An agent-based framework for distributed intrusion detections

Dayong Ye
University of Wollongong

An Agent-Based Framework for Distributed Intrusion Detections

A thesis submitted in fulfillment of the
requirements for the award of the degree

Master by Research

from

UNIVERSITY OF WOLLONGONG

by

Dayong Ye

School of Computer Science and Software Engineering
May 2009

Dedicated to
Zhen Ye and Tonghua Wang

Declaration

This is to certify that the work reported in this thesis was done by the author, unless specified otherwise, and that no part of it has been submitted in a thesis to any other university or similar institution.

Dayong Ye
May 25, 2009

Abstract

Network application has become a part of our everyday life. With the increasing of convenience and popularity of network, more and more malicious users utilize network to obtain their vicious intentions. In order to protect network users' information security and privacy, various intrusion detection systems were proposed and developed in the last decade. Intrusion detection as an emerging technology has made great achievements in theory and practice, whose aim is to protect the confidentiality, integrity or availability of a system or resource. As a complex system, the development of an intrusion detection system includes many aspects, such as system architecture design, design and implementation of system components, system test in real cases, and so on.

Though many intrusion detection systems have been presented, most of them mainly focus on one or two aspects of intrusion detection systems. This thesis aims at providing a rudimentary solution for an agent-based Peer-to-Peer distributed intrusion detection framework. The major contributions of this thesis include the following five aspects.

1. Introducing a novel Peer-to-Peer framework which involve different agents on different peers;

2. Designing functionalities of each agent in the framework by using JACK/UML approach;

3. Representing knowledge of each agent about intrusion and detection according to employing ontology;

4. Developing an efficient task allocation protocol which is used to coordinate different hosts in the system to collaboratively detect distributed attacks;

5. Implementing and testing the framework in a reasonable manner by utilizing an agent development environment, i.e. $JACK^{TM}$.

In summary, this framework integrates agent technology, Peer-to-Peer architecture, ontology technique and a task allocation protocol. Implementation and experiments

show the potential applicability of this framework to real cases. In addition, this framework could help in development of a good intrusion detection system in open and complex environments.

Acknowledgements

Studying abroad is a tedious and tired journey. Without the help and support of many people, I cannot complete my research.

I am indebted to my supervisors, Associate Professor Minjie Zhang and Dr. Quan Bai. Their constant commitment and guidance was instrumental in the completion of this thesis, and in making it a fulfilling experience. I am grateful to Dr. Quan Bai for his kind help, encouragement and patient proofreading my thesis and research papers. I am also delightful for Associate Professor Minjie Zhang's enthusiasm for my everyday life. Furthermore, I thank the School of Computer Science and Software Engineering and the University of Wollongong for the financial support of conference attendance.

My thanks are extended to Mr. Shaojie Yuan, who often discusses with me in the lab and enriches my knowledge; and Mr. Guohua Yao, my house mate, who chats with me during our dinner time everyday and brings me a lot of fun.

I would like to express my deepest gratitude to my parents, Zhen Ye and Tonghua Wang, who always make their financial support, encouragement, understanding and love. Without their help, this thesis would not be finished. Thanks too, to my wife, Yun, for her constantly tolerating my selfishness and her delicious food. I hope she could forgive me for what I have done. I have dedicated this thesis to my parents and my wife for their patience, understanding and unconditional love.

Finally, thanks to all the anonymous reviewers of my research papers, and all my other dear friends and relatives who have supported me.

Publications

The followings are list of my research papers that have been already published during my Master study that is to be ended by the completion of this thesis.

- Dayong Ye, Quan Bai, and Minjie Zhang. BDI agent-oriented design for distributed intrusion detections. *Communications of SIWN*, 4:11-17, Jun. 2008.

- Dayong Ye, Quan Bai, and Minjie Zhang. Ontology-based knowledge representation for a p2p multi-agent distributed intrusion detection system. In *Proceedings of the 2008 IFIP International Workshop on Network and System Security (NSS 2008)*, pages 111-118, Shanghai, China, Oct. 2008.

- Dayong Ye, Quan Bai, and Minjie Zhang. P2P distributed intrusion detections by using mobile agents. In *Proceedings of the seventh IEEE/ACIS International Conference on Computer and Information Science (ICIS 2008)*, pages 259-265, Portland, Oregon, US, May 2008.

- Dayong Ye, Quan Bai, and Minjie Zhang. A mobile agent based peer-to-peer framework for distributed intrusion detections. In *Proceedings of the Eighth International Conference on Intelligent Technologies (InTech'07)*, pages 45-55, Sydney, AU, Dec. 2007.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Since the last decade, security issues, such as network intrusions, have become more and more serious with the growth of computer and network applications. Intrusion is a set of actions that attempt to compromise the confidentiality, integrity or availability of a resource. In order to prevent information from malicious attackers, Intrusion Detection Systems (IDSs) are used to detect various intrusions in network environments.

After Anderson [3] and Denning [11] presented the very first prototypes of IDSs, many works have been done on intrusion detection. Recently, with the development of artificial intelligence, multi-agents systems and intelligent agent technology provide a powerful paradigm for the modeling and developing of complex systems. Subsequently, many studies are undertaken on agent-based intrusion detection systems. The architectures of conventional agent-based IDSs are centralized which involve a central unit to monitor the entire system, such as [6], [31], [38], [81], and [84]. The centralized architectures have two obvious drawbacks. Firstly, the centralized forms may lead to a single point failure, because the failure of the central analyzer (e.g. the central analyzer is cracked by an attacker) would cause the whole system to be destructed. Secondly, the central unit is easy to become the bottleneck of the whole system when there are many simultaneous client requests. In an effort to overcome the two disadvantages, some researchers proposed agent-based Peer-to-Peer (P2P) architectures for IDS, e.g. [55] [79] [82]. However, most of these P2P architectures only allow hosts in a network to obtain information from their one-hop (direct linked) neighbors. This limitation may lead a system to make inaccurate decisions. The aim of this thesis is to provide a novel agent-based P2P framework for distributed intrusion detection, in which hosts derive information not only from the directed neighbors but also other indirect linked hosts if needed. Thereafter, the detailed design and knowledge representation of each agent in this framework are presented in this thesis as well. Furthermore, an efficient task allocation protocol which is utilized to allocate detection tasks among hosts is also introduced.

The goal of this chapter is to provide an overview of intrusion detection and the architectures of agent-based intrusion detection systems. Section 1.1 describes intrusions and their detections. Section 1.2 demonstrates the architectures and mechanisms of some current agent-based IDSs. Section 1.3 depicts the research concerns that remain open for agent-based IDSs in a P2P environment. Section 1.4 outlines this thesis, and also explains the outcomes of this research and how these outcomes are embodied in this thesis.

1.1 Intrusion and Intrusion Detection

A significant security problem for computer systems and networks is intrusion by users or softwares. User intrusion can take the form of unauthorized login to a machine or an authorized user illegally acquires privileges which are beyond those that have been authorized to that user. Software intrusion can be the form of a virus, worm, or Trojan horse. This thesis focuses on user intrusion.

From Internet Security Glossary [67], security intrusion and intrusion detection are defined as follows, respectively.

- *Security Intrusion*: An action, or a set of actions, which attempt(s) to gain access to a system or a resource without legal privileges to do so, or to prevent legitimate users from being served.
- *Intrusion Detection*: A security service that attempts to find suspicious events or actions which constitute an intrusion, and to provide real time warnings if possible.

In Subsection 1.1.1, a detailed description of various intrusions is provided, while an overview regarding intrusion detection is shown in Subsection 1.1.2.

1.1.1 Intrusion

There exist numerous categories of intrusions. In order to exhibit various intrusions systematically, much attention has been paid on attack taxonomies. Landwehr *et al.* [36] provided a taxonomy which is categorized according to genesis (how), time of introduction (when) and location (where). They contain subclasses of: *validation errors*, *boundary condition errors* and *serialization errors*, as a means of effecting an intrusion.

As detailed by McHugh [43], the common character of most taxonomies is that the categorization is based on the attacker's point of view. Thus, McHugh suggested a different way, which classifies attacks according to protocol layer, i.e. whether a completed protocol handshake is needed or not. Similarly, Guha [22] also provided an alternative categorization approach which relies on analysis of each layer of the TCP/IP protocol stack.

A taxonomy, which is based on the consequence of attacks, is proposed by DARPA Off-Line Intrusion Detection Evaluations [24] [41]. From this taxonomy, the sub-categories include *Denial of Service* (DoS), *Remote to Local* ($R2L$), *User to Root* ($U2R$) and *Probe*. Each of the four sub-categories are described as follows:

- *Denial of Service* (DoS): this attack utilizes any methods which can overload or crash some service in order to prevent other legitimate users from being served, e.g. SYN flood attack [64]. More details with regard to DoS attack can be found in [32].
- *Remote to Local* ($R2L$): unauthorized access from a remote machine to local ones, e.g. guessing password;
- *User to Root* ($U2R$): unauthorized access to local superuser (root) privileges, e.g., various "buffer overflow" attacks;
- *Probe*: surveillance and other probing, e.g., port scanning.

Joshi and Undercoffer [27] utilized ontology to classify attacks based on the following four criteria:

1. *Target of Attack*: the system or network component that is the target of an attack. The components may include the network protocol stack, the operation systems, applications and so on.
2. *Means of Attack*: the methods that are used by the attacker. This category involves input validation errors (buffer overflows, boundary condition errors, etc.), exploits and configuration errors.
3. *Consequences of Attack*: the end result of the attack. This category consists of DoS, $R2L$, $U2R$ and *Probe* as described above.
4. *Location of Attacker*: the location of the attacker, namely that whether the attacker is from the network or a local host.

In addition, Joshi and Undercoffer [27] also presented a complete ontology in a graphical form with Resource Description Framework (RDF) [58], which is easy to read and understand as shown in Figure 1.1. An ontology [21] is "an explicit specification of a conceptualization". Ontology is employed to support the sharing and reuse of formally represented knowledge among systems, such as Artificial Intelligence (AI) systems. Details about ontology will be depicted in Chapter 4.

The construction in Figure 1.1, presented in [27], is based on the notion that an intrusion is effected by some inputs from an attacker. These inputs are caused by some means and directed to some system components, and finally result in some consequences. Details regarding the notations in Figure 1.1 will be elaborated in Chapter 4. It should be noted that one attack instance could belong to two or more classes based on different criteria. For example, SYN flood attack [64] can be categorized into *Denial of Service*, if the criterion of classification is *Consequences of Attack*; meanwhile SYN flood attack [64] can also be labeled as *TCP Layer*, if the classification is based on *Target of Attack*.

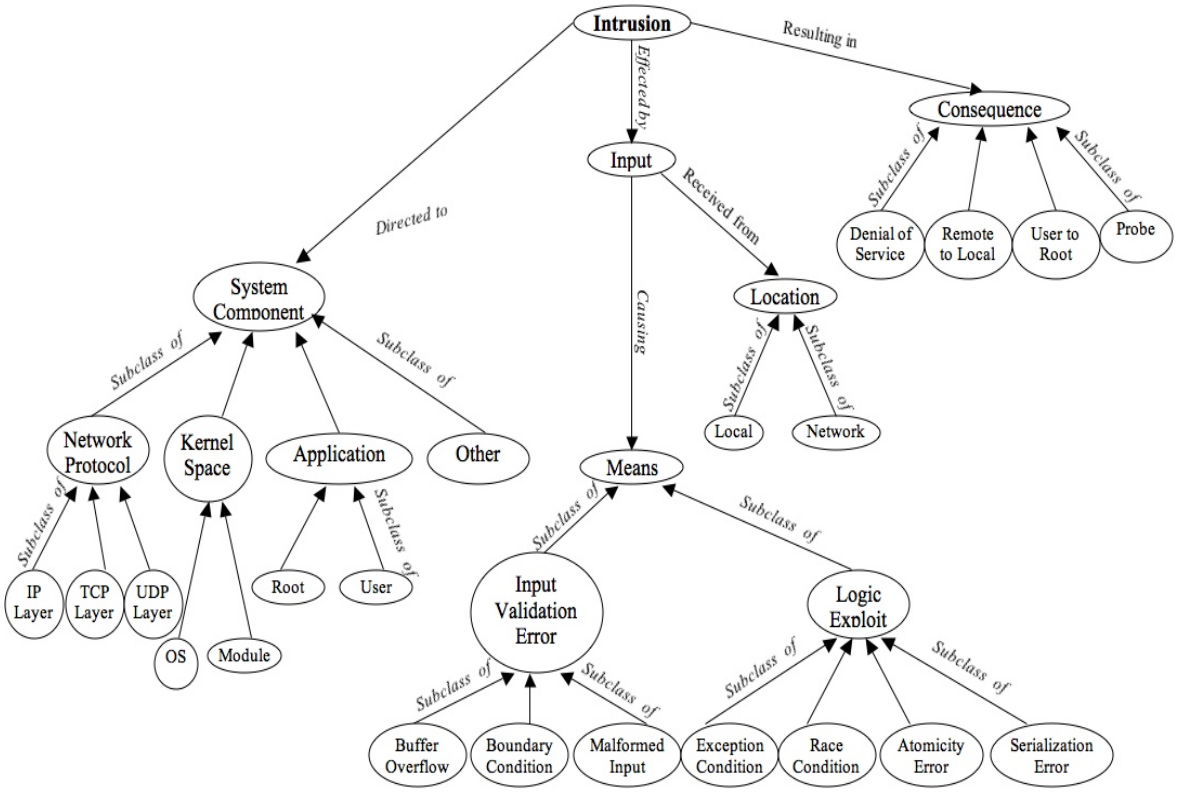


Figure 1.1: Attack Classification with Ontology

Generally, the objective of an intruder is to gain the access to a system or a resource, or to increase the intruder's privileges accessible on a system. In most cases, the target

system has been protected by a username and password combination. Since username is usually public, with the knowledge of a legitimate user's password, an intruder can log in to a system and execute all the privileges which are authorized to that legitimate user. Therefore, the first step of the intruder is to guess the password. Alternatively, the goal of an intruder might be to disable a system or a service. In this case, the intruder may employ various *DoS* attacks to make the target system inefficient, and thus the legitimate users are unable to be served. The instances of various *DoS* attacks can be found in [46].

In order to increase the probability of successful intrusion and conceal the traces, distributed attacks, which involve multiple decentralized host domains, are becoming a greater concern. Frincke [17] divided distributed attacks into the following three categories.

1. *Simple attacks*: Attacks are constituted by a series of actions (automated or interactive) initiated from a single host. This category actually includes those attacks aimed at a single host.

2. *Repeated pattern attacks*: Attacks are formed from a sequence of simple attacks, each independently taking place on a separate host or network and potentially issued from one or more host(s). Some network attacks, such as distributed port scanning, usually fall into this category.

3. *Multipoint attacks*: Attacks are combined from a set simple attacks. Compared with *Repeated pattern attacks*, individual actions may take place on different hosts and be launched from one or more distinct hosts. This type of attacks is more insidious compared with the other two categories. The attack is still widespread, but different activities target at different hosts, rather than a repetition of the same activity at victim hosts.

The following three examples are distributed attacks which can be matched with the aforementioned three types, i.e. *Simple attacks*, *Repeated pattern attacks* and *Multipoint attacks*.

1. *Doorknob-Rattling Attack* [71]: For this attack, the intruder tries a very few username and password combinations on several hosts that results in very few failed login attempts (e.g. guessing password) on each host. This type of attack is difficult to be detected unless the data related to failed login attempts are collected and correlated from several hosts in the network. This attack belongs

to the second category, i.e. *Repeated pattern attacks*. Figure 1.2 depicts the paradigm of this attack.

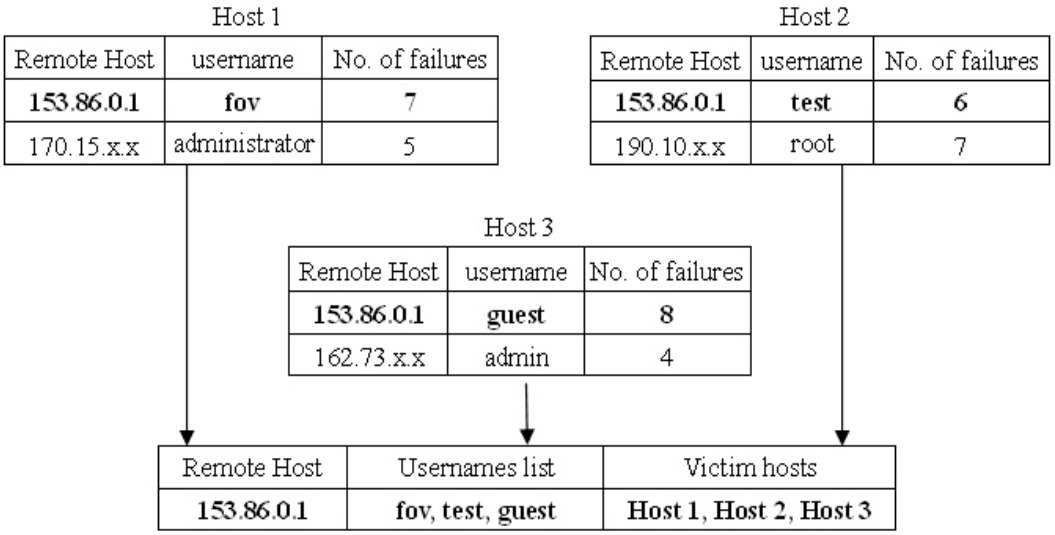


Figure 1.2: A Paradigm of Doorknob-Rattling Attack

In Figure 1.2, there are attempts from a remote host, whose IP address is 153.86.0.1,to login the three hosts, namely *Host 1*, *Host 2* and *Host 3*. According to the view of any one host, there are just several failed login attempts, and some hosts may not even consider this as a suspicious activity since the threshold with regard to the number of failed login attempts is different for each host. However, if the relevant data about failed login attempts could be correlated from several hosts, the subtle *Doorknob-Rattling Attack* would be discovered.

2. *Chain/Loop Attack* [34]: In *Chain attack*, the attacker moves across several hosts in order to hide his/her original trace, which is the host this attacker first successfully logged on. The result of this attack is in a chain of connection through many hosts. In *Loop attack*, the chain of connection forms a loop which makes it more difficult to trace the origin of connection. This attack is an instance of *Simple attacks*. Figure 1.3 demonstrates the paradigm of this attack.

   In Figure 1.3, it can be seen that at time $t1$, a remote host with IP address 153.86.0.1 successfully connected to *Host 1*. Then, at time $t2$, a connection was created between *Host 1* and *Host 2*. Thereafter, a link was initiated from *Host 2* to *Host 3* at time $t3$. Finally, at time $t4$, the user of *Host 4* logged on *Host 1* successfully. It is similar with *Doorknob-Rattling Attack* that any one host in the network cannot find out the *Chain/Loop Attack* only from its local record. Several hosts have to be cooperated to reveal this attack.

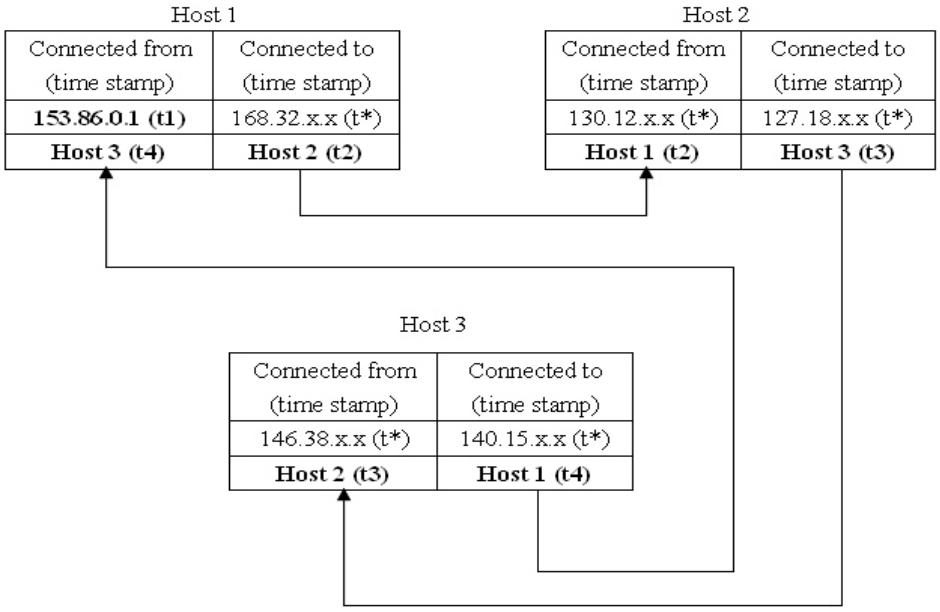


Figure 1.3: A Paradigm of Chain/Loop Attack

3. *Mitnic Attack* [50]: For this attack, an intruder first launches one type of *DoS* attacks (such as SYN flood attack [64]) to prevent a trusted host (e.g. *HostA*) from accepting incoming TCP connection requests (i.e. SYN packets). Then, the intruder tries to connect to another host (say *HostB* which trusts *HostA*) by spoofing *HostA*'s IP address and TCP port, which have been flooded, as source IP and source port. *HostB* is the intruder's real target. This attack can be classified into the third category, namely *Multipoint attacks*. Figure 1.4 shows the paradigm of this attack.

   Figure 1.4 displays the steps which will be followed by the attacker to initiate the *Mitnick Attack*. In this attack, *Host A* can detect a type of *DoS* attacks, while *Host B* may disclose a TCP sequence number prediction attack. Thus, both *Host A* and *Host B* cannot discover the *Mitnic Attack*, if they only check their own local records. This distributed attack is more insidious than *Doorknob-Rattling Attack* and *Chain/Loop Attack*, as it utilizes two different types of attacks.

This thesis uses the above three attacks as instances to test the proposed framework, since they are representatives of common distributed attacks and easy to simulate.

1.1.2 Intrusion Detection

In this subsection, the background about intrusion detection will be introduced. According to [72], intrusion detection can be classified as follows:

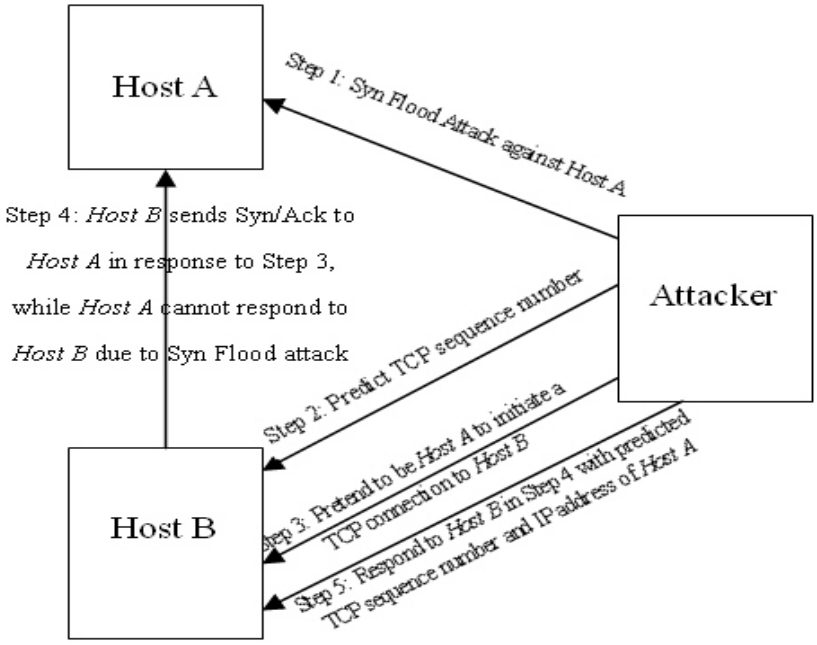


Figure 1.4: A Paradigm of Mitnick Attack

1. *Host-based Intrusion Detection* monitors any suspicious activities and events which occur on a single host. By utilizing host-based intrusion detection, both external and internal intrusions can be detected, while internal intrusions are usually difficult to detect with network-based intrusion detection. There are generally two approaches within host-based intrusion detection.

   - *Anomaly Detection* is based on detecting the deviation of the profile of legitimate users' behavior. The profile is formed according to data collection and correlation over a period of time, while detection methods against the profile are usually statistical ones.
   - *Signature Detection* relies on a set of predefined rules, which are employed to observe events occurring in the system, and then result in a decision that whether an activity is suspicious or not.

2. *Distributed Host-based Intrusion Detection*: Traditionally, host-based intrusion detection focused on a single host. However, many distributed attacks happened in recent years, whose targets may include several hosts, such as *Doorknob-Rattling Attack*, *Chain/Loop Attack* and *Mitnick Attack*. In order to detect these attacks, data collected and correlated from multiple hosts are needed. Therefore, distributed host-based intrusion detection is necessary, which is employed to cooperate and coordinate individual suspicious activities happening on each single

host in the network.

3. *Network-based Intrusion Detection* monitors and analyzes network traffic to identify suspicious activities. Network-based intrusion detection may involve several sensors distributed in different network segments. The sensors observe network transport and render their observation to one or more manager(s) (generally human interface) to analyze and lead to a decision regarding whether or not there is an intrusion in the network.

The above three categories of intrusion detection are sometimes overlapping and complementary. Thus, they are often collaborated together to perform intrusion detection. A distributed host-based IDS makes use of several host-based IDSs that can cooperate and communicate with each other. A network-based IDS focuses on network events and network devices. Distributed host-based IDSs and network-based IDSs can then be combined to monitor different parts of a network system and coordinate intrusion detection and response. The framework proposed in this thesis assembles a distributed host-based IDS and a network-based IDS.

