ACKNOWLEDGEMENTS

The author would like to thank her thesis advisor Dr. Jose, M. Vidal for his constant guidance, advice and encouragement. She is grateful to him for providing invaluable ideas and insights. She owes a huge debt of gratitude to him for always having time to answer her queries and doubts.

Also, the author would like to thank her committee members Dr. Michael, N. Huhns and Dr. Csilla, Farkas for their sound advice and helpful discussions.

Finally, the author would like to thank her parents and sister for their love and encouragement that fuel all of her endeavors.
ABSTRACT

Distributed Internet-based attacks on computer systems are becoming more prevalent. These attacks usually employ some form of automation and involve the compromise of many systems across the Internet, systems which are not necessarily owned by the same company or individual. The information needed to detect and neutralize these attacks is spread across many machines. A system administrator who wishes to detect and confirm these distributed attacks must constantly monitor his systems and communicate with other administrators around the world, a challenging task. This dissertation study presents the design and implementation of a multiagent system, built using FIPA-OS, in which agents responsible for different network realms communicate with each other in order to determine if certain suspicious events are actually part of a distributed attack, and to warn each other about possible threats.
# TABLE OF CONTENTS

ACKNOWLEDGEMENTS ii

ABSTRACT iii

LIST OF TABLES vii

LIST OF FIGURES viii

CHAPTER 1. INTRODUCTION 1
  1.1 Motivation 1
  1.2 Road Map 2

CHAPTER 2. BACKGROUND AND RELATED TECHNOLOGIES 4
  2.1 Background 4
    2.1.1 DIDS 5
    2.1.2 AAFID 5
    2.1.3 EMERALD 6
    2.1.4 CIDF 6
    2.1.5 Need for Open Security Protocol 7
  2.2 Related Technologies 8
    2.2.1 FIPA 8
    2.2.2 FIPA ACL 11
    2.2.3 FIPA-OS 14

CHAPTER 3. ARCHITECTURE 19
  3.1 Methodology 19
  3.2 System Design 20
CHAPTER 4. COMPONENTS OF THE SYSTEM

4.1 Events
4.2 Tasks
4.3 Conversations
4.4 FIPA-OS Agent

CHAPTER 5. IMPLEMENTATION

5.1 Overview
5.2 Database setup
5.3 System Implementation

CHAPTER 6. INTERACTION PROTOCOLS

6.1 Alert Protocol
6.2 Denial Of Service Alert Protocol
6.3 Query Suspicious Login Protocol
6.4 Query Echo Storm Protocol
6.5 Ignore Request Protocol
6.6 Quit Request Protocol
6.7 Query Illegal Process Protocol

CHAPTER 7. TEST RESULTS

7.1 Test Cases
7.2 Testing Denial of Service Alert Protocol
  7.2.1 Testcase 01
7.3 Testing Query Suspicious Login Protocol
  7.3.1 Testcase 01
  7.3.2 Testcase 02
  7.3.3 Testcase 03
  7.3.4 Testcase 04
7.4 Testing Query Illegal Process Protocol
  7.4.1 Testcase 01
7.5 Testing timeouts on Protocols
  7.5.1 Testcase 01
LIST OF TABLES

Table 2.1. FIPA Communicative Acts 12
Table 2.2. Pre-defined FIPA Message Parameters 13
Table 4.1. Unusual entries within log files 25
Table 4.2. Events associated with each classification of intrusions 26
Table 5.1. Event Table Schema 33
Table 5.2. Illegal Connection Event Schema 34
Table 5.3. Echostorm Event Table Schema 35
Table 5.4. Ignore Request Event Schema 36
Table 5.5. Illegal Process Event Schema 36
Table 5.6. Login Event Schema 37
Table 5.7. Too Many Packets Event Schema 37
LIST OF FIGURES

Figure 2.1. Agent Management Reference Model 9
Figure 2.2. Message Transport Reference Model 11
Figure 2.3. Architecture of the FIPA-OS 15
Figure 2.4. Logical Composition of MTS 17
Figure 3.1. Architecture of the current(MNSS) prototype system 23
Figure 5.1. Example FIPA complaint conversation between agents A and B 39
Figure 6.1. Alert Protocol. 42
Figure 6.2. Query Suspicious Login Protocol 47
Figure 6.3. Illustration of Echo Storm Attack 49
Figure 6.4. Query Echo Storm Protocol 50
Figure 6.5. Ignore Request Protocol 51
Figure 6.6. Quit Request Protocol 52
Figure 6.7. Query Illegal Process Protocol 54
Figure 7.1. Illustration of Alert Loops 87
CHAPTER 1

Introduction

1.1 Motivation

Security and privacy are growing concerns in the open distributed software systems community because of the Internet’s rapid growth and the desire for secure transactions. This desire has led to the advent of many security architectures and protocols that deal with authentication, cryptography, and authorization. One of the biggest risks to Internet survivability is the growing number of distributed and automated attacks by malicious intruders. The security industry has so far concentrated solely on the development of automated security programs that analyze the attacks within a single isolated system. These programs never use the Internet as a communication medium except when downloading updates from the central server---usually owned by the same company that produced the program. Security consortiums, on the other hand, concentrate on the publication of security alerts aimed at system administrators. None of these approaches manages to leverage the distributed automated nature of the Internet to serve as a vehicle for its own survival. Meanwhile more and more security incidents consist of a large series of widely distributed exploits, involving numerous systems, networks, operating systems and applications. Intruders often compromise multiple systems when they attack a target
site. At each compromised system, there may be signs of intrusive activities that agents of the respective systems discover. By gathering information from those systems (from agents of those systems), we can determine the widespread nature of attacks against our networked systems. It is also possible that a system may have been compromised and is serving as an unwitting participant in large-scale attacks against several sites. External contacts assist us in security monitoring, greatly extending our ability to detect intrusions. Therefore, the need arises for systems to cooperate with each other in order to manage such diverse attacks across networks and time.

