentAuthPolicy</td>
<td>urn:magicnet:names:AgentAuthPolicy:1.0:AgentAuthPolicy:1.0:PolicyId:</td>
<td>-</td>
</tr>
<tr>
<td>AgentAdoptionPolicies</td>
<td>urn:magicnet:names:AgentAuthPolicy:1.0:AgentAdoptionPolicies</td>
<td>-</td>
</tr>
<tr>
<td>AgentAccessControlPolicies</td>
<td>urn:magicnet:names:AgentAuthPolicy:1.0:AgentAccessControlPolicies</td>
<td>-</td>
</tr>
<tr>
<td>NS abbreviation</td>
<td>Namespace definition</td>
<td>Std.</td>
</tr>
<tr>
<td>xacml</td>
<td><a href="http://www.w3.org/2001/XMLSchema-instance">http://www.w3.org/2001/XMLSchema-instance</a></td>
<td>XACML</td>
</tr>
</tbody>
</table>

Table 6.1 - Entities and Namespaces Conventions

6.2.1.1 Agent Adoption Policies

The first macro set of policies defines rules for users who want to adopt mobile agents characterized by a specific role. For each user’s role, there is a list of mobile agent roles the user is allowed to adopt. This policy set includes a target defined in Figure 6.2. A new attribute for subject category definition called <&categoryAttribute;> is introduced. Such element is needed in order to distinguish between the two different policy sets. Therefore, in this context, its value is human-user. The same attribute in the second policy set has the value mobile-agent. Requests to PDP shall obviously use this attribute in order to distinguish between user’s and mobile agent’s authorization requests. XACML standards accept the use of self-introduced attributes where required. <&categoryAttribute;> has been defined in order not to override the standard Category attribute, which is used with a totally different meaning. Agent Adoption Policy set contains a collection of standard RBAC <xacml:PolicySet> that defines a set of permissions for each user role. This is achieved by the presence of standard RBAC elements, Permission

\(^7\) It corresponds to XACML 1.0 target composed by <AnySubject>/<AnyResource/> and <AnyAction/>.
Policy Set (PPS) and Role Policy Set (RPS). There is a PPS set for each user role containing all permissions with rules expressed as follows:

- **Target Resource** (&lt;xacml:Resource&gt;) contains the URI (RFC 2396) of mobile agent’s role that is adoptable. Regular expressions can be used here to select multiple roles, since standard function &lt;&amp;function;anyURI-regexp-match&gt; is used;
- **Target Action** (&lt;xacml:Action&gt;) contains the allowed actions for the target mobile agent’s role. Its value is “adoption” and is fixed, since no further operations are expected.

Figure 6.2 – Target Element for Agent Adoption Policies

Moreover, PPS can contain one or more references to other PPSs (&lt;PolicySetIdReference/&gt;) in order to use hierarchical RBAC model. It should be noticed that PPS shall never be evaluated directly. Indeed, they are referenced by RPS, which performs the effective mapping between user roles and permissions. On the other hand, RPS is composed of rules expressed as follows:

- **Target Subject** (&lt;xacml:Subject&gt;) containing the target user role (i.e. &lt;&amp;UserRole;administrator&gt;).

Resource access, in this case “agent adoption”, must be granted only if evaluation of all policies with an agent’s role as target resource (when multiple matches are present) has a permit effect. Since agent roles are defined hierarchically, it is then possible to inhibit a set of agent’s roles, thus explicitly creating a negative rule. This is achieved by the deny-overrides algorithm.

The element in Figure 6.3 is an example of an agent’s role as a resource for user-related policies. Regular expression can be used to describe the set of roles. Moreover, RPS shall contain one (and only one) reference to the proper PPS that assigns permissions to the selected subject. It also holds for the next set of policies.

---

8 Further work could be focused on an extension of expected actions to let the policy cover not only agent adoption (i.e. listing, etc.)
6.2.1.2 Agent Access Control Policies

This is the second macro set of policies needed in order to define proper resource access control for mobile agents, characterized by a specific role, by means of RBAC/XACML rules. For each mobile agent role, there is a list of resources that it is allowed to access when performing a specific action. Agent access control policy set contains a collection of standard RBAC <xacml:PolicySet> that defines a set of permissions for each agent’s role. As shown in the previous case, this is achieved by means of the PPS and RPS elements. There is a PPS set for each mobile agent’s role containing all permissions in terms of resources and actions. Rules are composed as follows:

- **Target Resource** (<xacml:Resource>) contains the URI that identifies the resource compliant with RFC 2396. Regular expressions can also be used here in order to select multiple resources at once by means of the standard function <&function;anyURI-regexp-match>.
- **Target Action** (<xacml:Action>) contains actions for the target mobile agent’s role (i.e. read, write, traverse, etc.).

This resource definition can also be enhanced to fit a particular scenario and to include resources not taken into account in this example (i.e. communications, Web service access, etc.). As supported by the hierarchical RBAC model, PPS can also contain one or more references to another PPS (element <PolicySetIdReference/>) in order to support inheritance between agent roles. Also, PPS shall never be evaluated directly. Indeed, they are referenced by RPS, which provides effective mapping between agents’ roles and their permissions.

RPS is composed using rules expressed as follows:

- **Target Subject** (<xacml:Subject>) containing the target agent’s role (i.e. &AgentRole;role0).

Resource access must be granted only if all policies with a target resource (when multiple matches are present) evaluate to a permit effect. Since resources can be defined hierarchically, it is possible to inhibit a set of resources explicitly creating a negative rule, by means of deny-overrides algorithm. An example of a resource could be the following:

```xml
<xacml:Resource>
  <xacml:ResourceMatch MatchId="&function;anyURI-regexp-match">
    <xacml:AttributeValue DataType="&xml;anyURI">
      &AgentRole;
      com.magicnet.se.DistributedInformationRetrieval.LocalFiles.SharedDocuments.Text
    </xacml:AttributeValue>
    <xacml:ResourceAttributeDesignator AttributeId="&resource;resource-id" DataType="&xml;anyURI"/>
  </xacml:ResourceMatch>
</xacml:Resource>
```

Figure 6.3 – Sample Resource Entry in the XACML Policy File
The element shown above selects the entire sub-tree of a local shared folder for any host belonging to the
defined network (a simple network mask 255.255.255.0)⁹. In the example, grant to access any resource
can be expressed as a file, as in UNIX systems. Anyway, agent resource description can have a format
different from the one introduced here, in fact, as complex as needed.

6.2.1.3 Agent Data Access Control Policies

These policies are used to apply access control to mobile agent’s baggage during mobile agent’s execution at an agent platform. Section 7.2.2.1.2(c) explains the use of these polices during mobile agents execution. In this section we will only explain the content of these polices. A simple data access control policy includes:

a) **Subjects**: in this case identities of other hosts/platforms in the route that can access agent’s baggage.

b) **Resource**: specific data which is contributed by the current runtime node (agent platform) to the agent’s baggage.

c) **Action**: *Read* or *Overwrite*.

d) **Description**: description of the policy.

e) **RuleCombiningAlgorithm**: XACML rule combining algorithm for this policy.

A sample policy based on the above attributes specifies which platforms are allowed to access which portion of the agent’s baggage and with what access rights, e.g. *read* or *overwrite*.

6.3 Mobile Agents Deployment Phase

We have explained trusted mobile agents creation and trusted mobile agents’ adoption phases in Chapter
3 and 4 respectively. In this section we will explain the next phase i.e. mobile agents’ deployment phase. This phase is very important for agent’s execution in the next phase (mobile agents’ execution phase). During this phase, users (those that have adopted agents in the trusted mobile agents adoption phase) are authenticated and then authorized in order to use specific mobile agents, while agents’ code is authenticated before their initialization.

6.3.1 Components in the Deployment Phase

There are four main components in the deployment phase:

---
⁹ For sake of simplicity, in the presented example IP validity is not checked. Anyway regular expressions can be as complex as needed.
6.3.1.1 Policy Administration Point (PAP) Server

Policy Administration Point (PAP) server, as per XACML [11], is the component that creates different XACML policies for a runtime environment. There are three different policy types that PAP manages: (1) agent policies, describing agent’s role authorizations and used by PDP in a runtime environment to decide whether or not an agent is authorized to access a resource; (2) user policies, describing users’ authorization to adopt an agent, i.e. which user can adopt which agents; and (3) agent data policies, describing agent platform authorizations to access agent’s baggage. The first two policies are manually created by a PAP administrator using graphical interface, while the third type of policies is created dynamically, as a PAP server thread, based on the Policy Creation Request, from runtime nodes.

6.3.1.2 Policy Information Point (PIP) Servers

Policy Information Point (PIP) [1] is the system entity that acts as a source of attribute values. We have mentioned above that we have two main RBAC policy types, i.e. agent policies and user policies. Therefore, we have two sources of attributes of agents and users. Mobile agents are treated as subjects that need access to runtime nodes and AF is the repository for the evaluated agents. Therefore, AF acts as the first PIP since it provides information about attributes related to agents. The second source of attributes is IDMS, since all users of the system are registered in the IDMS. Therefore, IDMS also acts as the PIP, since it provides attributes related to users.

6.3.1.3 Policy Decision Point (PDP) Server

“Policy Decision Point PDP is the system component responsible for all security decisions concerning authentication and authorization in the system” [1]. PDP performs two different fundamental functionalities. PDP is first supports user authentication and authorization in the agent’s deployment phase. In that context, PDP server acts as a credential collector and asserting entity that issues SAML tickets to users. Second, PDP handles authentication and authorization requests coming from runtime nodes’ PEPs (explained in section 7.2.1.2.3), evaluates applicable policies, and renders an authentication and authorization decision.

In our system we have designated a trusted component, called PDP Server, to perform these operations. PDP server has three policy repositories: one for users’ authorization (to adopt a certain agent), another for agents’ authorization (to perform some action with sensitive resources), and the third for agent’s data authorization (to read certain portion of agent’s baggage). Moreover, PDP server relies on Local Certification Authority (LCA) to verify user certificates and on Identity Management Server (IDMS) to manage user identities (and decide whether the user can be authenticated or not).

6.3.2 Operations in The Deployment Phase

After AO adopts agents using the process explained in section 5.2.3.2, he/she performs three important operations during the deployment phase.
6.3.2.1 Creation/Update of Policies

After agent’s adoption, AO will send request to PAP, as shown in Figure 6.5 in order to create (if already not created) or update (otherwise) policies for specific agent’s role \( r_1 \) and for the available resources. The details of XACML policies and conceptual methodology have already been explained in section 6.2. PAP creates policies and deposits them to the PDP Server. PAP takes the required attribute values from the PIP (AF, IDMS), creates/updates policies, and distributes them to the PDP Server.

![Figure 6.5 - Creation of Policies](image)

6.3.2.2 User Authentication

After creation of policies, the next step is user authentication. AO and PDP Server authenticate each other using the protocol described in section 5.2.3.3. As the second step, PDP verifies AO’s PK_{AO} certificate from a Local CA and then verifies user information obtained from the Identity Management System (IDMS). After successful verification, PDP server stores locally user’s credentials (C_{AO}). If any of the verifications fail, PDP will not issue a ticket to the user in the next step.
6.3.2.3 Agent Code Authentication (Before Initialization)

This procedure can be treated as a combination of the following two steps: a) agent pre-authentication; and b) user authorization. In the first step, single-sign-on protocol is used. AO sends to the PDP an authentication ticket request (A1_setup_req) containing agent’s credentials:

\[ A1_{\text{setup req}} = \{ C_{\alpha} \} . \]

\( C_{\alpha} \) contains three attributes \{\( H(\alpha_c) \), \( S_{\text{AC}} \), \( PK_{\text{AC}} \)\} used to authenticate agent’s code, thus preventing a malicious AO from executing an arbitrary agent’s code, claiming its correctness from the beginning or modifying a correct code afterwards (type IV attacks). There is no need to specify again \( C_{\text{AO}} \) if AO has already been authenticated, as described in the previous section. Furthermore, \( H(\alpha_c) \) will be used for agent’s code integrity, so that malicious third parties (type IV attacks) or other agents (type II attacks) cannot tamper with its code and the agent cannot cheat the platform (type I attacks). Alternatively, it has been chosen to send the entire PKCS7 PA package: \( C_{\alpha} = \{ \text{PKCS7}_{\text{PA}} \} \). This will enhance system security against type-IV threats. Agent is verified against its role, so that malicious user (third-party attacker with respect to the agent platform) cannot declare an arbitrary role for the adopted agent. Giving wrong privileges to an agent, even trusted, could result in a number of security attacks (i.e. exploiting an unexpected behavior to perform a DOS attack).

As an example, we can consider the following scenario: we assume that AO is authorized to use an agent \( \beta \) characterized by a high assurance level playing the role \( r_{\text{ds}} \), defined as “document shredder”. We can also assume that local policy contains the rule allowing “erase” action at the target “the entire hard disk”. Even though it reflects that \( \beta \) (and AO along with it) has high privileges, it doesn’t mean AO can arbitrarily detach \( r_{\text{ds}} \) from \( \beta \) and assign \( r_{\text{ds}} \) (and resulting privileges) to another trusted agent \( \gamma \). This operation can be disastrous, if \( \gamma \) has lower assurance level (i.e. if \( \gamma \) is a bugged “playlist organizer” agent), since \( \gamma \) could have the opportunity to unintentionally or maliciously harm the system. Therefore, agent’s role must be linked to the agent’s code. Thus, it is not possible to solve this issue sending just the following two 3-tuples:

\[ C_{\alpha} = \{ H(\alpha_c), S_{\text{AC}}, PK_{\text{AC}} \} \]

Since there is no proof of their correlation, the complete PKCS7_PA (content of) package must be sent. Otherwise, it is still possible to attach an arbitrary role to a trusted agent. Alternatively, \( S_{\text{PA}} \) could be used instead of \( H(\alpha_c) \), but this does not make sense, since agent is trusted and the new responsible person is Agent Owner. Moreover, if \( H(.) \) function is collision resistant, it does not offer less protection in terms of integrity than \( S_{\text{PA}} \), as long as it is kept private and not exposed to forgery (i.e. in transmission). Verification of \( C_{\alpha} \) agent credentials by PDP is a multistep process that involves the operations: a) Verification of received \( S_{\text{AC}} \) against received \( H(\alpha_c) \); b) Verification of \( PK_{\text{AC}} \) validity by the Local CA; c) Verification of \( S_{\text{PA}} \); and d) Verification of \( PK_{\text{PA}} \) validity by the LCA.

---

10We have referred different mobile agent security related threats described in [43]. We refer agent to platform” attacks as “Type I” attacks, “platform to agent” attacks as “type II” attacks, “agent-to-agent” attacks as “type III” attacks and “external entities to agent system” as “type IV” attacks.
If all operations produce correct result, PDP checks if AO can really adopt an agent $\alpha$ consulting authorization policies, i.e. if the user has authorization to adopt $\alpha$. Policies are expressed in a simple XACML format [1], i.e. Subject can only be an AO (its unique identifier); Resource can only be an agent’s role; Action is always “adoption”; Condition (of rule) based on agent’s role exact match or agent’s role partial match (with use of regular expressions). The second category Condition is useful to simplify policy definition, since agent’s role is defined in the form of an URI. Finally, PDP issues SAML ticket $tk_1$ to the AO. The ticket comprises a random number, an entity identifier, and a structure with all the proper information to be resolved by the PDP. The ticket also contains an expiration time in order to avoid replay attacks:

$$tk_1 = \{ \text{DN}_{AO}, \text{rnd}, \text{IP}_{PDP}, \text{t}_{\text{exp}} \};$$

PDP caches $tk_1$ in order later to properly authenticate the agent. AO receives $tk_1$, unpacks the agent from PKCS7_PA, and proceeds with its initialization. This step concludes the deployment phase. Figure 6.6 shows components and messages during user authentication/authorization and agent’s code authentication.

![Diagram of components and messages during user authentication/authorization and agent code authentication]

Figure 6.6 – Components and Messages during Agent Code Authentication and User Authentication/Authorization

---

11 It is important group policy rules according to regular expressions or URI instead of defining single rules for each possible agent role of the subset, which could result prohibitive (i.e. “permit adoption to all agents coming from a MagicNET platform”). Such expressivity degree is offered by XACML language only from its version 2.0.
Protection of Mobile Agent’s Code and Baggage

7.1 Mobile Agents Protection

Literature suggests preventive and detective techniques as a solution for mobile agents’ protection during execution at remote host. Preventive techniques include: allowing mobile agent movement only to trusted hosts, setting up organizational policies, using temper-proof hardware, obfuscate agent code, or use of encrypted functions [13]. Detective techniques include use of trusted third party execution tracing or replicating agents [13]. Each of the above techniques has its merits and demerits. The fundamental issue common to all is that they either detect malicious acts upon agents by a malicious host/platform or avoid malicious host. However, there is no mechanism to safeguard agents by catering threats emanating from malicious host/platform in a manner where agents execute in a foolproof process even on malicious host, i.e. even if host wants to spoil agent’s code, but still it cannot.

In this chapter we extend our infrastructure with Mobile Agents Execution Phase. This phase specifically provides mechanism for the protection of mobile agents and their baggage. In addition to simple agent’s baggage protection, we provide layered agent’s baggage protection approach, in which different platforms’ contributions/processing results can be hidden from subsequent hosts in the agent’s route. A host can decide at runtime while executing agents, which server in the route can read its contributions carried by the agents. This approach increases tremendously mobile agents’ distributed processing ability as compared to traditional approach in which mobile agent platform encrypts processing results with agent owner’s public key and thus only the owner could see the final results.

7.2 Mobile Agents Execution Phase

In the previous chapter we have explained mobile agents’ deployment phase. At the end of the deployment phase, mobile agent’s code has authenticated and user authorization has been checked. After successful agent’s code been authentication and user authorization, PDP issues a ticket to a user. This makes user a legitimate owner of the agent. As a next step, AO is ready to initialize an agent and launch it in the network. This is the starting point of mobile agents execution phase. This phase starts when AO initializes agents with default values, adds route information, which runtime nodes will receive agents in the network. Runtime nodes will then authenticate and then authorize mobile agents. During execution
phase agents migrate from one host to another, perform their activities at each host, as per their functional specifications, and return back to the management station.

7.2.1 Components in The Execution Phase

There are only two new physical components for the mobile agents execution phase, i.e. Runtime node and Key Distribution Server (KDS). PDP and PAP servers are also involved in the execution phase and already explained in section 6.3.1.

7.2.1.1 Key Distribution Server (KDS)

Key Distribution Server (KDS) has been introduced in our system in order to achieve agent’s baggage security against the threats emanating from malicious agent platform. Therefore, only authorized platforms at runtime nodes can access particular agent’s platform contributed data. We have used Group Secure Association Key Management Protocol (GSAKMP), RFC4535, for the implementation of KDS. KDS acts as a Group Controller/Key Server (GC/KS) [44]. KDS performs three functions: a) key creation, b) key distribution, and c) key access control. KDS interacts with PDP. PDP acts as an identity provider (asserting party) in the context of the SAML protocol [45], while KDS acts as a service provider.

7.2.1.2 Runtime Nodes

Runtime nodes are network hosts where mobile agents perform their operations. They are used in the agent’s execution phase. Runtime nodes have a physical existence. Each node has six different software components that provide different functionalities. These components are built based on the concept of Component-based Software Engineering, where each component encapsulates a set of related functions. These components are functionally related and perform different processes of a runtime node. The following are the components:

a) Agent Platform (AP)

b) Agent Server (AS)

c) Policy Enforcement Point (PEP)

d) Legitimacy Verification Component (LVC).

e) Data Storage Component (DSC)

f) Group Management Component (GMC)

It is important to mention that runtime node, along with all its components, is secure. Agent platform security has been covered in depth in literature and we are not covering any problem related to Agent platform security. However the code of MagicNET system and all its components, including runtime node, is encrypted. During packaging of the MagicNET system, we create class files, encrypt each class file, calculate hash, concatenate both and then save them with new file extension *.stx. During execution we load .stx files, verify hash and then decrypt the files, and finally execute them [80]. Detailed methodology for software protection (in our case MagicNET System) is explained in [80].
For the sake of simplicity, we will use the term “runtime node” in the rest of the thesis, where individual role of the component is not required to mention and to give holistic view. Figure 7.1 shows a sample runtime node along with its software components. These components perform their individual roles during agents’ execution at a runtime node.

7.2.1.2.1 Agent Platform (AP)

Agent platform provides all services necessary for mobile agent’s execution, such as communication, registration, transport, storage, and discovery services. Mobile agents migrate from one host to another and execute at remote hosts within the environment using the functions provided by mobile agents’ platform.

7.2.1.2.2 Agent Server (AS)

Agent Server is a Java RMI-based component of a runtime node. Agent Server provides three core services:

a. Agent Administration: This service is responsible for accepting incoming mobile agents.

b. Server Administration: This service is responsible for performing administrative functions of the server, like accepting/fetching certificates, logging server operations, and server basic operations like, restart, stop and refresh server, etc.

c. Authentication: this service is responsible for the authentication of a runtime node. During server startup it fetches certificates from a LCA. As the next step, it authenticates with the PDP server using mutual authentication protocol mentioned in section 5.2.3.3. PDP in return sends an authentication ticket $t_{RN_{nodeId}}$ back to the server. This ticket can be used by runtime node at any time to prove its authenticity.

$$t_{RN_{nodeId}} = \{DN_{RN_{nodeId}}, \text{rand}, IP_{pdn}, t_{exp}\}$$

SAML ticket contains a pseudo random number, IP of the PDP, and expiration time. PDP caches this ticket, which allows the PDP to verify it in the future. Communication between PDP and runtime node is encrypted, as both during mutual authentication protocol exchange $KeyEnchiperment$ certificate, so they can encrypt further communication between them.

Agent Server is tightly coupled with other components of a runtime node. Since Agent Server receives important information needed by other components, strong interaction is required.
7.2.1.2.3  **Policy Enforcement Point (PEP)**

Policy Enforcement Point (PEP) [1] is the system entity that performs access control by making decision requests and by enforcing authorization decisions. In our system there is a PEP component at each runtime node. It performs two main functions: a) protects sensitive resources from incoming mobile agents. Mobile agents that want to access sensitive resource must own a valid SAML ticket issued by the PDP server during initial agent’s authentication phase. PEP will use the ticket to create authentication and authorization requests to the PDP, b) PEP receives requests from GMC (mentioned in section 7.2.1.2.6) to access a particular data portion of an executing mobile agent. PEP converts this authorization request to SAML authorization request and sends it to the PDP. It waits for a SAML response from the PDP. PDP evaluates applicable policies and in case of a positive response, PDP issues a SAML authorization assertion to the KDS. PEP receives group keys from KDS and it passes them on to GMC to execute appropriate functions (explained in section 7.2.2.1.2(c)). PEP has low level visibility and can recognize and manage native requests, thus converting requests in the native request format of the XACML canonical form and converting authorization decisions from the XACML canonical form into the native response format [1].

7.2.1.2.4  **Legitimacy Verification Component**

This is an important component located at each runtime node responsible for protection of the node from a set of threats originating from SAML ticket misuse. Legitimacy Verification Component (LVC) receives SAML ticket, calculates actual agent’s code hash and verifies that it matches the hash contained inside the ticket. This ensures that SAML ticket really belongs to the claiming agent and prevents any agent to use stolen tickets (agent-to-agent threat) and prevents AO to attach wrong tickets to agents (external entities-to-agent system threat). LVC only verifies that agent possesses a legitimate ticket, but does not directly verify agent’s code or ticket’s integrity. PDP is the entity that verifies ticket’s integrity, eventually detecting if the ticket has been tampered or if it is invalid, thus preventing any originating misuse. Furthermore, by verifying tickets, PDP also indirectly verifies agent’s code integrity, since agent’s ticket and agent’s code correspondence are ensured by the LVC and any code modification violates such correspondence.

7.2.1.2.5  **Data Storage Component (DSC)**

DSC allows saving data in an XML format and appending it to the mobile agent’s baggage during execution. Every DSC is already equipped with XML schema, which guides runtime node to construct data in a uniform and standard manner. Each runtime node follows the underlying XML schema in order to create XML file which stores data.

The following schema facilitates APs to add their data to mobile agents’ baggage by first providing their identity (Agent Platform ID), then adding data to it with respect to tags. In the future it could be possible to extend this XML to provide more flexibility. Last, but not least, the data itself is appended, which is stored in the form of an encrypted string “content”. Agent carries this XML in the form of a string. Each runtime node can contribute its data to agent’s baggage. Agent’s data is usually the value of agent’s
attributes in the code. As every agent is different from other agents, it is impossible to have unique XML schema that suits every agent. We propose a simple data structure, “content”, which keeps agent’s data in the form of a string. The format of the string will be as follows:

\[
\text{Content} = "\langle\text{Variable}\rangle|\langle\text{value}_1\rangle; \langle\text{Variable}\rangle|\langle\text{value}_2\rangle; \ldots \ldots \langle\text{Variable}\rangle|\langle\text{value}_n\rangle"
\]

Suppose an agent has three variables: X, Y, Z and the values of these variables are A, B, C respectively. Then the final content string will be

\[
\text{Content} = "X|A;Y|B;Z|C"
\]

Hence, in this way a runtime node can contribute multiple of variables in the agent’s baggage as required. These variables can be of a basic or of a complex type. In case of complex type, we serialize the object and store it in a byte form. At the time of retrieval of data, the content string is first decrypted with a specific symmetric key and then the resulting content string is parsed in order to get the variable values.

\[
\text{<?xml version="1.0" encoding="utf-16"?>}
\text{<xsd:schemaAttributeFormDefault="unqualified" elementFormDefault="qualified" version="1.0"}
\text{xmlns:xsd="http://www.w3.org/2001/XMLSchema">}
\text{<xsd:element name="AgentPlatform">}
\text{<xsd:complexType>}
\text{<xsd:sequence}>
\text{<xsd:element name="Data">}
\text{<xsd:complexType>}
\text{<xsd:sequence}>
\text{<xsd:element name="content" type="xsd:string" />}
\text{<xsd:attribute name="tag" type="xsd:string" />}
\text{<xsd:complexType>}
\text{</xsd:element>
\text{</xsd:sequence>}
\text{<xsd:attribute name="id" type="xsd:string" />}
\text{</xsd:complexType>}
\text{</xsd:element>
\text{</xsd:sequence>}
\text{</xsd:schema>}
\]

Table 7.1: XML Schema of The Agent’s Baggage

7.2.1.2.6 Group Management Component (GMC)

GMC at runtime node creates a group of runtime nodes that can access data contributed by this particular runtime node (section 7.2.2.1.2(c) explains the use case). Each runtime node can act as a Group Owner (GO) or Group Member (GM). The choice depends on whether it creates a new group that can access data contributed by this runtime node or whether it acts as a consumer, i.e. wants to read particular portion of mobile agent’s baggage contributed by some other runtime node. The concept of GM and GO is based on the GSAKMP standard [44].

In the context of GMC, it is important to define a group. A group in MagicNET comprises runtime nodes that can access a particular data portion of an agent’s baggage during agent’s executions. Each data portion has a tag, assigned by the DSC, used as a unique group’s name. GMC has the following two functions:
a) GMC acting as GO (referred in the text as GMCGO)
b) GMC acting as GM (referred in the text as GMCGM)

**GMC acting as GO:** GMCGO is the policy creation authority for the group. All potential *runtimes nodes* in a group recognize GMCGO as an entity with authority to specify policy for a group. This information is specified in a policy token (PT) created by the GMCGO. The policy reflects protection requirements for particular agent’s baggage. The GMCGO determines security rules appropriate for the data being protected by the group keys. These rules are reflected as an XACML policy at the PDP (section 7.2.2.1.2(c) explains in detail). GMCGO sends PT to KDS to create a new group.

Policy token (PT) contains
- Identification: unique identification of group data and PT:
- Access Control Rules: These rules specify who can access encryption key for the baggage.

The PT is sent to a KDS. KDS will verify the signature of the PT in order to ensure that it comes from a valid GMCGO. After verification, KDS will create a group and a random symmetric key for this group and save the key in a table corresponding to a *runtime node*’s ID, as shown in Table 7.2. This key is then sent back to a GMCGO at the *runtime node* to encrypt data portion of agent’s baggage.

<table>
<thead>
<tr>
<th>Data Tag</th>
<th>Runtime Node</th>
<th>Symmetric key</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data 1</td>
<td>RN-1</td>
<td>K1</td>
</tr>
<tr>
<td>Data 2</td>
<td>RN-2</td>
<td>K2</td>
</tr>
<tr>
<td>Data 3</td>
<td>RN-3</td>
<td>K3</td>
</tr>
<tr>
<td>Data 4</td>
<td>RN-4</td>
<td>K4</td>
</tr>
</tbody>
</table>

*Tables 7.2: Data-RuntimeNode-Key Mapping*

**GMC acting as GM:** GMCGM has two responsibilities: a) GMCGM verifies that PT is signed by the recognized GMCGO and KDS is authorized to send group keys, as specified in PT, and b) sends request to a PEP to fetch group keys, in order to access required data. If key retrieval request is generated, PDP receives authorization request from PEP. PDP evaluates applicable policies, based on the *Target* information received from the PEP in the form of an authorization request, and renders decision ‘allow’ or ‘deny’. It sends authorization assertion to KDS. Depending on the decision, if keys are allowed to be delivered, KDS sends keys to the requesting PEP. The PEP passes them to the GMCGM in order to open specific portion of agent’s baggage.

As per the concept of SAML, PDP acts as an identity provider, while KDS acts as a service provider, while GMCGM at *runtime node* acts as a subject.

**7.2.2 Operations in The Execution Phase**

**7.2.2.1 Agent’s Life Cycle**

There are eleven stages in agent’s life cycle described in section 3.1.2. Agent creation and owning phase related stages are already covered in the previous chapters. We will start from agent launching stage in
agent deployment phase and explain each stage role in the process of protection of a mobile agent and its baggage.

7.2.2.1.1 Agent Launching

AO, after receiving a ticket from the PDP, in the last phase sets $tk_1$ as initialization parameter, so that $\alpha$ is linked to the $tk_1$ for all sensitive operations. AO initializes (adds signed agent route info, etc.) and then launches an agent $\alpha$ from the AO station (Management Station, MS). MS exact role is explained in section 8.2.2.1. As a next, but not required step, we can imagine that $\alpha$ migrates from the local AP to another runtime node AP and may eventually return the results back to the AO, after visiting all hosts in the route.

Usually, an agent consists of three components: Header, Code, and Data. Header contains information, like hash to assert agent’s integrity. Code contains the code of an agent (agent functionality), and Data carries any data contributions (baggage) made by different runtime nodes during agent’s execution. Prior to agent launching, AO creates hash of an agent and then signs hash with his/her private key $SK_{AO}$. AO then attaches its identity $AO-ID$, its public key certificate $PK_{AO}$, and signature $SAO$ to agent’s code, packaged in the form of PKCS7 SignedData. This process covers the integrity issues of an agent. AO then creates a random content encryption key for an agent to be encrypted. For each recipient (runtime node) content-encryption key is encrypted by recipient’s public key. Recipient’s public certificates are accessible to AO from the LCA. This provides code security mechanism, as only the intended runtime nodes can decrypt the code and execute it. This package is in the form of PKCS7 EnvelopedData. The complete package is in the form of PKCS7 SignedAndEnvelopedData, denoted as PKCS7$_{AO}$. Figure 7.2 shows the structure of the PKCS7$_{AO}$ package.

![Figure 7.2: Protected Mobile Agent – PKCS7 Signed and Enveloped Data Package](image)

7.2.2.1.2 Agent Hosting and Execution

When an agent has been launched into a network, the agent is received by a destination runtime node (let’s assume that the first node is denoted as RN-1) which provides hosting environment for mobile agent’s execution. Mobile agent migrates from one runtime node to another, specified in its route. Once runtime node receives an agent, it executes it by first opening the PKCS7$_{AO}$ package, verifies code’s
authenticity, decrypts agent’s code, and then executes it. We believe our solution is better than any other
solution, because our encryption of agent’s code prevents unauthorized runtime nodes from opening or
modifying agent’s code. Only authorized runtime nodes, having corresponding private key, can decrypt
agent’s code. Once the code is decrypted, runtime node executes the byte code by using MagicNET Class
Loader. So we have achieved yet another security mechanism, i.e. Code Execution Protection. The code
will be loaded by the class loader into the JVM, and since after it is loaded in the JVM, even runtime node
is unaware of agent’s execution and code. Moreover, if the code is already in the JVM, it cannot be
tempered or illegally modified. It is important to mention that we have not used traditional java class
loading concepts. We have used eclipse implementation of OSGi framework [85] to dynamically load
agents. The use of OSGi framework allow us to install and start encrypted agents (in form of eclipse plug-
s), dynamically without restarting runtime node.

However, here the question is how runtime node allows remote code to execute without any verification
and without knowing how it will execute. Before execution, visiting mobile agent is being authenticated
and authorized by the PDP Server using processes explained in the next two sections.

a) Agent Authentication (after Initialization at Runtime Node)

The first step before mobile agent’s execution at each AP is mobile agent’s authentication. It starts when
an agent α needs to execute at the AP and perform some sensitive operation using some platform’s
resource (i.e. to write a file “doc.txt”, where AP is configured to treat that file as sensitive and to protect
file access). For this purpose we use XACML, SAML, and RBAC standards for human user
authentication and authorization of mobile agents. An agent α requests PEP to perform an operation op1
using local resource resx. An ad-hoc runtime node component, called LVC, receives the ticket tk1,
calculates \( H(\alpha_c) \), and verifies whether it matches \( H(\alpha_c) \) contained inside the ticket. If that is not the
case, it denies the request.

This step, involving LVC, is required in order to ensure protection against ticket theft, ticket misuse, or
agent’s code alteration. PEP receives SAML ticket tk1, operation to be performed op1, and target
resource resx. Then it translates a native request format into XACML canonical form, acting as a context
handler [1]. It translates previous objects respectively to subject, action, and resource. Then PEP sends
SAML authentication request A1req:={tk1} to the PDP. PDP verifies that all tk1 parameters properly
match and returns a SAML authentication assertion. PEP caches the reply received from the PDP in order
to avoid authentication of further requests from the same agent.

b) Agent Authorization

After receiving positive SAML authentication response, PEP sends SAML authorization request (A2req)
to the PDP:

\[ A2req:={tk1, Target( Resource, Subject, Action)}; \]

PDP verifies tk1 validity by comparing it against the cached version and then, according to the role r1
and the local XACML policies, decides to grant or deny access. Policies are in the form of the standard
XACML Profile for Role Based Access Control (RBAC) [30], explained already in Chapter 4. PDP communicates its decision by sending SAML authorization response \( A2_{res} \). PEP receives \( A2_{res} \) and grants or denies operation to the target resource, based on the received response. After these two steps, it has been established that agent’s code is fully trusted and it has the required permission to execute on the runtime node. After this, mobile agent executes its desired functionality at the runtime node, creates results, and then migrates to the next runtime node, specified in its route. Figure 7.3 shows components and messages during agent’s authentication and authorization at a runtime node.

![Figure 7.3: Components and Messages during Agent Authentication and Authorization](image)

c) Mobile Agent Baggage Protection

During execution at runtime nodes, agents collect or receive data from some node, required by another runtime node in the path or to be delivered only to the AO. In that case, it is required to encrypt data in such a way that it can be read only by authorized runtime nodes or by the AO. Authorized runtime nodes are members of a group that can read this particular data. There can be two scenarios regarding protection of data:
The first scenario is that runtime node wishes to hide its data from other runtime nodes (the same as GSAKMP group creation mechanism), but wants also to allow some of the runtime nodes in the same domain to see the data. In this case runtime node, using GMCGO, will send SAML based authentication request containing authentication ticket (tkRN, which was previously received by Agent Server at runtime node during its initialization, mentioned section 7.2.1.2.2) to the PDP. PDP verifies the integrity of the ticket and matches it with the ticket it had stored in its cache. Upon successful verification, PDP issues SAML authentication assertion, which is then forwarded to the GMCGO. Now, runtime node, using GMCGO will send a PT (created by it) and authentication assertion received from PDP to KDS to request a symmetric key to encrypt data. KDS also stores PT locally for future use (explained in scenario 2). KDS will process this information and will generate a new symmetric key, specific for this runtime node. KDS stores this key in a table along with the data tag name, i.e. data1; so that if other runtime nodes ask for a key to particular data, it can be given to them if they are authorized to access it. At this time KDS also sends PT to all group members specified in the PT. AO will be the default group member in every PT. Authentication between KDS and group members will be performed in a same way as described in section 5.2.3.3.

Figure 7.4: Scenario 1, Agent Baggage Protection
This data tag name can also be considered as a group name. KDS then returns the symmetric key, generated for this particular data tag, to the runtime node. Runtime node, using GMC_GM, encrypts the data with that symmetric key. Meanwhile, KDS sends the received PT to the PAP and PAP generates XACML policies, based on the attributes contained in the PT. These policies are then distributed to the PDP. Now, as the final step, KDS sends the PT to other runtime nodes, specified in the PT. AS is used to receive the PT at runtime nodes. Figure 7.4 shows different components and their interactions in the first scenario.

In the second scenario, when a runtime node wishes to use or read data encrypted by another runtime node, it asks for a symmetric key from the KDS (the same as GSAKMP key download feature) in order to decrypt particular data portion in an agent’s baggage. GMC_GM asks PEP at the runtime node to send request to the KDS for the required key. KDS redirects PEP to the PDP for authentication. PDP request appropriate attributes for runtime node authentication. PEP sends ticket (t_kRN-) received during runtime node initialization, mentioned in section 7.2.1.2.2. It is important to note this authentication is runtime node authentication. PDP compares the ticket with locally cached ticket and returns SAML authentication assertion. PEP then sends data tags that need to be accessed and action to be performed in the form of authorization request. PDP evaluates applicable polices and returns authorization assertion.

![Figure 7.5: Scenario 2: Agent Baggage Protection](image)

In the case of a positive response, PEP sends this authorization assertion and the PT, received by runtime node as a result of group creation in the previous scenario to the KDS. KDS compares the PT stored locally with the received PT in order to identify group membership. If it matches, it verifies authorization assertion as it is signed by PDP. On successful verification of authorization assertion, KDS fetches requested keys and returns to PEP. PEP forwards these keys to GMC_GM. KDS sends the requested keys to the PEP. These keys are then given to the GMC_GM which decrypts data tags using the received keys.
and performs the required operation with the decrypted data. Agent then continues its execution and performs the desired functions, as it was supposed to perform. Figure 7.5 shows the second scenario.

Table 7.3 shows data-runtimeNode-group mapping by PDP. Data1 contributed and encrypted by Runtime Node 1 (RN-1) will only be allowed to be seen by Runtime Nodes 1, 2, 3 and Agent Owner. Data2 contributed and encrypted by Runtime Node 2 can only be used by Runtime Node 1, 4 and Agent Owner. Data3 contributed and encrypted cannot be seen by any other platform, but the Agent Owner. Three runtime nodes as group owners provide policy tokens. Each policy token is saved with its group information.

<table>
<thead>
<tr>
<th>Data Tag</th>
<th>Runtime Node Contributed</th>
<th>Allowed Group Members</th>
<th>Policy Token</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data1</td>
<td>RN-1</td>
<td>RN-1,RN-2,RN-3, AO</td>
<td>P1</td>
</tr>
<tr>
<td>Data2</td>
<td>RN-2</td>
<td>RN-1,RN-4, AO</td>
<td>P2</td>
</tr>
<tr>
<td>Data3</td>
<td>RN-3</td>
<td>AO(Agent Owner)</td>
<td>P3</td>
</tr>
</tbody>
</table>

Table 7.3: Data-RuntimeNode-Group Mapping

Let’s assume that the requesting runtime node is authorized to access data1 and data3, but not data2, which means that requested runtime node is a group member of data1 and data3 (refer to Table 7.3). KDS retrieves the keys for those particular data tags. KDS retrieves the symmetric keys for data1 and data3, i.e. K1 and K3.

7.2.2.1.3 Agent Transfer

After agent’s execution finishes at some node, the next step is a transfer of the agent to the next runtime node. Before the agent is sent to the next runtime node, current runtime node checks whether it made any contribution to the data baggage or not. If data has been added during execution at this runtime node, then it was either encrypted or left in clear. At this time, if data contribution has to be made by the AP, then GMCGO at the runtime node follows the first scenario explained above: it gets the key and encrypts the data and adds it to the agent’s baggage.

After data \( D_{RN-1} \) has been encrypted and added to the baggage by the runtime node RN-1, the agent is now ready to be transferred to the next runtime node in the route, i.e. RN-2. As data has been changed in the baggage, RN-1 creates a new PKCS7 package, i.e. PKCS7RN-1. This new PKCS7RN-1 package contains both data, i.e. baggage, and initially received PKCS7 package from the Agent Owner, i.e. PKCS7AO signed by the RN-1. For the enveloping process of the PKCS7RN-1, a content encryption key is generated, i.e. \( K_{RN-1} \) and already signed content is then encrypted by it. Now, \( K_{RN-1} \) is then encrypted by the public key of RN-2, i.e. \( PK_{RN-2} \).

Upon transfer, RN-2 receives the PKCS7RN-1 package. It decrypts the package using its private key, retrieves the content encryption key \( K_{RN-1} \), and decrypts the contents. It then extracts contents, i.e. encrypted baggage and PKCS7AO, both of which are signed by the RN-1. PKCS7AO contains agent’s code, which is recovered by RN-2 using its private key. RN-2 is now ready to execute the received mobile agent. Figure 7.6 shows content of PKCS7 package at different runtime nodes.
7.2.2.1.4 Agent Return

At this point, agent had traversed every single runtime node specified in its route. Now, agent’s baggage has different layers of data in it, some are encrypted and some are not. Agent returns back to the AO. If AO has to see all the data encrypted by multiple platforms, it interacts with the KDS. AO authenticates itself with the KDS using the same mechanism as in the step 5.2.3.3.

After authentication, AO presents PT to the KDS in order to prove its membership in the group. As the next step, KDS asks PDP for authorization, as AO has been added to the group by default. PDP authorizes KDS and KDS sends the keys of all runtime nodes to the AO. AO then performs integrity check on an agent by verifying hash. This process is performed in order to make sure that agent (agent’s code) has not been changed during its roaming through network. There is no possibility that agent’s code is tampered, because of our protection mechanism, as mobile agents travel only encrypted and they are decrypted only when already loaded in the JVM. Even if malicious runtime node make the copy of agent code and
change it, but it is of no use as it will not able to recreate signedAndEnveloped package for the rest of runtime nodes nor able to sign the route, as it is not AO. Therefore, even malicious runtime node cannot tamper with it. Eventually, AO retrieves the complete agent’s baggage and then performs any required task using agent’s baggage.

### 7.2.3 Secure Communication between Mobile Agents

Previous section 7.2.2, explained in detail mobile agent’s code and data protection during the execution phase. Agent-based architecture has been used to demonstrate mobile agents’ protection mechanisms. According to this architecture, mobile agents are launched from a host machine, called Agent Home (in our case MS). Agents traverse predefined hosts specified in their route, execute remotely, and eventually return to Agent Home. This architecture bypasses agent-to-agent communication and prevents mobile agents from threats originating from malicious agents, as mobile agents are not required to communicate with other mobile agents during their execution at remote host. However, it significantly deprives mobile agent-based applications from the benefits of mobile agents’ paradigm. Mobile agents being a “social entities”, communicate with each other in order to get performance and goal-related benefits. For example, inter-agents communication is essential when an operation is distributed between different agents in order to enhance system throughput through parallelism. In this situation, different agents need to share each others’ intermediate processing results, regardless of the agent platform on which they are executing. Agents may collaborate with other agents in order to accomplish complex tasks. Collaboration may be with static agents (agent platform), with the members of agent’s own team, or remotely, with other independent agents, executing at other agents servers (agent platforms). Mobile agents’ collaboration is an essential aspect of a mobile agent system and the prerequisite for the effective and secure collaboration is to mitigate all security threats during mobile agents’ communication. Among those threats masquerading, repudiation, denial of service, and unauthorized access [43] are of major concern.

Therefore, in order to get benefits of mobile agents’ collaboration, we have extended our system to support secure agent-to-agent communication. We propose an authentication protocol designed for agents’ identity verification. It is based on the concept of mutual authentication, i.e. both communicating agents authenticate each other before their communication.

All runtime nodes (agent platforms) mutually authenticate each other using protocol mentioned in the section 5.2.3.3. All communications between runtime nodes are encrypted and signed. The protocol shown in Figure 7.8 provides mobile agent-to-agent mutual authentication. Each agent possesses a ticket, which it presents to the AP in section 7.2.2.1.2(a). The same ticket is being used for agent-to-agent mutual authentication.

The authentication protocol designed for agents’ identity verification is based on the concept of mutual authentication, i.e. both communicating agents will authenticate each other. Suppose there are two agents, A and B, at two different runtime nodes, RN-1 and RN-2 respectively. The following are secure communication requirements that must be ensured for each participating agent: security against masquerading, unauthorized access, denial of service, and repudiation.
In order to fulfill those requirements for secure communication between agents, the following authentication protocol is designed:

1. Agent ‘A’ generates a message containing four elements: (a) ticket $TK_{AgentA}$, (b) randomly generated agent’s initialization $ID_{AgentA}$ created when agent was registered with the host agent platform, (c) distinguished name $DN_{RN-1}$ of the runtime node RN-1, and agent’s role $RAgentB$. Since RN-2 can have multiple agents running simultaneously, that is why $RAgentB$ is needed in order to distinguish the agent that agent ‘A’ wants to communicate to. This message is then enveloped into PKCS7 SignedAndEnvelopData package, $PKCS7_{A(RN-1)}$, in order to ensure message confidentiality and integrity. Agent Platform (AP) then sends this message to RN-2.

$$PKCS7_{A(RN-1)} = \{ TK_{AgentA} , ID_{AgentA} , R_{AgentB} , DN_{RN-1} \}$$

2. RN-2, upon receiving the message, decrypts the package $PKCS7_{A(RN-1)}$. It then retrieves the content of the package. RN-2 checks the certificate and $DN_{RN-1}$ in order to verify whether the package came from the RN-1. Once the verification of RN-1 is completed, RN-2 retrieves the ticket $TK_{AgentA}$. In order to check the integrity and validity of this ticket, PEP RN-2 sends SAML authentication request to the PDP: $Req = \{ TK_{AgentA} \}$. PDP verifies if $TK_{AgentA}$’s parameters properly match and returns SAML authentication assertion.

Once, the identity of RN-1 and authenticity of the $TK_{AgentA}$ have been verified, AP at RN-2 sends back a PKCS7 package to the RN-1, with the contents: pre initialization ticket of agent ‘B’ $TK_{AgentB}$, distinguished name of RN-2: ($DN_{RN-2}$), Agent A’s random initialization $ID_{AgentA}$, as received in the $PKCS7_{A(RN-1)}$, and Agent B’s random initialization $ID_{AgentB}$. This package is denoted as $PKCS7_{B(RN-2)}$.

$$PKCS7_{B(RN-2)} = \{ TK_{AgentB} , ID_{AgentA} , ID_{AgentB} , DN_{RN-2} \}$$

3. RN-1 receives the package $PKCS7_{B(RN-2)}$ and decrypts it. Once the package is decrypted, it checks the certificate and $DN_{RN-2}$ in order to verify whether the package came from RN-2. It then verifies the authenticity of the ticket $TK_{AgentB}$, as explained in the previous point. If the verification is successful, agents start the communication.

Figure 7.8 shows mobile agent-to-agent authentication protocol. An initiator and responder runtime node interacts with PDP in order to accomplish mutual authentication.
Figure 7.8: Agent-to-agent Authentication Protocol
Part 2 and 3 described the security infrastructure for mobile agents. Part 4 explains network security system based on mobile agents. Chapter 8 explains the conceptual architecture of the proposed network security system. In this chapter we will also give the motivation for the use of mobile agents’ technology for network security. This chapter gives overview of an application that is developed on the top of our mobile agents security infrastructure, explicitly it means it uses trusted and secure mobile agents that are built using our security infrastructure.
Network Security System

8

8.1 Network Security Problems

8.1.1 Intrusions and Damages

Security and protection of computer networks and their resources is one of the most important IT activities today. Most organizations no longer take for granted that their networks and applications are secure and, therefore, use all kind of protection tools and products. But, even after installing various protection mechanisms, performing continuous monitoring of security logs, and running extensive penetration tests, security personnel spend considerable time chasing incidents, preventing penetrations or solving problems after intrusions and damages. More or less, everybody has already realized that “secure the perimeter” approach does not prevent the tide of incidents, intrusions and damages, because current techniques and products do not provide effective solutions [46]. In spite of all the efforts, we are almost daily witnessing intrusions, damages, and stolen valuable Government and/or corporate information.

Over the last several years, the trends and styles of intrusions have been changing [47]. Intrusion profiles have enhanced from simple methods, like tracing and recovering passwords, social engineering attacks [48], and exploiting simple software vulnerabilities to more sophisticated methods, like exploiting protocol flaws, defacing web servers, installing sniffer programs, denial of service attacks, distributed denial-of-service attacks, or developing command and control networks using compromised computers to launch attacks. CERT Coordination Center confirmed in the recent “CERT/CC Experiences Vulnerability Report” [47] there has been significant exponential increase in discovered vulnerabilities: from 171 in 1997 to 7236 in 2007. This increase in vulnerabilities and intrusion profiles has also dramatically increased the number of security incidents in the past few years. These statistics show an alarming situation in which expertise of intruders is increasing, complexity of network and system administration functions is increasing, ability to react fast enough is declining significantly and along this, vendors continue to produce software with inherent vulnerabilities. In addition to direct attacks and penetrations by humans (hackers or insiders), one of the additional rising problems in today's networks is the existence of malicious bots and bot networks [49]. Most botnets are created in order to conduct malicious actions, such as conducting Denial of Service (DoS) attacks, stealing user identities, installing keyboard loggers to record keystrokes, or generating e-mail spam.
8.1.2 Current Solutions and Their Weaknesses

Several ID/IP research solutions and many commercial products emerged in the past provide protection against intrusions at a host or at a network level. These traditional solutions, like antivirus, firewall, spyware and authentication mechanisms provide security to some extent, but still face the challenge of inherent system flaws, OS bugs, and social engineering attacks. Back in 1980, James Anderson [50] proposed the concept of intrusion detection. Then in 1988, three IDS models have been proposed based on the approach to detect intrusions: Anomaly Detection, Misuse Detection, and Hybrid Detection [51]. Anomaly Detection IDS produces high rate of false positives. Misuse Detection produces have smaller number of false positives, but the problem is that signature databases need to be regularly updated, as their detection capability is based on them.

One of the major problems with current IDSs is that they cannot detect and respond to new attacks in real time, because most of them require updates of attack signatures, usually provided by network administrators. It is really very difficult for network administrators to analyze large logs generated by network traffic, to identify attacks, and to respond to them in real time. The consequences are new, often distributed attacks, based on the window of opportunity for an attacker, because of the delay in attack identification and response by network administrators [47]. Our system based on mobile agents solves very effectively this problem.

Another serious problem with the current ID/IP systems is that they produce large logs, which cannot be efficiently used and utilized. With so many available security solutions, both open source and commercial products, the problem is not to obtain security related data, but to reasonably process the large amount of data. Those solutions, in order to be effective, report several thousand ‘events’ a day, the number rising to near absurd totals in the government, commerce and also open university networking infrastructures. This situation quite clearly raises a number of issues. It becomes practically impossible to analyze every logged snippet of information due to the sheer volume of collected data. Consequently, more critical attacks may go unnoticed. Security administrators either never process relevant attacks data or process them too late. Security analyst must have an almost superhuman speed, capabilities and understanding of the information being presented [52].

In addition, it is generally accepted today that software has inherent security vulnerabilities [53]. Usually, system and network administrators do not discover these vulnerabilities in real time, because of the large size of their networks and their inability to have access to all the information about the discovered vulnerabilities. In fact, it should be advantageous that, as soon as the patch is released, it is installed where it is required. Our system is capable to detect vulnerabilities due to new installed software (explained in section 8.2.3.1.2), report existing vulnerabilities and also automatically fetch and distribute patches to their target machines.

8.2 Network Security System based on Mobile Agents

The major drawback in all solutions available at the time of this research is their methodology of protection. First, the methodology is reactive: reaction starts when there is already an intrusion in
progress. Second, there is no learning mechanism at a network level to study and learn about intrusions and to provide protection against the same intrusion to the rest of the network. Third, there are no preventive measures taken against foreseeable threats that can turn into intrusions, based on existing vulnerabilities in the system, which become the cause of zero-day attacks. There is a need to revisit existing methodologies with an intention to improve them by applying the concept of immune system to achieve comprehensive security for information systems. In this chapter, we present the network system that functions in six stages to secure information systems against intrusions. Our system is based on the concepts of prevention, deterrence, detection, response, and learning. We used mobile agents along with different ID/IP products (called sensors) to achieve the desired results of our comprehensive security system.

![Security Solution along Attack Time Line](image)

Figure 8.1: Security Solution along Attack Time Line

Figure 8.1 shows a simple case of intrusion. The first line shows attacker’s perspective, i.e. the steps attackers usually perform when trying to compromise a host. First, they look for vulnerabilities on a host that can be exploited. Once they find the vulnerability, they exploit and violate the access control. After the attack is launched against the host, we term this attack as successful intrusion. Attacker looks for next target after successful intrusion. In order to address every move of an attacker, we should have multifaceted or defense-in-depth approach that provides layered protection. Hence, we have proposed six different mechanisms, combined one after the other in order to protect hosts in a network.

As the first step, our system detects and eliminates vulnerabilities, which are usually being exploited by intruders. Second, our system provides access control mechanism that blocks all illegal accesses based on predefined security policy. However, it is possible that attacks can overcome the above two preventive measures. Therefore, as the third step, we use mechanisms to detect intrusion attempts in real-time based on use of Intrusion Detection Systems (IDS). As the fourth step, an Intrusion Response System (IRS) is activated in response to alerts generated by the IDS in order to limit intrusions and damages. The fifth step is use of Post Intrusion Vulnerability Analysis mechanism, which detects the cause of intrusion by identifying the vulnerability that had been exploited by the attacker. Once that specific vulnerability has been identified, the system applies preventive measures against the likely exploitation of the same vulnerability. Finally, all hosts in the network must be kept up-to-date with respect to security...
technologies, security configurations, and updates installed. The Figure 8.1 shows our approach using the attack timeline based on the immune system concept.

There are numerous research papers and reports emphasizing and suggesting security for mobile agents and their platforms, many of them mentioned in Chapter 2. The proposed system is based on the security infrastructure for mobile agents described in Chapters 3 to 7. The described network security system is a useful and effective application based on mobile agents. It is expected that mobile agents, as the new computing paradigm, will show several advantages compared to the current network security technologies and products: efficient discovery of vulnerabilities, accurate and prompt monitoring of events, filtering and analysis of system logs, and intelligent decisions for local (host) or global (network) environments, reactions in real-time to undesirable, illegal or unauthorized events, and simplified network security management. Because of these effects and advantages, it is expected that the network security system based on mobile agents will improve effects of security products and technologies used in current computer systems and networks.

8.2.1 The Functions of The Proposed Solution

We will cover the following four major functions of the network security system:

8.2.1.1 Analysis of Vulnerabilities and Software Patches

The first function of mobile agents is to assist network and system administrators to analyze their installed IT components in the network and to detect potential vulnerabilities. For this purpose mobile agents use three techniques: (a) vulnerabilities identified and reported in various vulnerability databases, (b) their own testing of new, undetected vulnerabilities, and (c) creating and using sophisticated SNORT vulnerability rules.

To perform these functions a team of mobile agents is assembled and launched, manually or automatically – prescheduled, to scan vulnerabilities at remote hosts in a network. Mobile agents reach remote host, get their profile, and bring back the results. These profiles are then compared with entries in the vulnerability databases (NVD, OSVDB). At the same time the agents handle software patches for those vulnerabilities.

8.2.1.2 Intrusions Detection

The second function of mobile agents is to detect malicious activities in the network. For this purpose mobile agents provide three groups of functions: (a) analysis of large volume of data in various logs and generation of effective reports, (b) detection of and reaction to host-based intrusion attempts in real time, and (c) detection of and reaction in real time to distributed intrusion attempts.

In order to perform these functions, different teams of mobile agents are assembled and launched, among them a team capable to analyze logs on remote hosts generated by sensors like, SNORT, Osiris [54][63], and Microsoft Windows firewall. Mobile agents reach remote host, analyze logs and in case of serious problems, report back to the security administrator. At the same time, the second team of mobile agents
reaches remote host, stays there and continuously monitors and analyzes SNORT logs and activities. In the case of any suspicious activity, they immediately call other agents for reinforcements. Finally, the third team of mobile agents detects activities that are below intrusion threshold, but cannot be ignored (explained in section 8.2.3.2 through Doorknob-Rattling Attack example). Agents analyze system logs, infer and correlate information from different hosts, all with an intention to identify distributed intrusion attempts.

8.2.1.3 Intrusions Response

The third function of mobile agents in our system is to timely react to reinforcement requests in case of intrusion attempts. In this case a team of mobile agents is automatically dispatched to a remote host under attack. The team is fully aware of the type of the attack, as reported by the static agents that discovered the attack, and what are the required reactions and responses. Mobile agents can act on their own behalf or they can also mutually cooperate in order to instruct or guide individual tools on remote hosts to appropriately apply the response. One of the immediate actions the agents perform is to close the port being used for an attack. They also immediately update local firewall configuration tables. In addition, they migrate to other hosts in order to perform the same preventive action(s) in case of a distributed attack.

8.2.1.4 Network Security Management

The fourth function of mobile agents is to assist network administrators by keeping the network up-to-date against new potential threats. For this function, a team of mobile agents is launched, which roam through the network, visit different systems, analyze and install different services or security software. For this purpose mobile agents use the following techniques. (a) connectivity and status of remote hosts are checked and reported; (b) configuration of remote hosts are checked and recorded; (c) security configuration management related tasks are applied; (d) mapping of SNORT rules and identified vulnerabilities. In order to perform these functions, a team of mobile agents is automatically assembled and launched, they interact with system logs and tools installed at remote hosts, and perform the desired security management related tasks.

8.2.2 The Components of the System

The system has five components: Figure 8.2 shows these components.

8.2.2.1 Management Station

Management station (MS) is the component of the system used by the security administrator (Agents’ Owner) in order to perform various management functions with mobile agents. It provides GUI to the administrator to perform various tasks, like launching of agents, communicating with agents, and receiving reports from agents. Moreover, the same station is also used to manage two servers, vulnerabilities database server and intrusions server. MS has associated server as well, we call it Administrative server/Management server as well in the text.
8.2.2 Mobile Agents Platforms

Mobile agents’ platform must be installed at each server (*runtime node*, mentioned in section 7.2.1.2) and at each workstation (*runtime node*) where mobile agents can arrive and execute. There are two other components as well at each *runtime node*:

- Agents’ Server: a server whose function is to accept and execute mobile agents and to provide environment for their execution;
- IDS/IPS Components: used to detect attacks and malicious activities. IDS/IPS components include Snort, Osiris, Nessus™ and native firewall. We refer to these IDS/IPS components as sensors in our system. These components are in fact static agents that perform certain predefined tasks.

8.2.2.3 Mobile Agents

Mobile Agents are the key component of the system. Different teams of mobile agents are manually or automatically launched from the management station. They perform their designated tasks at remote hosts and bring back their results to the management station. They can also reside at remote hosts and continuously perform their monitoring and analyses tasks. Different types of mobile agents have been described in section 8.2.3 to achieve the desired functionality of our network security system. Security Administrator at the management station adopts trusted mobile agents using the methodology explained in Chapters 3 to 5. We assume that in our network security system administrator has already adopted all the desired agents. Hence, security administrator acts as mobile agents’ owner.
8.2.2.4 Management Servers

Our system has two types of management servers: (a) Intrusion Server used to response to reinforcement requests from different agents, in case of an attempt of an intrusion, (b) Vulnerabilities Server used to collect up-to-date information about different vulnerabilities and patches. It also hosts vulnerabilities database. It is used to response to different mobile agent team requests for vulnerability analyses at target runtime nodes.

8.2.2.5 Vulnerabilities Database

Vulnerabilities database contains up-to-date information about the latest vulnerabilities. The information about different vulnerabilities is continuously updated from various sources, like OSVDB, NVD, Security Focus, and Nessus™. System Manager can also manually add new vulnerabilities into the database in order to further strengthen the vulnerability database. We call our vulnerability database “Comprehensive Vulnerabilities Database” (CVDB).

CVDB contains detailed information about all vulnerabilities. Any vulnerability has a number of attributes, which describe every aspect of it. They include static as well as dynamic information about the vulnerabilities. Static information is the information which is available in advance when a database entry is created, while dynamic information is added continuously, based on available patches and hosts information about different vulnerabilities on them. CVDB scheme comprises two different levels of information. The first level includes four tables that can be considered “static”, loaded when a new vulnerability is discovered and the related general information is available. Different tables of the CVDB schema are shown in Figures 8.3 and 8.4. Each vulnerability instance provides CVSS score and vector, vulnerability description, online references, eventual patches, etc, while dynamic information is built over period of time and it is based on the patches available for different vulnerabilities and vulnerable host in the network.

Figure 8.3: CVDB Tables (static)

Figure 8.4: CVDB Tables (dynamic)
8.2.3 Operations of the System

The operations of the proposed system are divided into four groups. Figure 8.5 shows the layered architecture of the system. These groups are as follows:

a. Vulnerability Analysis (VA)
b. Intrusion Detection (ID)
c. Intrusion Response (IR)
d. Security Management (SM)

The proposed solution is tightly coupled with a number of mobile agents which provide flexibility, scalability, platform independence, and reliability to the system. Moreover, mobile agents can execute asynchronously, autonomously and can dynamically adapt in different stages of attacks timeline and thus they provide robust security mechanism. At the abstract level, these mobile agents are acting as middleware between ID/IP components and the overall system’s core functionality.

![Figure 8.5: Layered Structure of Network Security System](image)

Figure 8.5 shows conceptual abstract structure of network security system. Our system can be deployed on any heterogeneous system. On each system we will install different ID/IP components, e.g. SNORT [54], Osiris [63] and etc. Mobile agents provide inputs and take outputs from these components. Mobile agents can take themselves actions on the outputs of the ID/IP components or they can forward them as a
message to administrative interface. Administrative interface will be at the management station, explained in section 8.2.2.1. Mobile agents perform their individual roles in the context of group they belong. The next section explains each group’s operations and mobile agents involved in each group.

8.2.3.1 Vulnerability Analysis (VA)

There are four functions in this group:

- Vulnerabilities Record Management
- Pre-Intrusion Vulnerability Analysis
- Post-Intrusion Vulnerability Analysis
- Patches Management and Enforcement

8.2.3.1.1 Vulnerabilities Record Management

Vulnerabilities Record Management provides to security administrator up-to-date and rich information about vulnerabilities. In our system this has been achieved by using three vulnerability databases: NVD, OSVDB and Security Focus [59] [60], and by generating IDP System’s own database, called Comprehensive Vulnerabilities Database (CVDB). CVDB is a rich database which contains relevant information about most of the known vulnerabilities today. VAS updates CVDB daily, weekly, or monthly, depending on a local security policy.

CVDB generation process is managed by the database management engine (DBME). DBME provides to system administrator up-to-date and rich information about vulnerabilities. Such functionality can be achieved by analyzing any database with entries in an XML format and whose structure is defined either by an XML Schema Definition file (XSD) or an SQL/MySQL schema file. Currently, the system uses NVD and OSVDB databases [59][60]. Moreover this “engine” also scans SecurityFocus database, storing all the information needed in the database. CVDB can be updated by the DBME daily, weekly or monthly, depending on the selected policy.

![Download National Vulnerability DB Files](https://example.com/download.png)

Figure 8.6 (a): Download NVD Data Files
DBME parses the target database schema file and shows the structure, along with description of every field, in the left side of the panel in the form of a tree. Administrator double clicks on a field on the left–side of the panel. The chosen fields are then displayed in the right–side panel. For each field previously selected, the system administrator selects the corresponding CVDB field using a drop–down menu. This process is very important for interoperability between target database and CVDB, as administrator map the fields of two different vulnerabilities database. Figure 8.6 shows the snapshot of the implementation.

Figure 8.6(b): Matching between Target DB and CVDB [74]

8.2.3.1.2 Pre-Intrusion Vulnerability Analysis

Pre-Intrusion Vulnerability Analysis provides capability to query remote servers regarding their profile with respect to different software components installed on those servers. It is necessary to acquire host for an effective vulnerability analysis. A team of mobile agents is assembled and dispatched to automatically scan vulnerabilities at remote hosts. There are three types of mobile agents involved in this phase: Agent_Vulnerability_Analyzer, Agent_Host_Scanning and Agent_Scanning_Report. Security administrator launches Agent_Vulnerability_Analyzer from his/her computer to a host or multiple hosts in the network. Once an agent reaches one of the remote hosts in the selected route, it fetches host’s profile containing information about the software components installed and their attributes. This is done using jRegistryKey [61]. This agent will check host profile against the vulnerability database, looking for known vulnerabilities detected in software installed the scanned machine.
The other way to analyze the network is to send `Agent_Host_Scanning` agent to the targeted hosts. It executes local Nessus daemon in the background, which scans the host. After its execution is completed, Nessus generates a report in the XML format. Once the scanning is completed, `Agent_Host_Scanning` agent launches an `Agent_Scanning_Report` agent, which sends the detailed scanning report back to the security administrator. When `Agent_Scanning_Report` reaches the security administrator's workstation, it notifies the administrator how many vulnerabilities have been detected (Figure 8.7), allowing the administrator to check the report immediately or later. In the case the administrator wants to check the report immediately, it will be transformed into the more “human-readable” HTML format by using XSL Transformer and then shown using web browser integrated into the system.

### 8.2.3.1.3 Post-Intrusion Vulnerability Analysis

Most of the IDS and Intrusion Response Systems (IRS) consider their task accomplished after they detect and respond to an intrusion. It can be assumed that the intruder will try to exploit the same vulnerability at a different system in the network. **Post-Intrusion Vulnerability Analysis** provides capability to search for the particular vulnerability that was being exploited by an intruder in the most recent attack. The main purpose of this analysis is to identify the exploited vulnerability at a particular host and then apply preventive measure to the rest of the network against specific vulnerability. This activity is performed with the help of the `Agent_Post_Intrusion_Vulnerability_Analyzer` agent. As soon as IRS finishes its task and reports to the Administrative Server (the Server from where agents were launched the first time), the Administrative Server automatically launches `Agent_Post_Intrusion_Vulnerability_Analyzer` agent in order to analyze the particular host previously under attack, by analyzing the system logs and logs generated by SNORT. Management Station stores vulnerability information in the Vulnerabilities DB in the `Vulnerable_Host_Data` table.
8.2.3.1.4 Patches Management and Enforcement

CVDB tables contain two important attributes about patches corresponding to some vulnerability. The first is patch in table Reference, a Boolean value that indicates if the related URL refers to a patch that can be eventually downloaded. During the vulnerability assessment process, many instances of the Agent_Vulnerability_Analyzer agent move through the network in order to scan hosts. Each agent brings back the configuration of the host scanned and compares the collected information with the information available in the CVDB. If some vulnerability is found, the agent will check the Reference table of the CVDB to see if there is a patch available online. The ideal scenario is when the agent can autonomously download the patch from a given Internet address. Unfortunately, patch’s URLs provided by vendors do not always refer directly to the package/file that should be downloaded. Therefore, once vulnerability is discovered, the best that can be done is to list all the references for the vendor of the vulnerable product that refers to a patch (Figure 8.8). An administrator can then select one of them and press “Download” button. Then, a web page of the corresponding URL will be opened (Figure 8.9).

![Listing patches to be downloaded](image)

Figure 8.8: Listing of Available Patches [74]
The administrator can then download the patch and store it in the local server. Once the patch is downloaded, the information about the patch for that vulnerability is stored in the CVDB by setting the boolean field `local_patch` as `true` in the table `Vulnerable_SW_Data`. In addition, the path that indicates where the patch is located in the local server is also saved in the field `local_location` in the table `Patches`. With this information, it is also possible to display all the patches already downloaded and currently stored on the server, selecting “List Available Patches” from Patches menu. From here, clicking on “Send Patch”, the selected patch will be transferred to and installed in the target host by the `Agent_Patch_Installer` agent. For installation of some patches we require human intervention. `Agent_Patch_Installer` agent carries a simple executable file (the available patch) and executes it on the target host. From now on, every time an agent discovers any host vulnerable to the same vulnerability, it will look up the CVDB to check if a patch is available and it will find the path that refers to where the patch is stored in the local server. In this way, the agent can perform patches management autonomously without intervention of the system administrator.

### 8.2.3.2 Intrusion Detection System (IDS)

In an IDP system, a team of mobile agents is dispatched into the network in response to a suspicious alarm. They analyze static data related to different intrusion attempts, like DoS attack, DDoS, Doorknob-Rattling Attack [62], etc, and then request the reinforcement from the Security Administrator.
First, analysis of logs and reporting is performed on logs generated by system events and sensors, like SNORT or Osiris on remote hosts. Analysis of logs notifies Security Administrator about the intrusion. Agent_\_IPS\_Logs agent along with the Agent_\_Leader agent analyzes these logs, filter them, extract relevant and useful information, and display results to the Security Administrator.

Second, Host–based Intrusion Detection (HID) subsystem monitors the network hosts for different types of intrusions and reports as soon as some intrusion is attempted, so that appropriate response measures by IRS can be applied. Host monitoring is being performed by continuously analyzing SNORT logs. Agent_Host_Monitoring agent is not initially present at remote hosts. Security administrator launches Agent_Host_Monitoring agent. It, along with Agent_\_Leader agent, reaches remote hosts and permanently stays there. They continuously inspect log entries generated by SNORT, OSIRIS and as soon. If they find any intrusion based on SNORT rules, they ask for help from the administrative server.

Third, Distributed Intrusion Detection (DID) subsystem works equivalent as Host–based Intrusion Detection, but it is activated by the detection of suspicious activity below the intrusion threshold, by ID components. Agent_Host_Monitoring agent provides the signal for the activation of the DID. DID creates Suspicious Host List (SHL), specific to each type of the attack, and then logs the addresses of each host generating the same type of alert. Figure 8.10 shows the structure of the SHL.

<table>
<thead>
<tr>
<th>Host IP</th>
<th>Attack Type</th>
<th>Proved</th>
<th>Response Applied</th>
</tr>
</thead>
<tbody>
<tr>
<td>192.168.8.22</td>
<td>Doorknob-Rattling</td>
<td>True</td>
<td>False</td>
</tr>
<tr>
<td>192.168.17</td>
<td>Distributed port Scanning</td>
<td>True</td>
<td>False</td>
</tr>
</tbody>
</table>

Figure 8.10 The Structure of the Suspicious Host List (SHL)

At this instance of time DID launches Agent_Distributed_Intrusion_Analyzer agent (ADIA) along with Agent_\_Leader agent and Agent_Distributed_Messenger agent to visit each host listed in the SHL, to analyze data from system logs, and then to correlate and aggregate them with system logs of other entries in the SHL in order to create traces of the attack. If the result is positive, then Agent_Distributed_Messenger agent immediately returns back to the Administration Server and sets the proved field in the SHL as true for hosts under the distributed intrusion attack, followed with a message to the administrator. Before launching an ADIA, DID creates a static Agent_SHL_Monitoring agent that constantly monitors SHL for any change in the proved fields of different entries.

In order to understand the described scenario, take as an example a Doorknob-Rattling Attack in which attacker tries several common password and username combinations on many computers that result in failed login attempts [62]. These failed login attempts, as events, are logged in the system log. Agent_Host_Monitoring agent detects these events, marks them as suspicious, and reports to the administrator. DID then creates the SHL entry, fills host IP field, and marks attack type as Doorknob-Rattling. DID then launches Agent_Distributed_Intrusion_Analyzer agent along with the Agent_\_Leader agent and Agent_Distributed_Messenger agent to visit each host listed in the SHL,
and to analyze data from system logs for Doorknob-Rattling attack. When it is detected that a remote host is under Doorknob-Rattling attack, this is reported to the DID by the Agent_Distributed_Messenger agent and which also asks for reinforcements.

### 8.2.3.3 Intrusion Response System (IRS)

IRS is tightly coupled with the IDS and it is activated as soon as the IDS detects some intrusion at remote hosts for both cases, i.e. Host–based Intrusion and Distributed Intrusion. The function of the IRS is to prevent intrusions in real-time. It launches Agent_Intrusion_Response agent to counter the attack by any method, like blocking the address from where the attack initiated, closing the port, shutting down service or program, or shutting down remote host which is under attack in order to stop contamination in the network.

Agent_Intrusion_Response agent along with Agent_Leader agent is automatically launched (event based) from the Administrative Server to the host that reported an intrusion. Agent_Intrusion_Response agent reaches the desired host, creates the response, and then reports to the Administrator. The response is implemented using decision tables. Decision table is a mechanism that associates each attack with a specific response. It is the basis for static mapping and does not consider any other factor, except the attack type. Agent_Intrusion_Response agent is using firewall to implement its responses against intrusions. It blocks port or IP address in order to response to intrusion.

As soon as the Agent_Distributed_Messenger agent marks, proved field as true Agent_SHL_Monitoring agent launches Agent_Intrusion_Response agent along with Agent_Leader agent. Agent_Intrusion_Response agent applies appropriate response to all hosts in the SHL that have confirmed intrusion and reports back to the Administrator.

### 8.2.3.4 Security Management System (SMS)

SMS also uses mobile agents to perform management tasks in the network. The purpose of these tasks is to keep all hosts updated with respect to number of security services, solutions, or security configurations. SMS performs the following tasks:

Test connectivity of remote host, query remote host configurations, and apply a number of security management tasks to remote hosts and exploited vulnerability and SNORT rule mapping. SMS performs these tasks with the help of a number of mobile agents as follows:

Test Connectivity: Security administrator tests the connectivity of remote hosts by launching Agent_Get_IP agent along with the Agent_Leader agent. Agent_Get_IP agent reaches remote host if it is running and connected to a network, gets its IP, and reports to the security administrator.

Host Configuration is acquired with the help of the Agent_Configuration_Inquiry agent that moves to the remote host, checks the configuration of a remote host, and returns results to the administrator. Configuration of remote hosts includes information about OS type, OS version, users currently logged on,
SNORT running status, number of SNORT rules and Osiris database generation date. These configuration items help security administrator to familiarize himself/herself with various security services running on remote host and with their status.

Configuration Management: it is performed by Agent_Configuration_Management (ACM) agent. ACM at remote hosts interacts with firewall and SNORT sensors in order to implement security management tasks. These security management tasks are to enable/disable firewall, to enable/disable port, to enable/disable services, to enable/disable programs running, to run SNORT, and to add SNORT rule.

Agent_Post_Intrusion_Vulnerability_Analyzer agent updates Vulnerabilities DB with exploited vulnerability, then the SMS launches Agent_Vulnerability_Responder agent in the network that updates the rest of remote hosts with a SNORT rule that detected the exploitation of that specific vulnerability. The mapping between SNORT rules and Vulnerabilities: In an ideal situation vulnerabilities should have their patches available and security administrator should update a network in advance. In practice, however, there is a usually time gap between discovery of vulnerabilities and availability of their patches. This time gap can become a window of opportunity for an attacker. In order to cater this problem, in our system, as soon as an intrusion attempt is discovered and responded to for the first time, a SNORT rule will be created for that specific vulnerability. Agent_Vulnerability_Responder agent will use it later. Currently in most of the IDP systems security administrator is performing this task, but in our future research intelligent mobile agents, based on network conditions, would build new SNORT rules.

8.3 Conclusions

The described system shows all benefits, as expected, for detection and prevention of attacks and penetrations. It automates and simplifies maintenance of various system components at remote hosts, it provides more efficient reaction and protection against attacks in real time, and it simplifies management of distributed intrusion detection and prevention systems. Contrary to the current commercial products available on the market, based on closed and proprietary approaches, our system is compatible with multiple ID/IP products and can be easily applied to such products for their interoperability, combined use, and improved maintenance and administration. Finally, the system is more efficient than existing solutions, since it minimizes human interventions and decisions. It is based on the open architecture and specifications, so it can also be easily extended by creating and deploying new agents and teams.
PART FIVE

CONCLUSION

Part 5 concludes the thesis. Chapter 9 explains the methodology used for validation of the system. Chapters 10 concludes and specifies future research areas.
Validation of The System Design

9.1 Validation Methodology

A basic goal when designing secure systems is that they should not have any weaknesses or vulnerabilities that could be exploited by attackers. The question is how to identify that the system design will not leave any loopholes that can be potentially dangerous, thus can serve as the motivation for an attacker to exploit system weaknesses to fulfill his malicious desire that can result in security violations. For that purpose, there should be some sort of a metrics. A metrics generally implies a system of measurement based on quantifiable measures [76].

NIST report “Direction in Security Metrics Research”, published in August 2009, gave overview of the security metrics area, which may help researchers to develop measures and systems of measurements for computer security. They identified those aspects of security measurements that have been used in the past and which are considered pertinent to security matrices. These aspects can help end-users, like designers of security systems, to measure security properties of their systems. These aspects include:

a) Correctness and Effectiveness
b) Leading Versus Lagging Indicators
c) Organizational Security Objectives
d) Qualitative and Quantitative Properties
e) Measurement of Large versus the Small

Based on the system or organizational security requirements, evaluators used different aspects for security measurements. We will use the fourth option, i.e. Qualitative and Quantitative Properties, to measure the security properties of our system.

Our choice of Qualitative and Quantitative Properties to create security metrics for the measurement of security properties of our system is based on the definition of metrics proposed by “SSE-CMM: Systems Security Engineering Capability Maturity Model, International Systems Security Engineering Association (ISSEA)”...

“At a high-level, metrics are quantifiable measurements of some aspect of a system or enterprise. For an entity (system, product, or other) for which security is a meaningful concept, there are some identifiable attributes that collectively characterize the security of that entity. Further, a security metric (or combination of security metrics) is a quantitative measure of how much of that attribute the entity possesses”
For this definition, it can be easily established that in order to measure security properties we have to identify the attributes that contributes towards the security of the entity. Therefore, for our MagicNET system, we will evaluate its design by looking at the quantities and qualitative properties of the system.

9.1.1 Qualitative Properties

All security mechanisms proposed in the MagicNET System are based on existing proven security technologies. We have used different standards/technologies/mechanisms, like PKCS7, XACML, GSAMKMP, Apache Rampart, Single Sign-On, SAML and FIPS-196 strong authentication protocol. The use of proven technologies has strengthened the security of our system.

9.1.2 Quantitative Properties

Based on the definition mentioned in the previous section “identifiable attributes that collectively characterize the security of that entity”, we first identify different entities within our system. As the next step, we identify different security requirements with respect to those entities. In this regard NIST’s publication on mobile agent security [43] provides the basis for mobile agents’ security requirements and threats. They mention two basic entities, i.e. mobile agents and agent platforms. Agent platforms provide an execution environment for mobile agents. They have identified security requirements such as confidentiality, integrity, accountability, availability and anonymity for mobile agents and agent platforms. In order to cover different research problems related to mobile agents’ security, we have identified new entities explained in previous chapters. In our security metrics for MagicNET system, we will consider all those entities as well.

Metrics 1: (Entities vs Security Properties)

<table>
<thead>
<tr>
<th>Security Requirements / Actions/operations</th>
<th>Confidentiality</th>
<th>Integrity</th>
<th>Non Repudiation</th>
<th>Availability</th>
</tr>
</thead>
<tbody>
<tr>
<td>Agent Creator: AC</td>
<td>AC creates different agents and deposit them to AF</td>
<td>After mutual authentication (FIPS-196) with AF, AC deposit PKCS7 signedAndEnvelopedData denoted as PKCS7\textsubscript{AC} to AF.</td>
<td>Availability is not an issue for AC, as we can have $n$ of ACs in our system.</td>
<td></td>
</tr>
<tr>
<td>Agent Trust Appraiser: ATA</td>
<td>ATA fetches the list of agents from AF waiting for appraisal and deposits them back after appraisal</td>
<td>After mutual authentication (FIPS-196) with AF, ATA also deposits PKCS7 signedAndEnvelopedData denoted as PKCS7\textsubscript{ATA} to AF.</td>
<td>Although we have proposed single ATA in our infrastructure, we can assign this role to multiple authorities. However, in the case of a single ATA, availability is not an issue as its role is passive.</td>
<td></td>
</tr>
<tr>
<td>Agent Privilege Authority: PA</td>
<td>PA fetches the list of agents from AF waiting for privileges</td>
<td>After mutual authentication (FIPS-196) with AF, PA also deposits PKCS7 signedAndEnvelopedData denoted as PKCS7\textsubscript{PA}</td>
<td>The same argument as for ATA</td>
<td></td>
</tr>
</tbody>
</table>
Mobile Agent Registrar: AR
- AR queries AF for available agents and publishes them at the UDDI
- AR mutually authenticate (FIPS-196) with AF.
- AR authenticates UDDI using ISO/IEC 9798-3 mechanism. Confidentiality of information is not important here, as AR just registers agent name and its service (which is agent role assigned by PA, e.g. DCoumentSearcherAgent).
- AR is a passive entity, availability requirement does not apply to AR.

Agent Owner: AO
- AO adopts mobile agent from AF and sends Agents in the network, interacts with KDS, PAP, PDP and runtime nodes.
- In all interactions AO mutually authenticates itself with other interacting entity. Moreover it exchanges KeyEnchiperment Certificate, so that all future correspondence can be encrypted. AO sends PKCS7 signedAndEnvelopedData to runtime nodes.
- The same as AR

Agent Factory: AF
- Repository of Agents during agent development cycle
- Mutually authenticates with all entities, those communicate with it. All information in and out to AF is signed and encrypted.
- For web services security we used WSO2 WSAS along with Rampart. Rampart implements WSS core security specifications 1.1.

UDDI Server
- Core UDDI role
- Used JUDDI and adopted one-way authentication protocol ISO/IEC 9798-3
- Not required
- Not required

PDP Server
- Policies Repository
- These servers also mutually authenticate interacting entity and exchange keyEnchiperment Certificate so that all future correspondence can be encrypted between the parties. Wherever require send simple signed information too.
- In a small domain there will be single instance of these servers.

PAP Server
- Policies Creation Authority

KDS
- Keys Repository

Runtime Node
- Receive and Send agents
- Only receives PKCS7 signedAndEnveloped Agents.

IDMS
- Possesses information about the users
- Not directly part of our system. We have not handled security requirements for these entities

LCA
- Local Certification Authority, CA

In the following metrics we will cover security properties with respect to the threats identified by NIST’s publication on mobile agents’ security [43].

**Metrics 2 (Threats vs Security Mechanisms)**

<table>
<thead>
<tr>
<th>Threat</th>
<th>Fulfilled</th>
<th>Mechanism</th>
</tr>
</thead>
<tbody>
<tr>
<td>Agent-to-platform (runtime node) threats</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Masquerading</td>
<td>Yes</td>
<td>Agent pre-authentication at management station and LVC component verification at runtime node</td>
</tr>
<tr>
<td>Unauthorized access</td>
<td>Yes</td>
<td>The complete authentication and authorization process at management station and runtime node</td>
</tr>
<tr>
<td>Denial of service</td>
<td>Yes</td>
<td>LVC at runtime node</td>
</tr>
<tr>
<td>External entities to agent system threats</td>
<td></td>
<td></td>
</tr>
<tr>
<td>------------------------------------------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Copy &amp; Reply</td>
<td>Yes</td>
<td>SAML ticket includes expiration time</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Agent-to-agent Threats</th>
</tr>
</thead>
<tbody>
<tr>
<td>Masquerading</td>
</tr>
<tr>
<td>Denial of Service</td>
</tr>
<tr>
<td>Repudiation</td>
</tr>
<tr>
<td>Yes</td>
</tr>
<tr>
<td>Using the protocol mentioned in section 7.2.3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Baggage Protection</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unauthorized Access</td>
</tr>
<tr>
<td>Eavesdropping</td>
</tr>
<tr>
<td>Alteration</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Agent Protection</th>
</tr>
</thead>
<tbody>
<tr>
<td>Masquerading</td>
</tr>
<tr>
<td>Eavesdropping</td>
</tr>
<tr>
<td>Unauthorized Access</td>
</tr>
</tbody>
</table>

Metrics 1 and 2 shows different entities and their attributes. All these attributes collectively contribute towards security of individual entities and thus collectively towards security of the entire system. These attributes are in fact different mechanisms that have been adapted to fulfilled individual entities security requirements with-respect-to confidentiality, integrity, repudiation and availability, thus protecting these entities from threats, such as masquerading, eavesdropping, unauthorized access, copy and reply, and denial of service.

### 9.2 Development Results

Security requirements against different system entities identified in matrix 1 and the threats identified in matrix 2 are being addressed in the MagicNET system design through different security mechanisms. Applied security mechanisms are based on proven technologies and standards. These standards/technologies include PKCS7, XACML, GSAKMP, Apache Rampart, Single Sign-On, SAML, FIPS-196 authentication protocol, Snort, Nessus, Osiris, National Vulnerability Database (NVD), Open Source Vulnerability Database (OSVD), and Security Focus.

The uses of proven technologies along with the adoption of different standards build confidence in the system security design. However, important challenge, in case of system implementation, is combining different technologies together to achieve system goal. As a proof of concept we have developed a
prototype of MagicNET Infrastructure and Network Security system using mobile agents. The goal of implementation was not to develop comprehensive mobile agents system by following specifications, e.g. Foundation for Intelligent Physical Agents (FIPA) specifications, with every feature to support mobile agents based applications. Rather, the purpose of implementation was to prove that these different standards and technologies can be used together to achieve system goals (research problems). Our implementation has two main parts a) development of security infrastructure for mobile agents, b) Network Security system using mobile agents. Part b) is based on different mobile agents and their role in different operations of the system, as mentioned in section 8.2.3 and implementation details have been explained in the last chapter. This section will focus on part a) implementation details, up to mobile agents’ adoption in deployment phase.

Snapshots of different interfaces and description of different security steps in the background are the following:

a) **Initial Panel – MagicNET Login Screen**

![Initial Panel - MagicNET Screen](image)

After displaying initial panel the system asks user for authentication. Authentication can be username and password based or it can be smart card based.

![Initial Panel - MagicNET Login Screen](image)

(Smart card or Username & Password based login)
Each entity (server or human user) identified in metric 1 must be registered before in the system through IDMS. Users can be registered using the following interface of the IDMS.

![Register Person Interface](image)

Server can be registered using the Server Registration Interface of IDMS.
After a user logs in, the main interface of the MagicNET system is displayed, as shown in figure 9.5.

b) **MagicNET – The Main Interface and Global Functions of the System**
As the next step, user must fetch certificates from the Local CA. User will submit certificate request using Local Security Management GUI.

![Submit Certificate Request Interface](image)

Once certificate request has approved by the Local CA, user can fetch certificates by using “Fetch Certificate” option in Local Management GUI.

![Submit Digital Signature Request](image)

![Submit Key Exchange Request](image)

![Submit Authentication Request](image)

Figure 9.6 Submit Certificate Request Interface
Once certificates are downloaded and saved locally, user can view his/her certificates and certificate chain using “List Local Certificates” function:
For simplification of testing we use the same user for four different roles, i.e. Agent Creator, Agent Appraiser, Privilege Authority and Agent Owner. However in a real deployment they must be four different users of the system.

9.2.1 Interfaces at Creation and Validation Phase

After the initial setup of the system, as mentioned in the previous section, initial setup steps are same same for AO, ATA, PA and AC. As the next step we describe individual operations in each phase, mentioned in section 3.1.2. We cover in our implementation only High Assurance level mentioned in section 4.2.

c) Agent Factory Management Interface and Functions

As the first step, we start Agent Factory Server (AF). AF has two main parts: a) RMI Server, and b) WSAS Application Server. RMI Server covers the functionality required in Creation and Validation Phase, while WSAS Application Server covers functionality required in the Agent Adoption Phase.

AF starts and makes connection with database. A database used for storage of intermediate developed agents, as mentioned in section 4.3.2. After that, AF fetches certificates from the Local CA and saves
them locally. Next time when the server starts, it reads its certificates from the local database. This process is the same for our all servers (PDP Server, KDS and Runtime Nodes) in the system.

We have developed AF modules as Eclipse plug-ins. There are three plug-ins:

com.magicnet.agentfactory  
com.magicnet.agentfactory.shared  
com.magicnet.agentfactory.ui

com.magicnet.agentfactory plug-in contains the core logic of the AF Server. It exposes different RMI Services for AC, ATA and PA. At the same time, it provides the mechanism for mutual authentication with AC, ATA and PA, as mentioned in section 5.2.3.3.

com.magicnet.agentfactory.ui provides management GUI for AF. It provides features related to AF management, like certificates (verify, fetch, renew), logging and Agents Management (list, view, delete). The following snapshot shows basic GUI of AF.

com.magicnet.agentfactory.shared: This package contains some shared modules that are required by both AF GUI and AF core business logic.

d) Agents Creation Interface and Functions

The next interface is for Agent Creator, who is the first entity in the Creation and Validation Phase. As soon as user switches to “Agent Creation”, agent creation station performs mutual authentication with the AF, as mentioned in section 5.2.3.3. In case of successful mutual authentication, we receive message, “Authentication process completed successfully at Agents Creation side”. At the end of mutual authentication, both interacting parties exchange KeyEncipherment certificate. Figure 9.8 shows AF certificates as well. During mutual authentication, digital signature certificate is being exchanged while at the end of authentication, we exchange keyEncipherment certificate.
Now as the next step, we start with our main use case where AC has an agent and he/she wants to deposit it to the AF for appraisal and privileges assignment. Using the interface shown in Figure 9.11, AC selects an agent. A unique agentUUID will be assigned by the system. AC provides agent description and finally deposits the agent to the AF. All the checks related to agents correctness and reliability are performed before agent deposits to the AF. Theses checks includes :- code style, potential programming problems, name shadowing and conflicts, deprecated and restricted API, unnecessary code, generic types check and annotations. We have proposed to use Eclipse JDT[86] for these checks.

It is important to note here that agentDescription and agentCode are signed by the AC first, then put it in a container (mentioned in next paragraph), converted into byte array created as signedAndEnveloped package, and deposited to the AF. AC business logic and GUI has been implemented in the form of Eclipse plug-in:

com.magicnet.agentcreation
Figure 9.11 Agent Creator: Deposit Agent
As soon as the AF receives the package, it opens the package, verifies AC signature of the container and the deposit it as such in the database (under column “CreatorContributions”). We have implemented AF database using SQL Server. Figure 9.12 shows the scheme of AF database implemented using SQL server.

![Figure 9.12(a): Agent Factory Server Database Schema.](image)

```
public class AgentDTO implements Serializable {
    private static final long serialVersionUID = -4658624935817920851;

    private String agentUUID, agentCreatorSignedCode, agentDescription, agentCreatorDigitalCert;  
    private String agentAParoleAssigned, agentPareilgeAuthenticationDigitalCert;
    private String agentFileName;  

    public AgentDTO() {} // public AgentDTO(String agentUUID, String agentCreatorSignedCode, String agentDescription,
                        // String agentCreatorDigitalCert, String agentAParoleAssigned, String agentPareilgeAuthenticationDigitalCert, String agentFileName) {
```

In the last section we have mentioned that AC puts his signed contributions in the container. This container is very important as it carries particular agent information back and forth between AC, ATA and PA. AC, ATA and PA contribute their contributions to the container, signedAndEnvelope deposit to the AF. In order to verify signatures of entities involved in the creation and validation phase, container also contains Digital Certificates of these entities. Certificates are stored locally when package is transferred to the ATA, PA or AO. Received certificates can also be verified by the Local CA using Local Security Management GUI (as shown in Figure 9.8).

e) Agents Appraisal Interface and Functions

The next entity in our system is Agent Trust Appraisal (ATA), and as per section 4.3.3.2, ATA fetches the agents pending for appraisal. ATA applies various tests, adds agent task specifications, and signs them. Initial startup steps and mutual authentication of ATA with AF is the same as for the AC.
ATA first sends request to AF to fetch agents, those still pending for appraisal. AF queries its database and fetches all those agents whose “AppraisalStatus” is still pending in the database. As in database contained PKCS7 SignedAndEnveloped contributions, AF first opens the SignedAndEnveloped package and then again repackages it in the PKCS7 SignedAndEnveloped package and sends it to ATA. This time the recipient is ATA.

ATA receives the package, opens it and in the case of a valid package the following message appears and ATA stores the package into its local database.
As the next step ATA list available agents from its local database. It applies various tests, as mentioned in section 4.3.3.2. Figure 9.15 shows that ATA assigned Task 1 and Task 2 along with their descriptions.

Task specifications are also signed by ATA. ATA puts them into Agent Container, referred in Figure 9.12, and finally creates a new PKCS7 SignedAndEnveloped package and deposits to the AF. We have created the following Eclipse plug-in for handling agent appraisal related functions:

com.magicnet.agentappraisal
The next entity in our system is Privilege Authority (PA), and as per section 4.3.3.3, interaction of the PA is similar to ATA. The only difference is that PA, based on task specifications assigned by ATA, assigns role to an agent. This role is very important, because agents are later being searched by web services.
using this role. Second, this role is also used by PAP to create different policies for execution environment.

PA fetches agents pending for privileges assignment. PA verifies task specification and then as per the requirement of tasks assigns a role to an agent, as mentioned in section 4.3.3.3. PA signs the role assigned and adds it to agent container, as referred in Figure 9.12. Initial startup and mutual authentication of the PA with the AF is same as for the AC.

PA first sends request to AF to fetch agents still pending for privilege assignment. AF queries its database and fetches all those agents having “PrivilegeAuthStatus” still pending in the database. As in database contained PKCS7 SignedAndEnveloped contribution from ATA, AF first opens the SignedAndEnveloped package and then again repackages it into PKCS7 SignedAndEnveloped package and sends it to the PA. This time the recipient is PA.

PA receives the package, opens it and in the case of valid package, PA stores the package into its local database. Later PA lists available agents locally pending for privilege assignment. PA selects one agent to assign a role to it.

![Figure 9.17: PA Interface, List of Agents Pending](image)

After selection of an agent, a module at the PA verifies Agent Task Specifications by verifying ATA’s signature. Similarly also verifies AC’s signature of the agent code and agent description. PA then assigns a role, in this example role is CONNECTION_VERIFIER. We have also created an Eclipse plug-in that provides the complete PA functionality.

com.magicnet.privileges
9.2.2 Interfaces for the Adoption and Deployment Phase

We use agent factory application server, i.e. WSAS application server, in the adoption phase. For this purpose we have used Eclipse WTP plug-in [79] to create eclipse web project com.magicnet.agentfactory.web. WTP plug-ins allowed us to create and publish two web services.
We have integrated WSAS within the environment of the MagicNET system. WSAS can be started and stopped within our MagicNET system. Figure 9.19 shows the WASA tool bar inside MagicNET System.

![WSAS Integration with MagicNET](image1)

Figure 9.19: WSAS Integration with MagicNET

We have created two web services: a) **AgentProviderService** and b) **SelectedAgentService**. As per section 5.2.3.1 **AgentProviderService** provides list of agents available to AR or AO. Similarly **SelectedAgentService** as per section 5.2.3.2, allows AO to fetch selected agents from the AF. Figures 9.20 and 9.21 show the two services published at WSAS. Appendix C shows the two WSDL files.
Figure 9.20: AgentProviderService published at WSAS

Figure 9.21: SelectedAgentService published at WSAS

Published services can be tested using the TryIt functionality provided by Management GUI of WSAS. We show here only the output of SelectedAgentService. The SelectedAgentService takes two arguments,
selected agent role and its unique ID. Web service response is an agent container, which contained signed contributions of the AC, ATA and PA.

Figure 9.22: TryIt WSAS Feature: SelectedAgentService

**g) Agents Adoption Interface and Functions**

The next step in the implementation of the MagicNET system is the adoption of mobile agents by Agent Owner. AO uses management station to adopt agents as per his preferences and needs. Management Station (MS) in fact acts as a web services client, for the web services described in previous section. AO switch to *Agent Adoption* and selects publish agent menu button. In the background, through web service client, it fetches the list of agents available at the AF. User selects one agent and presses download button. Behind the download button *SelectedAgentServices* is invoked. MS receives an agent. It verifies signatures of the AC, ATA and PA contributions. After successful verification, it saves it locally. This step concludes agents’ adoption phase, as per section 5.2.3.2

Figure 9.23 (a): Management Station Interface

We have implemented MS in three Eclipse plug-ins. These are
com.magicnet.managementstation contains the code related to GUI, management server administration, and web services clients. com.magicnet.managementserver contains the core business logic of the management server. Management server during its life time interacts with other servers, like KDS, PAP Server, PDP Server and Runtime Nodes. The role of the management server (MS) has been explained in detail in Chapters 6, 7 and 8. com.magicnet.management.shared contains the common classes and packages used by management station, as well as, by management server.

Before mobile agents’ adoption, web services url are fetch from UDDI server. Web services are published at UDDI server by AR.
After the adoption of mobile agents by the AO, AO as per section 6.3.2.1 sends request to the PAP Admin to create XACML policies. PAP Admin uses administration GUI to create different policies, as per methodology mentioned section 6.2. Few interfaces from PAP administration GUI are as follows. We will not mention every interface due to space limitation. PAP admin creates policies based on the XACML 2.0 standard.
Figure 9.25: PAP Admin Interface: Create Policy

As soon as PAP Admin creates policies and deposits them to the PDP Server, it sends confirmation message to the AO at the MS. Now agents can be launched in the network. AO launches agents in the network using the interface shown in Figure 9.25. AO selects route information from the available runtime node servers in IDMS. As soon as launch button is pressed, MS performs mutual authentication (as per section 5.2.3.3) with the first runtime node, signs agent code, adds ticket mentioned in section 7.2.2.1.1 appends its digital certificate, adds route information and finally create PKCS7 signedAndEnveloped and sends it to the first runtime node.
Figure 9.26: Management Station: Agent Launch

After this step agent execution phase starts. There is no GUI in agents execution phase, but there are many steps that are being executed during agents’ execution at runtime nodes which have been explained in section 7.2.2. Based on agent’s role and functionality agent perform its activities and brings back results to MS. We have used different agents in the Network Security System. The detail purpose of each agent has been explained in section 8.2.3.

9.3 Validation Methodology Conclusions

Our validation methodology for the MagicNET system was based on three concrete blocks. After the identification of research problems, we intended to address in MagicNET system. We first designed MagicNET system using the existing proven security technologies and standards. Second, we evaluated our design by identifying two security matrices. We identified the identities, their security requirements, threats and mechanism to cater those threats. Finally, we developed a prototype implementation which proves that different technologies can be used in combination in order to achieve desired system goals.
10 Conclusions and Future Research

10.1 Conclusions

Mobile agents’ technology started almost two decades ago. Initially it was as perceived very interesting and useful to research community and industry. This is mainly due to its attractive features such as mobility, autonomous operation, collaboration, communication, intelligence, reactive properties, adaptively, and etc. However, we still have failed to see wider adoption of mobile agent applications in the real life on a large scale. “European coordination action for agent-based computing”, agentLink [77] has listed around 1200 research papers, 28 ongoing projects and 128 different mobile agents software on their website till 2006. These figures clearly show the extent of work that has been done in mobile agents’ research. Now, the question is why mobile agents based applications are still not adopted in the real life. Based on our literature survey we have concluded that the main reasons for the negative trend towards mobile agent-based applications are two: a) interoperability issues between different mobile agent software, and b) security issues.

In our research we have focused on the security issues. NIST special publication on mobile agents security in year 1998 provide the basis for mobile agents security research. They have identified security requirements for mobile agents and threats that are needed to cater. However, this report is 12 years old in the last decade we have seen significant work on mobile agents’ security as well. Still there are some open issues. Second, NIST publication discussed very simple model of mobile agents, which has mainly two entities: agents and agent platform. We felt that there is the need to address mobile agents’ security issues in a holistic manner. For this purpose there is a need to address security requirements for the complete mobile agents’ development and execution life cycle. We interpret these security requirements in the form of security infrastructure for mobile agents. This infrastructure should not only provide security architecture and security mechanisms for agents’ runtime execution, but also all infrastructural components along with methodology for the creation, classification, adoption, and validation of mobile agents before their actual deployment in real-environment, along with protection of mobile agent code and its baggage during execution.

We have further classified the infrastructure features into six research questions. These research questions precisely helped us to identify our research problem. These questions are: a) How to create trusted mobile agents, b) How to adopt trusted mobile agents in a secure manner, c) How to establish trust in mobile agents code during deployment, d) How to protect mobile agents during execution, e) How to establish
secure communication between two mobile agents during execution, and finally f) As a proof of concept show how mobile agents can be used for network security.

Our proposed network security system uses multifaceted approach in order to eliminate network security threats. Mobile agents have been used for vulnerabilities analysis, intrusion detection and response, and for network management. Second, in the case of mobile agents’ infrastructure, we have identified the key components of infrastructure along with conceptual methodologies for mobile agents’ assurance, acquisition, authorization, and protection. We have adopted well established security standards like, RBAC, XAML, SAML, PKCS7, GSAKMP and Apache Rampart for the design and implementation of these methodologies. We have evaluated our solution against known threats in the area of mobile agent security. We are confident that our infrastructure is based on well established standards and caters each threat in a holistic manner.

10.2 Future Research

There are a number of security challenges that still exist in mobile agents’ paradigm. Future research can be carried out to address the following issues.

- We have proposed mobile agent baggage protection mechanism. However there is no mechanism so far that addresses mobile agent’s baggage access control.
- In our proposed solution access control policies are applied in a specific domain. There is the need to research how mobile agents from one domain can execute in another domain with the same access rights.
- We have not addressed security issues for mobile agents cloning and reproduction. How parent mobile agents access rights transfer to a child agent. It would be an interesting research problem. Similarly, how to identify agents clones in order to give them appropriate access rights.
- In our infrastructure we have used single web service to adopt mobile agents. Future research can be done where mobile agents autonomously register their web services.
- In our infrastructure user adopts mobile agents for specific purpose. It would be interesting to propose more dynamic behavior for mobile agents, i.e. mobile agents can adopt other mobile agent during execution based on their needs and requirements. There would be a lot of security challenges in runtime adoption of mobile agents.
References:


83. Awais Shibli, Alessandro Giambruno and Sead Muftic “Security Architecture and Methodology for Authorization of Mobile Agents”, accepted for publication in the International Journal for Internet Technology and Secured Transactions (IJITST), Vol. 5, ISSN 1748-569X (Print), ISSN: 1748 - 5703 (Online).


## Appendix A  Abbreviations and Conventions

Table A.1 describes all abbreviations used in this thesis.

<table>
<thead>
<tr>
<th>Abbreviations</th>
<th>Explanation</th>
<th>Abbreviations</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>AF</td>
<td>Agent Factory</td>
<td>PKCS</td>
<td>Public-Key Cryptography Standards</td>
</tr>
<tr>
<td>AC</td>
<td>Agent Creator</td>
<td>PKCS#7</td>
<td>Cryptographic Message Syntax Standard</td>
</tr>
<tr>
<td>AP</td>
<td>Agent Platform</td>
<td>PK_{AC}</td>
<td>Public Key Certificate - Agent Creator</td>
</tr>
<tr>
<td>AC</td>
<td>Agent Creator</td>
<td>PK_{AA}</td>
<td>Public Key Certificate - Agent Trust Appraisal</td>
</tr>
<tr>
<td>ACL</td>
<td>Access Control Lists</td>
<td>ABAC</td>
<td>Attribute based Access Control</td>
</tr>
<tr>
<td>ATA</td>
<td>Agent Trust Appraisal</td>
<td>PK_{PA}</td>
<td>Public Key Certificate - Privilege Authority</td>
</tr>
<tr>
<td>AR</td>
<td>Agent Registrar</td>
<td>PEP</td>
<td>Policy Enforcement Point</td>
</tr>
<tr>
<td>AO</td>
<td>Agent Owner</td>
<td>PA</td>
<td>Privilege Authority</td>
</tr>
<tr>
<td>AES</td>
<td>Advanced Encryption Standard</td>
<td>PKI</td>
<td>Public Key Infrastructure</td>
</tr>
<tr>
<td>AS</td>
<td>Agent Server</td>
<td>PT</td>
<td>Policy Ticket</td>
</tr>
<tr>
<td>CA</td>
<td>Certification Authority</td>
<td>PDP</td>
<td>Policy Decision Point</td>
</tr>
<tr>
<td>CMS</td>
<td>Certificate Management System</td>
<td>PPS</td>
<td>Permission Policy Set</td>
</tr>
<tr>
<td>DES</td>
<td>Data Encryption Standard</td>
<td>PAP</td>
<td>Policy Administration Point</td>
</tr>
<tr>
<td>DLL</td>
<td>Dynamic Link Library</td>
<td>PT</td>
<td>Policy Token</td>
</tr>
<tr>
<td>DSC</td>
<td>Data Storage Component</td>
<td>RSA</td>
<td>Rivest-Shamir-Adleman encryption algorithm</td>
</tr>
<tr>
<td>DOS</td>
<td>Denial of Service</td>
<td>RBAC</td>
<td>Role Based Access Control</td>
</tr>
<tr>
<td>DN</td>
<td>Distinguished Name</td>
<td>RMI</td>
<td>Remote Method Invocation</td>
</tr>
<tr>
<td>FD</td>
<td>Functional Description</td>
<td>RPS</td>
<td>Role Policy Set</td>
</tr>
<tr>
<td>FIPS</td>
<td>Federal Information Processing Standards</td>
<td>RAM</td>
<td>Random Access Memory</td>
</tr>
<tr>
<td>GSakM</td>
<td>Group Secure Association Key Management Protocol</td>
<td>RFC</td>
<td>Request for Comment</td>
</tr>
<tr>
<td>GC</td>
<td>Group Controller</td>
<td>S_{AC}</td>
<td>Signature Agent Creator</td>
</tr>
<tr>
<td>GB</td>
<td>Giga Bytes</td>
<td>S_{PA}</td>
<td>Signature Privilege Authority</td>
</tr>
<tr>
<td>GMC</td>
<td>Group Management Component</td>
<td>S_{AA}</td>
<td>Signature Agent Trust Appraisal</td>
</tr>
<tr>
<td>GM</td>
<td>Group Member</td>
<td>SAML</td>
<td>Security Assertion Markup Language</td>
</tr>
<tr>
<td>HTTP</td>
<td>Hypertext Transfer Protocol</td>
<td>SSL</td>
<td>Secure Socket Layer</td>
</tr>
<tr>
<td>IP</td>
<td>Internet Protocol</td>
<td>SOAP</td>
<td>Simple Object Access Protocol</td>
</tr>
<tr>
<td>IDMS</td>
<td>Identity Management System</td>
<td>SHA</td>
<td>Secure Hash Algorithm</td>
</tr>
<tr>
<td>ISO</td>
<td>International Organization of Standardization</td>
<td>UDDI</td>
<td>Universal Description Discovery and Integration</td>
</tr>
<tr>
<td>JUDDI</td>
<td>Java based Universal Description Discovery and Integration</td>
<td>URI</td>
<td>Uniform Resource Identifier</td>
</tr>
<tr>
<td>JVM</td>
<td>Java Virtual Machine</td>
<td>WSD</td>
<td>Web Services Description</td>
</tr>
<tr>
<td>KDS</td>
<td>Key Distribution Server</td>
<td>WSS</td>
<td>Web Service Security</td>
</tr>
<tr>
<td>KS</td>
<td>Key Server</td>
<td>WSO\textsubscript{2} WSAS</td>
<td>WSO\textsubscript{2} Web Service Application Server</td>
</tr>
<tr>
<td>LCA</td>
<td>Local Certification Authority</td>
<td>XACML</td>
<td>eXtensible Access Control Markup Language</td>
</tr>
<tr>
<td>LAN</td>
<td>Local Area Network</td>
<td>XML</td>
<td>Extensible Markup Language</td>
</tr>
<tr>
<td>LVC</td>
<td>Legitimacy Verification Component</td>
<td>CERT</td>
<td>Computer Emergency Response Team</td>
</tr>
<tr>
<td>-------------------------</td>
<td>----------------------------------</td>
<td>-----------------------</td>
<td>-------------------------------------------------</td>
</tr>
<tr>
<td>MagicNET</td>
<td>Mobile Agent Intelligent Community Network</td>
<td>CC</td>
<td>Coordination Center</td>
</tr>
<tr>
<td>Mbps</td>
<td>Mega bits per second</td>
<td>DOS</td>
<td>Denial of Service</td>
</tr>
<tr>
<td>MIC</td>
<td>Message Integrity Check</td>
<td>IDS</td>
<td>Intrusion Detection System</td>
</tr>
<tr>
<td>CVDB</td>
<td>Comprehensive Vulnerability</td>
<td>CVSS</td>
<td>Common Vulnerability Scoring System</td>
</tr>
<tr>
<td>HID</td>
<td>Host-based Intrusion Detection</td>
<td>DID</td>
<td>Distributed Intrusion Detection</td>
</tr>
<tr>
<td>SMS</td>
<td>Security Management System</td>
<td>SHL</td>
<td>Suspicious Host List</td>
</tr>
<tr>
<td>VAS</td>
<td>Vulnerability Analysis System</td>
<td>OSVDB</td>
<td>The Open Source Vulnerability Database</td>
</tr>
<tr>
<td>MS</td>
<td>Management Station</td>
<td>IRS</td>
<td>Intrusion Response System</td>
</tr>
</tbody>
</table>

Table A.2 describes all conventions used in this thesis.

<table>
<thead>
<tr>
<th>Conventions</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>r&lt;1&gt;</td>
<td>Role assigned to agent</td>
</tr>
<tr>
<td>tAR</td>
<td>Timestamp</td>
</tr>
<tr>
<td>A1setup-req</td>
<td>Authentication ticket request</td>
</tr>
<tr>
<td>β</td>
<td>An agent</td>
</tr>
<tr>
<td>α</td>
<td>An agent</td>
</tr>
<tr>
<td>Cα</td>
<td>Agent α credential</td>
</tr>
<tr>
<td>tk&lt;1&gt;</td>
<td>A ticket issued to an agent</td>
</tr>
<tr>
<td>H(xyz)</td>
<td>Hash of xyz</td>
</tr>
<tr>
<td>IP PDP</td>
<td>IP of PDP Server</td>
</tr>
<tr>
<td>texp</td>
<td>Timestamp of ticket given to runtime node</td>
</tr>
<tr>
<td>op1</td>
<td>Operation α want to execute</td>
</tr>
<tr>
<td>resx</td>
<td>Some local resource at runtime node</td>
</tr>
<tr>
<td>dataN</td>
<td>Data contributed to visiting agent from a runtime node</td>
</tr>
<tr>
<td>ADIA</td>
<td>Agent_distributed_intrusion_analyzer</td>
</tr>
<tr>
<td>ACM</td>
<td>Agent_configuration_management</td>
</tr>
</tbody>
</table>
Appendix B\textsuperscript{12} Snort

Snort is an Open Source cross-platform lightweight network intrusion detection tool that is basically used for network traffic monitoring in order to detect suspicious network traffic. SNORT is petty and roughly 100 kilobytes, powerful tool with an ease of configuration and flexibility. It can be used as a network sensor in MagicNET infrastructure. It monitors the TCP/IP traffic and can raise an alert in case of a suspicious activity accompanying enough data to make an informed decision. It expedites the detection of novel attacks by development and inclusion of new rules in the rule database.

Snort is a packet sniffer and logger which is based on libpcap and can also be used as a lightweight network sensor. It is notable for featuring rules based logging in order to perform content pattern matching and also to the detection of a variety of attacks and probes, such as buffer overflows, port scanning, CGI attacks and much more. SNORT supports features like real-time alerting capability, where it sends alerts to syslog, Win popup messages or a can log it in the separate "alert" file.

Snort comprises of mainly three components: packet decoder, detection engine and real-time logging and alerting. The network interface card is set into the promiscuous mode to sniff the packet at libpcap to provide packet sniffing and filtering. The packet is handed to packet decoder system to efficiently decode the packet according to each layer of the TCP/IP protocol stack from data link layer to the application layer. In this phase pointers are set inside the packet which is used for analysis by detection engine.

Since Snort detects the suspicious activity on the basis of rules already defined in the rule data base, this confines it to detect only the known attacks. These rules are loaded in two dimensional linked list termed as Chain Header and Chain Options by a rule parser. Chain Header contains a list of common attributes and Chain Options have a list of corresponding modifier options. Whenever a packet arrives, the rule chains are scanned recursively to find a match. It triggers an event specified in the rule definition when a match is found.

Snort provides a simple, yet flexible and powerful way to write rules to detect diverse number of suspicious activities. The Snort rules can be mainly divided into 4 fields. First field is an action to be taken in case of rule match. There can be three types of actions available: pass, log, or alert. Pass directive is used to drop the packet. Log directs Snort to log the whole packet in log file mentioned on command line parameters. The alert action is used in case of most sever situation where an event notification is directly sent in the form of Windows popup messages or to an alert file specified in the command line parameter with additional information to facilitate later analysis. The second field contains the information related to the packet source, like source IP address and source port. The third field contains the information as follows:

\begin{verbatim}
Action Src-IP Src-Port -> Dest-IP Dest-Port Options
Alert any any -> 10.10.1.0/24 80 (content: "/cgi-bin/phf"; msg: "PHF probe!";)
\end{verbatim}

\textsuperscript{12} Appendix B has taken from [64]. It is reproduced here for reader’s complete understanding.
Snort facilitates alerting and logging being selected by command line switches. It supports logging in two formats: in decoded format and tcpdump format and alerting is done through five different ways: Syslog, WinPopup, full, fast or no alert generation.

**Snort Configuration**

```plaintext
var HOME_NET 192.168.1.0/24
var EXTERNAL_NET any
var DNS_SERVERS $HOME_NET
var SMTP_SERVERS $HOME_NET
var HTTP_SERVERS $HOME_NET
var SQL_SERVERS $HOME_NET
var TELNET_SERVERS $HOME_NET
var SNMP_SERVERS $HOME_NET
var SHELLCODE_PORTS !80
var ORACLE_PORTS 1521
var AIM_SERVERS
[64.12.4.0/23, 64.12.24.0/23, 64.12.28.0/23, 64.12.161.0/24, 64.12.163.0/24, 64.12.200.0/24, 205.188.3.0/24, 205.188.5.0/24, 205.188.7.0/24, 205.188.9.0/24, 205.188.153.0/24, 205.188.179.0/24, 205.188.248.0/24]

var RULE_PATH ..\rules

preprocessor frag3_global: max_frags 65536
preprocessor frag3_engine: policy first detect_anomalies
preprocessor http_inspect: global \ 
iis_unicode_map unicode.map 1252
preprocessor http_inspect_server: server default \ 
profile all ports { 80 8080 8180 } oversize_dir_length 500
preprocessor bo
preprocessor telnet_decode
preprocessor xlink2state: ports { 25 691 }
```
output alert_csv:alert.csv
timestamp,sig_generator,sig_id,sig_rev,msg,id,dst,src,proto,iplen

include classification.config

include reference.config

include $RULE_PATH/local.rule

The Command line used to run SNORT to log events in CSV format into SNORT log directory.

SNORT -vde -i 2 -c ..\etc\SNORT.conf -l ..\log

**Osiris**

In order to perform the task of host integrity sensor, Osiris has been used. It can be used to monitor the changes to the file systems of network of hosts. It detects the changes to the host file system and logs it into a log file on the respective host, and in case it monitors a number of hosts in the network then it logs it into a central management system. For this purpose, it takes periodic snapshots of the file system and stores it in an internal database. The system administrator can later schedule it to perform periodic scans. Whenever a discrepancy is found in the data stored in the previous database and newly scanned data, an event will be logged into the log file. It also supports monitoring of any change to user lists, group lists, and kernel modules or extensions. Osiris is cross platform tool supporting a diverse range of operating systems like Windows, Free BSD, and Linux etc. Unlike other host integrity tools, Osiris does not have any signature data base to detect malicious changes to avoid the complexities in management. The other benefit of Osiris is that it detects all types of changes irrespective of the existence of its signature in the database. Signature based host integrity monitoring tools lack this special feature. It also facilitates the user to configure Osiris to perform the automated scheduled scans from time to time.
Appendix C  Web Services Descriptions at AF

Agent Provider Service: WSDL

```xml
<wsdl:definitions xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/
xmlns:wsaw="http://www.w3.org/2006/05/addressing/wsdl"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/
xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/
xmlns:soap12="http://schemas.xmlsoap.org/wsdl/soap12/
xmlns:soap12="http://schemas.xmlsoap.org/wsdl/soap12/
xmlns:soap12="http://schemas.xmlsoap.org/wsdl/soap12/
xmlns:soap12="http://schemas.xmlsoap.org/wsdl/soap12/
xmlns:soap12="http://schemas.xmlsoap.org/wsdl/soap12/
xmlns:soap12="http://schemas.xmlsoap.org/wsdl/soap12/
xmlns:soap12="http://schemas.xmlsoap.org/wsdl/soap12/
xmlns:soap12="http://schemas.xmlsoap.org/wsdl/soap12/
xmlns:soap12="http://schemas.xmlsoap.org/wsdl/soap12/
xmlns:soap12="http://schemas.xmlsoap.org/wsdl/soap12/
xmlns:soap12="http://schemas.xmlsoap.org/wsdl/soap12/
xmlns:soap12="http://schemas.xmlsoap.org/wsdl/soap12/
xmlns:soap12="http://schemas.xmlsoap.org/wsdl/soap12/
xmlns:soap12="http://schemas.xmlsoap.org/wsdl/soap12/"
targetNamespace="http://service.web.agentfactory.magicnet.com">
  <wsdl:documentation>AgentProviderService</wsdl:documentation>
  <wsdl:documentation>AgentProviderService</wsdl:documentation>
  <wsdl:types>
    <xs:schema attributeFormDefault="qualified" elementFormDefault="qualified"
targetNamespace="http://service.web.agentfactory.magicnet.com">
      <xs:element name="requestAvailableAgentsResponse">
        <xs:complexType>
          <xs:sequence>
            <xs:element minOccurs="0" name="return" nillable="true" type="xs:string" />
          </xs:sequence>
        </xs:complexType>
        <xs:complexType>
          <xs:element name="requestAvailableAgentsResponse">
            <xs:complexType>
              <xs:sequence>
                <xs:element minOccurs="0" name="return" nillable="true" type="xs:string" />
              </xs:sequence>
              <xs:complexType>
                <xs:element>
                  <xs:complexType>
                    <xs:element name="requestAvailableAgentsRequest" />
                    <xs:element name="requestAvailableAgentsResponse" />
                    <wsdl:message name="requestAvailableAgentsRequest" />
                    <wsdl:message name="requestAvailableAgentsResponse" />
                    <wsdl:portType name="AgentProviderServicePortType">
                      <wsdl:operation name="requestAvailableAgents">
                        <wsdl:input message="ns:requestAvailableAgentsRequest" wsaw:Action="urn:requestAvailableAgents" />
                        <wsdl:output message="ns:requestAvailableAgentsResponse" wsaw:Action="urn:requestAvailableAgentsResponse" />
                      </wsdl:operation>
                    </wsdl:portType>
                    <wsdl:binding name="AgentProviderServiceSoap11Binding" type="ns:AgentProviderServicePortType">
                      <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="document" />
                      <wsdl:operation name="requestAvailableAgents">
                        <soap:operation soapAction="urn:requestAvailableAgents" style="document" />
                        <wsdl:input>
                          <soap:body use="literal" />
                        </wsdl:input>
                        <wsdl:output>
                          <soap:body use="literal" />
                        </wsdl:output>
                      </wsdl:operation>
                    </wsdl:binding>
                    <wsdl:binding name="AgentProviderServiceSoap12Binding" type="ns:AgentProviderServicePortType">
                      <soap12:binding transport="http://schemas.xmlsoap.org/soap/http" style="document" />
                      <wsdl:operation name="requestAvailableAgents">
                        <soap12:operation soapAction="urn:requestAvailableAgents" style="document" />
                        <wsdl:input>
                          <soap12:body use="literal" />
                        </wsdl:input>
                        <wsdl:output>
                          <soap12:body use="literal" />
                        </wsdl:output>
                      </wsdl:operation>
                    </wsdl:binding>
                    <wsdl:binding name="AgentProviderServiceHttpBinding" type="ns:AgentProviderServicePortType">
                      <http:binding verb="POST" />
                      <wsdl:operation name="requestAvailableAgents">
                        <http:operation location="AgentProviderService/requestAvailableAgents" />
                        <wsdl:input>
                          <mime:content type="text/xml" part="requestAvailableAgents" />
                        </wsdl:input>
                      </wsdl:binding>
                    </wsdl:binding>
                  </xs:complexType>
                </xs:element>
              </xs:complexType>
            </xs:sequence>
          </xs:complexType>
        </xs:complexType>
      </xs:element>
    </xs:schema>
  </wsdl:types>
</wsdl:definitions>
```
<wsdl:operation>
  <wsdl:output>
    <mime:content type="text/xml" part="requestAvailableAgents"/>
  </wsdl:output>
</wsdl:operation>

<wsdl:service name="AgentProviderService">
  <wsdl:port name="AgentProviderServiceHttpSoap11Endpoint"
    binding="ns:AgentProviderServiceSoap11Binding">
    <soap:address location="http://192.168.172.128:9762/services/AgentProviderService.AgentProviderServiceHttpSoap11Endpoint"/>
  </wsdl:port>
  <wsdl:port name="AgentProviderServiceHttpsSoap11Endpoint"
    binding="ns:AgentProviderServiceSoap11Binding">
    <soap:address location="https://192.168.172.128:9443/services/AgentProviderService.AgentProviderServiceHttpsSoap11Endpoint"/>
  </wsdl:port>
  <wsdl:port name="AgentProviderServiceHttpSoap12Endpoint"
    binding="ns:AgentProviderServiceSoap12Binding">
    <soap12:address location="http://192.168.172.128:9762/services/AgentProviderService.AgentProviderServiceHttpSoap12Endpoint"/>
  </wsdl:port>
  <wsdl:port name="AgentProviderServiceHttpsSoap12Endpoint"
    binding="ns:AgentProviderServiceSoap12Binding">
    <soap12:address location="https://192.168.172.128:9443/services/AgentProviderService.AgentProviderServiceHttpsSoap12Endpoint"/>
  </wsdl:port>
  <wsdl:port name="AgentProviderServiceHttpEndpoint"
    binding="ns:AgentProviderServiceHttpBinding">
  </wsdl:port>
  <wsdl:port name="AgentProviderServiceHttpsEndpoint"
    binding="ns:AgentProviderServiceHttpBinding">
    <http:address location="https://192.168.172.128:9443/services/AgentProviderService.AgentProviderServiceHttpsEndpoint"/>
  </wsdl:port>
</wsdl:service>

<wsdl:documentation>SelectedAgentService</wsdl:documentation>
<wsdl:types>
  <xs:schema attributeFormDefault="qualified" elementFormDefault="qualified" targetNamespace="http://service.web.agentfactory.magicnet.com">
    <xs:element name="requestParticularAgent">
      <xs:complexType>
        <xs:sequence>
          <xs:element minOccurs="0" name="agentId" nillable="true" type="xs:string"/>
          <xs:element minOccurs="0" name="agentRole" nillable="true" type="xs:string"/>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
  </xs:schema>
</wsdl:types>

Selected Agent Service: WSDL

<wsdl:definitions xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
  xmlns:wsaw="http://www.w3.org/2006/05/addressing/wsdl"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"
  xmlns:soap12="http://schemas.xmlsoap.org/wsdl/soap12/"
  targetNamespace="http://service.web.agentfactory.magicnet.com">
  <wsdl:documentation>SelectedAgentService</wsdl:documentation>
  <wsdl:types>
    <xs:schema attributeFormDefault="qualified" elementFormDefault="qualified" targetNamespace="http://service.web.agentfactory.magicnet.com">
      <xs:element name="requestParticularAgent">
        <xs:complexType>
          <xs:sequence>
            <xs:element minOccurs="0" name="agentId" nillable="true" type="xs:string"/>
            <xs:element minOccurs="0" name="agentRole" nillable="true" type="xs:string"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
    </xs:schema>
  </wsdl:types>
<xs:complexType>
  <xs:sequence>
    <xs:element minOccurs="0" name="return" nillable="true" type="xs:string" />
  </xs:sequence>
</xs:complexType>

<wsdl:message name="requestParticularAgentRequest">
  <wsdl:part name="parameters" element="ns:requestParticularAgent"/>
</wsdl:message>

<wsdl:message name="requestParticularAgentResponse">
  <wsdl:part name="parameters" element="ns:requestParticularAgentResponse"/>
</wsdl:message>

<wsdl:portType name="SelectedAgentServicePortType">
  <wsdl:operation name="requestParticularAgent">
    <wsdl:input message="ns:requestParticularAgentRequest" wsaw:Action="urn:requestParticularAgent"/>
    <wsdl:output message="ns:requestParticularAgentResponse" wsaw:Action="urn:requestParticularAgentResponse"/>
  </wsdl:operation>
</wsdl:portType>

<wsdl:binding name="SelectedAgentServiceSoap11Binding" type="ns:SelectedAgentServicePortType">
  <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="document"/>
</wsdl:binding>

<wsdl:binding name="SelectedAgentServiceSoap12Binding" type="ns:SelectedAgentServicePortType">
  <soap12:binding transport="http://schemas.xmlsoap.org/soap/http" style="document"/>
</wsdl:binding>

<wsdl:binding name="SelectedAgentServiceHttpBinding" type="ns:SelectedAgentServicePortType">
  <http:binding verb="POST"/>
</wsdl:binding>

<wsdl:service name="SelectedAgentService">
  <wsdl:port name="SelectedAgentServiceHttpSoap11Endpoint" binding="ns:SelectedAgentServiceSoap11Binding">
    <soap:address location="http://192.168.172.128:9762/services/SelectedAgentService.SelectedAgentServiceHttpSoap11Endpoint"/>
  </wsdl:port>
</wsdl:service>
<soap:address
  location="https://192.168.172.128:9443/services/SelectedAgentService.SelectedAgentServiceHttpsSoap11Endpoint" />
</wsdl:port>
<wsdl:port name="SelectedAgentServiceHttpsSoap12Endpoint"
  binding="ns:SelectedAgentServiceSoap12Binding">
  <soap12:address
    location="https://192.168.172.128:9443/services/SelectedAgentService.SelectedAgentServiceHttpsSoap12Endpoint" />
</wsdl:port>
<wsdl:port name="SelectedAgentServiceHttpSoap12Endpoint"
  binding="ns:SelectedAgentServiceSoap12Binding">
  <soap12:address
    location="http://192.168.172.128:9762/services/SelectedAgentService.SelectedAgentServiceHttpSoap12Endpoint" />
</wsdl:port>
<wsdl:port name="SelectedAgentServiceHttpEndpoint"
  binding="ns:SelectedAgentServiceHttpBinding">
  <http:address
    location="http://192.168.172.128:9762/services/SelectedAgentService.SelectedAgentServiceHttpEndpoint" />
</wsdl:port>
<wsdl:port name="SelectedAgentServiceHttpsEndpoint"
  binding="ns:SelectedAgentServiceHttpBinding">
  <http:address
    location="https://192.168.172.128:9443/services/SelectedAgentService.SelectedAgentServiceHttpsEndpoint" />
</wsdl:port>
</wsdl:service>
</wsdl:definitions>
Appendix D  Sample Agent Data Access Control Policy

<Policy PolicyId="RuntimeNode1Policy" RuleCombiningAlgId="urn:oasis:names:tc:xacml:1.0:rule-combining-algorithm:first-applicable"
  <Description>This policy will evaluate access control request for runtime nodes contributions</Description>
  <Target>
    <Subjects>
      <Subject>
        <SubjectMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
          <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">GroupRT-1</AttributeValue>
        </SubjectMatch>
      </Subject>
    </Subjects>
    <Resources>
      <Resource>
        <ResourceMatch MatchId="urn:oasis:names:tc:xacml:2.0:function:anyURI-regexp-match">
          <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">Data1</AttributeValue>
          <ResourceAttributeDesignator AttributeId="urn:oasis:names:tc:xacml:1.0:resource:resource-id" DataType="http://www.w3.org/2001/XMLSchema#anyURI"/>
        </ResourceMatch>
      </Resource>
    </Resources>
    <Actions>
      <Action>
        <ActionMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
          <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">Write</AttributeValue>
          <ActionAttributeDesignator AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id" DataType="http://www.w3.org/2001/XMLSchema#string"/>
        </ActionMatch>
      </Action>
    </Actions>
  </Target>
  <Rule RuleId="BaggageRule" Effect="Permit">
    <Description>this rule applies to contributions made by different runtime nodes in a group</Description>
    <Target>
      <Subjects>
        <Subject>
          <SubjectMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
            <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">GroupRT-1</AttributeValue>
          </SubjectMatch>
        </Subject>
      </Subjects>
      <Resources>
        <Resource>
          <ResourceMatch MatchId="urn:oasis:names:tc:xacml:2.0:function:anyURI-regexp-match">
            <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">Data1</AttributeValue>
            <ResourceAttributeDesignator AttributeId="urn:oasis:names:tc:xacml:1.0:resource:resource-id" DataType="http://www.w3.org/2001/XMLSchema#anyURI"/>
          </ResourceMatch>
        </Resource>
      </Resources>
      <Actions>
        <Action>
          <ActionMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
            <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">Write</AttributeValue>
            <ActionAttributeDesignator AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id" DataType="http://www.w3.org/2001/XMLSchema#string"/>
          </ActionMatch>
        </Action>
      </Actions>
    </Target>
  </Rule>
</Policy>

141
<AttributeValue
   DataType="http://www.w3.org/2001/XMLSchema#string">Write</AttributeValue>
   <ActionAttributeDesignator AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"
   DataType="http://www.w3.org/2001/XMLSchema#string"/>
</ActionMatch>
</Actions>
</Target>
</Rule>
</Policy>
# Appendix E  List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>NIST- Agent System Model</td>
<td>14</td>
</tr>
<tr>
<td>3.1</td>
<td>MagicNET System Phases</td>
<td>31</td>
</tr>
<tr>
<td>3.2</td>
<td>Agents Development and Execution Steps</td>
<td>33</td>
</tr>
<tr>
<td>3.3</td>
<td>MagicNET System</td>
<td>36</td>
</tr>
<tr>
<td>4.1</td>
<td>Creation and Validation of Trusted Mobile Agents</td>
<td>42</td>
</tr>
<tr>
<td>5.1</td>
<td>Mutual Authentication Protocol</td>
<td>47</td>
</tr>
<tr>
<td>5.2</td>
<td>Components and Steps in the Process of Adoption of Trusted Mobile Agents</td>
<td>48</td>
</tr>
<tr>
<td>6.1</td>
<td>Root Element of a Sample PolicySet Definition</td>
<td>52</td>
</tr>
<tr>
<td>6.2</td>
<td>Target Element for Agent Adoption Policies</td>
<td>54</td>
</tr>
<tr>
<td>6.3</td>
<td>Sample Resource Entry in the XACML Policy File</td>
<td>55</td>
</tr>
<tr>
<td>6.4</td>
<td>Sample Resource Entry in XACML Policy File</td>
<td>56</td>
</tr>
<tr>
<td>6.5</td>
<td>Creation of Policies</td>
<td>58</td>
</tr>
<tr>
<td>6.6</td>
<td>Components and Message during Agent Code Authentication</td>
<td>60</td>
</tr>
<tr>
<td>7.1</td>
<td>Runtime Node Components</td>
<td>63</td>
</tr>
<tr>
<td>7.2</td>
<td>Mobile Agent- PKCS7 Signed and Enveloped Data</td>
<td>67</td>
</tr>
<tr>
<td>7.3</td>
<td>Components and Message during Agent Authentication and Authorization</td>
<td>69</td>
</tr>
<tr>
<td>7.4</td>
<td>Scenario 1: Agent Baggage Protection</td>
<td>70</td>
</tr>
<tr>
<td>7.5</td>
<td>Scenario 2: Agent Baggage Protection</td>
<td>71</td>
</tr>
<tr>
<td>7.6</td>
<td>PKCS7 Packages during Agent Migration at Different Runtime Nodes</td>
<td>73</td>
</tr>
<tr>
<td>7.7</td>
<td>Messages and Components after Agent returns to its Owner</td>
<td>73</td>
</tr>
<tr>
<td>7.8</td>
<td>Agent-to-agent Authentication Protocol</td>
<td>76</td>
</tr>
<tr>
<td>8.1</td>
<td>Security Solution along Attack Time Line</td>
<td>81</td>
</tr>
<tr>
<td>8.2</td>
<td>Configuration of the Network Security System</td>
<td>84</td>
</tr>
<tr>
<td>8.3</td>
<td>CVDB Tables (static)</td>
<td>85</td>
</tr>
<tr>
<td>8.4</td>
<td>CVDB Tables (dynamic)</td>
<td>85</td>
</tr>
<tr>
<td>8.5</td>
<td>Layered Structure of Network Security System</td>
<td>86</td>
</tr>
<tr>
<td>8.6(a)</td>
<td>Download NVD Data Files</td>
<td>87</td>
</tr>
<tr>
<td>8.6(b)</td>
<td>Matching between Target DB and CVDB</td>
<td>88</td>
</tr>
<tr>
<td>8.7</td>
<td>Scanning Results Notification</td>
<td>89</td>
</tr>
</tbody>
</table>
Figure 8.8  Listing of Available Patches  
Figure 8.9  Browser displaying the URL chosen in Figure 8.8  
Figure 8.10  The Structure of the Suspicious Host List (SHL)  
Figure 9.1  Initial Panel- MagicNET Screen  
Figure 9.2  Initial Panel- MagicNET Login Screen  
Figure 9.3  User Registration Interface  
Figure 9.4  Server Registration Interface  
Figure 9.5  MagicNET Applications Panel  
Figure 9.6  Submit Certificate Request Interface  
Figure 9.7  Fetch Certificates Interface  
Figure 9.8  List Local Certificates Interface  
Figure 9.9  Agent Factory Management Interface  
Figure 9.10  Agent Creator: Authentication  
Figure 9.11  Agent Creator: Deposit Agent  
Figure 9.12(a)  Agent Factory Server Database Schema.  
Figure 9.12(b)  Agent Information Container Class  
Figure 9.13  ATA Interface, Fetch Agents  
Figure 9.14  ATA Interface, List Agents  
Figure 9.15  ATA Interface, Deposit Agents to AF after Appraisal  
Figure 9.16  ATA Interface, Assign Task Specifications  
Figure 9.17  PA Interface, List of Agents Pending  
Figure 9.18  PA Interface, Privileges Assignment  
Figure 9.19  WSAS Integration with MagicNET  
Figure 9.20  AgentProviderService published at WSAS  
Figure 9.21  SelectedAgentService published at WSAS  
Figure 9.22  TryIt WSAS Feature: SelectedAgentService  
Figure 9.23(a)  Management Station Interface  
Figure 9.23(b)  Web Services URL Fetch  
Figure 9.24  PAP Admin Interface: Create Target  
Figure 9.25  PAP Admin Interface: Create Policy  
Figure 9.26  Management Station: Agent Launch