1.2 Agent-Based Intrusion Detection Systems

According to [10], a hierarchical IDS generally consists of three logical components, i.e. sensors, analyzers and managers. Sensors are located at the bottom level of the hierarchy and output data to analyzers, which in turn report up to a manager, located at the top most level of the hierarchy.

- *Sensors*: Sensors (sometimes called as monitors or detectors) are responsible for collecting data and preprocessing the data in a common format for further analysis. Different IDSs usually have different common formats. The input for a sensor may be from any part of a system, which includes network packets, log files, and system call traces. Sensors collect and forward this information up to the analyzer.

- *Analyzers*: Analyzers receive input from one or more sensors or from other analyzers. The analyzer is responsible for determining if an intrusion has occurred. The output of this component is an indication that an intrusion has occurred or not or the confidence of an intrusion happening. The output may include evidences which support the conclusion that an intrusion occurred. The analyzer might provide guidance about what actions to take against the intrusion.

- *Managers*: Managers (sometimes called as directors or consoles) of an IDS manage the whole IDS, including controlling and configuring the behavior of the system and/or reporting the output to security officers.

Thus, according to the above description, the standard architecture of an IDS can be graphically represented in Figure 1.5.

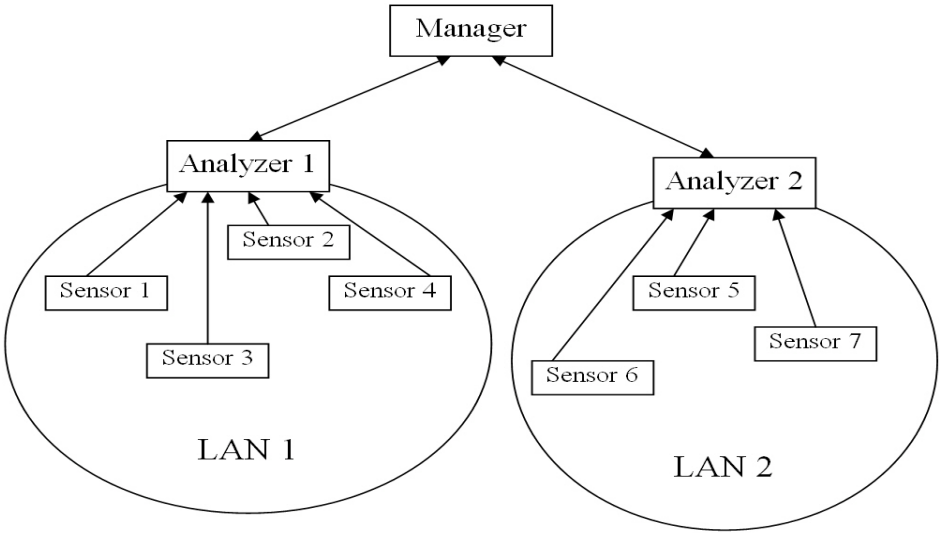


Figure 1.5: A Standard Architecture of IDS

With the development of artificial intelligence, many studies have been undertaken on agent-based intrusion detection systems. Russell and Norvig [61] defined that an agent is anything that can be viewed as perceiving its environment through sensors and acting upon that environment through actuators. Particularly, a software agent can receive keystrokes, file contents, and network packets as sensory inputs and then acts autonomously on the environment by displaying on the screen, writing files, and sending network packets. Software agents are widely used in intrusion detection, as they have the following benefits [4].

1. Agents can be added to and removed from a system without affecting other components in the system, since they are independently running, i.e. autonomous, entities. Therefore, when new agents join in an IDS or existing agents leave an IDS, there is no need for the IDS to restart.

2. Agents can reconfigure themselves without having to restart by using some mechanisms.

3. Before including agents into a complex environment, they can test on their own.

4. Many different agents can cooperate to form a group to fulfill a complex task. Each agent in the group performs simple functions and exchanges information with each other to obtain more complex results, which are difficult to derive by any one of these agents.

There have been several agent-based systems for intrusion detection, which are worthy to look through. In [6], a distributed intrusion detection framework based on autonomous and mobile agents was presented. It makes use of an administrator agent to create analyzer agents and send them toward the stations to be analyzed. There is a crisis agent to create a new administrator agent if the administrator falls in breakdown. In a large network, the administrator needs to create many analyzer agents and send them out. This is a heavy burden for the administrator, and the administrator will become the bottleneck of the whole system. Hence, the scalability might be an issue. DIDMA [31] is a distributed intrusion detection system using mobile agents. It is aimed at building a distributed IDS which places static agents at every host and the network along with a centralized Mobile Agent Dispatcher and IDS console. Although it has better scalability and is platform independent, it still faces a security problem, namely that if a hacker cracks mobile agent dispatcher or IDS console, the whole system will fail. The Multi-agent based Intrusion Detection Architecture [84] is a hierarchical architecture too. It consists of four types of agents, including basic agent, coordination agent, global coordination agent, and interface agent. If a basic agent encounters a complex task it is unable to handle, a coordination agent is created dynamically. The coordination agent communicates with other basic agents and directs them to perform certain functions cooperatively. When the coordination agent encounters a complex task it is unable to handle, it will give a report to the global coordination agent. In this system, the global coordination agent might be a weakness for intrusions, because if the global coordination agent was cracked, no other agents could replace it and the system would break down.

The above three proposed architectures of agent-based IDSs, i.e. [6], [31] and [84], have centralized architectures, which have the risk of single point failure and poor scalability. In order to overcome these two drawbacks, P2P architectures of agent-based IDS are presented. In [55], a P2P intrusion detection system based on mobile agents is introduced. This IDS gives up traditional hierarchical architecture. Hosts in a LAN monitor each other. They periodically send mobile agents to their neighbors to detect intrusions. When anomalous behaviors are detected, the observer neighbor initiates a voting process. It sends a mobile agent with a voting sheet to

other neighbors of the compromised host for a cooperative decision. This system can completely avoid single point failure problem. However, each host in the network has to store critical information of its neighbors which is a burden for each host. In addition, periodic detection may result in network overload increasing and have negative impact on the system overall performance. The basic principles of the system proposed in [79] are similar to those of the system described in [55]. According to [79], host in the network dispatches a mobile agent to its neighbors only when a suspicious incident is observed in that host instead of periodically sending mobile agents to its neighbors. Although this system overcomes some drawbacks of the one proposed in [55], it still has a few limitations. Because the host in the network only asks its neighbors for collaborative decision, it might not detect some specific distributed attacks, which may simultaneously attack multiple hosts in a network, such as *Doorknob-Rattling Attack* and *Mitnic Attack*.

As introduced in the previous paragraphs, most current agent-based IDSs have some limitations and drawbacks, specifically on the aspects of single point failure and detection accuracy. The framework, introduced in this thesis, adopts a P2P architecture, which can avoid single point failure. Furthermore, Intrusion detections in this framework are not only relied on direct-linked neighbors of a host, but also other hosts in the network if necessary. In this way, the original host can obtain further information to achieve a more accurate decision.

1.3 Research Concerns

The research concerns regarding agent-based P2P intrusion detection framework include:

1. A general architecture which outlines the components of the framework and functions of each component. For an agent-based architecture, the components are various agents which take on different tasks. Many architectures have been proposed. Most of them are based on centralized control. Although few ones are based on P2P style, they have other limitations as aforementioned. The architecture presented in this thesis is also based on P2P style, but alleviates the drawbacks which exist in current P2P based architectures.

2. Detailed design of each component (agent) in the framework. Even though many research work about application of intelligent agents in intrusion detection has

been done during the last decade, very few provide detailed design of each agent in proposed systems. In this thesis, the design and implementation of Belief-Desire-Intention (BDI) agents in our framework are introduced. A BDI agent [56] is able to continuously reason about beliefs, goals, and intentions and act accordingly. Detailed description of BDI agents will be provided in Chapter 3.

3. Knowledge representation of each agent in the framework. In a distributed domain, each individual IDS has to implement and interpret the rules with the same detection language in order to cooperate with each other. However, since IDSs can be developed by different vendors and different vendors may exploit different detection languages, the interoperability among various IDSs may become an obstacle. This shortcoming might be mitigated by using ontology technique, as ontology provides software systems with the ability to share a common understanding of information and enables software systems to have greater capability to reason the information. Many research efforts on application of ontology in network security have been done. Nevertheless, they mostly stop at initial proposal or focus on framework design without detailed representation of intrusion or attack and relevant detection knowledge with ontology. In this thesis, the design of ontology based knowledge representation for each agent in our framework is proposed.

4. Task allocation can be employed in our framework for detecting distributed attacks. When detecting some distributed attacks, e.g. *Doorknob-Rattling*, *Chain-loop Attack* and *Mitnic Attack* (mentioned in Subsection 1.1.1), the information from only one host is not sufficient. The host, which initiates a distributed detection, has to allocate detection tasks to other hosts. Thus, an efficient task allocation protocol is necessary. Task allocation in distributed environments has been studied thoroughly, but very few published works considered P2P environments. In addition, many task allocation protocols proposed in distributed environments assumed that there is a central manager which takes charge of task allocation. In this thesis, a novel task allocation protocol which is suitable for intrusion detection in P2P environments is presented.

5. Effective detection strategies are worthy to study in order to handle the current attacks and the variants of existing attacks.

6. Agent negotiation and coordination models could be used for task allocation in P2P environments for intrusion detection.

7. The communication security among different hosts in a P2P environment should also be considered, which involves message authentication and data integrity check.

1.4 Thesis Structure and Outcomes

This thesis starts with a presentation and review of developments in intrusion detection, and describes three specific distributed intrusions which are utilized to test our proposed framework. The difficulty of detecting distributed attacks in P2P environments is that the victim of such an attack is not only one host and any host does not have a global view of the P2P environments. The research concerns of this area are considered to be eight-fold (recall Section 1.3). The first five are tackled in this thesis as a part of the full approach to intrusion detection in P2P environments.

The major contributions of this thesis involve:

1. An agent-based P2P framework for distributed intrusion detection is proposed, which overcomes the disadvantages existing in the current related works;

2. Each agent in the framework is designed and implemented, which has not been undertaken by most related research;

3. Knowledge of each agent is represented by exploiting ontology which can ease communication and information exchange among different hosts;

4. An efficient task allocation protocol is presented for assigning detection tasks among various hosts in P2P environments.

The rest chapters of this thesis are organized as follows:

Chapter 2 reviews current related works, which include architectures and design of agent-based intrusion detection systems, intrusion detection representation mechanisms, task allocation protocols, and resource retrieval in P2P environments.

Chapter 3 introduces an agent-based P2P framework and its detailed design for distributed intrusion detection. This framework does not have a central controller which can avoid single point failure, and allows hosts to request help from not only direct linked neighbors but other hosts if needed.

Chapter 4 gives an ontology-based knowledge representation for each agent in the proposed framework. By making use of ontology, the capability of communication and information exchange among agents in our framework is enhanced.

Chapter 5 presents an efficient task allocation protocol, which is based on Contract Net Protocol [69] but more suitable for task allocation in P2P environments.

Chapter 6 demonstrates the test of our framework which is implemented by using the tool JACK^{TM} (Java Agent Compiler and Kernel) [23], and discusses the test results.

Chapter 7 concludes this thesis, with providing advantages and limitations of this study, and presents the future work.

Chapter 2

Related Research and Literature Review

The agent-based framework proposed in this thesis includes four parts, i.e. general architecture, detailed design of each agent, knowledge representation for each agent, and task allocation protocol for detecting distributed intrusions. In this chapter, we first review the related research about the architectures of agent-based IDSs and agent design in Section 2.1, and then introduce the literature regarding knowledge representation for intrusion detection in Section 2.2. In Section 2.3, we discuss the relevant task allocation protocols in distributed environments and resource search mechanisms in P2P environments. Finally, this chapter is summarized in Section 2.4.

2.1 Architecture and Design of Agent-Based Intrusion Detection Systems

There are numerous agent-based architectures proposed for intrusion detection research. This section depicts several typical architectures.

The AAFID project [4] proposed a distributed IDS architecture including three components that are various agents, transceivers and monitors. Agents are responsible for detecting suspicious events on a protected host and each agent is for a specific type of attack. Agents report their findings to the appropriate transceiver. Transceivers are per-host entities that oversee the operation of all the agents running in their host. Transceivers also report their results to one or more monitors which surveil the operation of several transceivers. In that case, monitors have the ability to access network-wide data and are able to perform high-level correlation to detect intrusions which involve several hosts. The shortcoming of this architecture is that a transceiver has to report to more than one monitors to provide redundant information to resist the failure of one of the monitors.

Qin *et al.* [54] deployed a number of light-weight agents, called ID (Intrusion Detection) agents, to various network components. They proposed multiple ID agents and each of them specializes in a certain category of intrusions. For example, the host-based ID agents can analyze audit data, system call traces, or user shell command streams to monitor applications and user behaviors. On the other hand network ID agents are responsible for network level attacks, such as *DoS*. The proposed architecture is hierarchical and divides the protection and analysis scope to Local-analysis, regional-analysis and Global-analysis. The ID agents are deployed locally to detect intrusive behaviors on network components. An ID Correlator manages some local ID agents and combines the security alarms sent by local ID agents. An ID Correlator is responsible for a region and reports its findings to the ID Manager, which is responsible for the whole network. Similar as most other research on application of agents in intrusion detection and response, the focus of Qin *et al.* is on the distribution of detection and response functions across a domain or several domains rather than the intelligent behavior of agents. Furthermore, the architecture proposed in [54] is hierarchical, which has the potential risk regarding single point failure.

Another agent-based IDS architecture, which was presented in [26], is also a hierarchical one that consists of several lightweight agents. These agents involve static data cleaning agents for obtaining and rendering information from system logs and audit data, low-level agents which monitor and classify ongoing activities, and high-level agents, namely data mining agents, that use machine learning methods to acquire predictive rules for intrusion detection.

The CIDS (Cougaar-based intrusion detection system) [8] provided a hierarchical security agent framework. In CIDS, a security node consists of four different agents and a number of such nodes form a security community. The advantage of CIDS is that an individual agent is responsible for each functional module which makes future modification easy. Compared to other related research, the most significant contribution of CIDS is that it is not only an initial design but also has been implemented although it only concentrated on single node intrusion detection.

A multi-agent based dynamic hierarchical distributed intrusion detection system was presented in [81]. There are four types of agents in this system. They are tracer agent, basic agent, supervisor agent and monitor agent. The supervisor agent is dynamically voted from basic agents in a Local Area Network (LAN) and takes charge of the LAN. Monitor agent is dynamically voted from supervisor agents in the whole system and monitors the running states of the system and interacts with administrator.

In some extent, it avoids the single point failure problem. However, in real hierarchical networks, some specific hosts cannot be substituted, such as servers and routers. If these devices were cracked, no one could replace them and the system would be crashed. Besides, their research stopped at initial architecture without detailed design.

Boudaoud and McCathieNevile [5] presented an intelligent agent-based model for security management, which is composed of three plans: the *user plan*, the *intelligence plan* and the *kernel plan*. The *user plan* represents the security policy-based model, which involves the administrator and the security policies. The administrator specifies security policies to apply to the network. The *intelligence plan* conveys the intelligent agent-based model that includes a multi-agent system and a BDI-based information model. The *kernel plan* indicates the event-based model, which consists of network to secure and the security events occurring in it. The authors attempted to employ BDI agent model in their research, but, like most related works, lacked detailed design and implementation. Furthermore, the functional architecture of the model is still centralized and hierarchical.

In [65], Shajari and Ghorbani explained the design of the Fuzzy Adaptive Survivability Tool (FAST) agents and their intelligent behaviors. A FAST agent uses BDI logic as the reasoning framework to decide on desirable response plans. This is the only research work which referred to the design and implementation of BDI agents in intrusion detection. However, they only considered intrusion detection and response on a single host without attempting to enable the FAST agents to cooperatively monitor and mitigate the attack in a distributed domain.

According to the survey of current related research, most of them only presented an architecture. Although some published works mentioned detailed design, they concentrated more on a single host rather than in a distributed environment, which is attempted to be dealt with in this thesis.

2.2 Intrusion Detection Language

In order to recognize intrusive behaviors effectively, a well defined representation scheme of intrusion or attack signatures is inevitable. There have been several attack languages proposed in the literature to handle this problem. These languages can be categorized as Event, Response, Reporting, Correlation, Exploit and Detection Languages [76], each of which is described as follows.

- **Event Languages**, such as [19] and [78], are used to describe "events". These events may be system logs or network traffic records, which are usually the basic input for security analysis. This class of languages is primarily concentrated on the specification of data format and structure.

- **Response Languages** are employed to specify the actions, which have to be taken in response to the observation of intrusions or attacks. To the best of our knowledge, there is not a well defined and accepted response language. Instead, most IDSs use library functions written in general purpose languages, e.g. C or Java, to represent response actions.

- **Reporting Languages**. One of the possible responses to an attack is reporting the attack event to a human security officer or an application. Reporting languages, such as [7], [29] and [15], are utilized to describe warnings which contain information about an attack, such as source of the attack, target of the attack, type of the attack if known, time of the attack, etc. A reporting language could also be used as an event language at a higher level, for example as input to correlators in a distributed IDS.

- **Correlation Languages**. In a distributed IDS, a correlator or manager may receive many alerts from different IDSs. Then, the correlator or manager correlates these alerts and attempts to recognize some complex attacks. Hence, correlation languages are used to specify the relationships among different attacks or suspicious events in an effort to identify distributed attacks. An example is [40].

- **Exploit Languages** are harnessed to depict the steps, which have to be followed to perform an intrusion or attack. Usually, exploit languages are executable and general purpose languages, e.g. C, C++ and Perl. There are also languages that are explicitly developed to support the scripting of attacks, such as [48] and [45].

- **Detection Languages** are devised to represent intrusion detection. These languages provide mechanisms and abstractions for identifying the manifestation of an attack, such as [13], [52] and [59].

A detailed explanation of these attack languages is provided as follows.

LogWeaver [19] is a log auditing tool, which provides a well-defined syntax and grammer for users to write signatures (rules). LogWeaver takes a system log as input and processes the system log according to a signature (rule) file, which defines the type

of events that are to be monitored and reported on. In addition, LogWeaver is capable of performing regular expressions and making correlations among events. The logic, which is utilized by LogWeaver, depends on *model checking* [60].

The Internet Engineering Task Force [7] attempted to establish a standard and widely accepted computer intrusions data model, namely Intrusion Detection Message Exchange Format (IDMEF) Data Model and eXtensible Markup Language (XML) Document Type Definition. Concretely, the Internet Engineering Task Force defines a data model in [7], which is representation of data exported by an IDS, and it also specifies the data formats and exchange procedures for inter or intra IDS(s) information exchanges. Considering the catholicity, the data model is defined and implemented in XML [78], which is a simple and very flexible format designed to ease the exchange of various data on the web.

The Common Intrusion Detection Framework (CIDF) [29] aimed at developing protocols and application programming interfaces to make information and resources able to be shared among IDS research projects. Furthermore, CIDF also enables developed IDS components to be reused by multiple systems. The components in CIDF exchange data in a standard format, which is based on a reporting language, i.e. the Common Intrusion Specification Language (CISL) [15].

Production-Based Expert System Toolset (P-BEST) [40] is a correlation language, which supports users to write inference rules for reasoning and acting based on facts asserted into its fact base or derived from external events. However, according to [12], the language, provided in P-BEST, lacks the ability to recognize events.

CASL (Custom Attack Simulation Language) [48] is a development tool. It is designed to test real network security holes by directly manipulating networks. In other words, CASL is intended to simulate attacks against hosts in order to see if those hosts are vulnerable to those attacks.

ADeLe [45] is an attack description language designed to model a database of known attack scenarios. This database is then used to configure the probes and detection engines of a given IDS or to test the detection capabilities of a given IDS (by means of attack replay). ADeLe relies on a largely accepted intrusion detection framework, i.e. IDMEF [10]. Compared with CASL [48], ADeLe considered three parts of attack languages, namely response, exploit and detection, not only concentrating on exploit part, like CASL.

STATL [13] is an attacker-oriented intrusion detection language which is based on extensible state transition. STATL enables users to describe computer intrusions as

a set of actions which an attacker has to follow to perform an attack. This language presents mechanisms to represent an attack as a combination of states and transitions. Nonetheless, it lacks constructs for assembling sub-events into larger events.

Bro [52] is a real-time and network based IDS. The language used in Bro is a detection language, which is called "Bro Language". The target of the "Bro Language" is to describe security policies, which specify the reactions when a specific event happens. According to Paxson [52], the language is environment sensitive.

SNORT [59] is a highly configurable host-based and network-based IDS. The functions provided by SNORT involve real-time packet capture, protocol analysis, and content searching and matching. SNORT utilizes its own detection language to define rules, which have two parts, namely header and options. The header contains information about the rule action, source/destination IP addresses and so on. The options include information regarding the type of attack, the message to be sent when an event is generated, etc.

In addition to formal attack languages, there are also attack signature specification mechanisms, such as [39] and [49].

Lin *et al.* [39] proposed abstraction-based misuse detection, including misuse signature specification and adaptable detection strategies. Due to the use of high-level concepts, a misuse signature can represent misuse in a simple form and with high expressiveness. However, the authors only considered host-based attacks without exploring network-based attacks which are very pervasive and serious in computer systems nowadays.

Ning et al. [49] extended the work in [39] to distributed environments. The authors exploited event relationships to represent attack signatures and derived system views from signatures to provide a more concise view of what has happened or is happening in the systems. This approach allows signatures to accommodate unknown variants of known attacks. Nevertheless, the extensibility and knowledge sharing are limitations of this approach.

From the above description, the majority of the attack languages specifically address one aspect of intrusion detection, and they are particular to specific domains, environments and systems. Consequently, they are not extensible and communicable among heterogeneous systems, and their semantics are often vague and lack grounding in any formal logic. Ontology which can avoid these disadvantages is utilized to represent agent knowledge in this thesis. Undercoffer et al. [75] compared ontology with an emerging standard, i.e. IDMEF data model encoded in XML, and provided

the following three conclusions.

- Representation: ontology is able to model the attributes and characteristics of a domain.
- Information sharing: ontology can represent the existence of an instance of the domain in a way that is understandable by any other entities which possess the specific ontology.
- Reasoning: ontology has the capability of aggregating particular instances of the domain in a knowledge base and concluding that some larger, or more comprehensive, instances of the ontology exist.

Some related works about the application of ontology in intrusion detection are introduced in the following paragraphs.

Raskin *et al.* [57] introduced and advocated the use of ontologies for information security. In stating the case for using ontologies, they claimed that an ontology organizes and systematizes all of the phenomena (intrusive behaviors) at any level of detail, consequently reducing a large diversity of items to a smaller list of properties.

Undercoffer *et al.* [75] produced an ontology which specifies a model of computer attack using the DARPA Agent Markup Language together with Ontology Inference Layer. They transitioned from traditional taxonomies and attack languages to ontologies and ontology specification languages. This is the only work we reviewed which referred to formally defining ontology for intrusion detection. However, they only concentrated on building an IDS ontology on a single host without either detailed representation of intrusion or attack with ontology or considering detection in a distributed domain. In addition, their IDS is deprived of some of the benefits which an intelligent agent can offer, such as autonomy and mobility.

Fang *et al.* [14] presented a novel fraud detection method based on ontology and ontology instance similarity. According to the measurement of the similarity of ontology instances, the proposed system can determine whether a user is defrauded. Compared to other detection methods, this method can reduce data model cost. However, they only focused on representing user behaviour with ontology rather than representing intrusion or attack with ontology, which will be described in this thesis.

In [25], the authors proposed a cooperative detection framework relied on the ontology, which unified the network and host features on a single host. Although the detection becomes more flexible and the global locality information to support cooperation can be provided, they only considered information correlation on a single host

to detect intrusion without between hosts to discover suspicious distributed attacks.

According to [68], Simmonds *et al.* listed some common taxonomies of network security attacks, demonstrated the relationship between them, and defined an extensible ontology for network security. However, this paper is just a proposal to initiate the design of ontology for network security attacks without any details about how to represent attacks or intrusions with ontology.

Vorobiev and Han [77] described several web services security threats, such as probing attacks, CDATA field attacks and so on. In addition, they depicted these attacks ontologies. Nonetheless, these ontology representations are comparatively rough and cannot be used in real cases.

From the above depiction, most current related work about ontology for network security lacks representation of intrusion or attack and relevant detection knowledge. This thesis adopts ontology technique to represent intrusion and detection knowledge. In this way, hosts in our proposed framework can share their common understanding of information due to ontology application and detect distributed intrusions taking the advantage of agent communication and cooperation.

2.3 Task Allocation Protocols and Resource Search Mechanisms

In an effort to detect distributed attacks, e.g. *Doorknob-Rattling Attack* [71], in a P2P environment, an efficient task allocation scheme which allocates tasks to other hosts for collaborative detection is essential. In this Section, we first provide some relevant research outputs with respect to task allocation in distributed environments. There is little, if any, published work about task allocation in P2P environments, but many research works regarding resource search in P2P environments have been done. Since resource search is the first step of task allocation, some literature about resource search in P2P environments is worth to exhibit.

2.3.1 Task Allocation in Distributed Environments

Recently, many mechanisms for task allocation have been proposed. Some of them investigate the task allocation problem in a centralized manner. Zheng and Koenig [85] presented reaction functions for task allocation to cooperative nodes. The objective is to find a solution with a small team cost and each task to be assigned to the

exact number of different nodes. This work assumed that there is a central planner to allocate tasks to nodes. Kraus et al. [35] proposed an auction based task allocation protocol which allows nodes to form coalitions with time constrains before allocating tasks. This protocol assumed each node knows the capabilities of all others, and one manager is responsible for allocating tasks to all coalitions. Theocharopoulou et al. [73] presented an approach for allocating temporally interdependent tasks to homogeneous or heterogeneous cooperative nodes in dynamic large-scale networks. Their contribution is combining searching, task allocation and scheduling as a synthesized problem to deal with. However, this method is based on a centralized way.

Centralized fashion can make the allocation process efficient and effective in a small system since the central planner has a global view of the system and it understands which nodes are good at which tasks. In that case, communication overhead during allocation processes could be reduced. However, the centralized fashion also has several notable disadvantages. The first one is that in some environments, it is difficult to have such a central controller, such as P2P environments in which no one node has a global view but only the local prospect about direct linked neighbors. Secondly, when the central planner is out of order or cracked by some attackers, task allocation will suffer a big trouble in this environment. Finally, the scalability in such an environment is limited because when too many nodes exist, the central controller has to maintain much information to hold the global view and respond plenty of request messages from nodes. In this case, the CPU and memory usage of the central controller and network bandwidth consumption drastically raise. To conquer these disadvantages, task allocation in distributed environments has also been investigated. A classic task allocation protocol for distributed environments is Contract Net Protocol (CNP) [69]. CNP was aimed to cope with problem-solving communication and control for nodes in a distributed problem solver. This protocol facilitates distributed control of cooperative task execution (called *task sharing*) with efficient inter-node communication. *Task sharing* is a process which is carried on between nodes with tasks to be executed and nodes that may be able to perform those tasks. The CNP was then evolved in [70] by adding another concept, *result sharing*. *Result sharing* is a form of cooperation in which individual nodes assist each other by sharing partial results. However, CNP was only a preliminary work and many details have to be done.

According to [37] and [66], the authors developed distributed algorithms with low communication complexity for forming coalitions in large-scale networks. Although

their work was pursued in distributed environments, it is still necessary to form coalitions before allocating tasks. Abdallah and Lesser [1] provided a decision theoretic model in order to limit the interactions between agents and mediators. Mediators in that paper mean the nodes which receive the task and have connections to others. Mediators have to decompose the task into subtasks and negotiate with other nodes to obtain commitments to execute these subtasks. However, their work concentrated on modeling the decision process of a single mediator. A scalable and distributed task allocation protocol was presented in [62]. The algorithm adopted in this protocol is based on computation geometry techniques, but the prerequisite of this approach is that agents' and tasks' geographical positions are known.

Weerdt et al. [80] proposed a distributed task allocation protocol in social networks. This protocol only allows neighboring nodes to help with a task which might result in high probability of abandon of tasks when neighbors can not offer sufficient resources.

As described in the first paragraph of this subsection, resource search is the first step of task allocation. Thus, some related works regarding resource search in P2P environments are depicted in the following subsection.

2.3.2 Resource Search in P2P Environments

Currently, the P2P architecture is a popular and effective paradigm for distributed computing and resource sharing. A P2P system [9] is an overlay system between nodes interconnected by an underlying physical network. The users that are located on the nodes in a P2P system are enabled to establish connections with other users (nodes) to acquire and share resources with each other in a decentralized manner. Compared with the traditional client-server architecture, the P2P architecture has the following advantages [33]:

1. *Improved Scalability*: Because of the lack of central servers, the information storage and computational cost are distributed among the peers in the P2P system which avoids the central server bottleneck problem and, thus, makes the P2P system easy to extend.

2. *High Reliability*: The P2P system has a high reliability due to its eliminating centralized coordinators. That means the P2P system can still survive even if part of peers are out of order.

3. *Increased Flexibility*: The P2P system allows individual nodes to join and leave the system frequently to share information directly with each other without the

help of dedicated servers. Thus, this manner gives users unlimited freedom. Each node plays as a server and a client simultaneously in a P2P system.

With the increasing popularity of resource sharing in P2P environments, many efficient resource search approaches have been presented. One of the most famous search algorithms in P2P environments is Gnutella algorithm [18] which is a flooding like scheme. It contacts all accessible nodes within a predefined number of hops. The advantage of Gnutella is its simplicity for implementation, while the disadvantage is the huge overhead during contacting many nodes. In order to improve the original Gnutella algorithm, some researchers proposed modified versions. The algorithm, presented in [30], is one of the variations of Gnutella algorithm. It has peers randomly choose only a part of their neighbors to forward the query to. This method can reduce the average message production in some extent, but still contacts a large number of peers. Another example of revised Gnutella algorithm is proposed in [42], which utilized iterative flooding with increasing user defined number of hops in order to search as small depth as possible. However, in some cases, this approach might produce even larger loads than the original Gnutella algorithm. Another contribution of [42] is its use of *Random Walk*. In *Random Walk*, the requesting node sends out k query messages to an equal number of randomly chosen neighbors. Each of these messages has intermediate nodes forward it to a randomly chosen neighbor at each step.These query messages are called "*Walkers*". A *Walker* terminates either with a success or a failure. The limitation of *Random Walk* is that its success rate totally depends on network topology and random choices. The aforementioned search methods are brute-force schemes, as they just attempt to traverse the network to find the objectives without intelligence. Conversely, some other researchers provided intelligent search approaches for P2P systems.

Kalogeraki *et. al* [30] also provided an intelligent version of modified Gnutella algorithm, in which nodes store query-neighborID tuples for recently answered requests from (or through) their neighbors in order to rank them. When a peer initiates a requesting, it identifies all queries similar to the current one. The peer, then, selects several of its neighbors, which have returned the most results for these queries, to forward query messages to . In [83], the authors proposed a local indices method. Each node indexes the objects stored at all nodes with a certain radius r and can answer queries on behalf of them. The search is performed in the same manner as the one in Gnutella algorithm. Distributed Resource Location Protocol (DRLP) is presented in [44]. Peers have no idea about the location of an object, and forward the query to each of their neighbors with a certain probability. If any object is found, the

query takes the reverse path to the requester, storing the document location at those peers. In the consequent queries, peers with location information can directly contact the specific nodes. The target of these intelligent approaches is improving resource discovery accuracy, but the message production is also increased during the process of building intelligence.

In this thesis, an Efficient Task Allocation Protocol (ETAP) is proposed for task allocation in P2P environments to collaboratively detect distributed attacks. Compared with [85], ETAP does not need a central planner. Against [37], it is not necessary for ETAP to form coalitions among nodes before allocating tasks. Unlike GDAP presented in [80] which allows only neighboring nodes to help with a task, ETAP enables nodes to allocate tasks not only to their neighbors but also other nodes in the system based on a novel resource search mechanism that will be described in Chapter 5. In this way, the nodes could have more opportunities to achieve solution of their tasks.

2.4 Summary

This chapter reviews the state of the art of agent-based IDS's various aspects in regard to our targeted research questions mentioned in Chapter 1. We briefly introduced the architectures of agent-based IDSs and some efforts which attempt to implement the architectures. We then discussed the knowledge representation of intrusion detection, which consists of a review of attack languages and ontology-based knowledge representation. After that, various current task allocation protocols are elaborated, which is accompanied by an overview of resource search schemes in P2P environments. In the next chapter, we present our solution for the first two problems outlined in Section 1.3, i.e. general agent-based intrusion detection architecture and detailed design.

Chapter 3

A Novel P2P Agent-Based Framework for Distributed Intrusion Detections

Several research issues have been outlined in Chapter 1. Aiming at solving these problems, a P2P agent-based framework is proposed in Section 3.1, and the detailed design of each agent in the framework is introduced in Section 3.2. Finally, this chapter is summarized in Section 3.3. For simplicity, the terms *node*, *host* and *peer* are used interchangeably throughout this thesis.

3.1 Framework Architecture

The framework is composed of six types of agents which are Monitor Agent, Analysis Agent, Executive Agent, Manager Agent, Retrieval Agent and Result Agent. The former four agents are static agents that are inquiline on hosts, while the latter two are mobile agents that can travel among hosts if needed. Consideration of the security and flexibility of the system, each host in the framework has to be equipped with the four static agents. This framework is independent of specific network topology. Figure 3.1 demonstrates the general architecture and the interaction between hosts of this framework. The following sessions describe each component of the framework in detail.

3.1.1 Monitor Agent

Monitor Agent is like a host monitor which fixes at a host. The responsibility of Monitor Agent is collecting and preprocessing information of both system audit records and network traffic for further analysis, such as system file operation and network connection.

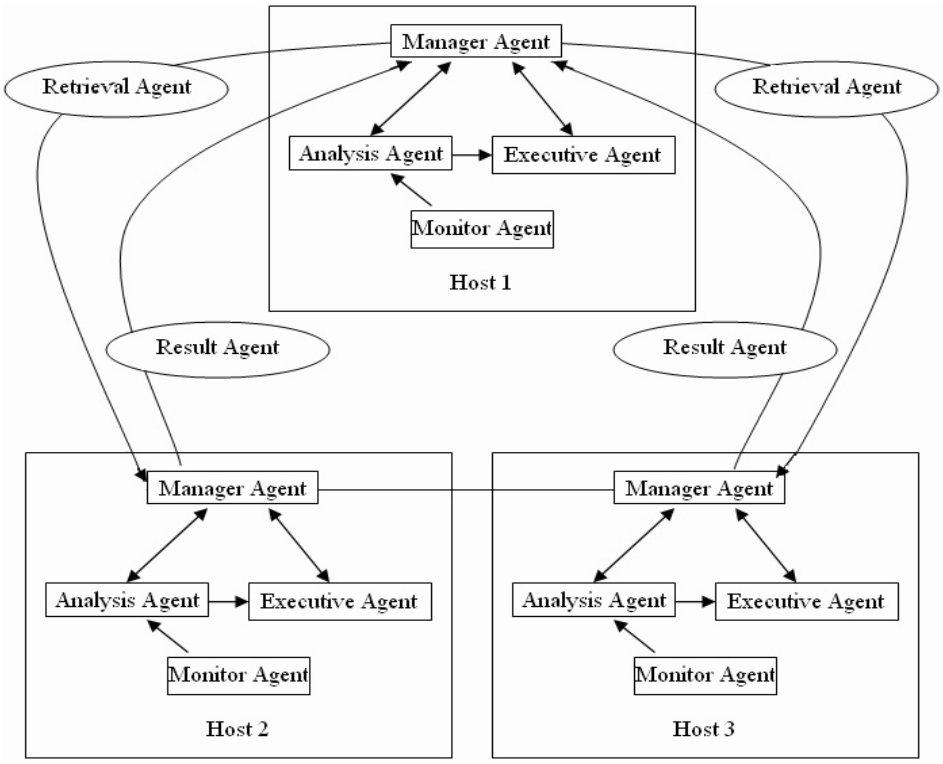


Figure 3.1: Architecture of the Framework

3.1.2 Analysis Agent

Analysis Agent integrates and analyzes the information received from Monitor Agent. In the framework, each host in the network has a local knowledge base which stores some critical information, such as attack signatures, intrusion patterns, system file size, and so on. If Analysis Agent can confirm an intrusion or attack, it will send a notification to Executive Agent to quarantine damaged file or cut off network connection. If Analysis Agent suspects that a distributed attack occurs, it will request Manager Agent for help and store the suspicious activity.

3.1.3 Executive Agent

Executive Agent is responsible for executing tasks depending on the notification of Analysis Agent. These tasks include restoring corrupted files, preventing network connection, etc.

3.1.4 Manager Agent

As introduced in Subsection 3.1.1, the Monitor Agent of a host is the agent that collects information from the host. However, to detect distributed attacks, it is not sufficient to collect information only from a single host. Hence, another three kinds of agents (i.e. Manager Agent, Retrieval Agent and Result Agent) are included in the framework to collect related information from multiple hosts of a network.

The Manager Agent is the agent that manages retrieval processes. It takes charge of Retrieval Agent and Result Agent, including dispatching, retracting and communicating with these two agents. A Manager Agent also maintains a neighborhood list of the host which the Manager Agent resides on. Neighborhood list is a list which contains IP addresses of the direct-linked neighbors of a host. Obviously, the Manager Agent of a host can easily calculate the number of neighbors of the host by checking the length of the neighbor list.

When a host connects/disconnects with another host, the Manager Agent of the host modifies the neighborhood list by adding/removing related information to/from the list. In addition, each Manager Agent has a Retrieval Agent Recorder (*RAR*) which is used to store Retrieval Agent Identifiers. Retrieval Agent Identifier (*RAID*) is used to distinguish different Retrieval Agents. *RAID* is also generated by the Manager Agent. We define *RAID* as the format "HostName0001". "HostName" means the name of the host which dispatches the Retrieval Agent, while "0001" means the serial number of the Retrieval Agent, for example, the first group of Retrieval Agents which perform the same task is "0001", the second group is "0002", and so on. Retrieval Agent Recorder is used to store Retrieval Agent Identifiers in order to avoid a Retrieval Agent traveling the hosts which this Retrieval Agent or other Retrieval Agents with the same *RAID* has/have already visited. Each Manager Agent has a *RAR*.

If a Manager Agent originates the mobile agent for a traveling detection, this Manager Agent is called as an *Initiator*. When an *Initiator* receives a request from an Analysis Agent for deciding a distributed attack, it will dispatch Retrieval Agents to inform other hosts to check whether they have the similar records from the same suspicious remote host. Then, each Manager Agent of the hosts, which have been visited by those Retrieval Agents, will send a Result Agent back to the *Initiator*. The *Initiator* will correlate the information and confirm whether this suspicious activity is a distributed attack. If so, the *Initiator* will broadcast this information to other hosts in the network and notify the local Executive Agent to take actions.

3.1.5 Retrieval Agent

Retrieval Agent moves to other hosts and lets their Analysis Agents check whether there are the similar records from the same suspicious remote host. There are four main types of information that Retrieval Agent needs to maintain, which are source IP address from where the original host dispatches this Retrieval Agent, characters of the incident, Retrieval Agent Identifier, and Time to Live (TTL). TTL, which is generated by an *Initiator*, is used to demonstrate the number of rest hosts the Retrieval Agent needs to visit. The Retrieval Agent will be discarded when the value of TTL reaches zero or there is no more host to be traveled.

3.1.6 Result Agent

Result Agent with a result record will be sent back by each Manager Agent, which has been visited by the Retrieval Agent, to the *Initiator*. Then, the *Initiator* tallies all the result records to make a final decision.

3.1.7 Agent working process

The working process of agents in our framework is described by a sequence diagram in Unified Modeling Language (UML) notations shown in Figure 3.2. Monitor Agents are fixed at hosts of the network and monitor the local activities of hosts. It collects and preprocesses relevant information, and reports the information to Analysis Agent. The Analysis Agent analyzes the information and decides whether there is an intrusion or attack based on the local knowledge base. If so, the Analysis Agent informs the Executive Agent to take actions against the intrusion or attack. However, if the Analysis Agent suspects that a distributed attack is occurring, it asks the upper level Manager Agent for help. When the Manager Agent receives a request from Analysis Agent, this Manager Agent becomes an *Initiator*, and then analyzes the request and dispatches Retrieval Agents to gather information for determining whether some suspicious activities in different hosts could be combined to form a distributed attack. Then those hosts, visited by Retrieval Agent, will send Result Agents with necessary information back to the *Initiator*. The *Initiator* will make a final decision based on the information it received. If there is a definite distributed attack, the *Initiator* will broadcast this detection result to other hosts in the network, and inform the local Executive Agent to take actions.

Six categories of agents and the working process, which consist of the foundation

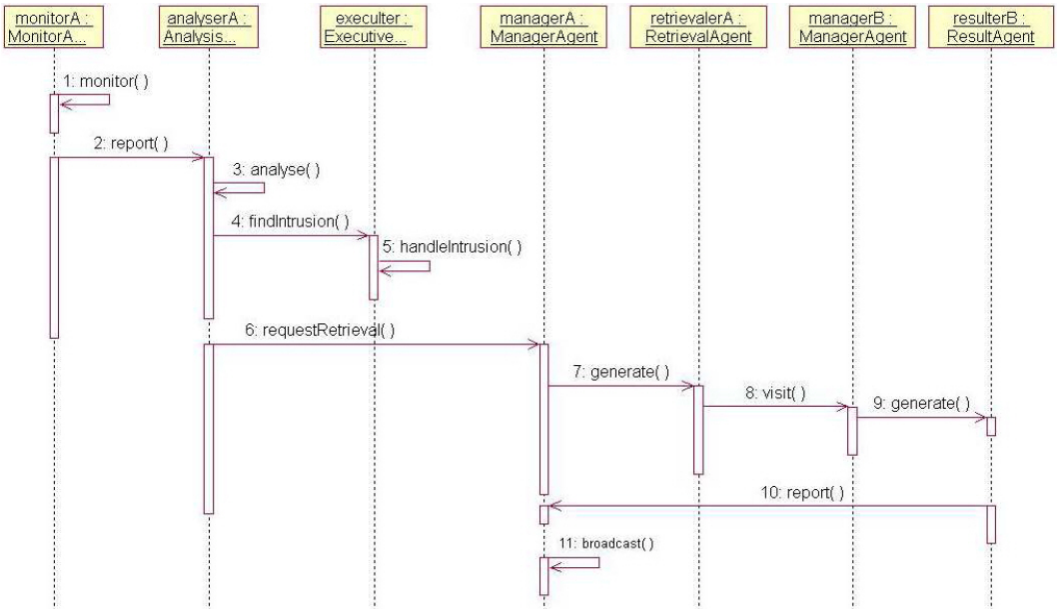


Figure 3.2: Retrieval Process

of the framework, have been described. In the next section, the detailed design of each agent will be introduced.

3.2 Detailed Design of the Framework

This section focuses on the design and implementation of agents in the framework. The brief overviews of BDI agents and JACK^{TM} agent development environment are presented in Subsections 3.2.1 and 3.2.2, respectively. After that, the design of each agent in the framework will be described in detail.

3.2.1 Overview of BDI agents

A BDI agent [56] is able to continuously reason about beliefs, goals, and intentions and act accordingly. There are four major concepts in the BDI architecture:

- *Beliefs* of an agent are information about the environment in which the agent stays. Beliefs include previous and current environment states and inference rules, which allow forward chaining to lead to new beliefs.

- *Desires* are goals assigned to the agent. They represent objectives or situations that the agent would like to accomplish or bring about.

- *Intentions* are commitments by an agent to achieve particular goals. Intentions represent the deliberate states of the agent, namely that what the agent has chosen to do.

- *Plans* are sequences of actions that an agent can perform to achieve one or more of its intentions.

3.2.2 JACK^{TM} Agent Development Environment

In this thesis, JACK^{TM} is used to design and develop agents in our framework. JACK^{TM} [23] is an agent oriented development environment, which is built on and integrated with the Java programming language. It includes all components of the Java development environment as well as offering specific extensions to implement agent behaviors. JACK^{TM} also provides programming constructs for representing and implementing reasoning. There are five main class-level constructs in JACK^{TM}:

- *Agent* is used to define the behavior of an intelligent software agent. The behavior includes capabilities an agent has, what type of messages and events the agent responds to and which plans it will use to achieve its goals.

- *Event* represents an occurrence which the agent must respond to by using a predefined plan. Events involve foreign events which are received from other agents, or interiorly generated events which correspond to some conditions happening.

- *Plan* is used by the agent to achieve its goals and handle its designated event.

- *Capability* allows the functional components that make up an agent to be aggregated and reused. A capability can be made up of plans, events, beliefsets and other capabilities that together serve to give an agent certain abilities. An agent can, in turn, be made up of a number of capabilities, each of which has a specific function attributed to it.

- *BeliefSet* represents agent beliefs according to employ a generic relational model. It has been specifically designed so that a beliefset can be queried using logical members. Logical members are like normal data members, except that they follow the rules of logic programming (as in programming languages like Prolog).

The agents in our framework are implemented by use of the JACK^{TM} development environment. The agents operate according to their beliefs about the attack signatures,

intrusion patterns and the current status of the network, and use their predefined plans to deal with the events which happen in the network. Events include attack incidents and inter-agents communication.

An extension of UML for designing JACK^{TM} agents (JACK/UML) has been proposed by Papasimeion and Heinze [51]. In [51], the UML high level stereotypes are used to represent the five main class-level constructs in JACK^{TM}, i.e. *Agent*, *Event*, *Plan*, *Capability* and *BeliefSet*; and the UML uni-directional association level stereotypes are utilized to define relationships between and behaviors of agents, such as *posts*, *handles*, *modifies*, *private*, etc. Table 3.1 lists a set of high level stereotypes for JACK^{TM} class-level constructs, while Table 3.2 defines a sequence of association level stereotypes for agents behaviors in JACK^{TM}. Figure 3.3 gives a simple example. This example illustrates the case that *MonitorAgent* has a private beliefset *SystemAuditRecord* which can post an event *NewAuditRecord*, and the *MonitorAgent* uses a plan *AuditRecordPreprocess* to handle this event.

Stereotype	Description
<<agent>>	Class level stereotype that defines a JACK agent
<<event>>	Class level stereotype that defines a JACK event
<<plan>>	Class level stereotype that defines a JACK plan
<<capability>>	Package stereotype that defines a JACK capability
<<beliefset>>	Class level stereotype that defines a JACK beliefset

Table 3.1: UML High Level Stereotypes for JACK^{TM}

Stereotype	Description
<<posts>>	Indicates a beliefset or a plan posting an intra agent event
<<sends>>	Indicates a plan sending an inter agent event
<<uses>>	Indicates an agent using a plan
<<handles>>	Indicates an event handled by a plan
<<modifies>>	Indicates a beliefset which a plan can modify
<<private data>>	Indicates a private beleifset owned by an agent

Table 3.2: UML Association Level Stereotypes for JACK^{TM}

The design of the six types of agents in our framework is described in the following subsections.

3.2.3 Monitor Agent

The responsibility of Monitor Agent is collecting and preprocessing information of both system audit records and network traffic for further analysis, which has been depicted

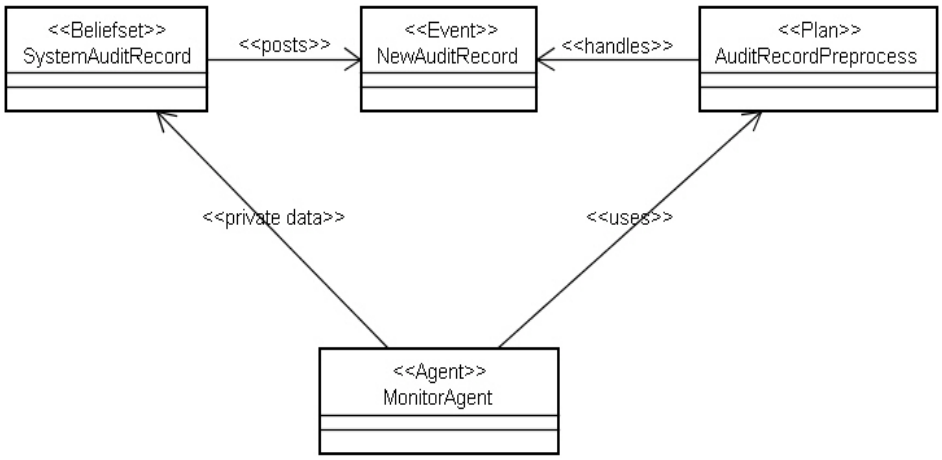


Figure 3.3: A Simple Example of Designing JACK^{TM} Agent with UML

in Subsection 3.1.1. The implementation of Monitor Agent using JACK/UML is described as follows (also demonstrated in Figure 3.4).

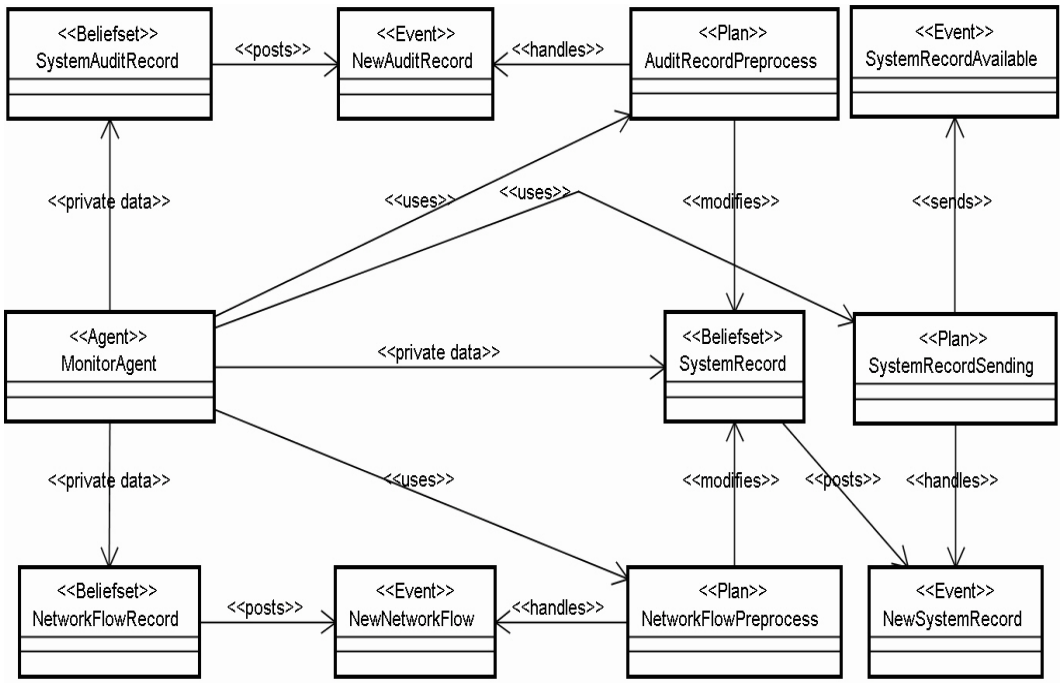


Figure 3.4: Design of Monitor Agent with JACK/UML

The Monitor Agent has the following *Beliefsets*:

1. *SystemAuditRecord*: stores system operation records, such as user login, file access, and so on.

2. *NetworkFlowRecord*: stores network data flow, such as remote client connection, port scanning, and so on.

3. *SystemRecord*: stores both system operation records and network data flow with a predefined format which have been preprocessed by Monitor Agent, and this beliefset will be further analysed by Analysis Agent to find intrusion or attack.

The following *Events* are defined in the Monitor Agent:

1. *NewAuditRecord*: is posted by *SystemAuditRecord* beliefset to remind the Monitor Agent that there is a new system audit record and it is ready for preprocessing.

2. *NewNetworkFlow*: is posted by *NetworkFlowRecord* beliefset to inform the Monitor Agent that a piece of new network data flow is received and ready for preprocessing.

3. *NewSystemRecord*: is posted by *SystemRecord* beliefset to notify the Monitor Agent that a new system record has been preprocessed and is ready for analyzing by Analysis Agent.

4. *SystemRecordAvailable*: is sent to Analysis Agent to inform that a new system record is ready for analysis.

A Monitor Agent uses the following *Plans*:

1. *AuditRecordPreprocess*: preprocesses the new system audit record for further analysis, handling the *NewAuditRecord* event.

2. *NetworkFlowPreprocess*: preprocesses the new network flow for further analysis, handling the *NewNetworkFlow* event.

3. *SystemRecordSending*: retransmits the new system record to Analysis Agent for further analysis, handling *NewSystemRecord* event and sending *SystemRecordAvailable* event.

3.2.4 Analysis Agent

Analysis Agent, as described in Subsection 3.1.2, integrates and analyzes the information received from Monitor Agent, and requests Manager Agent for help if needed. The implementation of the Analysis Agent by using JACK/UML is indicated as follows (also demonstrated in Figure 3.5).

The Analysis Agent has the following *Beliefsets*:

1. *KnowledgeBase*: stores some critical information, such as attack signatures, intrusion patterns, system file size, and so on.

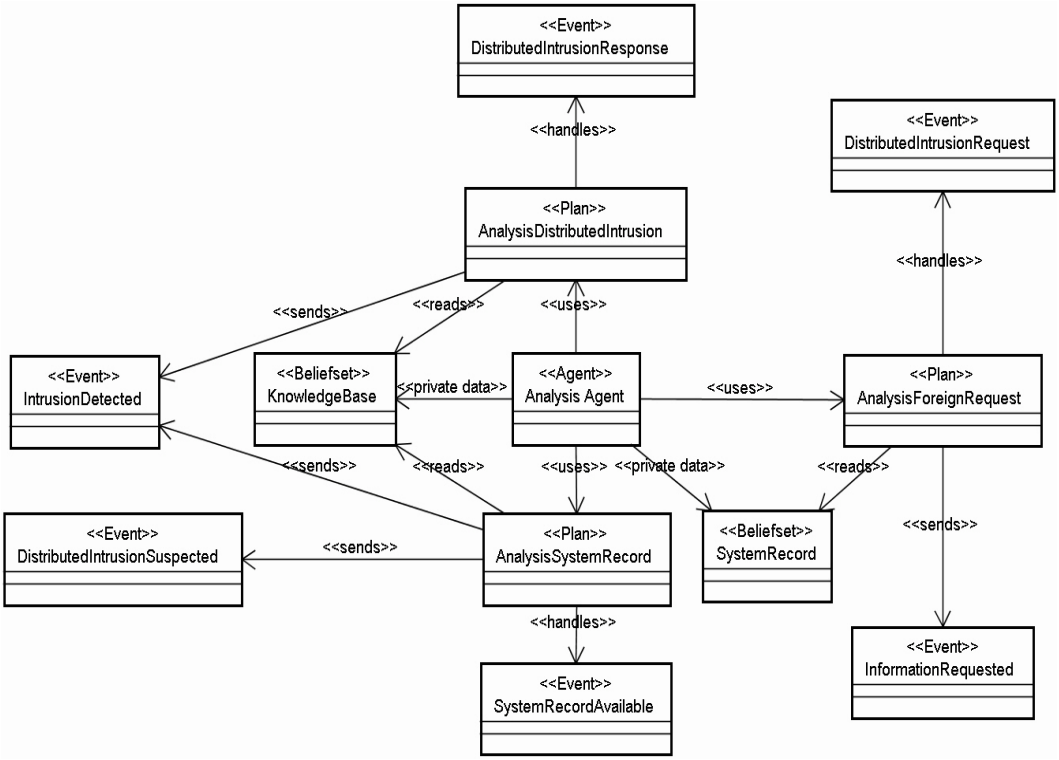


Figure 3.5: Design of Analysis Agent with JACK/UML

2. *SystemRecord*: the same as the above description in Monitor Agent.

The following *Events* are defined in the Analysis Agent:

1. *SystemRecordAvailable*: is sent to Analysis Agent to inform that a new system record is ready for analysis.

2. *IntrusionDetected*: is sent to Executive Agent to take actions against the intrusion.

3. *DistributedIntrusionSuspected*: is sent to Manager Agent to request help for collecting information from other hosts.

4. *DistributedIntrusionResponse*: is sent by Manager Agent with the collected information to Analysis Agent.

5. *DistributedIntrusionRequest*: is sent by Manager Agent to ask for collecting relevant information for other hosts.

6. *InformationRequested*: is sent to Manager Agent with the information the foreign Retrieval Agent requests.

The Analysis Agent uses the following *Plans*:

1. *AnalysisSystemRecord*: analyses the new system record received from Monitor Agent, handling the *SystemRecordAvailable* event, and sending the *IntrusionDetected* event or *DistributedIntrusionSuspected* event.

2. *AnalysisDistributedIntrusion*: analyses the result information received from Manager Agent which is collected from other hosts, handling the *DistributedIntrusionResponse* event and sending the *IntrusionDetected* event.

3. *AnalysisForeignRequest*: analyses the request from foreign Retrieval Agents and provides the relevant information, handling *DistributedIntrusionRequest* event and sending *InformationRequested* event.

3.2.5 Executive Agent

Executive Agent is responsible for executing tasks depending on the notification of Analysis Agent. The implementation of the Executive Agent using JACK/UML is described as follows (also demonstrated in Figure 3.6).

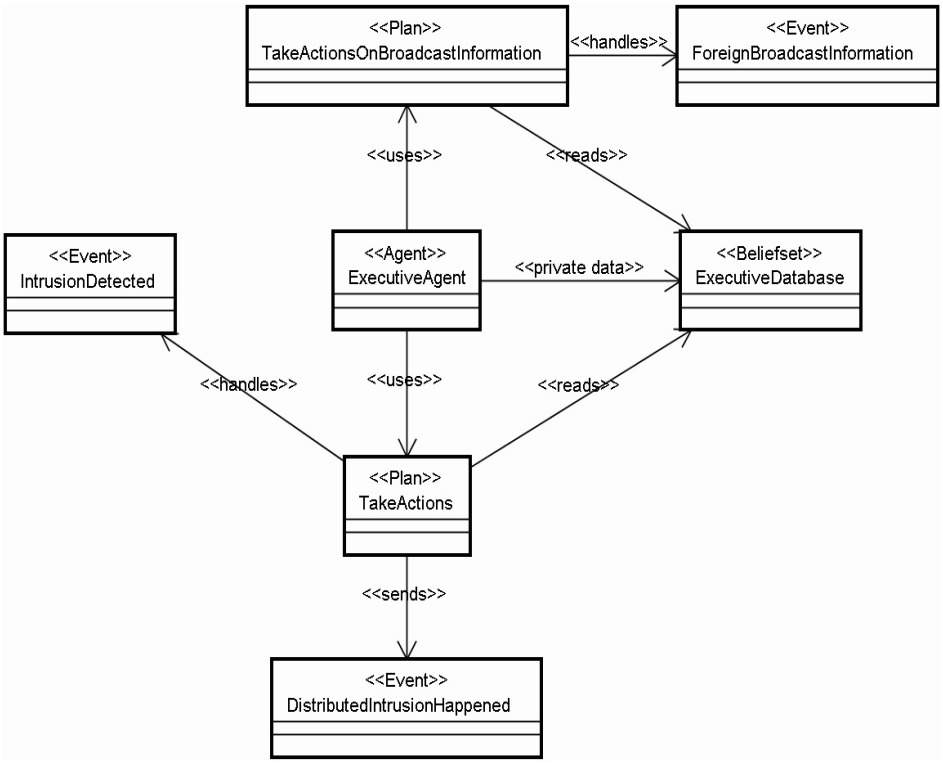


Figure 3.6: Design of Executive Agent with JACK/UML

The Executive Agent has the following *Beliefset*:

1. *ExecutiveDatabase*: stores the actions against each type of intrusion.

The following *Events* are defined in the Executive Agent:

1. *IntrusionDetected*: is sent by Analysis Agent to inform Executive Agent to take actions against the intrusion.

2. *DistributedIntrusionHappened*: is sent to ask Manager Agent to broadcast to the entire network that a distributed attack happened.

3. *ForeignBroadcastInformation*: is sent by Manager Agent to inform Executive Agent to take actions against a distributed attack which is notified by other hosts through broadcast.

The Executive Agent uses the following *Plans*:

1. *TakeActions*: takes specific actions to deal with an intrusion or attack, handling *IntrusionDetected* event.

2. *TakeActionsOnBroadcastInformation*: takes specific actions to deal with a distributed attack which is notified by other host, handling *ForeignBroadcastInformation*.

3.2.6 Manager Agent

The Manager Agent is the agent that manages retrieval processes. It takes charge of Retrieval Agent and Result Agent, including generating, dispatching, retracting and communicating with these two agents. Detailed description regarding Manager Agent can be found in Subsection 3.1.4. The implementation of Manager Agent using JACK/UML is described as follows (also demonstrated in Figure 3.7).

The Manager Agent has the following *Beliefsets*:

1. *NeighbourList*: stores IP addresses of the direct-linked neighbours of this host.

2. *RetrievalAgentRecorder*: stores Retrieval Agent Identifiers (RAID) in order to avoid a Retrieval Agent traveling the hosts which this Retrieval Agent or other Retrieval Agents with the same RAID have already visited.

The following *Events* are defined in the Manager Agent:

1. *DistributedIntrusionSuspected*: is sent by Analysis Agent to request Manager Agent to collect information from other hosts in the network.

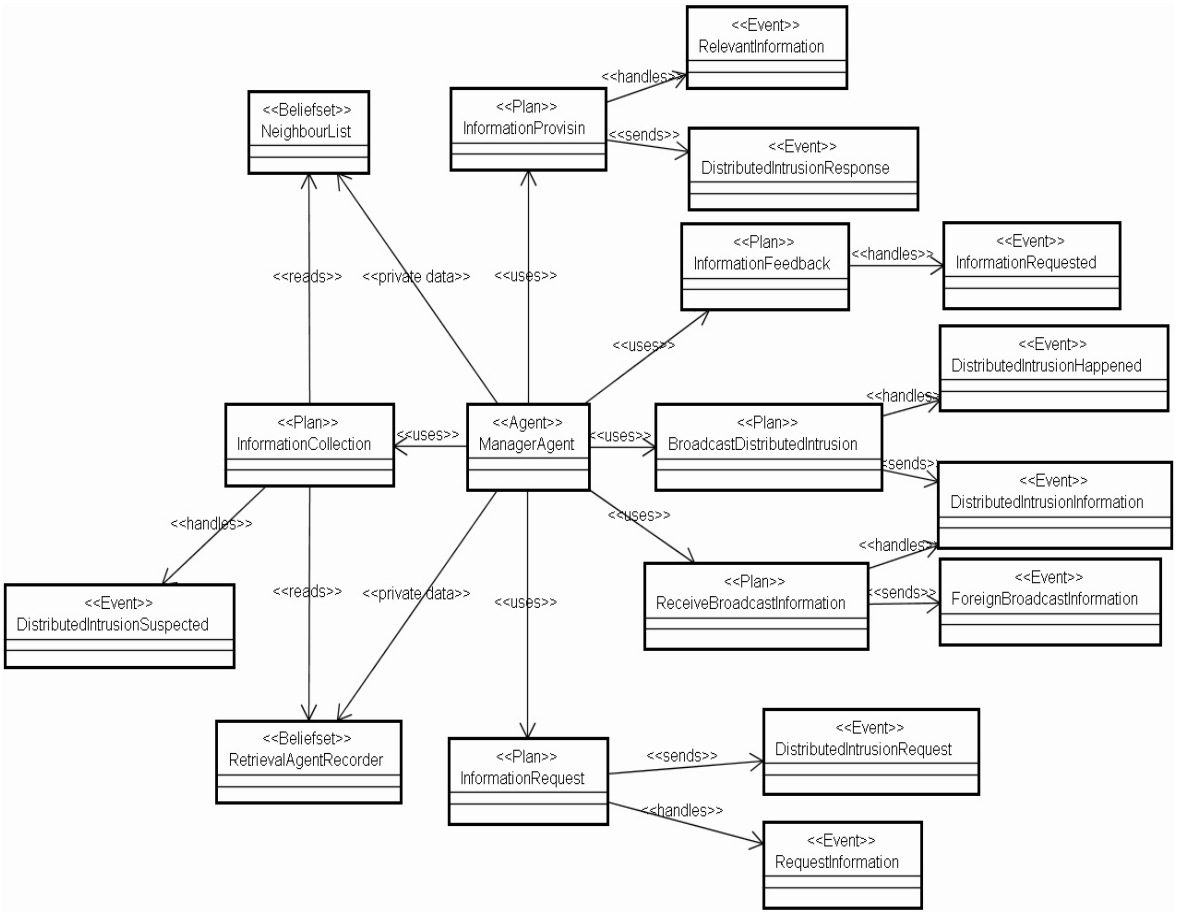


Figure 3.7: Design of Manager Agent with JACK/UML

2. *RequestInformation*: is sent by foreign Retrieval Agent which is from another host to request for help.

3. *InformationRequested*: is sent by Analysis Agent with the information the foreign Retrieval Agent requests.

4. *RelevantInformation*: is sent by foreign Result Agents with the information collected from other hosts.

5. *DistributedIntrusionRequest*: is sent to Analysis Agent to ask for collecting information.

6. *DistributedIntrusionResponse*: is sent to Analysis Agent with the collected information.

7. *DistributedIntrusionInformation*: is broadcasted to other hosts in the network.

8. *DistributedIntrusionHappened*: is sent by Executive Agent to ask Manager Agent to broadcast the distributed attack happened information to the entire network.

9. *ForeignBroadcastInformation*: is sent to ask Executive Agent to take actions against a distributed attack which is notified by other host through broadcast.

The Manager Agent uses the following *Plans*:

1. *InformationCollection*: generates Retrieval Agents and sends them to other hosts to collect information, handling *DistributedIntrusionSuspected* event.

2. *InformationFeedback*: generates Result Agent and sends it to the host which requests for specific information, handling *InformationRequested* event.

3. *InformationRequest*: asks local Analysis Agent to collect information, handling *RequestInformation* event and sending *DistributedIntrusionRequest* event.

4. *InformationProvisin*: provides the collected information to Analysis Agent for further analysis, handling *RelevantInformation* event and sending *DistributedIntrusionResponse* event.

5. *BroadcastDistributedIntrusion*: broadcasts the information that a distributed intrusion has been detected to the entire network, handling *DistributedIntrusionHappened* event and sending *DistributedIntrusionInformation* event.

6. *ReceiveBroadcastInformation*: receives broadcast information from other hosts, handling *DistributedIntrusionInformation* event and sending *ForeignBroadcastInformation* event.

3.2.7 Retrieval Agent

Retrieval Agent moves to other hosts and lets their *Analysis Agent*s check whether there are the similar records from the same suspicious attacker.

The implementation of the Retrieval Agent using JACK/UML is described as follows (also demonstrated in Figure 3.8).

The following *Event* is defined in the Retrieval Agent:

1. *RequestInformation*: is sent to the Manager Agent on the destination host to request it for help.

Figure 3.8: Design of Retrieval Agent with JACK/UML

3.2.8 Result Agent

Result Agent with a result record will be sent back by each Manager Agent, which has been visited by the Retrieval Agent, to the original Manager Agent that initiates the detection process. Then, the Analysis Agent, which receives the results from the original Manager Agent, tallies all the result records to make a final decision. The implementation of the Result Agent using JACK/UML is described as follows (also demonstrated in Figure 3.9).

Figure 3.9: Design of Result Agent with JACK/UML

The following *Event* is defined in the Result Agent:

1. *RelevantInformation*: is sent to the original Manager Agent with the information collected from the host on which the Result Agent resides.

3.3 Summary

In this chapter, a novel agent-based P2P intrusion detection framework is proposed and the detailed design of each agent in the framework is also provided. Compared with current related works, this framework can avoid single point failure and easy to be extended. We will utilize ontology to represent knowledge of each agent in this framework in the following chapter.

Chapter 4

Ontology-Based Knowledge Representation for Distributed Intrusion Detection

In order to empower the interoperability among hosts in our framework, ontology-based knowledge representation of each agent is presented in this chapter. A brief characterization of ontology is introduced in Section 4.1. The tool adopted to implement ontology and the knowledge representation of each agent in our framework are described in detail in Section 4.2. Summarization of this chapter is presented in Section 4.4.

4.1 Overview of Ontology

An ontology [21] defines a set of representational primitives to model a domain of knowledge. The representational primitives include classes (or sets), attributes (or properties), and relationships (or relations among class members). Therefore, ontology is designed for the purpose of enabling knowledge sharing and reuse between entities within a domain. In this thesis, these entities are various agents in our framework.

Resource Description Framework (RDF) [58] is employed to depict the ontology graph in this thesis. RDF is based on the idea that the things being described have properties which have values. The part that identifies the thing, which the statement defines, is called the *subject*. The part that identifies the property or characteristic of the subject, which the statement specifies, is called the *predicate*, and the part that identifies the value of that property is called the *object*. For example, in this sentence "champagne is made in France", "champagne" is *subject*; "made in" is *predicate*; and "France" is *object*. In a RDF graph, an ellipse is utilized to denote a class which may have several attributes. When two classes (ellipses) are connected by a directed edge, the edge dictates a relationship (*predicate*) between the two classes, where the class representing the *subject* is denoted by the start of the edge and the class representing the *object* is denoted by the end of the edge. Obviously, the example, "champagne is

made in France", should be represented as the form in Figure 4.1.

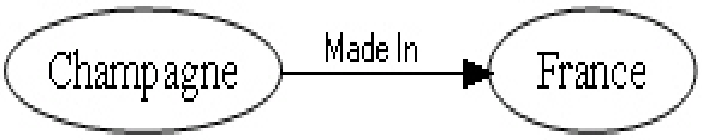


Figure 4.1: RDF relationship graph

Gruber [20] proposed a preliminary set of design criteria for ontologies that should be followed with during a design process.

1. *Clarity*: An ontology should effectively and correctly present the meaning of defined terms. The definition of terms or concepts should be explicit and independent of social or computational contexts. In addition, the definition should be complete in a domain rather than partial, and all definitions can be represented by natural languages.

2. *Coherence*: An ontology should support inferences that are consistent, at least logically consistent, with the definitions. Furthermore, the informally defined concepts should also be consistent, such as description in natural languages.

3. *Extendibility*: An ontology could accommodate variants of defined concepts in some extent. An ontology should offer a conceptual foundation to allow users to define new terms for their special uses based on the existing ontology without any modification.

4. *Minimal encoding bias*: An ontology should be defined on the knowledge level instead of the symbol-level encoding. An encoding bias limits knowledge sharing of the defined ontology, since different users may utilize different encoding mechanisms, which might not decode each other's ontologies. The relationship between an ontology and encoding schemes is similar with that between an algorithm and programming languages. An algorithm should be written by any programming languages, and analogously, an ontology should be represented by any encoding schemes.

5. *Minimal ontological commitment*: An ontology should make as few claims as possible with regard to the modeled domain in order to let users freely instantiate the ontology for the purpose of knowledge sharing.

4.2 Ontology Implementation

Figure 4.2 presents a high level view of our ontology which represents the knowledge of each agent in a peer in our framework. The representation of other peers is similar.

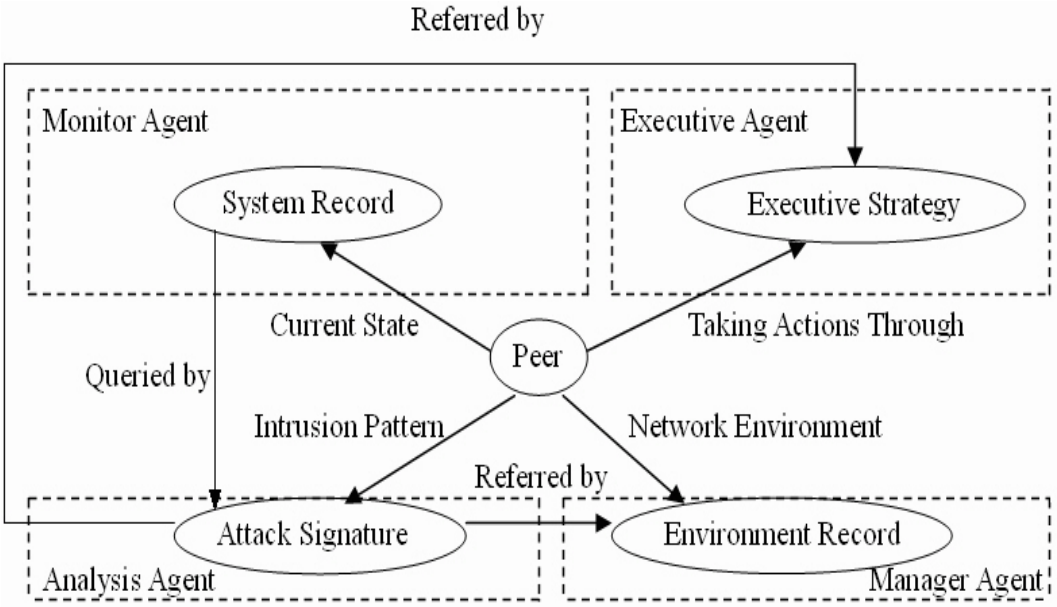


Figure 4.2: Ontology representation of agent knowledge in each peer

The central level of Figure 4.2 is the class *Peer*. *Peer* has the predicates *Current State*, *Intrusion Pattern*, *Network Environment* and *Taking Actions Through*. This construction is predicated upon the notion that the peer (a host in the network) contains attack signatures which are used to detect suspicious activities from the current state of the peer and it may need the information from other peers in the network (network environment) to cooperate detection, and finally the peer will take actions against the intrusion or attack.

Protege [53] is harnessed to implement the knowledge (beliefs) of each agent in our framework. Protege is a free and open source ontology editor and knowledge-base framework, which implements a rich set of knowledge-modeling structures and actions that support the creation, visualization and manipulation of ontologies in various representation formats. Protege ontologies can be exported into a variety of formats including RDF(S), OWL, XML, and so on. After implementation of knowledge representation with Protege, the representation is converted into *N-Triples*. The reason for choosing *N-Triples* is that *N-Triples* is more intuitionistic than XML and OWL. *N-Triples* [47] is a line-based and plain text format for representing the correct answers for parsing RDF/XML. The representation of *N-Triples* is of the following form:

<Subject> <Predicate> <Object>

The following sessions will describe knowledge representation of each agent in detail.

4.2.1 Knowledge of Monitor Agent

Monitor Agent is a host monitor which fixes at a host. The responsibility of Monitor Agent is collecting and preprocessing information of both system audit records and network traffic for further analysis. Figure 4.3 shows the ontology of knowledge of Monitor Agent. The knowledge of Monitor Agent is *System Record* which is about system status of the host. The class *System Record* consists of two components, i.e. *Network Flow Record* and *System Audit Record*. From Figure 4.3, it can be seen that three classes are defined in the ontology which are *System Record*, *Network Flow Record* and *System Audit Record*.

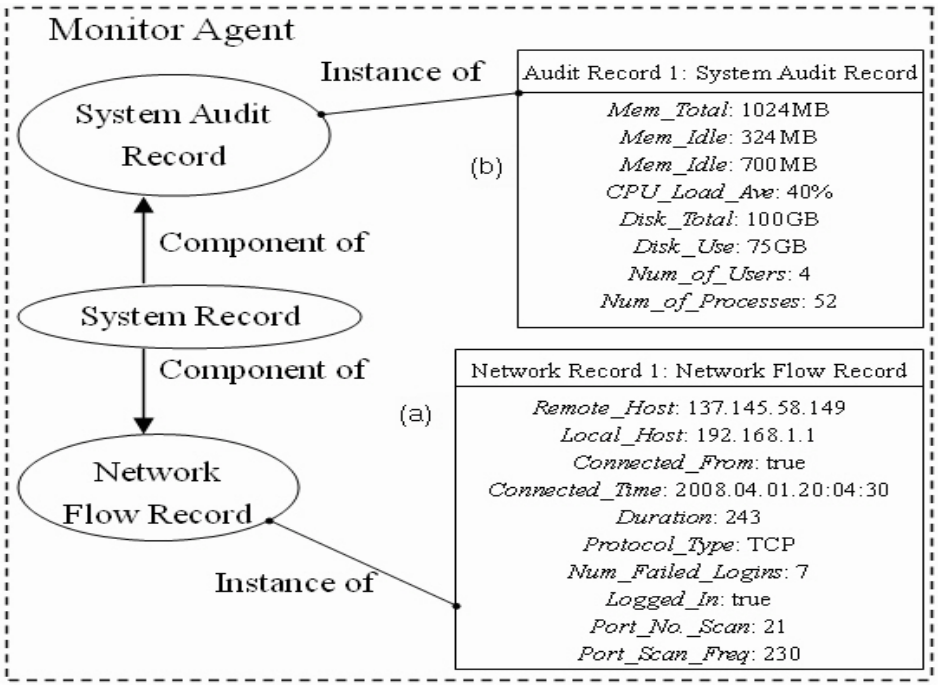


Figure 4.3: Monitor Agent Knowledge

The class *Network Flow Record* includes several attributes about network features which are IP address of remote host (*Remote_Host*), IP address of local host (*Local_Host*), whether the remote host connects to the local host or oppositely (*Connected_From*), the time when the remote host has a successful connection to the local host (*Connected_Time*), length (the number of seconds) of the connection (*Duration*), type of the protocol which the remote host uses to connect to the local host, e.g.

TCP, UDP, etc (*Protocol_Type*), the number of failed login attempts before a successful login (*Num_Failed_Logins*), whether the remote host successfully logged in the local host (*Logged_In*), the port number which has been scanned by the remote host (*Port_No._Scanned*), and the number of scanning to a specific port in the past two seconds (*Port_Scan_Freq*). In Figure 4.3, (*a*) is an example instance of the class *Network Flow Record*.

The class *System Audit Record* is inclusive of attributes representing the operating system state of the host, such as memory usage (*Mem_Total*, *Mem_Idle*, and *Mem_Use*), CPU usage (*CPU_Load_Ave*), disk usage (*Disk_Total* and *Disk_Use*), the number of current users (*Num_of_Users*), and the number of current processes (*Num_of_Processes*). In Figure 4.3, (*b*) is an example instance of the class *System Audit Record*.

The attribute, *Remote_Host*, representation with *N-Triples* (mentioned at the beginning of this section) is shown in Table 4.1 which means *Remote_Host* belongs to the class *Network_Flow_Record* and its type is *string*. The other attributes representation with *N-Triples* are analogous.

<http://www.owl-ontologies.com/unnamed.owl#Remote_Host> <http://www.w3.org/2000/01/rdf-schema#domain> <http://www.owl-ontologies.com/unnamed.owl#Network_Flow_Record>
<http://www.owl-ontologies.com/unnamed.owl#Remote_Host> <http://www.w3.org/2000/01/rdf-schema#range> <http://www.w3.org/2001/XMLSchema#string>

Table 4.1: An Example of N-Triples

4.2.2 Knowledge of Analysis Agent

Analysis Agent integrates and analyzes the information received from Monitor Agent (in Figure 4.2, the predicate *Queried by* representing this relationship). In our framework, on each host, the Analysis Agent contains many attack signatures which are used to discover intrusions or attacks through analyzing information from Monitor Agent. When Analysis Agent detects an intrusion or a attack, it will send a notification to Executive Agent to quarantine damaged file or cut off network connection. If Analysis Agent suspects that a distributed attack occurs, it will request Manager Agent for help.

The knowledge of Analysis Agent is *Attack Signature* which is about various intrusion or attack patterns. The class *Attack Signature* has two components, including

Network Attack and *Host Attack*. Figure 4.4 demonstrates the ontology representation of the three classes. In order to conveniently communicate and cooperate between Monitor Agent and Analysis Agent, the attributes of *Network Attack* and *Host Attack* are nearly the same as those of *Network Flow Record* and *System Audit Record* respectively, except the attributes, *Attack_Type* and *Num_of_Hosts*, which are additional in *Network Attack* and *Host Attack* to specify the type of an attack and how many hosts need to be detected. In this research, four subclasses are added to the class *Network Attack* which are *DoS*, *R2L*, *U2R* and *Probe* (described in Subsection 1.1.1). In Figure 4.4, (*a*) is an example instance of the class *R2L*.

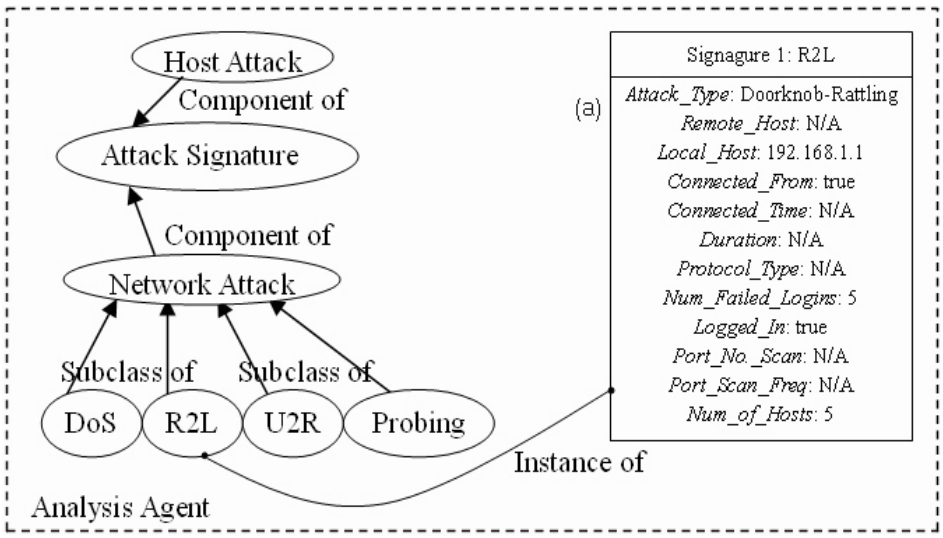


Figure 4.4: Analysis Agent Knowledge

4.2.3 Knowledge of Executive Agent

Executive Agent is responsible for executing tasks against intrusions or attacks which depend on the type of attack notified by Analysis Agent (in Figure 4.2, the predicate *Referred by* representing this relationship). These tasks include restoring corrupted files, preventing network connection, and so on. The knowledge of Executive Agent is *Executive Strategy* which is about how to take actions against intrusions or attacks. The example of knowledge representation with ontology of Executive Agent is demonstrated in Figure 4.5.

The attributes of *Executive Strategy* include type of attack (*Attack_Type*), type of actions to be taken (*Action_Type*), effect to which target (*Target*), and extent of the effect (*Extent*).

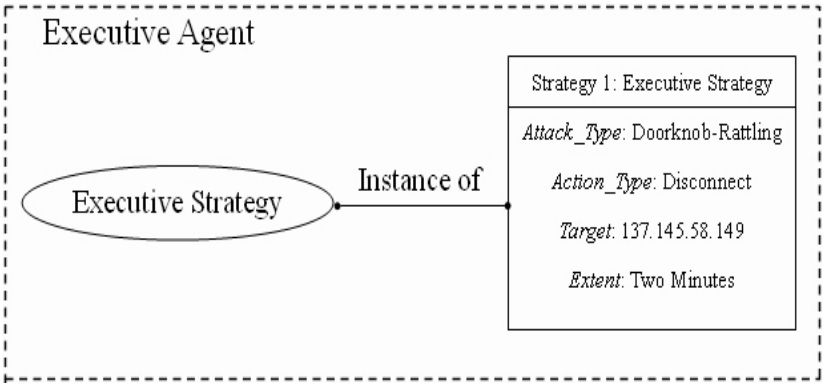


Figure 4.5: Executive Agent Knowledge

4.2.4 Knowledge of Manager Agent

As described in Subsection 3.1.4, the Manager Agent is the agent that manages retrieval processes. It takes charge of Retrieval Agent and Result Agent, including generating, dispatching, retracting and communicating with these two agents. The knowledge of Manager Agent is *Environment Record* which is about environment information of the host. The class *Environment Record* has two components, i.e. *Neighbor List* and *Retrieval Agent Record*. The ontology representation of the three classes is shown in Figure 4.6. The class *Neighbor List* contains relevant information about neighbor hosts (one-hop hosts) of the the host which the Manager Agent resides on. The class *Neighbour List* has attributes including IP addresses of neighbors (*Neig_IP_Addr*), host names of neighbors (*Host_Name*), and MAC addresses of neighbors (*MAC_Addr*). Obviously, the number of neighbors of a host is equal to the number of records the *Neighbor List* contains. The class *Retrieval Agent Record* is used to store Retrieval Agent Identifiers (RAID) in order to avoid Retrieval Agent traveling the hosts which it or other Retrieval Agents with the same RAID have already visited. The class *Retrieval Agent Record* has several attributes, including visited time (*Vis_Time*), from where (*IP_Address*) and the RAID (*RAID*). In Figure 4.6, (*a*) and (*b*) are example instances of the classes *Neighbor List* and *Retrieval Agent Record* respectively.

When an original Manager Agent, which initiates the detection process, receives a request from the Analysis Agent for deciding distributed attack, it will generate and dispatch Retrieval Agents to inform other hosts to check whether they have the similar records which could form a distributed attack (in Figure 4.2, the predicate *Referred by* representing this relationship).

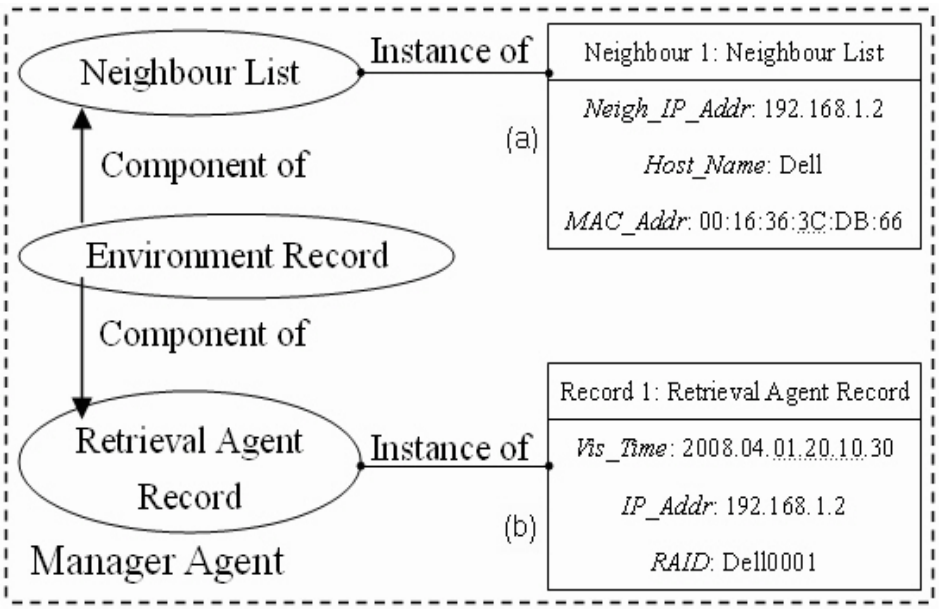


Figure 4.6: Manager Agent Knowledge

4.2.5 Knowledge of Retrieval Agent

Retrieval Agent, generated by Manager Agent, moves to other hosts and lets their Analysis Agents check whether there are the similar records to constitute a distributed attack. There are four types of knowledge that Retrieval Agent needs to maintain, which are source IP address from where the original host dispatches this Retrieval Agent (*Source_IP_Addr*), type of the suspicious attack (*Attack_Type*), remote IP address where the suspicious attack is from (*Remote_IP_Addr*), Retrieval Agent Identifier (*RAID*), and Time to Life (*TTL*). *TTL*, generated by an *Initiator*, is used to demonstrate the number of rest hosts the Retrieval Agent needs to visit. The Retrieval Agent will be discarded when the value of *TTL* reaches zero or there is no more host to be traveled.

4.2.6 Knowledge of Result Agent

Result Agent, also generated by Manager Agent, with a result record will be sent back by each Manager Agent, which has been visited by the Retrieval Agent, to the *Initiator*. The result record contains the information including source IP address from where the original host dispatches this Result Agent (*Source_Host*), type of the suspicious attack (*Attack_Type*), whether the original host has similar records (*Similar_Record*), and if so where these records are from (*Remote_Host*). Then, the local Analysis Agent which resides on the same host with *Initiator* tallies all the result records to make a final decision.

4.3 Example

We have created several instances of the class *R2L* and *Probing* in our ontology with Protege and maintained them as the knowledge of Analysis Agent. These instances are specific intrusions or attacks, such as *Doorknob-Rattling Attack*, *Chain/Loop-Attack* and *Mitnick Attack*. In this section, we only present the detection of *Doorknob-Rattling* attack as an example. The sponsor of *Doorknob-Rattling Attack* (mentioned in Section 1.1) tries a very few common username and password combinations on several computers that results in very few failed attempts on each host. This type of attack is hard to be detected unless the data related to failed login attempts are collected and correlated from all hosts in the network. The *N-Triples* representation of suspicious *Doorknob-Rattling Attack* is illustrated in Table 4.2.

<http://www.owl-ontologies.com/unnamed.owl#network_attack_Instance_1>
<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
<http://www.owl-ontologies.com/unnamed.owl#R2L>

<http://www.owl-ontologies.com/unnamed.owl#network_attack_Instance_1>
<http://www.owl-ontologies.com/unnamed.owl#Attack_Type>
"Doorknob_Rattling"^^<http://www.w3.org/2001/XMLSchema#string>

<http://www.owl-ontologies.com/unnamed.owl#network_attack_Instance_1>
<http://www.owl-ontologies.com/unnamed.owl#Connected_From>
"true"^^<http://www.w3.org/2001/XMLSchema#boolean>

<http://www.owl-ontologies.com/unnamed.owl#network_attack_Instance_1>
<http://www.owl-ontologies.com/unnamed.owl#Num_Failed_Logins>
"5"^^<http://www.w3.org/2001/XMLSchema#int>

<http://www.owl-ontologies.com/unnamed.owl#network_attack_Instance_1>
<http://www.owl-ontologies.com/unnamed.owl#Logged_In>
"true"^^<http://www.w3.org/2001/XMLSchema#boolean>

Table 4.2: N-Triples Notation for Suspicious Doorknob-Rattling Attack

In Table 4.2, it is noted that *Doorknob-Rattling Attack* is an instance of *R2L* and in this attack the victim host is connected from a remote host, the number of failed login attempt is at least 5, and the login is finally successful.

In order to query for the existence of a suspicious *Doorknob-Rattling Attack*, a rule should be defined which tests for the number of failed login attempts before a successful login. The query in Table 4.3 performs this test in JACK^{TM} [23] syntax. If the attributes, *Connected_From* and *Logged_In*, of any instance in the class *System*

Record matches the query, other attributes of that instance are instantiated. Then, the test about whether the number of failed login is more than 5 will be executed on each matched instance. If so, that instance is a suspicious *Doorknob-Rattling Attack* and the relevant information will be sent to Manager Agent to request help. The relevant information includes the type of attack, the IP address of remote host which connects to the local host, and the number of hosts which need to be visited.

```
SystemRecord.query (Connected_From=true, Logged_In=true)
=>
if (Num_Failed_Logins>=5) then
send("ManagerAgent", SuspiciousMessage(Attack_Type,Remote_Host,Num_of_Hosts))
```

Table 4.3: Query for Suspicious Doorknob-Rattling Attack

4.4 Summary

In this chapter, we utilize ontology to represent knowledge of each agent in our framework. The advantage of using ontology is that peers in the framework can easily share knowledge among each other. In the next chapter, an efficient task allocation protocol will be introduced to assign detection tasks to different peers for collaboratively detecting distributed attacks.

Chapter 5

Task Allocation in the P2P Framework

To cope with the issue of allocating tasks in a P2P environment, a decentralized task allocation protocol, i.e. Efficient Task Allocation Protocol (ETAP), is elaborated in this chapter. We first formally describe the task allocation problem in Section 5.1, and then propose the ETAP in Section 5.2. Section 5.3 gives the comparison tests of ETAP against Gnutella algorithm [18] and Greedy Distributed task Allocation Protocol (GDAP) [80]. Finally, this chapter is summarized in Section 5.4.

5.1 Problem Description

The description of task allocation problem is formalized in this subsection. Firstly, the definition of a *P2P system* is given.

- **Definition 5.1**: A *P2P system* is defined as an undirected graph written as $P2P = (P, E)$ where P is the set of peers in the system, namely $P = \{p_1, p_2, ..., p_n\}$ and $E = \{e_{12}, e_{13}, ..., e_{21}, e_{23}, ...\}$ indicates the set of edges which are existing relationships between two peers. For example, the edge $e_{ij} \in E$ means there is a connection between the peers p_i and p_j. Therefore, p_i and p_j are neighbors of each other.

Each peer $p \in P$ is defined as a tuple $<PeerID(p)$, $Neig(p)$, $Resource(p)>$, where $PeerID(p)$ is the identity of the peer, $Neig(p)$ is a list which indicates the neighbors of the peer, and $Resource(p)$ is a dataset which depicts the resource types and the number of resources for each type that the peer contains. Then, the definitions of the two terms *Initiator* and *Participant*, which are used throughout the rest of this thesis, are provided in **Definition** 5.2.

- **Definition 5.2**: Suppose there is a set of tasks $T = \{t_1, t_2, ..., t_n\}$ in a P2P system. The agent which requests help for its tasks is called *Initiator* and the agent which accepts and performs the announced tasks is called *Participant*.

Each task $t \in T$ is defined as a tuple, namely $<TaskID(t)$, $Resource(t)$, $Benefit(t)>$. In this tuple, $TaskID(t)$ is the identity of the task, the form of $Resource(t)$ is similar as $Resource(p)$ but $Resource(t)$ just contains a record which depicts the resource type and the number of this resource that is necessary for completing the task, and $Benefit(t)$ is the benefit gained when the task is completed successfully. The detailed illumination of $Benefit(t)$ relies on different situations and applications. In this thesis, $Benefit(t)$ is just set as a random integer number for simplicity.

Besides $TaskID(t)$, $Resource(t)$ and $Benefit(t)$, the query message for each task contains three more tuples, namely $PeerID(p_I)$ that is the ID of *Initiator*, $PeerID(p_i)$ indicating the ID of agent which forwards this query message last time, and TTL (Time-To-Live) which means the number of hops the query message could be forwarded.

In this thesis, it is assumed that each task $t \in T$ needs only one type of resource to finish and can be assigned to only one agent to accomplish, as task decomposition is not the concentration of this research. Task allocation, therefore, can be defined as follows.

- **Definition 5.3**: Given a finite set of tasks, recorded as $T = \{t_1, t_2, ..., t_n\}$, and a finite set of peers, written by $P = \{p_1, p_2, ..., p_m\}$ in a P2P system, *Task allocation* in this research is defined as attempting to allocate the n tasks to some or all of the m agents.

A successful task allocation should satisfy the situation that the *Participant* agent has the specified resource type which is matched the announced task's resource type, and the number of this resource the *Participant* agent contains should be more than the number of the resource which is needed for completing the announced task.

The *Initiator* prioritizes the tasks based on the efficiency of each task, and allocates tasks with their efficiency descending. In addition, the *Participant* also chooses the most efficient tasks to offer help every time. The definition of the efficiency of a task, $t \in T$, is described as follows.

- **Definition 5.4**: The *efficiency* of a task, $effi(t)$, is in the light of the ratio between the benefit gained from completing the task and the number of the resource that is required for accomplishing the task, i.e.

$$effi(t) = \frac{Benefit(t)}{Resource(t)}. \tag{5.1}$$

As aforementioned about $Benefit(t)$, *efficiency* in this thesis is not a very critical term either, which is only used to prioritize the tasks. Hence, we overlook the particular explanation of *efficiency*.

5.2 Principle of ETAP

In a P2P system, there is no peer that has a global view about the system but only the local prospect regarding its neighbors. This research focuses on how to allocate tasks, which are distributed among the peers in a system, appropriately to *Participant* peers particularly when the neighbors of the *Initiator* peer do not have sufficient resources for tasks. In this thesis, for simplicity, it is supposed that the P2P system architecture is fixed during task allocation process, which means that peers entering and leaving the system dynamically are not considered. Figure 5.1 briefly describes the interaction process between an *Initiator* and a *Participant*. In Figure 5.1, $Reso_{T_{(a)}}$ means the type and number of the resources which Agent a needs to finish its tasks, while $Reso_{neig}(a)$ indicates the the type and number of the resources that Agent a's neighboring agents contain. The idea of ETAP is illustrated as follows.

The *Initiator* peer, $p \in P$, attempts to find its neighboring peers to help with its tasks, $t_i \in T_p$ and $T_p \subseteq T$. Here, T_p means that the set of tasks that the *Initiator* peer has to allocate. The *Initiator* peer first sends resource query messages to its neighbors. These neighbors will respond the message with information about the types of resources they contain and the number of resources for each type, and the identities of them.

The *Initiator* peer then compares the available resources from its neighbors, i.e. $Reso_{neig}(p)$, with the resources required for its tasks, namely $Reso_{T_p}$. They are calculated as $Reso_{neig}(p) = \bigcup_{p_i \in Neig(p)} Resource(p_i)$ and $Reso_{T_p} = \bigcup_{t_i \in T_p} Resource(t_i)$, respectively. This comparison will result in one of the following two cases.

1. **Case One** ($Reso_{neig}(a) \supseteq Reso_{T_a}$): in this situation, *Initiator* directly requests help for tasks from its neighbors, as *Initiator*'s neighbors have enough resources to handle its tasks. *Initiator* begins with assigning the most efficient task(s). If more than one neighbors can solve one task, *Initiator* will allocate this task to the one which has the most number of available relevant resource. The neighbors receive and store the requests, and select the tasks with the highest efficiency to perform. When *Initiator* receives the responses from its neighbors, it finally sends contracts for the allocated tasks to *Participants*.

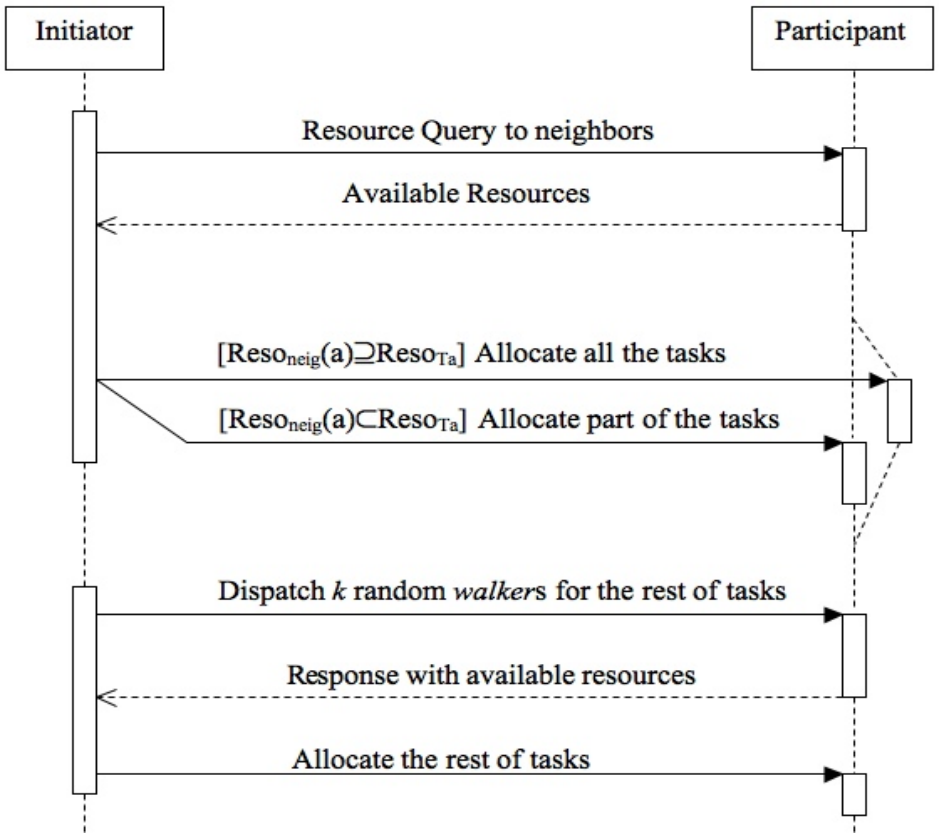


Figure 5.1: Interaction Process Between *Initiator* and a *Participant*

2. **Case Two** ($Reso_{neig}(a) \subset Reso_{T_a}$): in this case, *Initiator* requests help only for those tasks, $t_i \in T_a^{'}$, which can be handled by current available resources from its neighbors. *Initiator* starts with allocating the tasks, $t_i \in T_a^{'}$, also based on the efficiency of these tasks. When finishing assigning tasks, $t_i \in T_a^{'}$, *Initiator* attempts to send query messages in depth (not only to neighbors) for the rest of tasks which cannot be dealt with by using the resources of its neighbors, namely $t_j \in (T_a - T_a^{'})$.

   Initiator generates query messages which are called *Walkers* and dispatches them out. The definition of *Walker* is the same as that of query message introduced in Section 5.1. The number of *Walkers* for each task is set to k that can be adjusted by users, and the TTL of each *Walker* is also set by users. A *Walker* then is forwarded to a neighboring agent at each step. The probability with which a neighboring agent is chosen depends on the number of neighbors that each neighboring agent has, excluding the *Initiator*. The agent which connects with more neighbors has higher probability to be selected. The probability can be calculated according to Equation 5.2.

$$p_{a_i} = \frac{|Neig(a_i)| - 1}{(\sum_{a_i \in Neig(a_I)} |Neig(a_i)|) - |Neig(a_I)|} \tag{5.2}$$

In Equation 5.2, p_{a_i} is the probability with which the agent, a_i, is selected; as the description in Section 5.1, $Neig(a_i)$ is a list which indicates the neighbors of the agent, a_i, and $|Neig(a_i)| - 1$ designates the number of neighbors which agent a_i connects, excluding the *Initiator* agent a_I. For example, agent a_1 is the *Initiator* that has two neighbors, i.e. a_2 and a_3, while a_2 and a_3 connect with 4 and 8 neighboring agents separately, both excluding a_1. When a_1 selects a neighbor to pass a *Walker*, it chooses a_2 and a_3 with the probability $\frac{4}{4+8} = \frac{1}{3}$ and $\frac{8}{4+8} = \frac{2}{3}$ respectively.

The agent which receives a *Walker* will decrease the TTL by 1, before the agent forwards the *Walker* to any of its neighbors. If, after decrementing the *Walker*'s TTL, the TTL is found to be zero, the *Walker* will not be forwarded any more. Otherwise, the agent forwards the *Walker* to one of its neighbors with the same manner as the above description. During this process, if any agent has the relevant resources which are desired by *Walkers*, the agent sends a response message with its available resources back to *Initiator*. *Initiator*, then, selects the agent with the most number of available resource as *Participant*, like in **Case One**, and makes a contract with it.

In order to avoid query message replication, each agent keeps a State Record which contains the information about where the *Walkers* are from and forward to. The State Record is consisted of four attributes, i.e. $TaskID(t)$, $AgentID(a_I)$, $AgentID(a_i)$, and $AgentID(a_j)$. $TaskID(t)$, $AgentID(a_I)$ and $AgentID(a_i)$ have been mentioned in Section 5.1. $TaskID(t)$ and $AgentID(a_I)$ can be used to distinguish different tasks, and $AgentID(a_i)$ is exploited to avoid agent forwarding the *Walker* to the former agent which just sent this *Walker*. $AgentID(a_j)$ is the ID of the agent which is chosen for the *Walker* to be forwarded to. When a new *Walker* arrives, the agent checks the *Walker*'s $TaskID(t)$ and $AgentID(a_I)$ against its state record. If the same record has existed, the agent will forward the *Walker* to one of other neighbors, with the same way depicted above, to which this *Walker* has not been forwarded yet, and creates a new piece of record in its State Record with the four aforementioned attributes. If, in the extreme condition, all of the agent's neighbors have received this *Walker*, this *Walker* will be discarded no matter the value of its TTL.

In the concurrent situation that is one agent has been requested by two or more other agents nearly simultaneously, the agent responds their requests with First-Come-First-Service (FCFS) mechanism, and its available resources announced in each response message will exclude the former announced ones.

From the above description, it can be found that ETAP is based on both Gnutella algorithm [18] and *Random Walk* [42], but extends them. Gnutella algorithm can achieve high resource discovery accuracy, while *Random Walk* can lower message production. In addition, both of them do not need to have any pre-knowledge of the P2P system before searching resources. Thus, ETAP first employs Gnutella algorithm to request help from *Initiator*'s neighbors (setting the number of hops to 1), and then utilizes the refined *Random Walk* to query other agents for the rest of tasks. Instead of randomly choosing a neighbor to forward the *Walker* as in [42], a biased selection approach is employed when choosing next neihboring agent to pass the *Walker*, which relies on the number of neighbors that each neighboring agent has.

5.3 Test of ETAP

To test the performance of ETAP, we compare ETAP with the Gnutella algorithm [18] and the Greedy Distributed Allocation Protocol (GDAP) [80]. Gnutella is a popular and well-established searching algorithm which has been deployed in many real P2P systems. GDAP is utilized for allocating tasks in a distributed environment, but it only allows neighboring agents to help with a task. In this section, we first depict Gnutella and GDAP briefly. Then, the settings of a test environment and three criteria are introduced. Finally, the test results and the relevant analysis are illustrated.

5.3.1 Gnutella Algorithm

As described in Subsection 2.3.2, Gnutella algorithm [18] is exploited for resource search in P2P systems. It attempts to traverse all the peers in a P2P system. Gnutella's flooding like scheme is easy to be implemented, but produces a large number of messages due to contacting many peers. Although Gnutella algorithm has not been used in task allocation, it can be borrowed in our research as a standard for comparison.

5.3.2 Greedy Distributed Allocation Protocol

Greedy Distributed Allocation Protocol (GDAP) [80] is employed to handle task allocation problem in agent social networks. An agent social network is defined in [80] as an undirected graph where vertices are agents and each edge indicates the existence of a social connection between two agents. The task allocation process of GDAP is described briefly as follows. All manager agents try to find neighboring contractors to help them with their tasks. They start with offering the most efficient task. Out of all tasks offered, contractors select the task with the highest efficiency, and send a bid to the related manager. A bid includes all the resources the agent is able to supply for this task. If sufficient resources have been offered, the manager selects the required resources and informs all contractors of its choice. When a task is allocated, or when a manager has received offers from all neighbors but still cannot satisfy its task, the task is removed from its task list.

It can be seen that the main shortcoming of GDAP is that it only relies on neighbors which may cause many unallocated tasks due to the limitation of resources, while our research is trying to figure this problem out.

5.3.3 Test Setting

In order to compare the three protocols, ETAP, Gnutella and GDAP, we set a test environment for assessing them. Power-Law random graph [2], which is the topology of many real life P2P networks [28], is simulated for testing ETAP, Gnutella and GDAP. The feature of power-law topology is that although the average number of neighbors is the same as the normal random graphs, there are few peers with very high connectivity while most other peers with few neighbors.

There are four different setups used in this test.

- *Setup* 1: The number of peers and tasks in the P2P network are 50 and 30 separately. The number of types of different resources is 5 and each peer randomly has several of them. The average number of resources for each type is 30 and the average number of resources required by each task is also 30. The number of *Walkers* for each task, i.e. k, is 1 and the TTL value for each *Walker* is set to 5. In this evaluation, we suppose that each task only needs one type of resources because task decomposition is not considered in this thesis (as described in Section 5.1). The tasks are distributed uniformly on each peer. The exact number of resources of each resource type that a peer has and the number required

by a task are both distributed normally. In addition, the average efficiency of tasks is 10 and the exact efficiency of a task also satisfies normal distribution. The only changeable attribute is the average number of neighbors in this setup. *This setup is designed to show how different average number of neighbors influences the performance of the three protocols.*

- *Setup* 2: This setting is similar to *Setup 1* but with a few modifications. The TTL value for each *Walker* varies from 2 to 8. Furthermore, the average number of neighbors is fixed at 6. *The purpose of this setting is to test the adaptability of the three protocols.*

- *Setup* 3: In this setting, the average number of neighbors is fixed at 6. The number of peers fluctuates from 100 to 400 and the ratio between the number of agents and tasks is confirmed at 5/3. The proportion of the number of peers and resources types is set to 10/1. In order to match the fluctuation of the number of peers, the TTL value for each *Walker* transforms from 5 to 20. *This setup is used for demonstrating the scalability of the three protocols in different scale networks with a fixed average number of neighbors.*

- *Setup* 4: This setting is only for ETAP which is also similar to *Setup 1*. The only difference from *Setup 2* is that instead of adjusting TTL value, we adapt the number of walkers, i.e. k, from 1 to 4. Since Gnutella and GDAP do not contain the term *Walker*, they are neglected in this setting. *The target of this setting is to exhibit the relationship between the number of* Walkers *and the performance of ETAP.*

In this test, three criteria are used to estimate the performance of ETAP, Gnutella and GDAP.

1. *ER (Efficiency Ratio)*: The proportion of summation efficiency of completed tasks to the expected total efficiency of tasks in a P2P system, namely:

$$ER = \frac{\sum_{t_i \in T_c} effi(t_i)}{\sum_{t_j \in T} effi(t_j)} \tag{5.3}$$

where T_c is the set of completed tasks, T is the set of tasks in a P2P system, and $effi(t)$ is efficiency of the task (as described in Section 5.1). Higher ER means that more tasks can be allocated and solved, so the performance is better.

2. *Num*: The entire number of the communication messages transferring in the system during one task allocation process. Lower Num indicates that less communication messages are generated and transferred in the system. Therefore, the burden of a P2P system can be remitted more.

   In a P2P system with n tasks, m peers, k *Walkers* for each task and TTL for each *Walker*, the complexity of Num is $O(n^2(k \cdot TTL + m))$. This can be proved as follows. In each iteration in the worst case (i.e. a fully connected P2P system), for each of the $O(n)$ *Initiators*, $O(m)$ resource query messages are sent. Then, the $O(m)$ available resource response messages are generated and sent back to the *Initiators*. Next, each of the $O(n)$ *Initiators* allocates $O(n)$ tasks to their $O(m)$ neighbors. Thus, the messages created at this stage are $O(n)$. After that, each of the $O(n)$ *Initiators* creates $O(kn)$ *Walkers* for the rest of tasks and send them out in $O(TTL)$ hops. Hence, the messages generated during this phase are $O(kn \cdot TTL)$. Then, the $O(m)$ peers which have available resources will respond the $O(kn)$ *Walkers* and the messages are $O(nm)$. Finally, each of the $O(n)$ *Initiators* makes $O(n)$ contacts. There are $O(n)$ *Initiators* in the system. Therefore, during one task allocation process, the number of communication messages are $O(n(m+m+n+kn \cdot TTL+nm+n)) = O(n^2(k \cdot TTL + m))$. It should be noticed that several special conditions, such as message replication and concurrent request (mentioned in Section 5.2), do not affect the final analysis result.

3. *CoP (Coefficient of Performance)*: The ratio between the summation efficiency of successfully finished tasks and the number of communication messages (Num) during the task allocation process. CoP can be calculated by using Equation 5.4.

$$CoP = \frac{\sum_{t_i \in T_c} effi(t_i)}{Num} \tag{5.4}$$

   Generally, the performance of the protocol is more desirable if higher ER could be achieved and lower number of communication messages are created. Hence, observing either ER or Num only is not enough to determine the quality of the protocol. We, therefore, employ the CoP as the third metric to evaluate the three protocols. Higher CoP indicates better performance, since higher CoP means each message can derive higher efficiency.

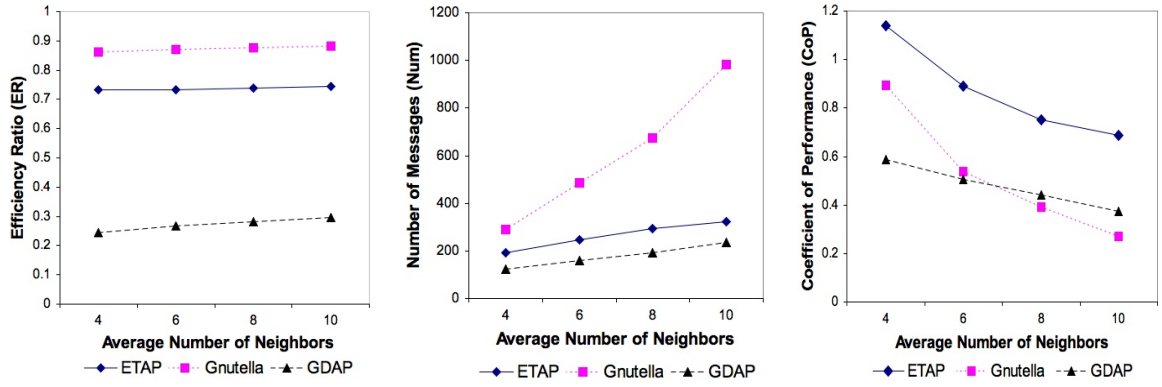
For convenience, we suppose that once a task has been allocated to a *Participant*, the *Participant* would definitely finish this task without failure.

5.3.4 Test Results

The test is performed on the four aforementioned setups for ETAP, Gnutella and GDAP. In order to achieve precise results, each assessment step was executed 30 times and the average data were obtained.

Test Results from Setup 1

This test is done on *Setup 1* as described in Subsection 5.3.3. The purpose of this test is to estimate the influence of different average number of neighbors on all the three protocols.



(a) Efficiency Ratio (ER)

(b) Number of Messages (Num)

(c) Coefficient of Performance (CoP)

Figure 5.2: Performance of different protocols on distinct average number of neighbors

Figure 5.2(a) demonstrates that *Efficiency Ratio*s (ER) of ETAP and Gnutella in different conditions are much higher and more stable than that of GDAP. This is because task allocation with GDAP only depends on neighbors of *Initiator*. Therefore, the more neighbors work on tasks, the more opportunities the tasks could be solved. Comparatively, ETAP and Gnutella rely on not only neighbors but also other peers if needed. This feature results in steady performance of ETAP and Gnutella. It is also shown that with more average number of neighbors, the performance of GDAP is improved continuously. The reason of this situation is that when there are more neighbors, *Initiator* has higher probability to derive sufficient resources for dealing with more of its tasks.

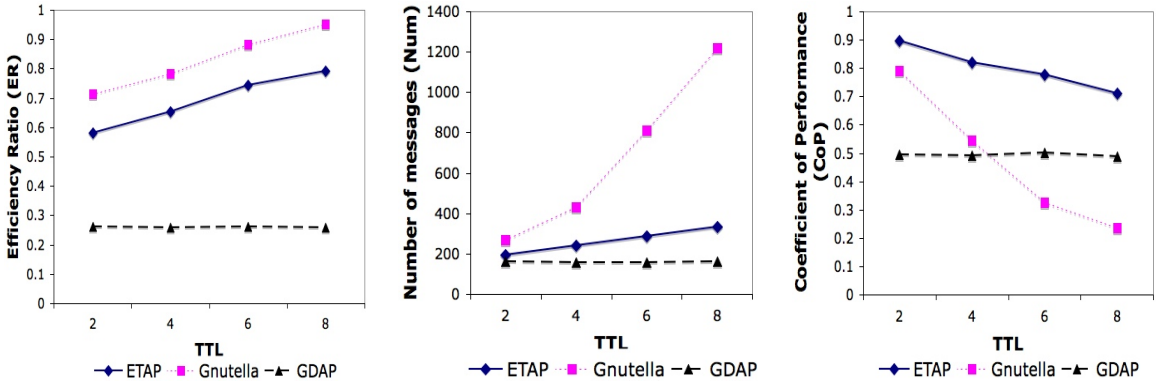
Figure 5.2(b) depicts the number of messages (Num) of the three protocols in different situations generated in allocation processes. As ETAP and Gnutella would request other peers for help when resources from neighbors are insufficient, the number

of messages of both ETAP and Gnutella is higher than that of GDAP. Thus the presentation of GDAP in this test is relatively good due to its consideration of neighbors only which could decrease the number of messages created during task allocation process. It is also found that the number of messages of Gnutella is much higher than that of ETAP and GDAP. This is because Gnutella is a flooding scheme which requests all the neighbors during each hop. In that condition, the number of messages rises dramatically. Compared with Gnutella, the number of messages of ETAP is a little higher than that of GDAP, since ETAP requests only one neighbor for the unallocated tasks during each hop (remember $k = 1$ in *Setup 1*).

Figure 5.2(c) shows the *Coefficient of Performance* (CoP) of the three protocols in different cases. With the average number of neighbors increasing, the CoP of all the three protocols declines, but the CoP of ETAP is higher than that of the other two protocols. It should also be found that when the average number of neighbors is more than 6 the CoP of Gnutella is lower than that of GDAP. This is because for Gnutella more neighbors bring much more communication messages. Hence, although the *Efficiency Ratio* of Gnutella increases with the average number of neighbors ascending, the number of communication messages rises much more.

Test Results from Setup 2

This test is done on the setting described in *Setup 2* which is employed to assess the adaptability of the three protocols.



(a) Efficiency Ratio (ER)

(b) Number of Messages (Num)

(c) Coefficient of Performance (CoP)

Figure 5.3: Performance of different protocols on distinct TTL value

Figure 5.3(a) provides that with TTL value ascending, the *Efficiency Ratio* (ER) of

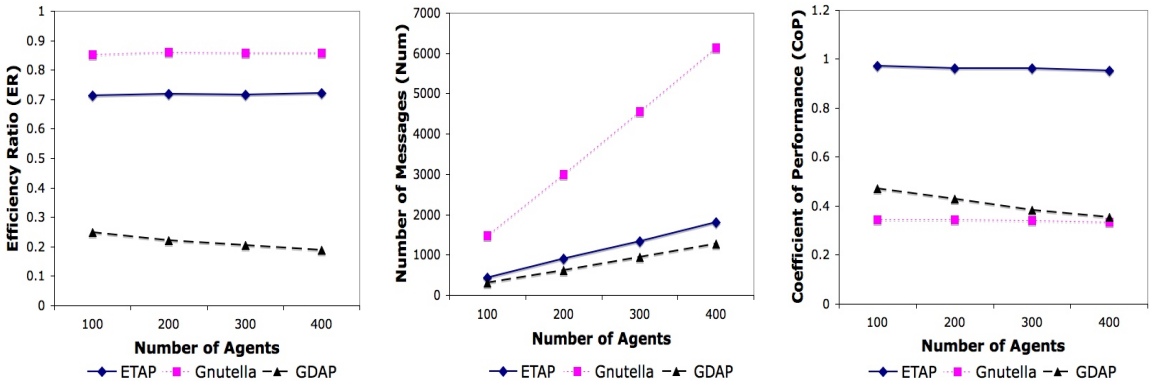
ETAP and Gnutella soars up. This can be explained that the higher the *TTL* value is, the more the task allocation steps are. Therefore, tasks could have more opportunities to be assigned.

From Figure 5.3(b), it is evident that the numbers of messages of ETAP and Gnutella increase gradually with *TTL* value rising. This is because more allocation steps generate more communication messages.

According to Figure 5.3(c), with the increase of *TTL* value, the *CoP*s of both ETAP and Gnutella decrease. The reason of this situation is similar to the one described in Subsection 5.3.4, namely that the increasing rate of *Efficiency Ratio* is less than that of the number of messages. Since *TTL* is not related to GDAP, the performance of GDAP keeps firm during the test process.

Test Results from Setup 3

This test is based on *Setup 3* which has been depicted in 5.3.3. The aim of this setup is to evaluate the scalability of the three protocols in different network scales.



(a) Efficiency Ratio (ER)

(b) Number of Messages (Num)

(c) Coefficient of Performance (CoP)

Figure 5.4: Performance of different protocols on distinct number of agents

According to Figure 5.4(a), we can see that with the increasing of network scale, the *Efficiency Ratio* (*ER*) of GDAP is continually descending while *ER*s of both ETAP and Gnutella keep stable and high. This case can be argued that when the network scale soars up, tasks and types of resources also rise proportionally. Although the average number of neighbors is fixed, more tasks and resource types might still lead to tasks unallocated if *Initiator*s request only neighboring agents. Compared with GDAP, benefited from requesting other peers, both ETAP and Gnutella can preserve

their performance.

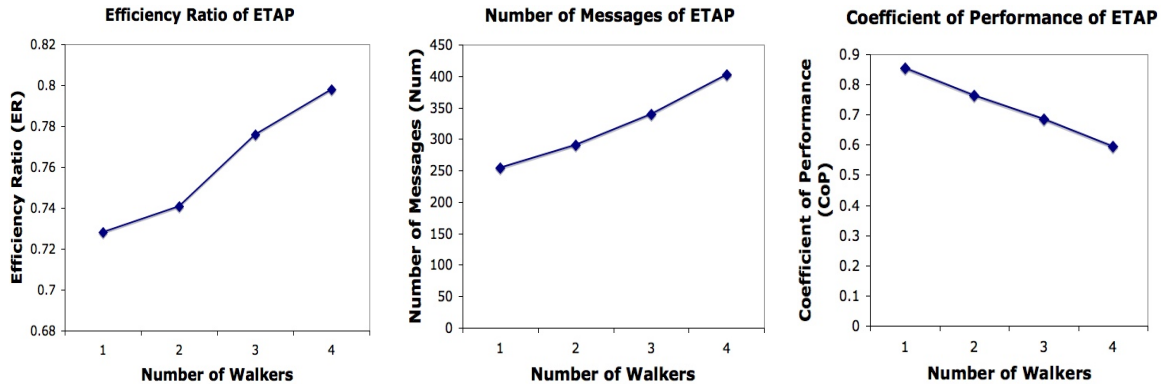
Figure 5.4(b) shows the number of messages (Num) of the three protocols in different network scales. All the three protocols generate more communication messages when there are more peers in the network. This is because although the average number of neighbors is confirmed, more large network scale is accompanied by more peers and tasks. Therefore, in order to allocate these tasks, more communication steps cannot be avoided which results in communication messages rising. On the other hand, the number of communication messages of GDAP always keeps a lower level than that of ETAP and Gnutella. This can be apparently explained that GDAP only relies on neighboring peers, and thus has less communication steps.

Figure 5.4(c) demonstrates the *Coefficient of Performance* (CoP) of the three protocols. It can be seen that the CoPs of both ETAP and Guntella keep almost steady while that of GDAP descends gradually with the increase of network scale. The reason of this result is described as follows. For ETAP and Gnutella, even though the number of communication messages ascends, the absolute value of efficiency derived by completing tasks also increases. Therefore, the CoP can keep stable. On the other hand, for GDAP, as the ER declines gradually while the number of messages soars up continuously, the CoP of GDAP decreases undoubtedly. It should be also noticed that the CoP of GDAP is higher than that of Gnutella, which indicates Gnutella creates a great deal of messages in large scale networks.

Test Results from Setup 4

As described in Subsection 5.3.3, this setup is only for ETAP since the only varying parameter in this setup is the number of *Walker*s, i.e. k, which is not relevant to either Gnutella or GDAP. The target of this setup is to assess how the number of *Walker*s influences the performance of ETAP.

From Figure 5.5(a) and 5.5(b), with the number of *Walker*s increasing, both *Efficiency Ratio* (ER) and number of messages ascend correspondingly. Nevertheless, the *Coefficient of Performance* drops gradually as depicted in Figure 5.5(c). This is because more number of *Walker*s brings higher ER but derives much more messages at the same time. Hence, it can be concluded that in most cases, $k = 1$ is a good choice unless ER is a very critical factor.



(a) (b) (c)

Figure 5.5: The performance of ETAP with different number of walkers

5.3.5 Discussion of ETAP

From the above description, it is obvious that the performance of ETAP is better than that of both Gnutella and GDAP, as each communication message in ETAP can achieve higher efficiency, i.e. higher CoP. Although Gnutella always derives the highest efficiency, the number of messages of Gnutella is very excessive and easily affected by several factors. On the other hand, GDAP generates the least messages but the efficiency of GDAP is low and could be easily disturbed by the average number of neighbors. Therefore, as depicted in the previous paragraphs and pictures, ETAP is more efficient, adaptable and scalable compared with both Gnutella and GDAP.

5.4 Summary

This chapter gives the detailed description of our ETAP, which is used to allocate detection tasks to other peers for collectively detecting distributed attacks. The benefit of ETAP is that it does not need a central planner or to form groups or coalitions before task allocation. Besides, peers with ETAP can request help for tasks not only from neighboring peers but other peers if needed. In the next chapter, we demonstrate the experiment results of our framework, and analyze these results.

Chapter 6

Test and Discussion

Our framework has been implemented by using JACK^{TM} [23]. In this chapter, intrusion detection experiment of the framework is provided, and ETAP is utilized as the detection mechanism. In order to contrast ETAP, Gnutella [18] and GDAP [80] are also set into the framework as detection schemes. We adopt the aforementioned three distributed attacks, i.e. *Doorknob-Rattling Attack*, *Chain/Loop Attack* and *Mitnick Attack*, as instances against our proposed framework for testing. Details of the three distributed attacks have been described in Section 1.1. The intrusion detection test metrics, which are employed to test our framework, are introduced in Section 6.1. Then, three scenarios for task allocation in detecting distributed attacks by using ETAP and the test results about each distributed attack are demonstrated in Section 6.2. Thereafter, discussion regarding the test is elaborated in Section 6.3. Finally, this chapter is summarized in Section 6.4.

6.1 Test Metrics of Intrusion Detection Systems

Ulvila and Gaffney [74] depicted that a system must be in one of the two states: either with an intrusion occurrence (I) or with no intrusion present (NI). The IDS might generate an intrusion alarm (A) or no alarm (NA). Therefore, three parameters used to test intrusion detection systems can be obtained, namely *false alarm rate*, *false negative rate*, and *detection rate*.

- *False alarm rate* means that an alarm arises with no intrusion event occurring, which is written as $P(A|NI) = \alpha$. Lower $P(A|NI)$ means better performance of an IDS.
- The meaning of *false negative rate* is that an intrusion happens with no alarm generated for it, which is written as $P(NA|I) = \beta$. Lower $P(NA|I)$ means better performance of an IDS.

- *Detection rate* depicts the ratio between the number of detected intrusions and the total number of attempted intrusions, which is written as $P(A|I) = 1 - \beta$. Higher $P(A|I)$ means better performance of an IDS.

False alarm rate and *false negative rate* are widely utilized to evaluate the performance of IDSs. However, for our framework, since the concentration is different from most proposed IDSs, we use *detection rate* as the evaluation metrics. Besides, *run time* is also employed as another evaluation metrics, which means the average time length of each detection process. In next section, three scenarios and the experimental detection of *Doorknob-Rattling Attack*, *Chain/Loop Attack* and *Mitnick Attack* will be demonstrated. The experimental detection is based on the two metrics, namely *detection rate* and *run time*.

6.2 Test of the Framework

In this section, the test setting is first depicted, and then three scenarios and the test results against each distributed attack are provided.

6.2.1 Test Setting

Since the contribution of this thesis is not detection algorithms, DARPA dataset [16], which is a famous dataset for testing intrusion detection algorithms, is not considered in this test. Instead, we manually input some normal and hostile records as the belief of agents on different peers, and initiate the detection process from a randomly selected victim peer. Each detection process is performed one hundred times and achieves the results regarding *detection rate* and *run time* which have been described in Section 6.1. In the following subsections, the relevant scenarios are provided first, and then the test results are exhibited. For convenience, the topology of the simulated P2P network is the same as that in the scenarios, i.e. Figure 6.1, 6.3 and 6.5. Due to the lack of test standard, we also mount Gnutella [18] and GDAP [80] into our framework as the detection mechanisms for comparison with ETAP.

6.2.2 Detection of *Doorknob-Rattling Attack*

In this subsection, a scenario regarding the detection of *Doorknob-Rattling Attack* is provided, and then the detection results within the framework are also given.

Scenario One: *Doorknob-Rattling Attack*

Figure 6.1 is an example of a P2P network (such as Ad hoc network [63]) which has four peers to be attacked by a remote host simultaneously. In this example, the task can be described as that the detection of *Doorknob-Rattling* attack needs to check failed login attempt records on several peers and this task is a high emergent task. It is also supposed that the number of tasks is 3 and the three tasks have the same task tuple. Following the definition about task in Subsection 5.1, the description could be matched as the tuple that $<TaskID(t) = Doorknob - Rattling$, $Resource(t) = Failed\_Login\_Attempt > 5$, $Benefit(t) = 10$, $Position(t) = Peer1$, $TTL(t) = 6>$. In this example, $Failed\_Login\_Attempt > 5$ is an assumed threshold of failed login attempt. That means if the number of failed login attempts is more than 5, this record would be considered as a suspicious one and the agent should initiate a detection process. Furthermore, $Benefit(t)$ can be circumscribed according to emergency of the task and, here, the number 10 is just an instance.

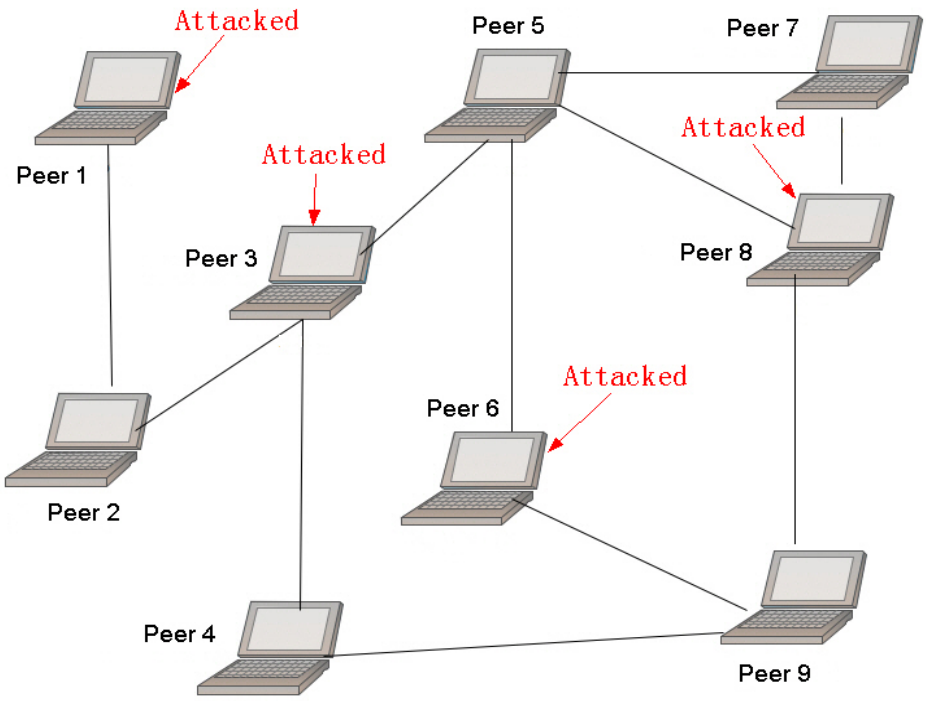


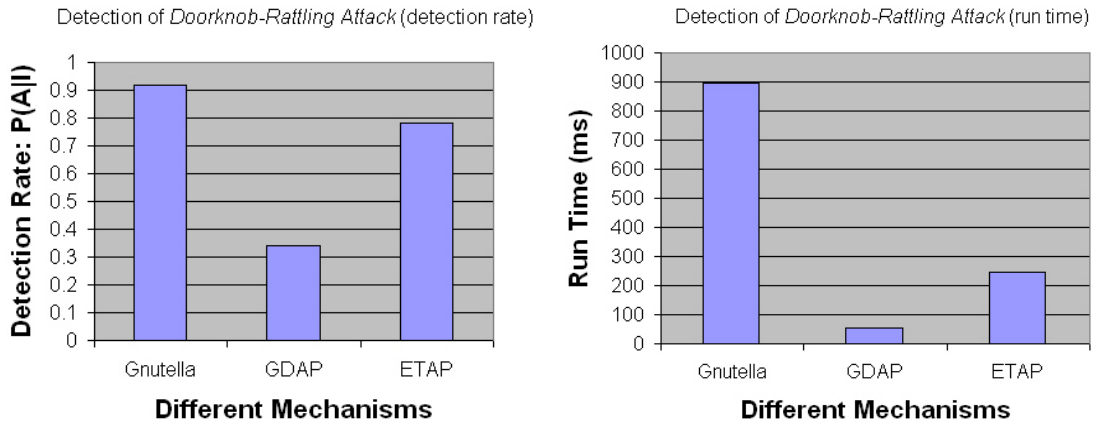
Figure 6.1: An example P2P network which has been attacked by *Doorknob-Rattling*

In terms of ETAP, *Peer 1* starts the task allocation process, namely detection process in this example, by first requesting its neighbors for help. However, the only neighbor of *Peer 1* is *Peer 2* which cannot supply enough resource for *Peer 1*. Then,

Peer 1 sends a *Walker* ($k = 1$ in this example) with $TTL = 6$ to *Peer 2* following the ETAP approach described in Subsection 5.2. *Peer 2* decreases 1 from TTL ($TTL = 5$ now) and sends the *Walker* to its neighbor, namely *Peer 3*. After that, *Peer 3* has the desired resource and sends a response message back to *Peer 1*. *Peer 3* then randomly chooses a neighbor and dispatches the *Walker* with $TTL = 4$. Here, we suppose *Peer 3* selects *Peer 5*. With the same process, *Peer 5* chooses *Peer 6* with $TTL = 3$. *Peer 6* sends a response message back and forwards the *Walker* with $TTL = 2$ to *Peer 9*. Subsequently, it is assumed that *Peer 9* randomly selects *Peer 8* with $TTL = 1$. Finally, *Peer 8* sends a response message back to *Peer 1* and decrements 1 from TTL. *Peer 8* finds $TTL = 0$ and discards the *Walker*. The *Initiator*, *Peer 1*, then make a decision about *Doorknob-Rattling Attack* based on these responses. Details with regard to the communication between different peers or different agents in one peer can be found in Section 3.2.

Detection within the Framework

For each test iteration, four randomly selected peers are victims and one of them initiates the detection process. The example can be found in Figure 6.1. It is defined that when three out of the four victim peers are discovered, the attack is successfully detected. Three detection mechanisms, i.e. Gnutella, GDAP and ETAP, are executed in each iteration. For Gnutella and ETAP, the TTL value is set to 4. The experiment results are demonstrated in Figure 6.2.



(a) (b)

Figure 6.2: Detection of *Doorknob-Rattling Attack* with different mechanisms

Figure 6.2(a) introduces the detection rates, i.e. $P(A|I)$, of three detection schemes.

It is obvious that Gnutella achieves a higher detection rate than GDAP and ETAP, because Gnutella attempts more peers in the network. GDAP derives the lowest detection rate as it only requests neighboring peers for help. The performance of ETAP is between Gnutella and GDAP. On the other hand, Figure 6.2(b) depicts the run time of each mechanism. Overtly, Gnutella needs the most time, while GDAP spends the least time. The reason is the same as that described about detection rate. Comparatively, ETAP obtains a balance between the two evaluation metrics, namely detection rate and run time.

6.2.3 Detection of *Chain/Loop Attack*

This subsection exhibits a scenario regarding the detection of *Chain/Loop Attack*, and the experimental detection within the framework.

Scenario Two: *Chain/Loop Attack*

Figure 6.3 exhibits an example that four peers in a P2P network have been intruded by *Chain/Loop Attack*. In this scenario, the task is to observe TCP or UDP connections among several peers in order to discover whether the connections could be formed into a "chain" or "loop". It is assumed that the number of tasks is 3, and the three tasks need different resources which are explained in the following. *Peer 1* is supposed to be the *Initiator*. As *Peer 1* finds out two dubitable connection records, it launches a task allocation process which is detection process in this example. One suspicious connection record is from a remote host with more than three failed login attempts, and the other is that the remote host later builds a connection to another peer in the network through *Peer 1*. According to the definition in Subsection 5.1, the task tuple of *Initiator*, i.e. *Peer 1*, can be described as $<TaskID(t) = Chain/Loop$, $Resource(t)$, $Benefit(t) = 10$, $Position(t) = Peer1$, $TTL(t) = 3>$, where $Resource(t)$ includes $Failed\_Login\_Attempt > 3$, $Connection = to$ and $Connection\_TimeStamp > t1$. In this example, $Failed\_Login\_Attempt > 3$ indicates a predefined finitude which is deemed to be a suspicious connection, $Connection = to$ demonstrates this connection is built from this peer to another peer in the network, and $Connection\_TimeStamp > t1$ means the connection time should be later than $t1$ at which $Peer1$ launches a connection to another peer, such as *Peer 2*. $Benefit(t) = 10$ is just an instance, which is the same as Scenario One in Subsection 6.2.2.

In the light of ETAP, *Peer 1* starts the task allocation process, namely detection process, by first requesting its neighbors for help. The neighbor of *Peer 1* is

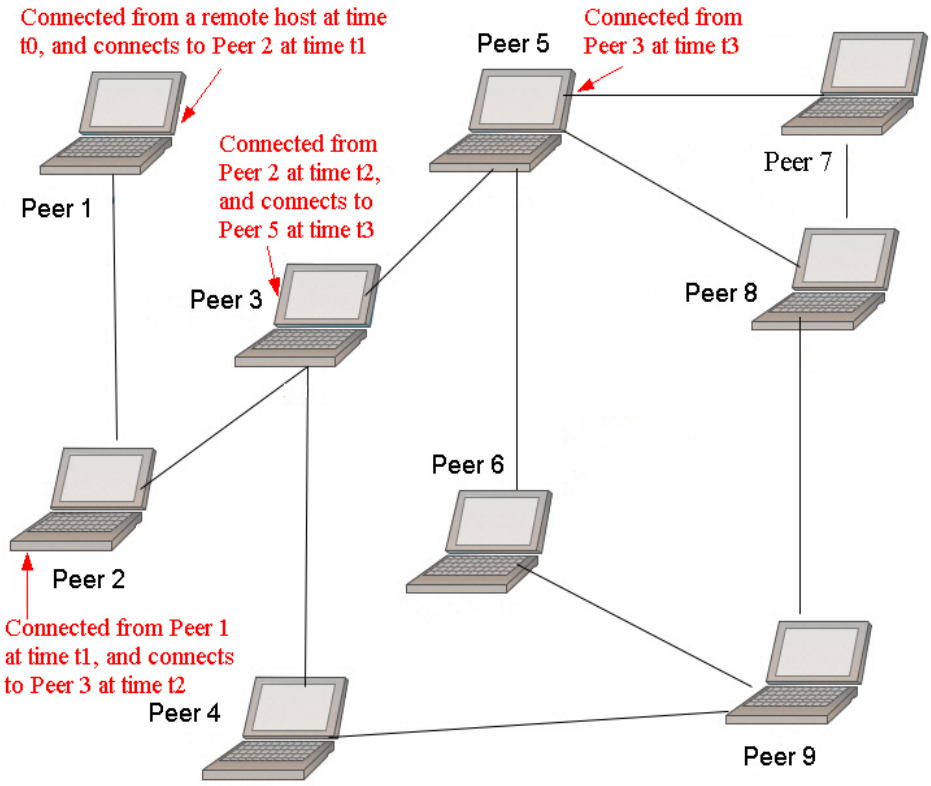


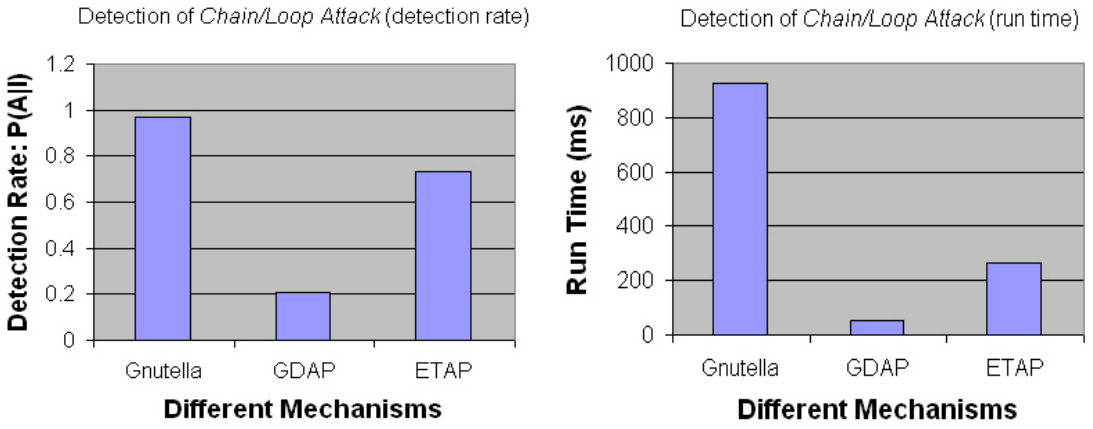
Figure 6.3: An example P2P network which has been attacked by *Chain/Loop*

Peer 2 which does not have sufficient resources for *Peer 1*, since *Peer 2* only has one connection record which can satisfy *Peer 1*'s desire. Therefore, *Peer 1* allocates one task to *Peer 2* and dispatches a *Walker* ($k = 1$ in this example) with $TTL = 3$ to *Peer 2*. *Peer 2* modifies the *Walker* as $<TaskID(t) = Chain/Loop$, $Resource(t)$, $Benefit(t) = 10$, $Position(t) = Peer1$, $TTL(t) = 2>$, where $Resource(t)$ includes $Failed\_Login\_Attempt > 3$, $Connection = to$ and $Connection\_TimeStamp > t2$. $t2$ is the time at which *Peer 2* establishes a connection to another peer. *Peer 2* then sends the modified *Walker* with $TTL = 2$ to one of its neighbors. The only neighbor of *Peer 2* is *Peer 3* in this example. With the same measure, *Peer 3* revises the *Walker* as $<TaskID(t) = Chain/Loop$, $Resource(t)$, $Benefit(t) = 10$, $Position(t) = Peer1$, $TTL(t) = 1>$, where $Resource(t)$ involves $Failed\_Login\_Attempt > 3$, $Connection = to$ and $Connection\_TimeStamp > t3$. The meanings of $t3$ is similar as $t1$ and $t2$. Subsequently, *Peer 3* responds to the *Initiator*, *Peer 1*, because *Peer 3* has the one connection record that *Peer 1* wishes. *Peer 3* selects the neighbor *Peer 5* to send the *Walker*, because there is a doubted connection record on *Peer 3* that indicates *Peer 5* is the next "node" in the potential *Chain/Loop Attack*. *Peer 5* then alters the *Walker*

as $<TaskID(t) = Chain/Loop$, $Resource(t)$, $Benefit(t) = 10$, $Position(t) = Peer1$, $TTL(t) = 0>$, and replies the *Peer 1*. Since TTL has become zero, *Peer 5* discards the *Walker* without further process. Finally, *Peer 1* correlates these responses to constitute the profile of a *Chain Attack*.

Detection within the Framework

For each test iteration, four peers, which are neighboring one by one, are imaginary victims, and the detection process is originated from the first or last peer in the "chain". Figure 6.3 demonstrates an example of this attack. It is set that when all of the four victims are exposed, the attack is successfully discovered. For Gnutella and ETAP, the TTL value is set to 5. Figure 6.4 shows the experimental detection results.



(a) (b)

Figure 6.4: Detection of *Chain/Loop Attack* with different mechanisms

Figure 6.4(a) displays the detection rates of three detection schemes. Gnutella undoubtedly obtains the highest detection rate, while the next one is ETAP and the last one is GDAP. For the run time of each detection mechanism, GDAP is the fastest while Gnutella is the slowest. ETAP is medium between GDAP and Gnutella. The sake is the same as the one described in Subsection 6.2.2.

6.2.4 Detection of *Mitnick Attack*

This subsection demonstrates a scenario about the detection of *Mitnick Attack*, and then displays the test results within the framework.

Scenario Three: *Mitnick Attack*

In Figure 6.5, a *Mitnick Attack* example, which is against a P2P network, is demonstrated. *Peer 1* is supposed to be the *Initiator*. When *Peer 1* discovers a TCP sequence number predication attack, it originates a task allocation process, i.e. detection process, in an effort to expose a *Mitnick Attack*. In this case, the number of task is 1, with the task tuple $<TaskID(t) = Mitnick, Resource(t), Benefit(t) = 10, Position(t) = Peer1, TTL(t) = 10>$, where $Resource(t)$ involves $Attack_Type = SYNFlooding$ and $Trusted_by_Peer1 = true$. Here, TTL is larger than the former two scenarios, since the other victim peer could be anywhere in the network.

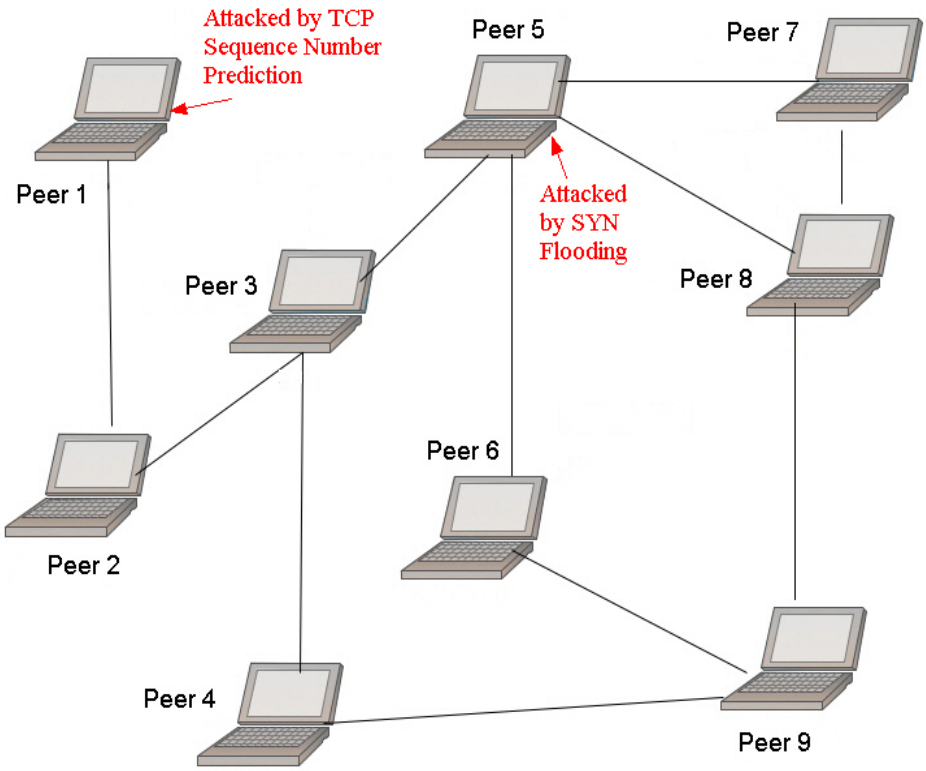


Figure 6.5: An example P2P network which has been attacked by *Mitnick*

According to ETAP, *Peer 1* launches the task allocation process by first requesting neighboring peers for help. The only neighbor of *Peer 1* is *Peer 2* which does not contain the interested resource. Then, *Peer 1* dispatches a *Walker* ($k = 1$ in this scenario) with $TTL = 10$ to *Peer 2*. *Peer 2* decreases one from TTL ($TTL = 9$) and transmits the *Walker* to its neighbor, *Peer 3*, which does not have the relevant resource either. Whereafter, *Peer 3* decrements TTL by 1 ($TTL = 8$), and selects one of its neighbors from *Peer 4* and *Peer 5* to pass the *Walker*. Here, it is assumed that *Peer 3*

chooses *Peer 4* as the next stop. The innocent *Peer 4* detracts 1 from TTL ($TTL = 7$) and sends the *Walker* to its neighbor *Peer 9*. With the same way, *Peer 9* selects *Peer 8* and transfers the *Walker* with $TTL = 6$, and *Peer 8* pushes the *Walker* with $TTL = 5$ to *Peer 7* supposingly. *Peer 7*, subsequently, sends the *Walker* with $TTL = 4$ to *Peer 5* which has the desired resource. *Peer 5* then responds the request to *Peer 1*, and declines TTL by 1. Since *Peer 3*, *Peer 7* and *Peer 8* have been visited, *Peer 5* issues the *Walker* with $TTL = 3$ to *Peer 6*. Then, *Peer 6* sends the *Walker* with $TTL = 2$ to *Peer 9*. As all the neighboring peers of *Peer 9* have been requested, *Peer 9* discards the *Walker* even though the TTL value of the *Walker* does not reach zero. The *Initiator*, *Peer 1*, synthesizes the response from *Peer 5* and results in a decision with regard to the existence of *Mitnick Attack*.

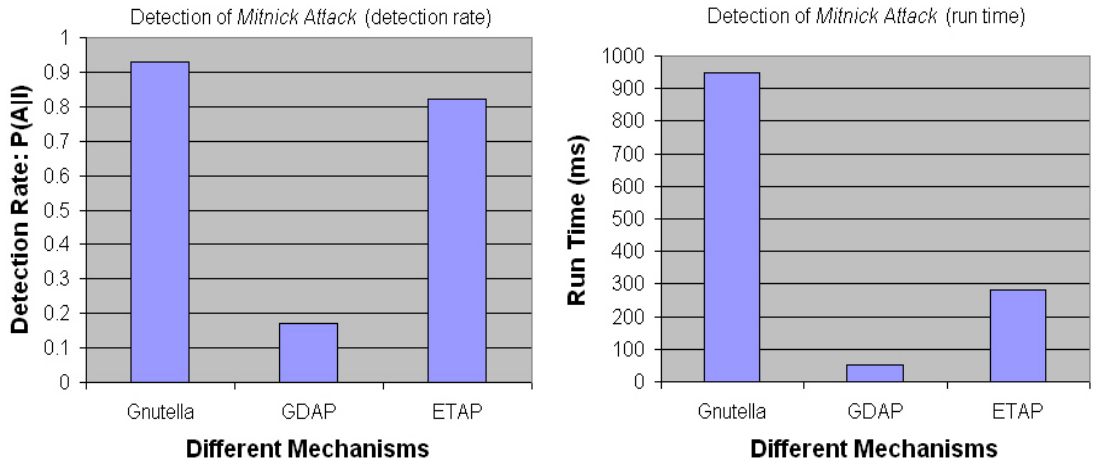
Detection within the Framework

According to the nature of *Mitnick Attack*, during each detection iteration, two randomly selected peers are assumed to be attacked. One peer is supposed to be attacked by TCP sequence number predicting and the other is attacked by SYN flooding attack. Since the peer attacked by SYN flooding attack usually has a network congestion, the detection process is assumed to be launched from the other victim peer. Figure 6.5 displays an example of this attack. It is enacted that when both of the two victim peers are found out, the attack is successfully detected. The three detection mechanisms, namely Gnutella, GDAP and ETAP, are all utilized to perform the detection processes. For Gnutella and ETAP, the TTL value is confirmed at 6. Figure 6.6 demonstrates the detection results.

In Figure 6.6(a), the detection rates of Gnutella, GDAP and ETAP are exhibited. Gnutella derives the higher detection rate than both of GDAP and ETAP, while ETAP achieves better detection rate than GDAP. In Figure 6.6(b), the performance of GDAP is the best, while Gnutella is the worst. The average consuming time of ETAP is between that of GDAP and Gnutella. The reason is the same as the one depicted in Subsection 6.2.2.

6.3 Discussion of the Test

Section 6.2 provided three scenarios and a detailed test for the proposed framework. According to the three scenarios, ETAP exhibits its potential capability of handling some real cases. However, people might argue that in these scenarios, other peers,



(a) (b)

Figure 6.6: Detection of *Mitnick Attack* with different mechanisms

other than *Peer 1*, might also initiate the detection process synchronizingly with *Peer 1*. This problem is called task duplication which may increase the redundant time and communication overhead. Agent negotiation mechanism might be borrowed to solve this issue in some extent, which is one of our future works.

Although the test results against the three simulated attacks, i.e. *Doorknob-Rattling*, *Chain/Loop* and *Mitnick* attacks, are analogous, there are two issues which are worthy to discuss.

1. According to Figure 6.2(a) and 6.4(a), the detection rate of Gnutella with regard to *Doorknob-Rattling Attack* is a little lower than that of *Chain/Loop Attack*. This can be explained that during the *Chain/Loop Attack* the victim peers are located relatively converged, and therefore they are easier to be discovered by using Gnutella algorithm which requests all accessible peers within the TTL value. However, the detection rates of both GDAP and ETAP about *Doorknob-Rattling Attack* are higher than those regarding *Chain/Loop Attack*. For GDAP, this is because not all the victim peers are neighbors of the *Initiator* in general, and thus it is hard to discover the attack by only requesting neighboring peers. For ETAP, the reason is that there is only one *Walker* which only travels to one neighbor each time. Due to the character of *Chain/Loop Attack* that usually only one neighbor of a victim peer is attacked, the *Walker* might miss victim peers during its traveling process and hence leads to detection failure.

2. Comparing Figure 6.2(a) and 6.6(a), the detection rates of Gnutella and ETAP

about *Doorknob-Rattling Attack* are almost the same as those for *Mitnick Attack*. The reason is that when the TTL value is set to be large enough, the other victim peers can be easily discovered. Nevertheless, the detection rate of GDAP regarding *Mitnick Attack* decreases slightly compared to that against *Doorknob-Rattling Attack*. This can be argued that in *Mitnick Attack*, two victim peers may be distributed far between each other, and therefore GDAP, which only asks neighbors for help, might result in detection failure.

From the above description, it can be concluded that ETAP which is proposed in this thesis achieves a good balance between the detection rate and run time, while Gnutella and GDAP focus more on one side. In addition, according to Chapter 5, ETAP is more flexible than Gnutella and GDAP by adjusting the number of *Walkers* and TTL value. Hence, compared to Gnutella and GDAP, ETAP is more suitable for detecting distributed attacks in P2P environments.

6.4 Summary

This chapter provides the experiment about our framework according to using three detection schemes, i.e. Gnutella, GDAP and ETAP which is proposed in this thesis. The test results demonstrate that ETAP fits our framework better than Gnutella and GDAP. In the next chapter, we conclude this thesis and introduce some future works.

Chapter 7

Conclusion

Thousands of computers in the Internet suffer the threat of various network attacks. In order to protect computers from being compromised, many IDSs have been proposed. Some of them are centralized, while others are decentralized. The primary objective of this thesis is to deal with the fundamental issues related to agent-based P2P framework for distributed intrusion detection, which include:

1. Architecture of the agent-based P2P framework;
2. Design of agents functionalities in the framework;
3. Knowledge representation of agents in the framework;
4. A task allocation protocol which can be adopted as a detection scheme for the framework.

In the rest of this chapter, we present the major contributions of this thesis and compare each contribution with current related research. Thereafter, we discuss the remaining problems of the proposed framework, and outline the future research directions.

7.1 Major Contributions of this Thesis

In Chapters 3, 4 and 5, we proposed a novel architecture of an agent-based P2P framework, and then designed the functions of each agent in the framework in detail. Thereafter, we represented the knowledge of each agent by using ontology, and finally introduced an efficient task allocation protocol for detecting distributed attacks in the framework. In the following subsections, we outline our contributions and compare them with current related works.

7.1.1 Architecture of the Agent-Based P2P Framework

The framework proposed in this thesis is based on P2P layout, and each peer in the framework contains multiple agents, each of which performs different functions. The benefits of the architecture are as follows.

1. Independent of particular network topologies;
2. Compared to centralized IDSs [31] [84] which have a central manager to handle issues in the systems, the proposed decentralized architecture can avoid single point failure and network congestion;
3. Against current P2P IDSs [55] [79] which remove the central manager but only allow peers to make queries among neighbors, the presented architecture allows peers to collaborate with not only direct linked neighbors but also other peers if necessary. In this way, peers may have more opportunity to achieve their goals.

Furthermore, many proposed agent-based IDSs stop at initial architecture design without detailed design and implementation of each agent. In this thesis, an agent development software, $JACK^{TM}$, is employed to design and implement agents in the framework. Compared to most relevant works, agents in our framework are not only introduced but also implemented. Contrasted with [65] which also implemented the agents but focused on detection on a single host, our framework concentrates on intrusion detection in a distributed domain.

7.1.2 Knowledge Representation of Agents

Many attack languages were proposed in the past decade, which can be classified as Event, Response, Reporting, Correlation, Exploit and Detection Languages [76]. These languages aim at representing various knowledge in intrusion detection. Nonetheless, attack languages mainly focus on a specific aspect of intrusion detection and they are usually dependent on particular systems and environments. Hence, the extensibility and communicability among non-homogeneous systems of these attack languages are incompetent. The advantages of adopting ontology to represent intrusion detection knowledge in this thesis include the following three facets [75].

1. Representation: ontology is able to model the knowledge of a domain, which involves the domain's attributes and characteristics;

2. Information sharing: ontology can represent the existence of an instance of the domain (model) in a way that is understandable by any other entities that possess the specific ontology;

3. Reasoning: ontology has the capability of aggregating particular instances of the domain in a knowledge base and concluding that some larger, or more comprehensive, instances of the ontology exist.

In this thesis, ontology is utilized to represent agents' knowledge in the proposed framework. Compared to related attack languages which focus on specific domains of intrusion detection, such as [13] and [19], our representation approach is more extensible and sharable. Against other ontology based intrusion detection representation methods which concentrate on a single host, such as [25] and [75], our approach focuses on intrusion and detection representation for a distributed environment. In this manner, peers in our framework can share their common understanding of information to discover distributed attacks through agent communication and cooperation.

7.1.3 A Task Allocation Protocol

In order to discover distributed attacks, allocating detection tasks to different peers is necessary. Thus, a potent task allocation protocol is needed. Various task allocation protocols have been developed. Some of them are centralized, which assume there is a central planner to schedule task allocation processes. Several protocols need the nodes in an environment to form groups before allocating tasks. Furthermore, some allocation mechanisms allow the nodes in an environment only request neighboring nodes for help.

In this thesis, an efficient task allocation protocol, i.e. ETAP, is proposed and utilized into our framework for distributed intrusion detection. Compared to [85], ETAP does not need a central planner. Against [37], it is not necessary for ETAP to form coalitions among nodes before allocating tasks. Unlike GDAP presented in [80] which allows only neighboring nodes to help with a task, ETAP enables nodes to allocate tasks not only to their neighbors but also other nodes in the system.

7.2 Remaining Problems and Future Work

Although several fundamental problems with regard to IDSs have been solved in this thesis, some remaining issues are still existent.

1. A well defined model about detection strategy is necessary, as detection strategies are the basis for an IDS. This model should be complete in order to handle as many types of attacks as possible, and could accommodate variants of existing attacks. However, in this thesis, our concentration is on building an agent-based P2P framework for distributed intrusion detection, not on detailed detection strategies which will be one of our future works.

2. During detection processes in our framework, it is assumed that the detection is initiated from one of the victim peers. Nevertheless, in real cases, other victim peers might also originate the detection processes for exposing the same attack. This condition would cause redundant detection processes and results in time and network consumption soaring up. How to deal with this problem is another future work of us.

3. The agent development tool, $JACK^{TM}$ [23], which is utilized to implement our framework, is not good for building mobile agent and network simulation. Therefore, the test results might have some bias, but the trends of the three detection schemes, i.e. Gnutella [18], GDAP [80] and ETAP, which are revealed by the test, is still worthy to notice. Since $JACK^{TM}$ is powerful for developing multi-agent systems with a free version, it is selected as development tool in this thesis. In the future, we will use some other methods to re-implement our framework and test it.

Bibliography

[1] S. Abdallah and V. Lesser. Modeling task allocation using a decision theoretic model. In *Proceedings of AAMAS'05*, pages 719–726, Utrecht, Netherlands, July 2005.

[2] William Aiello, Fan Chung, and Linyuan Lu. A random graph model for massive graphs. In *Proceedings of STOC'00*, pages 171–180, May 2000.

[3] James P. Anderson. Computer security threat monitoring and surveillance. Technical report, Fort Washington, PA: James P. Anderson Co., Apr. 1980.

[4] J.S. Balasubramaniyan, J.O. Garcia-Fernandez, D. Isacoff, E. Spafford, and D. Zamboni. An architecture for intrusion detection using autonomous agents. Technical Report 98-05, COAST Laboratory, Purdue University, 1998.

[5] K. Boudaoud and C. McCathieNevile. An intelligent agent-based model for security management. In *Proceedings of the Seventh International Symposium on Computers and Communications*, pages 877–882, Los Alamitos, CA, USA, Jul. 2002.

[6] D. Boughaci, H. Drias, A. Bendib, Y. Bouznit, and B. Benhamou. Distributed intrusion detection framework based on autonomous and mobile agents. In *Proceedings of the International Conference on Dependability of Computer Systems*, pages 248–255, May 2006.

[7] D. Curry and H. Debar. Intrusion detection message exchange format data model and extensible markup language (xml) document type definition. *http://tools.ietf.org/id/draft-ietf-idwg-idmef-xml-03.txt*, Feb. 2001.

[8] D. Dasgupta, F. Gonzales, K. Yallapu, J. Gomez, and R. Yarramsetti. Cids: An agent-based intrusion detection system. *Computer & Security*, 24(5):382–398, August 2005.

[9] Prithviraj Dasgupta. Adaptive sharing of large resources in p2p networks. In *Proceedings of AAMAS'05*, pages 839–845, Utrecht, Netherlands, July 2005.

[10] H. Debar and D. Curry. The intrusion detection message exchange format (idmef). *http://www.ietf.org/rfc/rfc4765.txt*, Mar. 2007.

[11] D. Denning. An intrusion-detection model. *IEEE Trans. on Software Engineering*, SE-13(2):222– 232, Feb. 1987.

[12] J. Doyle, I. Kohane, W. Long, H. Shrobe, and P. Szolovits. Event recognition beyond signature and anomaly. In *Proceedings of the Second IEEE SMC Information Assurance Workshop*, pages 17–23, West Point, NY, USA, Jun. 2001.

[13] S. Eckmann, G. Vigna, and R. Kemmerer. Statl: An attack language for state-based intrusion detection. *Journal of Computer Security*, 10(1/2):71–104, 2002.

[14] L. Fang, M. Cai, H. Fu, and J. Dong. Ontology-based fraud detection. In *Proceedings of International Conference on Computational Science*, pages 1048–1055, Beijing, China, May 2007. LNCS.

[15] R. Feiertag, C. Kahn, P. Porras, D. Schackenberg, S. Staniford-Chen, and B.Tung. A common intrusion specification language. *http://www.isi.edu/brian/cidf/drafts/language.txt*, Jun. 1999.

[16] D. J. Fried, I. Graf, J. W. Haines, K. R. Kendall, D. Mcclung, D. Weber, S. E. Webster, D. Wyschogrod, R. K. Cunningham, and M. A. Zissman. Evaluating intrusion detection systems: The 1998 darpa off-line intrusion detection evaluation. In *Proceedings of the 2000 DARPA Information Survivability Conference and Exposition*, pages 12–26, Hilton Head, SC, USA, Jan. 2000.

[17] D. Frincke. Balancing cooperation and risk in intrusion detection. *ACM Trans. Information and System Security*, 3(1):1–29, February 2000.

[18] Gnutella. The gnutella protocol specification v0.4. In *www9.limewire.com/developer/gnutella_protocol_0.4.pdf*, 2000.

[19] J. Goubault-Larrecq. An introduction to logweaver (v2.8). *http://www.lsv.ens-cachan.fr/ goubault/DICO/tutorial.pdf*, Sep. 2001.

[20] T. R. Gruber. Toward principles for the design of ontologies used for knowledge sharing. *In Formal Ontology in Conceptual Analysis and Knowledge Representation, Kluwer Academic Publishers, in press. Substantial revision of paper presented at the International Workshop on Formal Ontology*, 1993.

[21] T. R. Gruber. A translation approach to portable ontologies. *Knowledge Acquisition*, 5(2):199–220, 1993.

[22] B. Guha and B. Mukherjee. Network security via reverse engineering of tcp code: Vulnerability analysis and proposed solutions. *IEEE Networks*, 11(4):40–48, Jul./Aug. 1997.

[23] Jack Intelligent Agent User Guide. *http://www.agent-software.com*, 2009.

[24] J.W. Haines, L.M. Rossey, R.P. Lippman, and R.K. Cunningham. Extending the darpa off-line intrusion detections evaluations. In *DARPA Information Survivability Conference and Exposition II*, volume 1, pages 35–45, Anaheim, CA, USA, Jun. 2001.

[25] Y. He, W. Chen, M. Yang, and W. Peng. Ontology based cooperative intrusion detection system. In *Proceedings of International Conference on Network and Parallel Computing*, pages 419–426, Wuhan, China, Oct. 2004. LNCS.

[26] G. Helmer, J.S.K. Wong, V. Honavar, L. Miller, and Y. Wang. Lightweight agents for intrusion detection. *The Journal of Systems and Software*, 67(2):109–122, August 2003.

[27] A. Joshi and J. Undercoffer. On web, semantics, and data mining: Intrusion detection as a case study. In *Proceedings of the NSF Workshop on Next Generation Data Mining*, pages 62–68, Baltimore, Maryland, USA, Nov. 2002.

[28] M. Jovanovic, F. Annexstein, and K. Berman. Scalability issues in large peer-to-peer networks - a case study of gnutella. Technical report, University of Cincinnati, 2001.

[29] C. Kahn, D. Bolinger, and D. Schackenberg. Communication in the common intrusion detection framework v 0.7. *http://www.isi.edu/brian/cidf/drafts/communication.txt*, Jun. 1998.

[30] V. Kalogeraki, D. Gunopulos, and D. Zeinalipour-Yazti. A local search mechanism for peer-to-peer networks. In *Proceedings of CIKM*, pages 300–307, Mclean, Virginia, USA, November 2002.

[31] P. Kannadiga and M. Zulkernine. Didma a distributed intrusion detection system using mobile agents. In *Proceedings of the Sixth International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing, and First ACIS International Workshop on Self-Assembling Wireless Networks*, pages 238–245, May 2005.

[32] K. Kendall. A database of computer attacks for the evaluation of intrusion detection systems. Master's thesis, MIT, Department of EECS, 1999.

[33] Irwin King, Cheuk Hang NG, and Ka Cheung Sia. Distributed content-based visual information retrieval system on peer-to-peer networks. *ACM Trans. Information Systems*, 22(3):477–501, July 2004.

[34] Calvin Ko, Deborah A. Frincke, Terrence Goan, L. Todd, Heberlein Karl, Levitt Biswanath, and Mukherjee Christopher Wee. Analysis of an algorithm for distributed recognition and accountability. In *The first ACM Conference on Computer and Communication Security*, pages 154–164, Fairfax, VA, US, Nov. 1993.

[35] S. Kraus, O. Shehory, and G. Taase. Coalition formation with uncertain heterogeneous information. In *Proceedings of AAMAS'03*, pages 1–8, Melbourne, Australia, July 2003.

[36] C.E. Landwehr, A.R. Bull, J.P. McDermott, and W.S. Choi. A taxonomy of computer program security flaws. *ACM Computing Surveys*, 26(3):211–254, Sep. 1994.

[37] K. Lerman and O. Shehory. Coalition formation for large-scale electronic markets. In *Proceedings of ICMAS'00*, pages 167–174, Boston, Massachusetts, USA, July 2000.

[38] C. Li, Q. Song, and C. Zhang. Ma-ids architecture for distributed intrusion detection using mobile agents. In *Proceedings of the 2nd International Conference on Information Technology for Application (ICITA 2004)*, pages 451–455, Jan. 2004.

[39] J. Lin, X.S. Wang, and S. Jajodia. Abstraction-based misuse detection: High-level specifications and adaptable strategies. In *Proceedings of the 11th Computer Security Foundations Workshop*, pages 190–201, Rockport, MA, USA, Jun. 1998.

[40] U. Lindqvist and P.A. Porras. Detecting computer and network misuse through the productionbased system toolset (p-best). In *Proceedings of the 1999 IEEE Symposium on Security and Privacy*, pages 146–161, Oakland, California, USA, May 1999.

[41] Richard P. Lippmann, David J. Fried, Isaac Graf, Joshua W. Haines, Kristopher R. Kendall, David McClung, Dan Weber, Seth E. Webster, Dan Wyschogrod, Robert K. Cunningham, and Marc A. Zissman. Evaluating intrusion detection systems: The 1998 darpa off-line intrusion detection evaluation. In *Proceedings of DARPA Information Survivability Conference and Exposition. DISCEX '00*, volume 2, pages 12–26, Hilton Head, South Carolina, Oct., January 2000. IEEE Computer Society.

[42] C. Lv, P. Cao, E. Cohen, K. Li, and S. Shenker. Search and replication in unstructured peer-to-peer networks. In *Proceedings of ICS*, pages 84–95, New York, New York, USA, June 2002.

[43] J. McHugh. Testing intrusion detection systems: a critique of the 1998 and 1999 darpa intrusion detection system evaluations as performed by lincoln laboratory. *ACM Transactions on Information and System Security*, 3(4):262–294, Nov. 2000.

[44] D. Menasce and L. Kanchanapalli. Probabilistic scalable p2p resource location services. In *SIGMETRICS Perf. Eval. Review*, 2002.

[45] C. Michel and L. Me. Adele: An attack description language for knowledge-based intrusion detection. In *Proceedings of the 16th International Conference on Information Security*, pages 353–368. Kluwer, 2001.

[46] Jelena Mirkovic and Peter Reiher. A taxonomy of ddos attack and ddos defense mechanisms. *ACM SIGCOMM Computer Communications Review*, 34(2):39–54, Apr. 2004.

[47] N-Triples. *http://www.w3.org/2001/sw/RDFCore/ntriples/*, 2009.

[48] Secure Networks. Custom attack simulation language (casl). *http://download.matus.in/doc/eBooks/casl.ps*, Jan. 1998.

[49] Peng Ning, S. Jajodia, and X.S. Wang. Abstraction-based intrusion detection in distributed environment. *ACM Trans. Information and System Security*, 4(4):407–452, November 2001.

[50] S. NORTHCUTT. *Network Intrusion Detection: An Analysts Handbook*. New Riders, 1999.

[51] M. Papasimeon and C. Heinze. Extending uml for designing jack agents. In *Proceedings of 13th Australian Software Engineering Conference*, pages 89–97, Canberra, Australia, Aug. 2001.

[52] V. Paxson. Bro: A system for detecting network intruders in real time. In *Proceedings of the 7th Symposium on USENIX Security*, pages 2435–2463, San Antonio, TX, USA, Jan. 1998.

[53] Protege Platform. *http://protege.stanford.edu/*, 2009.

[54] X. Qin, W. Lee, L. Lewis, and J.B.D. Cabrera. Intergrating intrusion detection and network management. In *Proceedings of the 8th IEEE/IFIP Network Operations and Management Symposium*, pages 329–344, Florence, Italy, Apr. 2002.

[55] G. Ramachandran and D. Hart. A p2p intrusion detection system based on mobile agents. In *Proceedings of the 42nd annual southeast regional conference*, pages 185–190, Huntsville, Alabama, USA, 2004.

[56] A. Rao and M. Georgeff. Modeling rational agents within a bdi-architecture. In *Porceedings of the 2nd International Conference on Principles of Knowledge Representation and Resoning*, pages 473–484, San Mateo, CA, 1991. Morgan Kaufmann.

[57] V. Raskin, C. F. Hempelmann, K. E. Triezenberg, and S. Nirenburg. Ontology in information security: A useful theoretical foundation and methodological tool. In *Proceedings of Workshop on New Security Paradigms*, pages 53–59, Cloudcroft, New Mexico, USA, Sep. 2001. ACM Press.

[58] Resource Description Framework (RDF). *http://www.w3.org/RDF/*, 2004.

[59] M. Roesch. Snort, version 2.8.3.1, an open source nids. *availble via www.snort.org*, 2009.

[60] M. Roger and J. Goubault-Larrecq. Log auditing through model checking. In *Proceedings of 14th the IEEE Computer Security Foundations Workshop*, pages 220–236, Cape Breton, Nova Scotia, Canada, Jun. 2001.

[61] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach (Second Edition)*. Prentice Hall, 2003.

[62] Pedro V. Sander, Denis Peleshchuk, and Barbara J. Grosz. A scalable, distributed algorithm for efficient task allocation. In *Proceedings of AAMAS'02*, pages 1191–1198, Bologna, Italy, July 2002.

[63] PAOLO SANTI. Topology control in wireless ad hoc and sensor networks. *ACM Computing Surveys*, 37(2):164–194, June 2005.

[64] C.L. SCHUBA, I.V. KRSUL amd M.G. KUHN, E.H. SPAFFORD, A. SUNDARAM, and D. ZAMBONI. Analysis of a denial of service attack on tcp. In *Proceeding of the 1997 IEEE Symposium on Security and Privacy*, pages 208–223, Oakland, CA, US, May 1997.

[65] M. Shajari and A. A. Ghorbani. Agent-oriented design for network survivability. In *Proceedings of the 5th International Conference on Intelligent Systems Design and Applications*, pages 166–171, Wroclaw, Poland, Sep. 2005.

[66] O. Shehory and S. Kraus. Methods for task allocation via agent coalition formation. *Artificial Intelligence*, 101(1-2):165–200, May 1998.

[67] R. Shirey. Internet security glossary. *http://www.ietf.org/rfc/rfc2828.txt*, May 2000.

[68] A. Simmonds, P. Sandilands, and L.V. Ekert. An ontology for network security attacks. In *Proceedings of the Second Asian Applied Computing Conference*, pages 317–323, Kathmandu, Nepal, Oct. 2004. LNCS.

[69] R. G. Smith. The contract net protocol: High-level communication and control in a distributed problem solver. *IEEE Trans. Computers*, C-29(12):1104–1113, December 1980.

[70] R. G. Smith and R. Davis. Frameworks for cooperation in distributed problem solving. *IEEE Trans. Syst., Man, and Cyber.*, SMC-11(12):61–70, January 1981.

[71] S. Snapp and J. Brentano. Dids (distributed intrusion detection system): Motivation, architecture and an early prototype. In *Proceedings of the 14th NIST-NCSC National Conference on Computer Security*, Washintong, D.C., Oct., 1991.

[72] W. Stallings and L. Brown. *Computer Security: Principles and Practice*. Person Prentice Hall, 2008.

[73] Christina Theocharopoulou, I. Partsakoulakis, George A. Vouros, and Kostas Stergiou. Overlay network for task allocation and coordination in dynamic large-scale networks of cooperative agents. In *Proceedings of AAMAS'07*, pages 295–302, May 2007.

[74] Jacob W. Ulvila and John E. Gaffney. Evaluation of intrusion detection systems. *Journal of Research of the National Institute of Standards and Technology*, 108(6):453–473, November 2003.

[75] J. Undercoffer, A. Joshi, and J. Pinkston. Modeling computer attacks: An ontology for intrusion detection. *LNCS*, 2820/2003:113–135, 2003.

[76] G. Vigna, S. Eckmann, and R. Kemmerer. Attack languages. In *Proceedings of the IEEE Information Survivability Workshop*, 2000.

[77] A. Vorobiev and J. Han. Security attack ontology for web services. In *Proceedings of the Second International Conference on Semantics, Knowledge, and Grid*, pages 42–47, Guilin, Guangxi, China, Oct. 2006.

[78] W3C. Extensible markup language. *http://www.w3c.org/XML/*, 2009.

[79] X. Wang, J. Zheng, K. Xiao, X. Xue, and CK Toh. A mobile agent-based p2p model for autonomous security hole discovery. In *Proceedings of the Fifth International Conference on Computer and Information Technology*, pages 723–727, Sep. 2005.

[80] M. D. Weerdt, Y. Zhang, and T. Klos. Distributed task allocation in social networks. In *Proceedings of AAMAS'07*, pages 500–507, Honolulu, Hawaii, USA, May 2007.

[81] J. Wu, C. Wang, J. Wang, and S. Chen. Dynamic hierarchical distributed intrusion detection system based on multi-agent system. In *Proceedings of IEEE/WIC/ACM International Conference on Web Intelligence and International Agent Technology Workshops*, pages 89–93, Dec. 2006.

[82] K. Xiao, J. Zheng, X. Wang, and X. Xue. A novel peer-to-peer intrusion detection system using mobile agents in manets. In *Proceedings of the Sixth International*

Conference on Parallel and Distributed Computing, Applications and Technologies, pages 441–445, Dec. 2005.

[83] B. Yang and H. Garcia-Molina. Improving search in peer-to-peer networks. In *Proceedings of ICDCS*, pages 5–14, Washington, DC, USA, 2002.

[84] R. Zhang, D. Qian, C. Ba, W. Wu, and X. Guo. Multi-agent based intrusion detection architecture. In *Proceedings on the 2001 International Conference on Computer Networks and Mobile Computing*, pages 494–501, Oct. 2001.

[85] Xiaoming Zheng and Sven Koenig. Reaction functions for task allocation to cooperative agents. In *Proceedings of AAMAS'08*, pages 559–566, Estoril, Portugal, May 2008.