The implemented Multiagent Network Security System (MNSS) represents a security framework in distributed systems where an intelligent agent handles the security monitoring at each host. The agents are made responsible for alerting the system administrators about an attempted intrusion or misuse for a particular system. As such, the framework automates the task of distributed intrusion detection while minimizing the amount of human intervention needed. The long-term goals of this research are to develop and standardize all the information formats and all the interaction protocols needed for effective communication among cooperative distributed intrusion detection agents.

1.2 Road Map

The design and implementation of the MNSS system will be presented in the following chapters. Chapter 2 gives a brief overview of the related work and technologies. Architectural details of MNSS System are presented in Chapter 3. The components of
the system and implementation details of the MNSS system are given in Chapter 4 and
Chapter 5 respectively. Chapter 6 presents the interaction protocols developed to handle
suspicious events. Chapter 7 provides the conclusions and directions for future research
efforts.
CHAPTER 2

Background and Related Technologies

2.1 Background

An intrusion is defined as any set of actions that attempt to compromise the integrity, confidentiality or availability of a resource [Ste02]. Intrusion detection System (IDS) is a type of security management system for computers and networks. An Intrusion Detection system gathers and analyzes information from various areas within a computer or a network to identify possible security breaches, which include both intrusions (attacks from outside the organization) and misuse (attacks from within the organization). An IDS does not attempt to stop an intrusion as it occurs. Its role it to alert a system security officer that a potential security violation is taking place. Most of the commercial IDS products are based on centralized and single host monitoring. Hence they lack the ability to detect distributed attacks. However, a number of efforts have been made to come up with an effective distributed Intrusion Detection System.
2.1.1 DIDS

The Distributed Intrusion Detection System (DIDS) [Sna91] architecture combines distributed monitoring and data reduction with centralized analysis to monitor a heterogeneous network of computers. The DIDS components include the DIDS director, a single host monitor per host and a single LAN monitor for each LAN segment of the monitored network. The host and LAN monitors report the evidence of unauthorized or suspicious activity, while the DIDS director is primarily responsible for its evaluation. It also includes a host agent which handles all the communications between each of the host and LAN monitors and the DIDS director. The communications are based on the ISO Communication Management Information Protocol (CMIP) [Wil93], but these communications are limited to occur between the components of the DIDS system.

2.1.2 AAFID

The Autonomous Agents For Intrusion Detection (AAFID) [Jai98] system uses agents for data collection and analysis. Each host can contain any number of agents that monitor for interesting events occurring in the host. All the agents in a host report their results to a single transceiver. Transceivers perform some data reduction and report their results to one or more monitors. The agent does not have authority to directly generate an alarm. By combining reports from different agents, transceivers build a model of the status of their host, and monitors build a model of the status of the network they are monitoring. In AAFID architecture, agents do not communicate directly with each other. Instead they
send all their messages to higher level entities, namely transceivers where centralized analysis of those messages is performed.

2.1.3 EMERALD

In the EMERALD [Phi97] system agent technology has not been used. The architecture is hierarchical in which service monitors are deployed within a domain to perform analysis on network infrastructure, network services and logs. It includes domain-wide analysis covering misuse which is viable across multiple services and components; and enterprise-wide analysis covering multiple domains. The objective of the service analysis is to streamline and decentralize the surveillance of a domain’s network interfaces for activity that indicate the misuse or significant anomalies in operation. Information correlated by a service monitor is disseminated to other EMERALD monitors through an asynchronous communication model that is generally referred to as subscription based message passing. EMERALD’s client monitors are able to subscribe to receive the analyzed results produced by service monitors. Service monitors in turn disseminates the analyzed results asynchronously to the client subscribers. However the communication scheme is hierarchical, limiting access to the events or results between adjacent levels of hierarchy within their framework.

2.1.4 CIDF

All the above discussed systems focus on the development of a hierarchical framework for distributed intrusion detection and their communication models are limited to operate
within their IDS. Very little work has been focused on the nature of communication mechanism among different implementations of Intrusion Detection Systems. One notable exception is the Common Intrusion Detection Framework (CIDF) [Sta98] which has proposed some interfaces and languages to allow intrusion detection components to request information from other components. It consists of four discrete components namely Event generators (E-boxes), Event analyzers (A-boxes), Event databases (D-boxes) and Response Units (R-boxes). All four kinds of components exchange data in form of generalized intrusion detection objects (gidos) which are represented in a common standard format defined by Common Intrusion Specification Language (CISL) [Jon99]. A gido encodes the fact that an occurrence happened at particular time, or a conclusion about some events, or an instruction to carry out an action. However, this framework does not enable any automatic or intelligent cooperation among software agents.

2.1.5 Need for Open Security Protocol (Article)

For automated co-operation to happen there is a need for well defined domain languages and interaction protocols as explained in an article about Biological Network Security on SecurityFocus.com [Web05] which discusses the need for open and common mechanism for communicating among different security services.

The above projects and articles provided motivation to extend the current research on Distributed Intrusion Detection. The key effort is directed towards designing standard
event languages for describing suspicious events and vulnerabilities and also interaction protocols to handle inter-agent communications.

2.2 Related Technologies

2.2.1 FIPA

The Foundation for Intelligent Physical Agents (FIPA) [Web03], based at Geneva, Switzerland, is the leading standards-setting body for software agents and agent communication language. It is a non profit organization promoting the technologies and interoperability specifications that facilitate the end-to-end inter-working of intelligent agent systems in modern commercial and industrial settings. A FIPA compliant agent behavior is based on Belief, Desires and Intentions (BDI) architecture [Ger99]. The main responsibility of FIPA, on which this work is dependent, is in the area of specifying and maintaining an agent communication language, along with libraries of predefined communicative acts, interaction protocols and content languages.

The architecture of the FIPA agent platform is specified in the normative specifications “Agent Management Specification” [Web05]. FIPA agent platform provides the infrastructure in which agents can be deployed. It establishes the logical reference model for the creation, registration, location, communication, migration and retirement of agents.
The agent management reference model, shown in Figure 2.1, consists of the following logical components, each representing a capability set:

- An **Agent** is the fundamental actor in a domain which combines one or more service capabilities and may include access to external software, human users and communications facilities. An agent must have at least one owner and may support several notions of identity. For example, an Agent Identifier (AID) labels agent so that it may be distinguished unambiguously within the Agent Universe.

- An **Agent Platform (AP)** provides an infrastructure in which agents can be deployed. An agent must be registered on a platform in order to interact with other agents on that platform or indeed any other platforms. An AP consists of three capability sets Agent Communication Channel, Agent Management System and default Directory Facilitator.
- **Directory Facilitator (DF)** is a mandatory component of the AP. The DF provides yellow page service to other agents. Agents may register their services with the DF or query the DF to find out the services offered by other agents. Multiple DF’s may exist within an AP and may be federated.

- An **Agent Management System (AMS)** is a mandatory component of the AP which manages the creation, deletion and authentication of agents and provides a white page directory service for the agents that reside on the platform. The AMS exerts supervisory control over access to and use of the AP. Only one AMS will exist in a single AP. The AMS presents a directory of AIDs which contain transport addresses for agents registered with the AP. It stores the mapping between globally unique agent names (or GUID) and local transport addresses used by the platform. Each agent must register with an AMS in order to get a valid AID.

- An **Agent Communication Channel (ACC)** is an agent which uses information provided by the Agent Management System to route messages between agents within the platform and to agents resident on other platforms. All agents have access to at least one ACC. The ACC is the default communication method that connects all agents within an AP and between AP’s.

- A **Message Transport Service (MTS)** is the default communication method between agents of different APs. Detailed description can be found in [Web09] and a higher level representation of MTS is given in Figure 2.2.

- **Software** describes all non-agent, executable collection of instructions accessible through an agent. Agent may access software, for example, to add new services,
acquire new communication protocols, acquire new security protocols/algorithms, acquire new negotiations protocols, access tools which support migration etc.

Figure 2.2. Message Transport Reference Model

This is very brief and high-level description of the FIPA’s Agent architecture. A more detailed description can be found in [Web06].

2.2.2 FIPA ACL

The FIPA Agent Communication Language (ACL) [Web07] was designed as a language for inter-agent communication. It specifies the higher level communicative acts(called
FIPA performatives) as well as semantics to be implemented by a FIPA compliant agent.

The communicative acts denote the actions related to the communication.

The message structure in the FIPA complaint conversation comprises of

- A single FIPA communicative act (performative)
- One or more parameters

Comprehensive list of communicative acts included in FIPA are shown in Table 2.1.

**Table 2.1. FIPA Communicative Acts**

<table>
<thead>
<tr>
<th>Accept-Proposal</th>
<th>Agree</th>
<th>Cancel</th>
<th>Cfp</th>
</tr>
</thead>
<tbody>
<tr>
<td>Confirm</td>
<td>Disconfirm</td>
<td>Failure</td>
<td>Inform</td>
</tr>
<tr>
<td>Inform-If</td>
<td>Not-Understood</td>
<td>Propagate</td>
<td>Propose</td>
</tr>
<tr>
<td>Proxy</td>
<td>Query-if</td>
<td>Query-Ref</td>
<td>Refuse</td>
</tr>
<tr>
<td>Reject Proposal</td>
<td>Request</td>
<td>Request-When</td>
<td>Subscribe</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Request-Whenever</td>
</tr>
</tbody>
</table>
Comprehensive list of message parameters included in FIPA are shown in Table 2.2.

**Table 2.2. Pre-defined FIPA Message Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>: sender</td>
<td>Denotes the identity of the sender of the message.</td>
</tr>
<tr>
<td>: receiver</td>
<td>Denotes the identity of the intended recipient of the message.</td>
</tr>
<tr>
<td>: content</td>
<td>Denotes the content of the message; equivalently denotes the object of the action.</td>
</tr>
<tr>
<td>: reply-with</td>
<td>Introduces an expression which will be used by the agent responding to this message to identify the original message.</td>
</tr>
<tr>
<td>: in-reply-to</td>
<td>Denotes an expression that references an earlier action to which this message is a reply.</td>
</tr>
<tr>
<td>: envelope</td>
<td>Denotes an expression that provides useful information about the message as seen by the message transport service.</td>
</tr>
<tr>
<td>: language</td>
<td>Denotes the encoding scheme of the content of the action.</td>
</tr>
<tr>
<td>: ontology</td>
<td>Denotes the ontology which is used to give a meaning to the symbols in the content expression.</td>
</tr>
<tr>
<td>: reply-by</td>
<td>Denotes a time and/or date expression which indicates a guideline on the latest time by which the sending agent would like a reply.</td>
</tr>
<tr>
<td>: protocol</td>
<td>Introduces an identifier which denotes the protocol which the sending agent is employing. The protocol serves to give additional context for the interpretation of the message.</td>
</tr>
<tr>
<td>: conversation-id</td>
<td>Introduces an expression which is used to identify an ongoing sequence of communicative acts which together form a conversation.</td>
</tr>
</tbody>
</table>
Actions associated with some communicative acts which are extensively used in this work are as follows:

- *Inform*: Informs the recipient about the proposition which the sender believes is true.
- *Query-if*: Asks the recipient for the state of the given proposition
- *Request*: Asks the recipient to execute a given action

These acts occupy the head of a FIPA ACL message. They only specify the action that agent should take on the content of the message. The processing or reasoning about the message is left to the agent. The actual content of the message can be expressed in any language.

### 2.2.3 FIPA-OS

FIPA Open Source (FIPA-OS) [Web04], originating from Nortel Networks, is an agent platform that implements the FIPA standards. The platform supports communication between multiple agents using an agent communication language that conforms to the FIPA (Foundation for Intelligent Physical Agents [Web03]) agent standards. We chose FIPA-OS platform because our agents reside on multiple platforms and exhibit different behaviors. FIPA-OS can interoperate with other heterogeneous FIPA compliant platforms (e.g. JADE [Bel99]).
High-level Architecture

The architecture of FIPA-OS is shown in Figure 2.3. As it is referred in [Web04], FIPA-OS toolkit uses

- mandatory components (i.e. components required by ALL FIPA-OS Agents to execute)
- optional components
- components with switchable implementations

Mandatory Components

- The Task Manager (TM) provides the ability to split the functionality of the agent into smaller, disjoint units of works known as Tasks. Tasks are self-contained pieces of code that carry out some task and (optionally) return a result. They have the ability to send and receive messages, and have little or preferably no dependence on the agent they are executed within. This provides a number of benefits which can be found in [Web08].

![Figure 2.3. Architecture of the FIPA-OS](image)
- The Conversation Manager (CM) provides the ability to track conversation, as well as mechanisms for grouping messages of the same conversation together. The CM ensures that the protocol is being followed by both the Agents.

- The Message Transport Service (MTS) provides the ability to send and receive messages. It is logically split such that incoming and outgoing messages pass through a number of services within a “service stack”. Each service is a standalone component that performs some transformation on outgoing messages and inverse transformation on incoming messages. This logical composition of the MTS is shown in Figure 2.4.

- The Agent Shell in general provides following functionality:
  - sending messages
  - retrieving the Agents’ properties (profiles, AID, state) and locating platform Agents (DF and AMS)
  - registration with platform Agents
  - setting up Task’s
  - shutting down the Agent
  - initializes MTS, TM and CM components.
Optional Components

The Database Factory, Parser Factory and Choice Constraint Language (CCL) components are optional and do not have an explicit relationship with the other components within the toolkit. The Database Factory provides a simple mechanism for interacting with a number of database implementations in an object-oriented manner using a consistent interface. The Parser Factory component is designed to enable the content of a message to be expressed in a form containing just its semantic information, without concerning agent developers with the actual encoding syntax.

Switchable Implementations

The switchable implementations included as part of the FIPA-OS distribution for
each component include:

Message Transport Protocol

- Remote Method Invocation (RMI) (proprietary)
- Internet Inter-ORB Protocol (IIOP)

Database

- Memory Database
- Serialization Database

Parsers

- Semantic Language (SL)
- Agent Communication Language (ACL)
- Extensible Markup Language (XML)
- Resource Definition Framework (RDF)
CHAPTER 3

Architecture

3.1 Methodology

This application provides the capability to collect intrusion specific information from co-hosts through agents. Distributed attacks often leave trace information in log files, audit files, and processes left behind by an intruder. This trace information is used to search for suspicious events or connections that require further investigation. There are software packages (like TCP/IP daemon wrapper package [Web09]), which inspect the logging information and detect signs of intrusion. Once alerted by the intrusion detection software that an intrusion has been detected, we need to analyze that intrusion by investigating to what extent our systems or data have been compromised. We then respond to that intrusion based on the results of the analysis.

This analysis is carried out by the host agents and tries to answer the following questions:

- Which attacks are used to gain access?
- Which systems and data did the intruder access?
- What did the intruder do after obtaining access?
Since all intrusions differ from each other in many ways, agents use different analysis mechanisms for dealing with different types of intrusions. During the analysis, it is important for the agent to keep communication with other agents about an intrusion. These communications will help us gain additional information to protect our own systems.

An agent may receive an alert from other agents about some unexpected behavior due to an intruder. The agent can use this information to take preemptive or reactive measures. An agent can notify other agents about the evidence of attacks against its system originating from their systems, and vice versa. On the other hand, if an agent receives notification that an attack is being launched from its system the agent considers the possibility that its system may have been compromised by the intruder to hide his tracks. The agent will further investigate it and alert the system administrator if it finds any confirmed intrusion. During its investigation, an agent can request different events and alerts from other agents. At the same time, being co-operative, it responds to the requests for events or alerts from other agents. Consequently, the agent ends up sharing information with many agents for single possible intrusion detection.

3.2 System Design

A distributed communication model can be either clients-server or peer-peer model as explained in [Raj01]. While the client-server model deals with two participating entities, the peer-to-peer model deals with many peer entities.
The peer-to-peer model can be further classified into

- Event based model – In this model, any entity may produce events and any entity may consume events. Since entities are loosely connected, this model leads to higher scalability.

- Push based model – In this model, some entities are meant to strictly produce entities and some entities are meant to strictly consume entities. Since entities are tightly coupled, this model lacks flexibility.

This work involves the design and implementation of an **Event-based** security framework and provides a service of retrieving the information from a distributed network. This information is then used to detect intrusions on hosts. The architecture of the project is shown in Figure 3.1. In our framework each domain is represented by an agent. All computers in a domain submit their log files to a central database. We assume that these log files are inspected by an intrusion detection software which reports any suspicious events to the agent in its domain. Upon receiving suspicious events, an agent starts its analysis by actively communicating with agents of other domains. The results of the conversations with other agents are stored back as alerts or new events in the database.
In the context of our application, we adopt the following definitions from [Raj01]:

**Event**: An *event* is defined as any data that has not been analyzed.

**Alert**: An *alert* is defined as the result of analyzing a set of events that indicates some potential intrusive activity.

**Agent**: An *agent* is defined as an encapsulated software entity with its own state, behavior, thread of control, and ability to interact and communicate with other entities – including people, other agents, and systems. In our approach,

- the agent is autonomous in the sense that it capable of acting without direct external intervention; has some degree of control over its internal state and actions based on its own experiences.
- The agent is interactive in the sense that it communicates with the environment and other agents.
- The agent is intelligent in the sense that its state is formalized by knowledge (beliefs, goals, plans, assumptions) and interacts with other agents in a specific language.
- More importantly, the agent is cooperative and able to coordinate with other agents to achieve a common purpose.

The agents are implemented using FIPA-OS and communicate among themselves using FIPA-ACL.
Figure 3.1. Architecture of the current prototype system. Log files are inspected by Log Analyzer (LA). Any suspicious events become goals for the agent to act on. The responses received out of the conversations with other agents are stored back as alerts or new events in the database.
CHAPTER 4

Components of the System

4.1 Events

Suspicious acts are events which, while not a violation of policy, are of interest to an IDS. For example, commands which provide information about the state of the system may be suspicious. It is assumed that log files of every host are inspected by some software mechanisms at some predetermined time interval and any suspicious / unusual entries are documented. Some of the suspicious / unusual entries associated with respective log files are shown in Table 4.1.

The primary aim is to find out if these unusual entries could lead to any confirmed intrusion for which, additional information about that unusual entry is needed. Intrusions are classified based on the common characteristics of the intrusions - particularly the way the intrusion occurs and how it effects the target host. Each Event is defined in such a way to accommodate all the data related to the classification it is representing. Table 4.2 summarizes some suspicious events associated with each classification of intrusions. We employ some such classifications of characteristics (of intrusions) which includes some
well known attacks and derived a comprehensive set of events associated with those classifications.

<table>
<thead>
<tr>
<th>Type of Log File</th>
<th>Unusual Entries</th>
</tr>
</thead>
<tbody>
<tr>
<td>User Activity</td>
<td>Repeated Failed Logins.</td>
</tr>
<tr>
<td></td>
<td>Logins from unusual locations.</td>
</tr>
<tr>
<td></td>
<td>Unusual processes run by the user.</td>
</tr>
<tr>
<td></td>
<td>Unauthorized access to files.</td>
</tr>
<tr>
<td></td>
<td>Logins at unusual times.</td>
</tr>
<tr>
<td>Network connections</td>
<td>Connections from unusual locations.</td>
</tr>
<tr>
<td></td>
<td>Connections to unusual locations.</td>
</tr>
<tr>
<td>Web server activity</td>
<td>Repeated attempts to misuse the server.</td>
</tr>
<tr>
<td></td>
<td>Flooding activities that could cause a denial service problem.</td>
</tr>
<tr>
<td>Network traffic monitoring</td>
<td>Half open connections</td>
</tr>
<tr>
<td>Systems activity</td>
<td>Unexpected shutdowns.</td>
</tr>
<tr>
<td></td>
<td>Unexpected reboots</td>
</tr>
</tbody>
</table>
Table 4.2. Events associated with each classification of intrusions

<table>
<thead>
<tr>
<th>Type of Event</th>
<th>Classification of characteristics of Intrusions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Invalid or suspicious Login</td>
<td>Logins not logged for an abnormal length of time.</td>
</tr>
<tr>
<td>Event</td>
<td>Logins at unusual times.</td>
</tr>
<tr>
<td></td>
<td>Short login times.</td>
</tr>
<tr>
<td></td>
<td>Logins from unexpected locations.</td>
</tr>
<tr>
<td></td>
<td>Failed login attempts.</td>
</tr>
<tr>
<td>Illegal or Abnormal connection Event</td>
<td>Connections from / to unusual locations.</td>
</tr>
<tr>
<td></td>
<td>Half open connections.</td>
</tr>
<tr>
<td></td>
<td>Sudden spike in network traffic.</td>
</tr>
<tr>
<td></td>
<td>Telnet connections without output from w or who commands.</td>
</tr>
<tr>
<td>Unusual or Suspicious process Event</td>
<td>Processes that take a long time.</td>
</tr>
<tr>
<td></td>
<td>Processes with unusual start times.</td>
</tr>
<tr>
<td></td>
<td>Processes with high % of CPU time (a sniffer).</td>
</tr>
<tr>
<td></td>
<td>Processes without a controlling terminal.</td>
</tr>
<tr>
<td></td>
<td>Processes with unusual names.</td>
</tr>
<tr>
<td></td>
<td>Large number of processes at a time.</td>
</tr>
<tr>
<td>Suspicious or Unauthorized modification (of files) Event</td>
<td>Unexpected changes to password files or access control lists.</td>
</tr>
<tr>
<td></td>
<td>Unexpected size of file (may be Trojan).</td>
</tr>
<tr>
<td></td>
<td>Unfamiliar files.</td>
</tr>
<tr>
<td></td>
<td>System files that appear to have been modified recently.</td>
</tr>
<tr>
<td></td>
<td>Short systems files indicating that file has been edited or deleted.</td>
</tr>
</tbody>
</table>
### 4.2 Tasks

The Task Manager (part of the FIPA-OS architecture) provides the ability to split the functionality of an Agent into smaller, disjoint units of works known as Tasks. FIPA-OS defines a *Task* class as “self contained piece of code that carry out some tasks and optionally return a result, have the ability to send and receive the messages, and have little or preferably no dependence on the agent they are executed within”.

This provides many benefits:

- Tasks are highly re-usable - they can be used in many agents without having to re-write the same code / functionality.
- It is easy to debug, since tracking the flow of control is simple. Tracking can be done with debugging messages. Debugging messages were provided to indicate when task-interactions fail/are unhandled.
- An Agent can execute multiple Tasks at once – the Task Manager takes care of routing incoming messages and other events to the right Tasks.
• Conversation state is effectively encapsulated within a Task, reducing the manual tracking of Conversations to a bare minimum.
• Tasks can spawn child-tasks – this enables complex Task’s to be created through simply utilizing simpler Tasks within them.

In FIPA-OS Tasks are managed by Task Manager (TM). The entire TM component is event based. Every Task within TM has its own queue for messages. The TM processes the messages in the order they are generated for a particular Task. Tasks are based on event-based processing implying that events are delivered by dynamic invocation of various methods. For example, once a Task has been initialized, its startTask() method is invoked (the default implementation for this method provided by the Task super-class does nothing). This is where the main functionality of the Task should be initiated or carried out. Tasks can also be associated with a number of Agent conversations, ensuring that conversation updates are propagated to these Tasks. When a conversation update occurs, TM invokes the listener method handleX(Conversation conv), where X is the performative of the received message. In the case that no such method is defined within the handling Task, it invokes the listener method handleOther() - the default implementation replies to all messages with a “not-understood” response unless this method is overridden. More on the FIPA-OS Tasks can be found in FIPA-OS Developer’s Guide [Web08].
4.3 Conversations

FIPA-OS defines a conversation as “an instance of FIPA interaction protocol”. An interaction protocol defines a specific pattern of messages. Messages are generally sent as part of the conversation and follow the pattern defined in the interaction protocol. If messages sent are part of a conversation, and when a task sends a message, the conversation that message is part of is bound to that task. If messages adhere to an interaction protocol then a certain message order is reinforced. The Conversation Manager (part of the FIPA-OS architecture) provides the ability to track conversation state at the performative level, as well as mechanisms for grouping messages of the same conversation together. If a conversation is specified as following a particular protocol (FIPA or user-defined), the Conversation Manager will ensure that the protocol is being followed by both participants involved in the conversation.

4.4 FIPA-OS Agent

The FIPAOSAgent class (included in FIPA-OS) provides a shell for Agent implementation that can extend this class. The FIPAOSAgent shell is responsible for loading an Agent’s profile, and initializing the other components of which the Agent is composed.

Functionality provided by the FIPA-OS Agent Shell:

- Initialization – It creates these mandatory components in the order given below:
  - MTS (Message Transport Service)
  - Task Manager
○ Conversation Manager

These three components were discussed in section 2.2.3 of Chapter 2. At initialization of the Conversation Manager, references to the Message Transfer Service and Task Manager are passed to enable them to be dynamically bound to the CM. The FIPAOSAgent registers itself with Message Transport Service, Task Manager and Conversation Manager in the initialization phase. This initialization eases the use of services included in the FIPA-OS.

- Sending messages – This is accomplished by using the `forward()` method in either the FIPAOSAgent class or Task class depending on where in the agent implementation the message is being sent from. This method can be used to send messages to other agent in same AP or other AP. Agent shell takes the responsibility of transporting the messages to other agents via Conversation Manager and Message Transport Service.

- Retrieving the other Agents properties (profile, AID and state) and locating Platform Agents (AMS and DF) - Agent can get description information about other agents resident in the AP with numerous FIPAOSAgent methods. For example `searchDF()` method searches all agents that have certain description.

- Registration with Platform Agents - Agent itself can register with Agent Platform’s DF and AMS (FIPA standard requires agent to register with AMS to get valid agent ID) by using FIPAOSAgent methods `registerWithAMS()` and `registerWithDF()` respectively.

- Setting up Tasks – The FIPAOSAgent class provides direct access to the TaskManager and its associated Task methods.
• Shutting down the Agent – The Agents and its components can be cleanly shutdown by invoking the `shutdown()` method in the FIPAOSAgent class. This in turn invokes the `shutdown()` method of all the components of the Agent.
CHAPTER 5

Implementation

5.1 Overview

Given a suspicious event, an agent needs to analyze the abnormalities associated with that event and check if it belongs to the sequence of events that results in an intrusion. Each suspicious event is handled by a reusable Task class which is developed independent of the agents and types of intrusions agents are meant to deal with. Agents are provided with a variety of tasks and they gather information from other agents by invoking appropriate tasks.

5.2 Database Setup

The log files are stored as tables in the database. An audit record in the log file is considered as an Event. As we have already discussed in Section 4.1, there are a number of distinct Event types, which are known apriori, classified by the type of intrusion. Different types of Events are stored in different tables in the database. Events are tagged with data and can have any number of tag fields. The exact number and nature of the fields may be dependent on the type of the Event. Each Event can have a different but
Table 5.1. Event Table Schema

<table>
<thead>
<tr>
<th>Field</th>
<th>Null ?</th>
<th>Type</th>
<th>Size</th>
<th>Key</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EventID</td>
<td>Not Null</td>
<td>Varchar2</td>
<td>25</td>
<td>Primary Key</td>
<td>Unique ID to distinguish each event record.</td>
</tr>
<tr>
<td>TypeOf Event</td>
<td>Null OK</td>
<td>Number</td>
<td>3</td>
<td>-</td>
<td>Used to identify the specific Event from which we have to read the additional information about this Event.</td>
</tr>
<tr>
<td>Source Host Name</td>
<td>Null OK</td>
<td>Varchar2</td>
<td>64</td>
<td>-</td>
<td>Host name or the IP address of the source machine of the connection</td>
</tr>
<tr>
<td>Source Port</td>
<td>Null OK</td>
<td>Varchar2</td>
<td>5</td>
<td>-</td>
<td>Port number associated with connection on the source machine</td>
</tr>
<tr>
<td>Dest Host Name</td>
<td>Null OK</td>
<td>Varchar2</td>
<td>64</td>
<td>-</td>
<td>Host name or the IP address of the destination machine of the connection</td>
</tr>
<tr>
<td>Dest Port</td>
<td>Null OK</td>
<td>Varchar2</td>
<td>5</td>
<td>-</td>
<td>Port number associated with connection on the destination machine</td>
</tr>
<tr>
<td>Event Date</td>
<td>Not Null</td>
<td>Varchar2</td>
<td>10</td>
<td>-</td>
<td>Date on which the connection has been established.</td>
</tr>
<tr>
<td>Event Time</td>
<td>Not Null</td>
<td>Varchar2</td>
<td>25</td>
<td>-</td>
<td>Time when the connection has been established</td>
</tr>
<tr>
<td>User Name</td>
<td>Null OK</td>
<td>Varchar2</td>
<td>20</td>
<td>-</td>
<td>Username associated with the connection</td>
</tr>
<tr>
<td>Done Flag</td>
<td>Not Null</td>
<td>Varchar2</td>
<td>20</td>
<td>-</td>
<td>A flag to indicate the status of the event</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>a) 0 - not yet handled</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>b) 1 - is done and handled successfully</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>c) 2 - requires Sys Admin attention</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>d) 3 - cannot be handled by any Task class</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>e) 4 - incomplete, Task timed out</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>f) 5 - incomplete, cannot get information required</td>
</tr>
<tr>
<td>Comments</td>
<td>Null OK</td>
<td>Varchar2</td>
<td>64</td>
<td>-</td>
<td>Any additional comments written by the tasks upon handling the Event</td>
</tr>
<tr>
<td>Agents Chain</td>
<td>Null OK</td>
<td>Varchar2</td>
<td>64</td>
<td>-</td>
<td>All the agent names appended in the order Event has traversed</td>
</tr>
<tr>
<td>Time Stamp</td>
<td>Null OK</td>
<td>Varchar2</td>
<td>25</td>
<td>-</td>
<td>The timestamp which gets updated with the most recent time whenever it is handled by a task</td>
</tr>
</tbody>
</table>
fixed layout based on its type. All these different types of events have some common fields like ‘source host name’ of the connection, ‘destination host name’ of the connection, ‘date’ and ‘time’ when the connection has been established etc. All the common fields are stored in the generic Event record and the other types of events derive those fields from it. There is a ‘Type Of Event’ field with the Event record that uniquely identifies a specific Event. The database schemas for generic Event and other types of Event records are given in Tables 5.1 – 5.7:

**Table 5.2. Illegal Connection Event Schema**

<table>
<thead>
<tr>
<th>Field</th>
<th>Null?</th>
<th>Type</th>
<th>Size</th>
<th>Key?</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EventID</td>
<td>Not Null</td>
<td>Varchar2</td>
<td>25</td>
<td>Foreign Key References Event (EventID) Table</td>
<td>Unique ID that should match the EventID in the Event Table.</td>
</tr>
<tr>
<td>Alert Field</td>
<td>Null OK</td>
<td>Varchar2</td>
<td>64</td>
<td>-</td>
<td>Alert Name</td>
</tr>
<tr>
<td>Alert Message</td>
<td>Null OK</td>
<td>Varchar2</td>
<td>64</td>
<td>-</td>
<td>A short description on the alert generated.</td>
</tr>
<tr>
<td>TimeStamp</td>
<td>Null OK</td>
<td>Varchar2</td>
<td>25</td>
<td>-</td>
<td>The timestamp which gets updated with the most recent time whenever it is handled by a task</td>
</tr>
</tbody>
</table>
Table 5.3. Echostorm Event Table Schema

<table>
<thead>
<tr>
<th>Field</th>
<th>Null?</th>
<th>Type</th>
<th>Size</th>
<th>Key?</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EventID</td>
<td>Not Null</td>
<td>Varchar2</td>
<td>25</td>
<td>Foreign Key References Event (EventID) Table</td>
<td>Unique ID and should match the EventID in the Event Table.</td>
</tr>
<tr>
<td>IsIllegalProcess</td>
<td>Null OK</td>
<td>Varchar2</td>
<td>1</td>
<td>-</td>
<td>a) 0 – if no illegal process is associated with the connection</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>b) 1 – if an illegal process is associated with the connection</td>
</tr>
<tr>
<td>IsEchoRequest</td>
<td>Null OK</td>
<td>Varchar2</td>
<td>1</td>
<td>-</td>
<td>a) 0 – if there is no Echo Request generated by the host</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>b) 1 – if an Echo Request has been generated by the host</td>
</tr>
<tr>
<td>IsEchoPort</td>
<td>Null OK</td>
<td>Varchar2</td>
<td>1</td>
<td>-</td>
<td>a) 0 – if the port on the host side of the connection is not an echo port</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>b) 1 – if the port on the host side of the connection is an echo port</td>
</tr>
<tr>
<td>TimeStamp</td>
<td>Null OK</td>
<td>Varchar2</td>
<td>25</td>
<td></td>
<td>The timestamp which gets updated with the most recent time whenever it is handled by a task</td>
</tr>
</tbody>
</table>
**Table 5.4. Ignore Request Event Schema**

<table>
<thead>
<tr>
<th>Field</th>
<th>Null?</th>
<th>Type</th>
<th>Size</th>
<th>Key?</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EventID</td>
<td>Not Null</td>
<td>Varchar2</td>
<td>25</td>
<td>Foreign Key References Event (EventID) Table</td>
<td>Unique ID and should match the EventID in the Event Table.</td>
</tr>
<tr>
<td>Request Name</td>
<td>Not Null</td>
<td>Varchar2</td>
<td>64</td>
<td>-</td>
<td>The command name which the agent wants the receiving hosts to stop responding to</td>
</tr>
<tr>
<td>TimeStamp</td>
<td>Null OK</td>
<td>Varchar2</td>
<td>25</td>
<td>-</td>
<td>The timestamp which gets updated with the most recent time whenever it is handled by a task</td>
</tr>
</tbody>
</table>

**Table 5.5. Illegal Process Event Schema**

<table>
<thead>
<tr>
<th>Field</th>
<th>Null?</th>
<th>Type</th>
<th>Size</th>
<th>Key?</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EventID</td>
<td>Not Null</td>
<td>Varchar2</td>
<td>25</td>
<td>Foreign Key References Event (EventID) Table</td>
<td>Unique ID that should match the EventID in the Event Table.</td>
</tr>
<tr>
<td>Alert Field</td>
<td>Null OK</td>
<td>Varchar2</td>
<td>64</td>
<td>-</td>
<td>Alert Name</td>
</tr>
<tr>
<td>Alert Message</td>
<td>Null OK</td>
<td>Varchar2</td>
<td>64</td>
<td>-</td>
<td>A short description on the alert generated.</td>
</tr>
<tr>
<td>TimeStamp</td>
<td>Null OK</td>
<td>Varchar2</td>
<td>25</td>
<td>-</td>
<td>The timestamp which gets updated with the most recent time whenever it is handled by a task</td>
</tr>
</tbody>
</table>
### Table 5.6. Login Event Schema

<table>
<thead>
<tr>
<th>Field</th>
<th>Null?</th>
<th>Type</th>
<th>Size</th>
<th>Key?</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EventID</td>
<td>Not Null</td>
<td>Varchar2</td>
<td>25</td>
<td>Foreign Key</td>
<td>Unique ID that should match the EventID in the Event Table.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>References</td>
<td>Event (EventID) Table</td>
</tr>
<tr>
<td>UserName</td>
<td>Null OK</td>
<td>Varchar2</td>
<td>20</td>
<td>-</td>
<td>Username associated with the connection</td>
</tr>
<tr>
<td>ListOfCommands</td>
<td>Null OK</td>
<td>Varchar2</td>
<td>64</td>
<td>-</td>
<td>The commands executed by the user (used to track the activity of the user)</td>
</tr>
<tr>
<td>TimeStamp</td>
<td>Null OK</td>
<td>Varchar2</td>
<td>25</td>
<td>-</td>
<td>The timestamp which gets updated with the most recent time whenever it is handled by a task</td>
</tr>
</tbody>
</table>

### Table 5.7. Too Many Packets Event Schema

<table>
<thead>
<tr>
<th>Field</th>
<th>Null?</th>
<th>Type</th>
<th>Size</th>
<th>Key?</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EventID</td>
<td>Not Null</td>
<td>Varchar2</td>
<td>25</td>
<td>Foreign Key References Event (EventID) Table</td>
<td>Unique ID that should match the EventID in the Event Table.</td>
</tr>
<tr>
<td>Alert Field</td>
<td>Null OK</td>
<td>Varchar2</td>
<td>64</td>
<td>-</td>
<td>Alert Name</td>
</tr>
<tr>
<td>Alert Message</td>
<td>Null OK</td>
<td>Varchar2</td>
<td>64</td>
<td>-</td>
<td>A short description on the alert generated.</td>
</tr>
<tr>
<td>TimeStamp</td>
<td>Null OK</td>
<td>Varchar2</td>
<td>25</td>
<td>-</td>
<td>The timestamp which gets updated with the most recent time whenever it is handled by a task</td>
</tr>
</tbody>
</table>
5.3 System Implementation

Agent reads an Event record documented in the database and based on the ‘Type of Event’ field, it reads the specific Event record from the corresponding table in the database. After reading the specific Event, the agent passes it to all the registered tasks. The Tasks which can handle this type of Event get executed dynamically. The Task handles the Event by initiating a number of conversations with other agents and updates the Event from the responses of those agents. In the process, it may receive new Alerts or Events. Each Task is meant to follow an interaction protocol to deal with a specific type of Event. Based on the results returned by the tasks, agent may choose to invoke another Task or may conclude that no further investigation is required.

Conversation is comprised of a series of query and reply messages. The Task formulates several types of query messages depending on the type of suspicious event it is handling:

- Can it be explained by an authorized user?
- Can it be explained by known system activity?
- Can it be explained by authorized changes to programs?

Conversations are built using FIPA performatives. The content of the conversation is expressed in XML. All conversations initiated by the Task must bind to the protocol implemented by that Task. Outgoing messages are prevented from being sent if the conversation protocol is not being followed. Default responses are sent to incoming messages not following the protocol correctly. An example conversation between two agents A and B is shown in Figure 5.1. In this example agent A query’s agent B to inform
it whether the user ‘Paul’ is authorized or not using ‘Query-If’ FIPA performative. Agent B can reply that the user is authorized or unauthorized or query not understood using ‘Inform-If’ performative.

```
Query-If ( Sender: Agent-A
  Receiver: Agent-B
  Content: ( Inform-If
    SourceIP Address: 252.23.24.20
    DestinationIP Address: 219.29.27.28
    Date/Time: 19 SEPT 2002
    Username: Paul
    Content: Is user authorized?
    Reply-with: 0001
    Language: FIPA )
Inform-If ( Sender: Agent-B
  Receiver: Agent-A
  Content: ( SourceIP Address: 252.23.24.20
    DestinationIP Address: 219.29.27.28
    Date/Time: 19 SEPT 2002
    Username: Paul
    Content: Authorized / Not authorized / Not Understood
    In-reply to: 0001
    Language: FIPA )
```

Figure 5.1. Example FIPA complaint conversation between agents A and B
Once the Task has finished with all the conversations of the interaction protocol it has implemented, it updates the log files with the new events and alerts it received out of those conversations. If the Task times out by any reason, it marks the event incomplete. From the results obtained from the tasks, agent gathers sufficient information to determine whether the occurrence of suspicious activity leads to any confirmed intrusion. At last all confirmed evidences of intrusion, attempted intrusion are reported to an Internal Security Point of Contact. The user interface provided with this application shows the operations and status reported by the agents of the system. Also agent logs all the communication details (interaction protocol followed, messages sent, messages received) in a separate log file to view the details offline.
CHAPTER 6

Interaction Protocols

Each Task implements an interaction protocol for handling a particular type of event. The functionality of a Task can be further split into smaller, disjoint tasks. The tasks are decomposed into subtasks until a point a reached where they are no longer decomposable, or trivial to implement. Thus the tasks are loosely connected and their roles are symmetric. This leads to higher reuse and scalability as there is very little coupling.

The protocols developed for handling few suspicious events are presented in the following sections, along with examples of intrusion scenarios and discussions on how the protocols help in the detection of those intrusions.

6.1 Alert Protocol

Given the alert message and the host machines to which the alert should be sent, the Alert Protocol sends that alert message to the agents of those host machines. The AUML diagram of the protocol is given in Figure 6.1. