CHAPTER FOUR

TECHNOLOGIES

FOR

DEVELOPMENT

OF NIDSMAM
CHAPTER 4  
TECHNOLOGIES FOR DEVELOPMENT THE NIDSMAM SYSTEM

4. INTRODUCTION
In the present days hacking techniques turn into more developed and involved creativity and thinking outside expectations. Though, the existing security solutions do not provide the required precise preventative security for organizations which demands to provide new security enhancement and solutions. Intrusion detection systems are one of the most recent solutions in network security and become a concern for all researches and studies. Alternatively, software agents become a vital design model between operating distributed systems similar as network security systems based-snort IDS and expert systems and statistical algorithms because of their best advantages and pros. Cooperative multi-agent systems MAS are finding applications in a broad range of domains, including sensor networks, robotics, distributed control, collaborative decision support systems, and data mining. Cooperative MAS including a group of autonomous agents which interact with one another and that to improve a global performance measure. Through the literature survey of numerous studies and research on NIDS, multi-agents, Java expert system shell Jess, adaptive threshold algorithm, snort rules, etc. It is noticed that some of these research and studies dealt with these issues either a separate form or combination of one or more tools to evaluation results of the network traffic. Studies and research accomplished in the literature survey are addressed the detection of known attacks or unknown attacks which will support to reduce them on the network.

Diverse results and solutions to find the best methods suitable to reduce attacks, hackers and unauthorized access to networks and applications that are important to the public and private sectors. In this chapter, explain the importance of software tools and algorithms which are used in the design of this project. All the requirements which have been distributed in this chapter are fundamentals of the proposed system. In this chapter we covered multi-agent technology and Java expert system shell and snort rules network intrusion detection system and adaptive threshold algorithm and decision support system to enhancement in network security applications.
4.1 LITERATURE SURVEY OF MULTI-AGENTS

There are many research organizations and commercial organizations are engaged in the comprehension of agent applications, and a substantial number of agent construction tools have been comprehended. Some of the most interesting are AgentTool [DW00], ASL [KOE+98], AgentBuilder[Ret99], FIPA-OS[FIPAO], Grasshopper-2 [Gras], Beggent [KYH+99], RETSINA [SPW+96], MOLE [BHR+98], the Open Agent Architecture [MCM98] and Zeus [NNZ98]. AgentTool [DW00] is a graphical environment to create heterogeneous multi-agent systems. It is a sort of CASE tool, purposely oriented towards agent-oriented software engineering. The main advantages of this tool are the full support for the MaSE methodology which is developed by the same authors DeLoach and Wood [DW00] jointly with the tool and the independent from agent internal architecture with MaSE and agentTool. It is achievable to create multi agent systems made of agents with special internal architectures.

ASL [KOE+98] is an agent platform that supports the development in C/C++, Java, Jess, CLIPS and Prolog. ASL is based on the OMG’s CORBA 2.0 specifications. The use of CORBA technology facilitates flawless agent distribution and permits adding to the platform the language binding. It was supported by the used CORBA implementations. At the beginning, ASL agents used to converse by KQML messages, now the platform is FIPA acquiescent supporting FIPA ACL.

AgentBuilder [Ret99] is a tool for building Java agent systems based on two components one is the Toolkit and other are the Run-Time System. The Toolkit consists of tools for running the agent software development procedure while the Run-Time System make available an agent engine i.e. An interpreter which is used as implementation surroundings of agent software.

Agent Builder agents are based on a model obtained by the Agent-0 [Sho93] and PLACA [Tho94] agent models. FIPA-OS FIPA is one another software framework to build agent systems acquiescent to FIPA specification that has been realized by NORTEL. Similar type of framework offers the compulsory components realizing the agent platform of the FIPA reference model (i.e., The AMS, ACC and DF agents, and an internal platform message transport system), an agent shell and a template to create agents which communicate taking advantage of FIPA-OS agent platform.
Grasshopper-2 [Gras] is a clean Java based Mobile Agent platform, conformant to present agent standards, as defined by the OMG-MASIF (Mobile Agent System Interoperability Facility) [MBB+98] and FIPA specifications. Therefore, Grasshopper-2 is an open platform which enabling maximum compatibility and simple integration with other mobiles and intelligent agent systems. The Grasshopper-2 environment includes of a number of Agencies and a Region Registry which distantly connected via a selectable communication protocol. Some interfaces are specified to allow remote communications between the distinguished distributed components. Furthermore, Grasshopper-2 offers a Graphical User for easy to use admission to all the functionality of an agent system.

Bee-gent [KYH+99] is a software framework to develop agent systems acquiescent to FIPA specification which has been realized by Toshiba. Similar framework provides two types of agents: wrapper agents used to identify existing applications and mediation agents supporting the wrappers coordination by managing all their communications. Bee-gent also offers a graphic RAD tool to explain agents through state transition diagrams and a directory facility to locate agents, databases and applications.

RETSINA [SPW+96] offers reusable agents to realize applications. Each agent has four modules for communicating, planning, scheduling and monitoring the implementation of tasks and requests from other agents. RETSINA agents communicate through KQML messages.

MOLE [BHR+98] is an agent system developed in Java whose agents do not have an enough set of features to be considered really agent systems [GK94] [Sing99]. On the other hand, MOLE is important for the reason that it offers one of the best supports for agent mobility. MOLE agents are multi-thread entities identified by a globally unique agent identifier. Agents interact through two types of communication: through RMI for client/server interactions and through message exchanges for peer-to-peer interactions.

The Open Agent Architecture [MCM98] is an actually open architecture to recognize distributed agent systems in a number of languages, namely C, Java, Prolog, Lisp, Visual Basic and Delphi. Its main characteristic is its powerful facilitator that manages all the other agents in their tasks. The facilitator can take delivery of tasks from agents, decompose them and grant them to other agents.
Zeus [NNZ98] permits the rapid development of Java agent systems by giving that a library of agent components, by sustaining a visual environment for capturing user specifications, an agent building environment that consist of an automatic agent code generator and a collection of classes that form the building blocks of individual agents. Agents are collected of five layers: API layer, definition layer, organizational layer, coordination layer and at last communication layer. The API layer permits the communication with non-agent world. In 1999, a project at the Information-Technology Promotion Agency (IPA) in Japan engages an Intrusion Detection Agent (IDA) System [ATG99]. IDA is a standard host-based system that relies on mobile agents mostly to trace intruders amongst the variety of hosts involved in an intrusion.

In that year [QCP+99] chase an additional ambitious plan where the whole system functionally with mobile agents. However, only the architecture description has been offered, and no particulars have followed so far. In 2000, an IDS framework foundation on mobile agents has been described in [BM00]. Unluckily, the detection is dealt with superficially. Internationally, there have been few past attempts to take benefit of agents on the ground of intrusion detection as see in example [SZ00][DGY+05]. It is value mentioning the mobile-agents approach [WWZ+06] [DSW+04]. Moreover, there are further products such as Tritheme which is an ID underneath LPG license permitting the synchronized use of HIDS and NIDS approaches distributing the dissimilar functions under agents spread on the network under control [Tri01].

In 2002, Trapathi et al. [TAP+02] illustrates IDS which are developed as mobile application that roam the network to detect attacks and track intruders. MonALISA [Mon05] is a distributed and dynamic system capable to offer a full control and an overall monitoring of a complex system. The architecture of MonALISA is supported by autonomous entities able to collect, analyze and process data in distributed network collection of all the events happened in the host to that it is linked and storage the collected data in a sniffing file. The sniffer agent can replicate itself in order to improve the network charge. On the below the level, we are concerned to gather all the events happened through the network in real time. Sniffers are what are generally identified as sensors [Car01].
4.2 MULTI-AGENTS SYSTEMS

Today’s, the agent paradigm and developing agent technologies become a key for the implementation of flexible and scalable solutions for the open services market inside the information society. It grows to be a promising technology for designing and developing complex software systems such as security and distributed systems. An agent is a genuine or abstract entity, competent of performing on itself and on the environment. It can have a partial representation of this environment. It can communicate with other agents, and at last, its behavior is the product of its clarification, its knowledge and its interactions [Bal02] [Pin08]. There are many definitions of agents.

Agents can be defined by a set of their attributes like active, autonomous, goal-driven, and typically acting on behalf of a user or another agent. Agents are not a fresh paradigm, as they have been researched in the area of distributed artificial intelligence for numerous years. The widespread utilization of computers and their connectivity, mainly the web and the Java object-oriented programming language have provided a fresh inflow in the research, development, and deployment of agents. A specific inspiration for the use of agents is the vast amount of information available on the internet.

Agents have a considerable potential looking for information, filtering it, and extracting it from the source. The capability to represent and do something on behalf of the user represents a vital capability of agents and provides huge potential for their deployment. There are at least two communities at present pursuing agent research. These are the multi-agent system community and the mobile agents’ community. The multi-agent and intelligent system community pact frequently with stationary agents distributed on the network which communicate one another in order to follow a general goal. The other communities are deals with mobile agents as a paradigm for broadly distributed and heterogeneous systems. The multi-agents used to develop the projected security system, so this section will focus on multi-agents systems.

4.2.1 Categorizations of Agents

There are a lot of features that might be employed for the categorization of agents, the type of agents hardware or software, mobility, size, intelligence, capability to learn,
architecture, relation to other agents in the multi-agent system, and types of applications where they are implemented[Bad03].

4.2.1.1 Types
Hardware agents and software agents are two types of agents. Hardware agents are robots; they have sensors for monitoring their environment and effectors with which they carry out substantial actions. Software agents are programs that please one or more definitions specified above. While spreading of computer networks these are become interesting. The novel programming paradigm and agent-oriented programming are the foundation on the distribution of a software system onto a number of independent components (agents) linked with a net and problem solved by means of their communication.

4.2.1.2 Mobility
Agents can be either static or mobile. Static agents are constantly located at a particular place whereas mobile agents can vary their location. It is not so motivating to differentiate hardware agents is mobile or static. A robot's capability to move is just one of its potential actions. Alternatively, software agents that can cross computer networks set up a new computational paradigm that is pretty dissimilar from the one introduced by the use of static software agents. When a static agent wants particular action to be executed at specific remote site then it will send a message to an agent at that location with the demand for that action. A mobile agent acts differently. A mobile agent would broadcast itself to the remote site and call upon the action execution.

4.2.1.3 Size and Intelligence
Agents can be comes in a variety of sizes and can possess different amounts of intelligence. The intelligence of a software agent is comparative to its size. The intelligence of an agent is generally accomplished by encoding some information. If the bigger the knowledge base then the greater the agent's intelligence. The categorization of the software agents is as per their intelligence. It is same as the one that is based on their
size. Software agents can be roughly divided into three classes 1) big-sized agents 2) middle-sized agents and 3) micro-agents.

4.2.1.3.1 Big-Sized Agents
A big-sized agent resides in and controls either one or more computers. It shows sufficient capability to be useful even when it acts single with no other agents in a multi-agent system. Collaborating with new agents in a multi-agent system provide few additional abilities to a large sized agent.

A big-sized agent can be as large and as intelligent as an expert system. An example of similar type of agents is given in [HJF94] where a system for distributed medical care is described. A pharmacy agent is intelligent enough to recommend medicine for a patient. On the other hand, if the pharmacy agent assists with other agents (expert systems) then whole medical care system can be covered with a multi-agent system.

4.2.1.3.2 Middle-Sized Agents
A middle-sized agent is the one that is not sufficient itself without other agents in a multi-agent system or without additional software. Though, it is intelligent to execute some non-trivial tasks. A user-interface agent that acts single without other agents (it is not part of any multi-agent system) and performs few easy events can be categorize as a middle-sized agent. Examples of such middle-sized agents can be found in [BI97a] [BIB97a] [BI97b] [BIB97b]. Mobile agents are also called as middle-sized agents.

They pass through a computer network, and it is suitable to create them as little as possible. Enough space is not available for a huge agent's knowledge base. Some applications for example [Lan97] only extend a conventional programming language (Java, C, and C++) with the capability of a program to stop its implementation at one site. It transfers itself to some other site and carry on its implementation at another machine.

4.2.1.3.3 Micro-Agents
The term agent is also used in [Min86]. These agents are also called the society of mind agents which do not acquire any intelligence. Minsky [Min86] uses them for the justification of how human intelligence is achieved. The intelligence comes into view as a
global effect of the overall activity of more simple and unintelligent agents. Unluckily, there is still no theory that give details how this system works and which can be used for the improvement of such systems.

4.2.2 Learning

The capability of an agent to learn and to adapt to changes in its environment is a wanted specialty. It creates an agent strong when sudden changes occur. As per the learning and adaptation, there are two cases of agents one is agents able to learn and other is agents unable to learn. Agents use learning capabilities [Bad03]:

- For adapting to non-predicted changes in the environment that is being continuously changed.
- In the training process that takes place prior to productive agent usage.

The training process is frequently used as the last phase in the development of a personal digital assistant PDA. In this case, knowledge can also be written directly without learning. Nevertheless learning is a much better choice [Mae94], for the reason that most users find it very complicated to program their PDAs. A PDA is an agent that performs like a secretary in an office. It learns by monitoring the actions of its user. Agents are unable to learn which have to be programmed in such a way that nobody can shock them. In several fields, this is not so hard to achieve. An example of such an agent is library helper in [Tho94], which is programmed in the agent-oriented programming language PLACA. Library helper has its rules that include every situation when it should react.

4.2.3 Architecture

The categorization of agents concerning their architecture splits agents into three aggregates [WJ94]:

- Agents with a deliberative architecture,
- Agents with a reactive architecture, and at last
- Agents with hybrid architecture.

These two features reactivity and proactively decide three modules of agent architectures. If an agent is more fascinating in following its own goals rather than instantaneously reacting to events in its environment. Its architecture is a deliberative one, the agents
whose importance is on reactivity are erecting by using reactive architectures. Hybrid architectures are those which uniformly execute deliberative or proactive and reactive features of agents.

4.2.4 Relationship with other Agents
This feature is further connected to multi-agents systems than to particular agents. There are four classes of agents [Bad03]:

• Cooperative agents
• Competitive agents
• Semi-competitive agents
• Single agents

Cooperative multi-agents systems are the ones where agents cooperate within their work. Each agent helps another agent if help is wanted and potential. Competitive multi-agents systems are those where agents compete amongst themselves. Winner agents are permitted to perform their actions and to persuade the behavior of a system. Semi-competitive multi-agents systems are wherever between the over given cases. Agents in few semi-competitive multi-agents systems can be egotistic, but they are enthusiastic to cooperate for the reason that their gain will be bigger if they cooperate than it would be without cooperation.

4.2.5 Agent Applications
An agent can be categorized according to the area of application where it is implemented. In common, the multi-agent system approach and agent-oriented programming are appropriate for open system programming. Open systems have the below features [Bur94]:

• Continuous availability,
• Extensibility,
• Decentralized control,
• A synchronicity,
• Inconsistent information,
• Arms-length relationships.
4.2.5.1 Cooperative Problem Solving and Distributed Artificial Intelligence

The multi-agents system approach is appropriate for lots of problems within distributed artificial intelligence. Distributed artificial intelligence DAI is a part of AI connecting to all characteristic of building systems that acquire more than one entity performing intelligent actions. A manufacturing plant by means of robots can be observed as a multi-agents system. Every robot is an agent if there is a central computer that controls the taken as a whole manufacture which can also be seen as an agent. These agents communicate in their language; every agent is autonomous and has their goals. The events executes by an agent are determined by the internal condition of the agent and the message(s) it has got. A construction of such system is facilitated when the multi-agents system approach is used.

Standard multi-agents system techniques and few agent-oriented programming languages for agent programming are a superior choice than ad hoc solutions. An example of a multi-agents system referred in Distributed artificial intelligence is systems for distributed medical care illustrated in [HJF94].

4.2.5.2 Personal Digital Assistants and Intelligent Interfaces

With the spreading out of the Internet, another type of agent is becoming more and more popular. PDA agent is a computer program intended for performing easy and time-consuming tasks on behalf of its user.

A PDA can handle e-mail of its user or it can monitor or find fascinating newsgroups or web sites on the Internet and sieve interesting information [Mae94]. A PDA can maintain the appointment schedule of its user and autonomously make or cancel appointments [Mae94][MCF+94]. A PDA can communicate with further PDAs and perform few tasks that would or else have to be performed by the user. As proposed in [Mae94], a personal assistant should be able to improve its knowledge in the following means:

- By observing user actions,
- By user programming (the user can present examples of good actions),
- By receiving feedback from user,
- By sharing experiences with other agents.
A PDA acts like an intelligent interface to some application or to the Internet. An intelligent interface gets used to itself to the preferences of the user and builds the user work more suitably and proficiently.

One such an interface to the Internet that applies one agent (software robot) is illustrated in [EW94]. One more interesting approach is given in [YMK+00]. At this time an agent is located in a mobile device. The agent employs the information about its geographical coordinates for limit the search results of an Internet search engine. An adaptive user interface which composes of a lot of competitive agents which are described in [LW96]. Different settings for the interface are symbolized with different agents. The agents with the best user response are active and establish the interface. Examples of agents entrenched in interfaces are the Microsoft Excel Chart Wizard for graphic creation and Microsoft Wizard for Windows 2000 Internet settings.

4.2.5.3 Agents for Information Retrieval

Agents can be making use of information recovery. Suppose that each electronic source of information has an agent attached. When a user asks his/her agent for specific information then the agent will begin searching process. It will make use of a computer network to communicate with agents associated with ftp sites, databases, etc [Bad03].

4.2.5.4 Believable Agents

Believable agents are the agents used in the entertainment industry. Computer games and animated films utilize believable agents to accomplish human-like or animal-like features of their characters. As pointed out in [Bat94], emotions take part in a more significant role than intelligence in believable agent’s creation. A believable agent should be emotionally sensitive , and its emotions must be expressed authentically. A reactive architecture based on behaviors and emotions is depicted in [CF96].

4.2.5.5 Electronic Commerce

In [JW96] next to personal assistants, couples of additional probable software agent applications are declared. In the first one, agents are anticipated for an electronic marketplace. An agent could be sent to an electronic shop with specific quantity of
electronic money and orders the items from its user. It would be welcomed by a shop-
agent. If everything was in the order, they would do business. An agent could also go to or
call a restaurant and negotiate with a restaurant agent about reservations and meals. An
agent web-based market place has been developed at MIT Media Lab [CM96]. People
who sell goods generate selling agents while people interested in buying goods generate
buying agents. After the creation of an agent, its selling or buying policy is specified ,
and the agent is ready for negotiation with agents of the opposite type on behalf of its
proprietor.

4.2.5.6 Business Process Management
According to [JW96] agents could be also making use of it in business process
management. When two companies are going to work in partnership, the system that
usually has to be performed is to a certain extent complex.
A lot of paper work has to be completed between various departments in both companies
legal department, technical department, market department, etc. Software agents might do
an enormous part of this work. Papers [JFN+00a] and [JFN+00b] illustrate an agent-
based business process management system developed for British Telecom. As an
alternative of using a fixed inflexible workflow, negotiating agents are used. An approach
to business process management with mobile agents is described in [BIP99].

4.3 INTELLIGENT AGENTS AND NETWORK SECURITY
Centralized network security systems are not scalable because underneath intense
network load the systems experience from poor performance due to the load central
processing elements. One of the most extensively used description of the intelligent agent
is a system that enjoys the following features [BPG+01]:

➢ Autonomy: agents operate without straight interference of humans or others, and have
a few kind of control over their actions and internal state. Social capabilities of the agents
collaborate with other agents via similar kind of agent-communication language.
➢ Reactivity: agents recognize their environment and respond in an appropriately fashion
to modifications that occur in it.
Pro-activeness, agents do not only act in reply to their environment they are capable to demonstrate goal-directed behavior by taking the proposal. Mobile agent is a software object that is not bound to the system where they begin implementation but also it has the features of the intelligent agent beside the capability to migrate from one system in a network to another with their code and execution state.

Multi-agent system is a system platform that can supervise network of intelligent and mobile agents that effort together to solve a problem according to the information of each agent in the network. Intelligent agents have four main types which are: Simple Reflex agents which act only on the basis of the current percept. The agent function is based on the “if condition then action”, Model-Based Agent that manages moderately noticeable environments. Its current state is stored within the agent preserving similar type of structure which illustrates the part of the world which cannot be visible. This behavior needs information on how the world behaves and works, Goal-Based agents which agent decides amongst multiple possibilities and choosing the one which attains a goal state and Utility-Based agents which agent chooses the best action that attains their goals. These evaluations can be acquired through the use of an effectiveness function which plans a state to a measure of the utility of the state.

4.3.1 Development Standards of Intelligent Agent

Retroactively, agent technology had bears from diverse terminology and heterogeneity of technical approaches because of the not have of standards [Abd09]. In the present day, two main standards addressing interoperability of agents are available which are the Object Management Group's Mobile Agent System Interoperability Facility (MASIF) and Foundation for Intelligent Physical Agents (FIPA). On the other hand, these standards are distant from being total, needs validation and more improvements. MASIF is based on agent platforms, and it facilitates agents to migrate from one platform to another. FIPA specification is based on remote communication services using specific protocols. The former is mainly based on mobile agents travelling among agent system via CORBA interfaces and does not address inter-agent communication. The latter spot light on intelligent agent communication via content languages and do not declare much about mobility. FIPA adopt an agent communication model which can improved state of the
nature of collaboration and is more appropriate for integration with other AI technologies. MASIF adopts a mobile agent paradigm which is more suitable in situations where dynamic and autonomous swapping, replacement, modification, and updating of application components are required.

4.3.2 Specification of FIPA

The Foundation for Intelligent Physical Agents (FIPA) [Fip99] is an international non-profit association of companies and organizations contributing to the effort to create specifications of basic agent technologies. FIPA is foreseen not just as a technology for one application but also generic technologies for dissimilar application areas, and not just as autonomous technologies but as a set of vital technologies that can be integrated by developers to make complex systems with a high degree of interoperability.

FIPA is supports on two main assumptions. The initial is that the time to achieve consensus and to complete the standard should not be long. Mainly it should not act as a barrier on improvement rather than an enabler ahead of industries formulate commitments. The second is that only the external behavior of system components should be specified and leaving implementation details and its internal architectures to agent developers. In reality, the internal architecture of Jade is proprietary although it fulfills with the interfaces specified by FIPA. FIPA has been extensively renowned as the main standard in the area of agent-based computing. Figure 4.1 shows a summary of FIPA architecture. The below the description of FIPA model is indicated to [FIPA02].

![Figure 4.1](image-url)
4.3.2.1 Agent Platform

FIPA means a standard model of an agent platform which is represented in the following Figure 4.2. The Agent Management System AMS is the agent who exerts administrative control over admittance to and use of the Agent Platform. Only single AMS will be present in a particular platform. The logical reference model is recognized for agent creation, registration, communication, location, migration and retirement. The agent execution reference model consists of few components as described in Table 4.1 [Abd09]. The AMS offers white-page and life-cycle service, preserving a directory of Agent Identifiers AID and agent state. Every agent ought to register with an AMS in order to obtain a valid AID. The Directory Facilitator DF is the agent who grants the default yellow page service in the platform. The Message Transport System is also identified as Agent Communication Channel ACC which is the software component controlling the entire the exchange of messages inside the platform, including messages to/from remote platforms. Jade completely complies with this reference architecture, and when a Jade platform is launched, the AMS and DF are instantly formed. Additionally the Messaging Service (implementing the ACC component) is for all time activated to permit message-based communication. The agent platform can be dividing on a number of hosts. Normally (but not necessarily) only one Java application, and as a result only one Java Virtual Machine JVM, is executed on every host. Every JVM is a basic container of agents that gives a complete run time environment for agent execution and permits many agents to alongside implement on the equivalent host. The main-container is the container where the AMS and DF exist. The other container contains instead of link to the major container and grants a complete run-time environment for the implementation of any set of Jade agents.

![Figure 4.2 Illustrates Architecture FIPA Agent Platform](image-url)
<table>
<thead>
<tr>
<th>Component</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Agent platform (AP)</td>
<td>Provides the physical infrastructure in which agents can be deployed. The AP consists of the machine(s), operating system and agent support software. The internal design of an AP is an issue for agent system developers and is not a subject of standardization within FIPA.</td>
</tr>
<tr>
<td>Agent</td>
<td>Combines one or more service capabilities into unified and integrated execution model. An agent must have at least one owner; also it should be registered at a number of transport addresses at which it can be contacted.</td>
</tr>
<tr>
<td>Directory facilitator (DF)</td>
<td>The DF provides yellow pages services to other agent. Agents may register their services with the DF or query the DF to find out what services are offered by other agents.</td>
</tr>
<tr>
<td>Agent management system</td>
<td>The AMS exerts supervisory control over access to and use of the AP. Only one AMS will exist in single AP. The AMS maintains a directory of AIDs, also offers white pages services to other agents. Each agent must register with an AMS in order to get a valid AID.</td>
</tr>
<tr>
<td>Message transport service</td>
<td>The default communication method between agents on different APs. The reference model for agent message transport comprises three levels: message transport protocol (MTP); used to carry out the physical transfer of messages via the internet, these protocols such as IIOP (Inner-Orb Protocol), HTTP (Hypertext Transfer Protocol). Message Transport Service (MTS); supports the transportation of FIPA ACL messages between agents on any given AP and between agents on different APs. Agent communication Language (Acl); represents the payload of the messages carried by both the MTS and MTP.</td>
</tr>
<tr>
<td>Software</td>
<td>Describes all non-agents, executable collections of instructions accessible through an agent. Agents may access software, for example, to add new services, acquire new communications protocols or security protocols/algorithms, access tools which support migration, etc.</td>
</tr>
</tbody>
</table>

*Table 4.1 [Different Components of the Agent Management System of the FIPA]*
4.3.3.2 Agent Programming Languages

Study in multi-agent-based systems has in recent times guided to the development of practical programming languages and tools that are suitable for the implementation of these systems. Major agent programming languages have some primary platform which implements its semantics anxious with permitting general technique for appropriate aspects such as an agent communication and coordination. The most established languages will be escorted by some Integrated Development Environment IDE, projected to improve the efficiency of programmers by automating tire some coding tasks. Normally these will provide functionalities such as project management, creating and editing source files, refactoring, make and run process, and testing [BBD+06]. Agent-based programming languages and platforms are broadly executed using Java such as Aglets, MASON and Jade.

4.3.3 Applying Agents to Intrusion Detection: Related Work

A number of projects have achieved the utilization of agents in intrusion detection systems. The Computer Immunology Project [HWH+03] Java agents for meta-learning (JAM) and AAFID projects each one scrutinized the problem in diverse ways. The AAFID project [BJF+98] at Purdue’s COAST project is a flexible, distributed IDS infrastructure with some likeness to the MAIDS design. It has developed IDS by means of Perl agents for quick prototyping and cross-platform compatibility. The communications between agents are minor. Their project also evaluated the agent-based approach to intrusion detection. The Computer Immunology Project at the University of New Mexico [WFS+99] [FSH+97] discovered designs of IDSs based on thoughts collected by examining animal resistant systems. Small, individual agents would roam a distributed system, identify intrusions, and revolutionize the intrusions. One piece of the project developed intelligence of self for security related computer programs by observing the usual sets of system calls executed by the programs. This intelligence of self can be used to identify intrusions by discovering when a program executes an abnormal set of system calls. The JAM Project at Columbia University [LWS+98][SJP+97] make use of intelligent, distributed Java agents and data mining to learn models of scam and intrusive behavior that can be common between organizations.
4.3.4 Agent Life Cycle

According to agent platform life cycle in FIPA specification; these are represented in Figure 4.3 and are explained below:

- **Initiated**: the agent object is built but not registered itself, due to this it has neither a name nor an address and cannot communicate with further agents.

- **Active**: the agent object is registered with the AMS has its ordinary name and address and can have an access to all the several Jade features.

- **Suspended**: the agent object is presently blocked and its inner thread is suspended and due to this no agent behavior is being executed.

- **Waiting**: the agent object is blocked and waiting for something. Its internal thread is inactive on a Java monitor and will get up when quite a few states is met typically when a message arrives.

- **Deleted**: the agent is absolutely dead. The internal thread has finished its execution and the agent is no more registered with the AMS.

- **Transit**: a mobile agent goes into this state whereas it is migrating to the new location. The system continues to buffer messages and then that massage will be sent to its new location.

![Figure 4.3 Illustrates Agent Life-Cycle as Defined by FIPA](image-url)
4.3.5 Java Agent Development Framework (JADE)

Java Agent Development Framework JADE is broadly spread and used in the academic environment and also take advantage of its Open Source availability and its technical superiority. JADE is a software improvement framework aimed at developing multi-agent systems and applications conforming to FIPA standards for intelligent agents. It consists of two most important products: a FIPA-compliant agent platform and a package to develop Java agents. In order to develop the framework, it is necessary code his/her agents in Java. In JADE, it has been entirely coded in Java and an agent programmer Java. JADE is written in Java language and is prepared of a variety of Java packages as giving application programmers to both ready-made pieces of functionality and abstract interfaces for custom based on application dependent tasks. Java was the programming language of option because of it is having a lot of attractive features, especially directed to object-oriented programming in distributed heterogeneous environments, Figure 4.4 demonstrate Jade architecture [BCT10][Nag].

![Jade Platform Diagram](image)

**Figure 4.4 [Illustrates Jade Architecture]**

4.3.5.1 Basic JADE Concepts

- **Container**: It is a running instance of the JADE running environment which consists of a number of agents. A solitary Main Container has to always be active in a platform and also on all other containers register with it as soon as they can start.
4.3.5.2 Main Container

There are two special agents for main container:

- Agent Management system AMS that gives the naming service as the name make sure the uniqueness to create or destroy the agents.
- Directory Facilitator DF that gives a Yellow Pages service by way of which an agent can find other agents supplying the services he necessitate for achieve his targets.

4.3.5.3 Main Container Responsibilities

There are many responsibilities for main container:

- Managing the container table (CT) which is the registry of the object orientation and convey the addresses of all container nodes while creating the platform.
- Managing the agent descriptor table (GADT), which is the registry of all agents present in the platform, together with their current status and location?
- Hosting the AMS and the DF.

4.3.6 JADE Principles

JADE is amply developed in Java and is based on the subsequent dynamic principles: [BCP03]

- **Interoperability**: JADE is obedient with the FIPA specifications [FIPA]. As a result, JADE agents can interoperate with other agents and make available that they meet the terms with the identical standard.
- **Uniformity and portability**: JADE provides a standardized set of APIs that are self-determined from the fundamental network and Java version. Additional in details, the JADE run-time provides the same APIs both for the J2EE, J2SE and J2ME environment. In hypothesis, application developers could fix on the Java run-time environment at deploy-time.
- **Easy to use**: the complication of the middleware is out of sight behind a simple and axiomatic set of APIs.
- **Pay-as-you-go philosophy**: programmers do not necessitate using all the features provided by the middleware. Features that are not used do not need programmers to know anything regarding them neither adds any computational overhead.
4.3.7 JADE Features
The following is the list of features that JADE provide to the agent programmer: [BCP03]

- **Distributed agent platform:** the agent platform can be dividing between various hosts. And therefore only one application Java virtual machine is executed on each host. Agents are implemented as Java threads and live inside agent containers that make available the runtime support to the agent execution.
- **Graphical user interface:** to deal with various agents and agent containers from a remote host.
- **Debugging tools:** to help out in developing multi agents’ applications based on JADE.
- **Intra-platform agent mobility:** containing transfer of both the state and the code of the agent only when it essential.
- **Support to the execution of multiple:** parallel and concurrent agent activities by means of the behavior model. JADE schedules the agent behaviors in a non-preemptive fashion.
- **FIPA-compliant Agent Platform:** this contains the AMS (Agent Management System) and the DF (Directory Facilitator). These components are automatically started at the agent platform start-up.
- **Many FIPA-compliant DFs:** it can be started at run time so as to implement multi-domain applications, where a domain is a logical set of agents, whose services are publicized throughout an ordinary facilitator. Every DF inherits a GUI and all the standard capabilities defined by FIPA (i.e. Capability of registering, deregistering, modifying and searching for agent descriptions; and capability of federating within a network of DF’s).
- **Efficient transport of ACL messages:** it is contained by the similar agent platform. Actually, messages which transferred are encoded as Java objects, rather than strings, with the intention of avoid marshalling and unmarshalling procedures. After crossing platform boundaries, the message is automatically renewed to/from the FIPA compliant syntax, encoding, and transport protocol. This conversion is clear to the agent implementers that only require dealing with Java objects.
- **Library of FIPA interaction protocols:** is ready to be used.
- **Automatic registration:** and deregistration of agents with the AMS.
- **FIPA-compliant naming service**: at start-up agents are obtained their GUID (Globally Unique Identifier) from the platform.
- **Support for application**: defined content languages and anthologies.
- **In Process Interface**: to permit external applications to launch autonomous agents.

### 4.3.8 The JADE Platform

JADE is a high-performance software platform used for conveying targeted solutions to complex business problems. Figure 4.8 shows snapshots for Jade. JADE does extremely well in managing complex, highly connected data, relationships and business rules: [Jad11]

- **Wide, interconnected networks of objects**: for instance as the web of connections between facts in an intelligence management system.
- **Deep, hierarchical object structures and decision trees**: for instance as drilling down, crossing and investigating a ship’s cargo in a container management system.
- **Manipulating large collections of related objects**: for instance as following the exported logs of a nation in a supply chain management system, or regulate transactions in a financial management system.

### 4.4 OVERVIEW EXPERT SYSTEM

An expert system is a computer program designed to reproduce the problem solving behavior of a human who is an expert in a tight domain or propriety. In the opening days, expert systems only included expert knowledge. Currently on the other hand, any system using expert system technology is called an expert system, even if not including extremely specialized practice in a certain domain. For that reason, the term knowledge-based system is more suitable, even though the mainly people use the shorter term expert system. Expert system has come out from early work in problem solving, primarily because of the grandness of domain-specific knowledge. A human expert’s knowledge is particular to a problem domain. In much similar way, expert systems are designed to attend to a specific domain, called the knowledge domain.
4.4.1 Related Work

In the early 1980’s Stanford Research Institute SRI was developed intrusion detection systems and was called the Intrusion Detection Expert System IDES [DN85] [LTG+92]. IDES was a system that constantly monitored user performance and identified suspicious events as they take place. In IDES, intrusions could be flagged by detecting departures from established usual behavior patterns for individual users. As the analysis methodologies developed for IDES matured, scientists at SRI developed an improved version of IDES called the Next-Generation Intrusion Detection Expert System NIDES [ADF+95] [ALJ+95]. NIDE was single one from the some intrusion detection systems of its generation that could function in instantaneous for nonstop monitoring of user activity or could run in a batch mode for regular analysis of the audit data. Though, the prime mode of function of NIDES was to run in real-time. A flow chart illustrating the real time operation of NIDES is shown in Figure 4.5. Nothing like IDES, which is an anomaly detection system, NIDES is a hybrid system that has an improved statistical analysis engine. In both IDES and NIDES, a profile of usual behavior based on a chosen set of variables is maintained by the statistical analysis unit. This allows the system to judge against the recent activity of the user/system/network with the anticipated values of the audited intrusion detection variables stored in the profile and then flag an anomaly if the audited activity is adequately far from the anticipated behavior. Every variable in the stored profile replicates the extent to which a specific kind of behavior is looked like to the profile built for it under normal conditions. By this way, this is computed is by connecting every measure/variable to a related random variable. The frequency distribution is built and updated in due course, as further audit records are analyzed. It is computed as an exponential weighted total with a half-life of 30 days. This involves that the half-life value makes audit records that were collected 30 days in the past to contribute with half as much weight as recent records; those gathered 60 days in the past contribute one–quarter as much weight, and so on. The frequency distribution is kept back in the form of a histogram with possibility linked with every one of the probable ranges that the variable can obtain. The cumulative frequency distribution is subsequently built by using the prearranged set of bin probabilities. By using this frequency distribution, and the worth of the equivalent measure for the current audit record, it is
probable to compute a value that reflects how far away from the normal value of the measure the current value. The tangible computation in NIDES [ALJ+95] deliver a value that is associated with how abnormal this measure is combining the values get for every measure and taking into consideration the correlation between measures, the unit computes an index of how distant the current audit record is from the normal state. Records away from a threshold are flagged as feasible as intrusions. Though the techniques used in [DN85][LTG92][ADF+95][ALJ+95] have more than a few disadvantages: the procedures are responsive to the normality hypothesis. If data on a measure are not normally distributed then the procedures would yield a high false alarm. The techniques are mostly unvaried in that a statistical norm profile is created for a single measure of the activities in a system. Though, intrusions frequently affect multiple measures of activities jointly.

![Figure 4.5](flow_chart.png)

**Figure 4.5 [Flow Chart of Real Time Operation in NIDES]**

**4.4.2 Components of Expert System**

An expert system is generally composed of a knowledge base (information, heuristics, etc.), Inference engine (analyzes the knowledge base), and the end user interface (accepting inputs, generating outputs) as shown in Figure 4.6[JS88].
4.4.2.1 Knowledge Base

Building a knowledge base covers the process of gathering knowledge and transforming it into a form that can be processed by a computer as shown in Figure 4.7 everywhere the data collected is transformed to the form that can be understandable by the expert system. Knowledge systems have traditionally been building by knowledge engineers, who are people interviewing domain experts and formalizing their knowledge [JS88].

4.4.2.2 Inference Engine

An inference engine is the change of the facts and rules in the knowledge base into a computer program with the intention of a common user can run to obtain certain results and recommendations. The inference mechanism is basically building of the expert
system which is aware of the user input and matches it with the rules and facts of the expert system. Figure 4.8 expresses the inference engine [JS88].

4.4.2.3 User Interface

The user interface is the element of the expert system where the user interacts with the expert system. The user inputs the expert system with his field specific explanation of a new case and the expert system take action with the suitable advice and clarification to the user. All the interaction between the user and the expert system are executed in this component of the expert system.

4.4.3 Characteristics of Expert system

Expert system operates as a reactive with replies to questions, asks for illuminations, makes recommendations and broadly assists the decision-making process [Inf10]. Expert systems have several characteristics:

1) **Operates as an interactive system:** this intends an expert system which replies to questions, asks for illuminations, and makes recommendations, assists the decision-making process.

2) **Tools have the ability to sift (filter) knowledge:** it consists of storage and recovery of knowledge, mechanisms to increase and update the knowledge base on a long-lasting basis.

3) **Make logical inferences based on knowledge stored:** it consists of simple reasoning mechanisms are used, knowledge base must have means of take advantage of the
knowledge stored, else it is of no use e.g., learning all the words in a language, without knowing how to aggregate those words in the form a meaningful sentence.

4) **Ability to explain reasoning**: it consists of remembers logical chain of reasoning, improves user confidence in reference and acceptance of expert system.

5) **Domain-Specific**: it consists of a particular system provides a narrow area of specialization e.g., A medical expert system cannot be used to find faults in an electrical circuit, quality of advice presented by an expert system is dependent on the amount of knowledge stored.

6) **Capability to assign confidence values**: it can convey quantitative information which can understand qualitatively derived values, and can address inaccurate and incomplete data through assignment of confidence values.

7) **Applications**: it consists of the best suited those dealing with expert heuristics for solving problems, not a suitable choice for those problems that can be solved using strictly numerical techniques.

8) **Cost-effective alternative to human expert**: it consists of expert systems have become more and more popular because of their specialization. Even though in a narrow field, encoding and storing the domain-specific knowledge is economic process due to small size specialists in many areas are scarce and the cost of consulting them is high; an expert system of those areas can be helpful and cost-efficient option in the end.

### 4.4.4 Features of Expert System

Generally the features in expert systems are [Inf10]:

1. **Goal driven reasoning or backward chaining**: an inference technique which uses if-then rules to frequently break a goal into smaller sub-goals which are easier to evidence.

2. **Coping with uncertainty**: the capability of the system to reason with rules and data which are not exactly known.

3. **Data driven reasoning or forward chaining**: an inference technique which make use of the rules to assume a problem solution from initial data.

4. **Data representation**: the raw in which the problem specific data in the system is stored and accessed.

5. **User interface**: that section of the code which creates simple to use system.
6. Explanations: the capability of the system to clarify the reasoning process that it used to found a recommendation.

4.4.5 Rule Based Expert System

A rule based expert system is a model expert system which runs on the basic rules supplied in the knowledge base and the facts given as input from the user [Exp06]. The architecture of the rule based expert system is exposed in Figure 4.9.

![Figure 4.9 - A Typical Rule Based Expert System](image)

4.4.5.1 Knowledge base

The knowledge base of a distinctive rule based expert system includes of the rules that function the expert system. The rules are defined based on the specific domain on which the expert system operates. The rules may be stored in a database or as strings in a text file. The rules in the text file or database are in use as input by the inference engine to process the user request based on the facts existing in the working memory.

4.4.5.2 Working Memory

The running memory of the rule based expert system contains the user input stored in the sort of facts or facts loaded from an external file. The facts in the working memory are the facts that decide the current state of the system. The facts in the working memory differ from time to time based on the user input and functions of the expert system. The
facts in the working memory are loaded into the inference engine where it matches the rules loaded into the expert system and execute the suitable rule.

4.4.5.3 Inference Engine
Inference engine is similar to the central processing unit of the expert system. The matching of the rules and facts are completed in this stage. Inference engine acquires the rules from the knowledge base and facts from the working memory and pattern match the facts to the suitable rule. The rules matched are approved on to the agenda where it matches the most suitable rule and execute the rule.

4.4.5.4 Explanation Facility
This is the facility where the rules execute by the inference engine are transformed into a mode which can be comprehensible by the user. Depend upon the output of the explanation facility the user gets the solution or provides with additional facts to obtain a superior explanation of the problem.

4.4.5.5 User Interface
User interface is the element of the rule based expert system where the user acts together with the expert system by supplying the user input and receiving the suitable output.

4.4.5.6 Knowledge Acquisition Facility
The Knowledge Acquisition Facility KAF is software that supplies a conversation between the expert system and the human experts for the reason of obtaining knowledge from the human experts. The KAF places this obtained knowledge in the system’s database [JS88].

4.4.6 Java Expert System Shell Jess
JESS is short for Java Expert System Shell. It is a rule engine and scripting environment written totally in Sun's Java language by Ernest Friedman-Hill at Sandia National Laboratories in Livermore, CA. Its influential scripting language provides access to all of Java's APIs. It was initially motivated by the CLIPS [Jess expert system shell, but now, it
has developed into a distinct Java-influenced and rule-based environment. It supplies a tool to develop systems with clever interpretation capabilities. It has a speedy and competent algorithm which is known as Rete algorithm. Rete algorithm can create a network of pattern-matching nodes to resolve problems with rules. Primarily, it will perform the pattern matches, and then employ a set of memories to store information about the outcomes of the matches, and then provide out the obtainable matches. Throughout all the existing rule engines, Jess is very little, light, and is also one of the fastest engines. Jess is written in Java language that is effortlessly to assimilate with other Java based techniques such as JADE etc. Using Jess, the user can create design rules according to use knowledge, and then build Java software to reason the rules.

4.4.6.1 Java Expert System Shell Application

Jess can be used in interrelated methods; it can be a rule engine-a unique type of program that extremely proficiently employs rules to data [Fei03]. A rule-based program can have hundreds or even thousands of rules, and Jess will frequently apply them to data in the mode of a knowledge base. Over and over again the rules will stand for the heuristic knowledge of a human expert in few domains and the knowledge base will explain the state of a developing situation an interview and an emergency. In this situation they are agreed to make up an expert system. Expert systems are broadly used in a lot of fields. Amongst the latest applications of expert systems are as the reasoning part of intelligent agents in enterprise resource planning ERP systems, and in order validation for electronic commerce. Except the Jess language is too a general-purpose programming language and in addition it can straightly authorization all Java classes and libraries. Therefore, Jess is also repeatedly used as a dynamic scripting or rapid application development environment. At the same time Java code usually must be compiled prior to it can be run a line of Jess code is executed instantly upon being typed. This permits you to experiment with Java APIs interactively, and build up huge programs gradually. It is also extremely simple to lengthen the Jess language with fresh commands written in Java or in Jess itself, and so the Jess language can be personalized for particular applications. Jess is consequently useful in a broad range of circumstances. One application for which Jess is not so well suited is as an applet intended for Internet use. Jess’s size (a little hundred
kilobytes of compiled code) creates it too huge for applet utilize excluding on high-speed LANs. Additionally, few of Jess’s capabilities are vanished when it is used in a browser: for example, permission to Java APIs from the Jess language may not work at all due to security limitations in few browsers. While building Web-based applications using Jess should powerfully believe of using Jess on the server side.

4.4.6.2 Rete Algorithm

The exemplary expert system has a fixed set of rules whilst the knowledge base changes constantly [Fei03]. On the other hand, it is an experimental fact that in the majority expert systems much of the knowledge base is also quietly fixed from one rule operation to the other. Even though fresh facts appear, and old ones are removed at all times, the percentage of facts that modify per unit time is usually fairly little. For this reason, the understandable implementation for the expert system shell is very unproductive. This understandable implementation would be to maintain a list of the rules and endlessly cycle through the list checking every one’s left-hand-side (LHS) next to the knowledge base and performing the right-hand-side (RHS) of any rules that apply. This is incompetent because nearly all of the tests made on every cycle will have the similar results as on the preceding iteration. Though, the knowledge base is steady most of the tests will be repetitive. You might identify the rules finding facts approach and its computational complexity is the order of \((RFP)\), where \(R\) is the number of rules, \(P\) is the average number of patterns per rule LHS and \(F\) is the number of facts on the knowledge base. This solves considerably as the number of patterns per rule raises. Jess instead uses an extremely proficient method called as the Rete algorithm. The typical papers on the Rete algorithm [For82] grow to be on the basis for a total generation of fast expert system shells. In the Rete algorithm the incompetence explains above is relieved theoretically by remembering precedent test results transversely iterations of the rule loop. Simply new facts are tested against any rule LHSs. In addition, as we will be illustrated underneath new facts are tested against only the rule LHSs to which they are most likely to be appropriate. The Rete algorithm is implemented by building a network of nodes, each of which stand for one or more tests found on a rule LHS. Facts that are being added to or removed from the knowledge base are processed by this network of nodes. At the base of
the network are nodes representing individual rules. When a set of facts filters all the way down to the bottom of the network it has gone through all the tests on the LHS of a specific rule and this set becomes activation. The linked rule may have its RHS implemented executed if the activation is not invalidated first by the elimination of one or more facts from its activation set [Fei03].

Each rule has an inference cycle including of three stages: match, select and execute. In matching stage, the conditions of the rules are matched against the facts to check which rules are to be executed. The rules whose conditions are met are stored in a list called agenda for firing. From the list of rules, one of the rules is selected to execute or fire. The selections scheme may rely on priority, decency of usage, specificity of the rule, or on other measures. The rule selected from the list is executed by executing the actions in the right hand side of the rule. The action may be a motivation, executing a user determined or built-in function or executing a decision table, or differently. Where the decision table engine is used to execute decision tables [Sel10], Figure 4.10 illustrates pattern matching: Rules and Facts.

4.4.6.2.1 Advantages of Rete Algorithm

The primary advantage of Rete algorithm is speed as it carries the benefit of structural resemblance in rules. Frequently several rules include like patterns or group of patterns. Rete algorithm ponds the common elements so that they require not be calculated again [Sel10].
4.4.6.2.2 Disadvantage of Rete Algorithm
The primary cons of Rete pattern matching algorithm are that it is memory intensive. Saving the state of the system using pattern matches and partial matches substantial amount of memory. The space complexity of Rete is the order of (RFP), where R is the number of rules, F is the number of asserted facts, and P is the average number of patterns per rule. Severely written rules run slowly. Moreover if the all the facts were to be compared against all the patterns then also the speed cannot be accomplished. But this is only the worst case [Sel10].

4.5 OVERVIEW OF SNORT
Snort is an open source Intrusion Detection System IDS, Snort is a libpcap-based packet sniffer and logger that can be employed as a lightweight network intrusion detection system NIDS that can be deployed to monitor miniature TCP/IP networks and identify a large diversity of doubtful network traffic also explicit attacks. Snort can also be deployed speedily to fill potential holes in a network's security coverage, similar as when a fresh attack raises and profitable security vendors are slow to release new attack recognition signatures. In 1998 developed by Martin Roesch and now it is developed by Source fire Inc [PP03]. Snort is a tool for minute, eventually used networks , and it is accessible under the GNU General Public License [GNU89] and is at no cost for use in any environment. Snort can be configured and left operation for lengthy periods of time without necessitate monitoring or administrative maintenance and can as a result it also is utilized as an integral part of the majority network security infrastructures. There are three major modes in which Snort can be configured: sniffer, packet logger and network intrusion detection system. Sniffer mode reads the packets off of the network and displays in a constant flow on the console. Packet logger mode logs the packets to the disk. Network intrusion detection mode is the most difficult and configurable configuration permitting Snort to investigate network traffic for matches against a user defined rule set and performs numerous actions based upon what it sees. Therefore, Snort provides extremely user-friendly surroundings for addition of fresh rules to its rule-based database without inviting delays when a new attack pattern is detected in the network.
4.5.1 Snort Features
Snort is open source software it can be best execute as network intrusion detection and prevention system software; subsequent are the important features of Snort:
1) Snort is an open source, freeware IDS/IPS tool.
2) Source code of snort can be modified.
3) Snort can detect threats like stealth port scans, CGI attacks, buffer overflows, well-known backdoors and system vulnerabilities, and DDoS clients, and also alerts the user about them.
4) Alert file points any suspicious or malicious attacks.
5) Log file is created with tcpdump formats of incoming and outgoing data packets.
6) Snort can be used to inspect HTTP traffic.
7) Shared object rules are accessible in Snort and can be used.
8) Snort supports target-based intrusion detection.
9) It builds up a new signature to find vulnerabilities.
10) It records packets in their human-readable form from the IP address.

4.5.2 Architecture of Snort
Snort's architecture is paying attention to performance, simplicity, and flexibility. Basically, Snort is a packet sniffer, which receives packets and then processes them through the preprocessor and passes them to the detection engine to check them against a series of rules [Roes99]. Snort is logically separated into multiple components. These components work together to become aware of particular attacks and to yield output in a needed format from the detection system. Figure 4.11 shows how these components are arranged. A Snort-based IDS includes of the following main components: [Sen.]
Figure 4.11 Illustrates Components of Snort

- Packet Decoder: it fits captured packets into data structures and describes link level protocols. So, it captures the subsequent level, decodes IP, and TCP/UDP to get information regarding port addresses. Snort alerts for malformed headers are unusual TCP option.
- Preprocessors: they are like filters which describe things that should be checked afterward in Detection Engine module similar to suspicious connection attempt to some TCP/UDP port or too many UDP packets received during a port-scan.
- Rule files: text files with rule sets written by means of an identified syntax.
- Detection Plug-ins: those modules indicated from its description in the rule files, and they are proposed to recognize patterns every time a rule is assessed.
- Detection Engine: making utilization of detection plug-ins, it equivalent packets against the rules earlier charged into memory seeing as snort initialization.
- Output plug-ins: alerts, logs, extern files, databases

4.5.3 Technical of Snort Rules
Snort is one of the most popular signature-based IDS in the employ at the time current. It is an open-source packet sniffer or logger and network IDS. The packets that arrive to the
network interface are analyzed by these rules. And they are attempting to match their features with those received in the rules stored in its rule base. If a particular packet matches the premises of any rule, that rule is executed, and an exact action is generated to provide notice of this fact. Snort rules are statements that described has to be completed to the sniffed packets. Every rule must be on a single line. Snort requires a case of correct rule syntax on every line of a Snort rule file. It is mostly collection of the Rule Header and the Rule Option. The header part describes the traffic based on IP address and port and the all-purpose action to be done to it as will it are logged, escalated or ignored. The rule alternative part includes additional specific circumstances to further narrow down traffic to fit into the particular event that we want detected [Lao10]. The rules in Snort have the subsequent structure.

1. **Rule header**: contains the background information about the rule including:
   - **Rule action**: the main actions that will be carried out when rule conditions are met:
     - Pass: this act notify snort to disregard the packet. This action plays a significant task in momentum up snort procedure in cases where we do not want to pertain cheeks on certain packets.
     - Log: this action tells snort to log the packet in a way as defined for the duration of the configuration of the snort sensor.
     - Alert: the alert action is utilized to send an alert message when rule conditions are true for a special packet.
     - Activate: this can be utilized to trigger an alert and activate another rule to additional check the authenticity of the nature of traffic.
     - Dynamic: these rules are the ones activated by activate rules and only by activate rules. This is significant in doing more complex rules with less false positives.
     - User defined Actions: Snort permits users to generate their own custom actions. This could be sending mail and logging to custom files or even database.
   - **Protocol**: the protocol used by the packet being analyzed. Presently, Snort is aware of the following protocols: IP, TCP, ICMP and UDP.
   - **Source information**: it contains IP address and port of the source computer from where the packet developed. It is also feasible to use the key word ‘any’ to apply the rule on all packets regardless of the IP address or port number.
• Destination information: it contains IP address and port of the destination computer in the packet. The keyword ‘any’ can be used again with the same meaning as ever.

• Direction: the direction points are the flow of data from source to destination. It divides the source and destination IP on the rule structure. It can be any of these two values:
  → Uni directional – The IP on the left is the source and the other is the Destination.
    - E.g. `log udp any any -> any 80`
  <> Bi Directional – This will trigger the rule for any flow of data relating to the IP addresses on the rule header
    - E.g. `log udp any any < > any any`

2. **Rule options**: it includes alert messages and information on the parts of the packet that must be examined to conclude if the rule action should be taken or not. It is not always present, and it has more than 50 options existing to account for dissimilar necessities in the description of possible rules. The rule option is where the power of snort rules is coming from. It offers the ability to detect particular network activity and flexibility in rule generation. It can have a few number of rule options divided by semicolons and is always enclosed within parentheses. An option is included of three components:
   1. Option keyword – identifies the option to use
   2. Semicolon – separates the keyword from value
   3. Option value – the value to pass to the option keyword

Figure 4.12 shows sample sort rule structure [Lao10].
4.6 Platform Independent Language

JAVA is a high language with excellent support for string treatment such as finding regular expressions and performs categorization. It permits simple execution of shell commands and I/O redirection hence allowing a JAVA script to start another application and read its output. This skill is essential for gathering and reading network traffic. One more advantage of JAVA is interpreted rather than compiled. This removes the need for time-consuming recompilation every time a small change to the code is made. A drawback of using an interpreted language is that if a compiled language was used the software will not run as fast as it would. For this system, an interpreted language should be adequate, and the advantage mentioned above is more important. Would be used a compiled language to reduce the amount of processing power needed by the NIDS. This will be mostly important when running the software on a network with a lot of traffic.

Since the interpreters are available for most operating system and hardware combinations JAVA is also extremely portable and system independent. This permits the system to be tested on different systems only if required. Jess which is essentially a rule based engine developed on the Java platform. It has all the important features of Java. System particular libraries are helpful to supply the needed programming language support for any software to build up a complete system is not available in Java. So, Java is also called a platform independent language. It decreases the overhead of either addition of needed libraries for the developed system to run or to recompile the software every time while there is altering in the system that uses the software. All these advantages that attach to Java are as well demonstrate even in Jess because of which the system turn into a platform independent system.

4.7 OVERVIEW OF ANOMALY DETECTION ALGORITHM

Anomaly detection systems can detect every kind of intrusion that does not match with normal behavior of the system. In order to gain normal behavior of the system for anomaly detection, system frequently uses techniques based on artificial intelligence, machine learning etc. As a result of their capability to detect zero-day or new attacks, anomaly detection systems have acquired the attention of research and industry world. Anomaly detection systems work on the theory of monitoring run-time variation from
normal behavior. Therefore, an alarm is triggered if the run-time variation exceeds or falls below a certain threshold. Fundamental principle of operation of anomaly detection is shown in Figure 4.13. It may be clearly observed that as soon as the detected behavior deviates further than a fixed threshold from normal behavior it is classified as anomalous [Ali09]. Traffic anomaly detection is a typical task for network administrators who with familiarity can usually distinguish anomalous traffic from normal traffic. A lot of approaches have been anticipated to automate this task. Nearly all of them challenge to develop a satisfactorily refined model to stand for the full range of normal traffic behavior.

![Figure 4.13 - Basic Principle of Operation of Anomaly Detection]

4.7.1 Related Work

To identify anomalies different kinds of methods are presently used like rule based approaches, pattern matching, finite state machine, statistical analysis etc. Statistical analysis has been employed to identify both anomalies equivalent to network failures, as well as network intrusions [WZS02]. Lazarevic and Ertoz [LE03] propose diverse anomaly detection algorithms like administer SVMs which were exceptionally promising in detecting fresh intrusions since they had exceptionally high detection rate but very high false alarm rate also. Additionally, in the Mahalanobis based approach they were presently investigating the thought of defining a number of types of normal behavior and measuring the distance to each of them in order to recognize the anomalies.
The flooding detection system which was suggested by Wang et al. [WZS02] uses measured network data that illustrate TCP operations to detect SYN flooding attacks. The chronological modification point detection employed here makes use of the nonparametric cumulative sum CUSUM method. By means of this approach on trace-driven simulations it has been exposed that SYN flooding attacks can be detected with high accurateness and practically short detection times.

A number of recent network anomaly detectors by Ali [Ali09] offer techniques to compute optimal thresholds for their particular algorithms [JPB+04] [LCD04] [IK04]. Though, nearly every of these studies do not provide for the time fluctuating behavior of the input and as a result fail to offer adequate performance under fluctuating traffic conditions [AJM+08].

Few commercial IDS products also alter their thresholds in accordance with the input traffic characteristics [Peak] [Cisc]. As these products employ proprietary algorithms we cannot remark on the accurateness and enhancement that can be accomplished by these algorithms. The majority relevant previous work in this area is by Agosta et al. [AWC+07] suggested an anomaly detector which sets its threshold according to the differences monitored in input. On the other hand, no such common technique is existing that can cater for changing input and work with any anomaly detector. From the time when intrusion detection algorithms are frequently updated in reply to developing attack characteristics we dispute that a practical threshold adaptation technique as an alternative of being planned for a particular anomaly detection algorithm should flawlessly function with every given algorithm. Though, a general adaptive threshold tuning method that can automatically accomplish an appropriate operating point on the ROC plane for any existing real-time anomaly detector is not available. But few current anomaly detectors offer methods to compute optimum thresholds for their particular algorithms [JPB+04], [LCD04], [IK04]. Furthermore, few commercial network-based anomaly detection system products also set their thresholds in agreement with the input traffic characteristics [Peak] [Cisc].

SPADE (Statistical Packet Anomaly Detection Engine) by Staniford et al. [SHM02] is a statistical anomaly detection system that is accessible as a plug-in for Snort [Roe99], and it may be used for automatic detection of stealthy port scans. SPADE was one of the first
papers that suggested using the idea of an anomaly score to detect port scans in its place of using the established approach of looking at \( p \) attempts over \( q \) seconds. In [SHM02] Staniford et al. Used an easy frequency based approach to compute the anomaly score of a packet. The fewer times a given packet was seen the superior was its anomaly score. In other words, the authors describe an anomaly score as the degree of peculiarity based on recent past activity. Just once the anomaly score crossed a threshold the packets were forwarded to a correlation engine that was designed to detect port scans.

Though, the one chief disadvantage for SPADE are that it has a very high false alarm rate. This is due to the fact that SPADE classifies all unseen packets as attacks a part from of whether they are actually intrusions or not. Deri et al. [DLS+03] illustrate that in each network there are few global variables that can be beneficially used for detecting network anomalies, regardless of the type of users and equipment. The major thought is the design of IDS that employs both signatures based and anomaly based detection. Barford et al. [BPD+02] use wavelet analysis to take away from the traffic the unsurprising ambient part and then learn the differences in the network traffic rate. Network anomalies are detected by employing a threshold to a deviation score computed from the analysis. Thottan and Ji [TMJ03] take management information base MIB data gathered from routers as time series data and utilization an auto-regressive process to model the process. Network anomalies are identified by checking sudden changes in the statistics of the data. Wang et al. [HDG02] take the dissimilarity in the number of SYNs and FINs (RSTs) accumulated inside one sampling period as time series data and use a nonparametric CUSUM method to detect SYN flooding by identifying the change point of the time series. Siddhartha [Sid06] implement two of the algorithms -Entropy and Holt Winter in equivalent and online to recognize further about the types of anomalies generated and tapered down the traffic that is causing those anomalies. Zhang et al. [WZS02] illustrate the utilization of change point monitoring to identify denial of service attacks. The objective of change-point detection is to establish if the detected time series is statistically harmonized and if not to get the point in time when the change happens. Non-parametric CUSUM is yet again employed for the detection of DoS attacks. Zou et al. [ZGT+05] established a methodology for speedy detection of internet worms called trend detection. It is a stand on the fact that a worm in a premature stage transmits
exponentially with a constant positive exponential rate. The system attempts to detect this tendency.

4.7.2 Adaptive Threshold Algorithm

Adaptive Threshold Algorithm [SPF04] a simple and easy algorithm that identifies anomalies based on infringement of a threshold that is adaptively set based on present traffic measurements. Recurring discrepancies and directions are taken care of by using an adaptive threshold whose value is set based on an estimation of the mean number of the packets under consideration or the rate, either of which are calculated from the current traffic measurements. If \( x_n \) is the observed value in the \( n \)-th time interval, and \( \mu_{\bar{n}-1} \) is the mean approximation from measurements earlier to \( n \), then the alarm condition is signaled at time \( n \): [VSP04]

\[
\text{If } x_n \geq (\alpha + 1) \mu_{\bar{n}-1} \\
\text{Where, } \alpha > 0 \text{ is the parameter that the percentage above the mean value which we consider to be a sign of anomalous behavior. The mean } \mu_{\bar{n}} \text{ can also be calculated over some past time window or using an Exponentially Weighted Moving Average (EWMA) of the earlier measurements as given below.}
\]

\[
\mu_{\bar{n}} = \beta \mu_{\bar{n}-1} + (1 - \beta) x_n
\]

Straight application of the above algorithm would give up a large number of false alarms (false positives). An easy revision that may advances its performance is to indicate an alarm after the minimum number of consecutive violations of the threshold. In this case, the alarm condition is given by:

\[
\text{If } \sum_{i=n-k}^{n} 1 \{ x_i \geq (\alpha + \gamma \mu_{\bar{i}-1}) \} \geq k
\]

Then alarm signaled at time \( n \);

Where \( k > 1 \) is a parameter that point outs the number of successive intervals the threshold must be infringed for an alarm to be lifted. The tuning parameters of the on top of the algorithm is the amplitude factor \( \alpha \) for calculating the alarm threshold, A lot of consecutive threshold violations \( k \) prior to indication of an alarm, the EWMA factor \( \beta \), and the length of the time period above which traffic measurements are taken.
4.8 STRATEGIES TO FALSE POSITIVES AND FALSE NEGATIVES

4.8.1 The Distribution Causes of False Positives (false alarms)

False alarms can be categorized into a variety of causes based on important and specific includes the following [Tim10]:

1. **Reactionary Traffic Alarms:** traffic that is reasoned through an extra network event, often non malignant. An example of this would be a NIDS device activates an ICMP flood alarm at what time it is really numerous destination unreachable packets caused by equipment failure somewhere in the Internet.

2. **Equipment-Related Alarms:** attack alerts that are raised by strange, unrecognized packets produced by specific network equipment. Load balancers mostly raise these types of alarms.

3. **Protocol Violations:** alerts that are originated by unidentified network traffic frequently caused by badly or oddly written client software

4. **True False Positives:** alarms that are produced by IDS used for no obvious cause. These are mostly caused by IDS software bugs

5. **Non Malicious Alarms:** produced in the course of few real incidents that are non malicious in nature.

4.8.2 The Distribution Causes of False Negatives

False negatives are further complicated to measure than false alarms and may be identified as an intrusion detection device not issuing alerts on justifiable attacks. There are a number of potential causes for these include the following:

1. **Network Design Issues:** network design faults similar as inappropriate port straddling on controls and traffic exceeding the capability of a switch or hub add to these issues. Further issues consist of numerous entry point networks where the NIDS device cannot observe all incoming and outgoing traffic.

2. **Encrypted Traffic Design Flaws:** these issues occur because the IDS are not capable to recognize encrypted traffic. Placing the NIDS at the back VPN termination points and make use of SSL accelerators are superior ways to guarantee the NIDS is understood all traffic.
3. **Lack of Change Control:** a lot of times false negative conditions are generated by a short of communication among networking and security staff. Nearly all of the time this is in the form of the network or server. Modification that are not appropriately communicated to security staff. As a result, security is not capable to put into practice measures to mitigate the risk associated with modifications in security stance.

4. **Improperly Written Signatures:** even though the attack is recognized and the signature is developed the signature does not appropriately identify the attack or mutations of the attack because it has not been written appropriately.

5. **Unpublicized Attack:** the attack is not openly identified for that because vendors have no information and no signature is developed.

6. **NIDS Design Flaw:** the NIDS device basically does not grab the attack due to unfortunate design or signature implementation.

7. **Poor NIDS Device Management:** for a diversity of causes the NIDS device cannot be appropriately configured factors of contributing contains:
   - a. Exclusionary rules to decrease false alarms that are too common.
   - b. The device is beneath in excess of load and cannot accurately process all data.
   - c. Alarming is not configured correctly and last.
   - d. The system administrator has a poor understanding of the vulnerabilities and threats related with particular attacks.

4.9 **OVERVIEW OF DECISION SUPPORT SYSTEM**

4.9.1 **Definition Decision Support System**

A decision support system can be characterized in a lot of ways. Few definitions highlight hardware and software components; further spotlight mainly on task (i.e., Satisfying the information requires of decision makers) whilst a few even illustrate user interfaces, job functions, and data flow, definitions of decision support systems consist of: [Nce06]

➢ **Decision support system:** it is reactive software-based programmed information system proposed to assist decision maker accumulate helpful information from raw data,
documents, personal acquaintance, and business models to recognize and resolve issues and to make decisions. [Inf06]

➢ **Decision support system**: it is reactive programmed system that gets together and presents data from a broad range of resources to assist people build decisions. Applications are not solo information resources, similar as a database or a graphics program, but somewhat the mixture of integrated resources working together. [Web06]

➢ **Decision support system**: a unified and incorporated set of programs that distribute data and information and supply the capability to inquiry computers on an ad-hoc basis, evaluate information and forecast the effect of potential decisions. [Pcm06]

A powerful definition of a decision support system should includes: users who comprehend what the data mean and how they can be reached with, a technology system hardware, software, and user interfaces that manipulates, a data system incorporating data from multiple resources openly for the reasons and a decision making system (user-driven inside an organization).

Decision Support Systems can be identified as computer technology solutions that may be used to maintain complicated decision making and problem solving [PMF+02]. To account decision issues complexity and uncertainty we comprehend the DSS as a set of computer-based tools that supply decision maker with interactive capacities. It goal is to improve his understanding and information basis about considered decision problem through usage of models and data processing. According to [HWS+02], decision support systems fall into five classes:

1. **Communications-Driven DSS**: applies network and communications technologies to facilitate cooperation and communication.

2. **Data-Driven DSS**: highlight admission to and manipulation of a time-series of internal corporation data and occasionally external data.

3. **Document-Driven DSS**: integrates a diversity of storage and processing technologies to make available whole document recovery and analysis.

4. **Knowledge-Driven**: means to propose or recommend actions to managers. These DSSs are individual computer systems with specialized problem solving experience.
5. **Model-Driven DSS or Model-oriented DSS**: accentuates access to and manipulation of a model, e.g. statistical, financial, optimization and/or simulation. Easy statistical and analytical tools offer the most primary level of functionality.

### 4.9.2 Components of Decision Support System Technology

The typical DSS tool design is included of the components for: [IN]

- *Database management abilities with permission to internal and external data, information and knowledge.*
- *Powerful modeling roles accessed by a model management system and*
- *User interfaces that allow interactive communication between the user and system.*

DSSs is interactive computer-based systems proposed to assist decision makers make use of data and models to recognize and resolve problems and make decisions. The "system must aid a decision maker in solving unprogrammed, unstructured (or 'semistructured') problems...the system must possess an interactive query facility, with a query language that ...is ...easy to learn and use" [BHH+81]. DSSs aid managers/decision maker’s utilization and manipulation data, apply checklists and heuristics, and construct and use mathematical models. According to Turban [TE95] a DSS has four main characteristics: it integrates both data and models; it is designed to help managers in their decision processes in semi-structured or unstructured tasks, it supports instead of replaces managerial judgment and its purpose are to enhance the usefulness of decisions, not the effectiveness with which decisions are being taken. The components of decision support system demand for suggestion should address the planning, development, testing, and check of at least five main technology elements: [Nce06]

1. **An extract, transform, and load (ETL) mechanism**: this may basically be software code that allows data from numerous formats to be conveyed into the data warehouse throughout a manual or automated process or it can engage incorporation servers that permit one part of the system to converse with another and extract information through transparent processes.
2. A data warehouse: this depository should permit data from numerous sources to be housed for handling and analysis after ETL or it can be a data collection tool that honestly admission to existing, interoperable databases.

3. An analysis tool: applications that can aggregate and disaggregate data, execute statistical analysis or other processes, and transform uncooked data into useful information.

4. A reporting tool: an application that can interpret analytical results into graphic, tabular, or other formats that authorize users to further effortlessly read or understand results and communicate them undoubtedly to policymakers, parents, community members, educators, and others who may have a require to know or an interest in the result.

5. A user dashboard: an interface customized to convene the information requirements of key user groups such as teachers, principals, administrative staff, policymakers, and parents.

4.10 CHAPTER SUMMARY

Intrusion detection systems become one of the important protection solutions in network security world. Our efforts in chapter four included: a literature survey in multi-agents and classifications of agents that includes: type, mobility, size and intelligence, learning, architecture, relationship with other agent, agent applications. Explain in this chapter intelligent agents and network security includes development standards of intelligent agent, specification of FIPA, agent platform, and agent programming languages. We described related work applying agent to intrusion detection, we explain a JADE agent can be in one of several states, according to agent platform life cycle in FIPA specification. We describe JADE includes: JADE Principles, JADE Features, The JADE Platform, So; using multi-agent technology in developing intrusion detection systems provides many features that enhance the performance and productivity of these systems. On the other hand, multi-agent technology quickly becomes a key design paradigm for the dynamic and heterogeneous open, distributed environments. We explain overview expert system, components of expert system, characteristics of expert system, features of expert system, rule based expert system.
We describe Jess; it provides a tool to develop systems with intelligent reasoning abilities. It has a fast and efficient algorithm which is called Rete algorithm. Rete algorithm can build a network of pattern-matching nodes to solve problems with rules. In this chapter we explain Snort is an open source IDS, Snort is a libpcap-based packet sniffer and logger that can be used as a lightweight NIDS that can be deployed to monitor small TCP/IP networks and detect a wide variety of suspicious network traffic as well as outright attacks, and explain briefly features and architecture of Snort, we describe technical Snort Rules. They definition of snort in order to choose a snort rules for developing rule-based systems, implementing snort rules arises many challenges to obtain preemptive intrusion detection system. The extensive study of the chosen related work of adaptive threshold algorithm and more details about adaptive threshold algorithm. We explain strategies to false positives and false negatives and described definition decision support system and components of DSS.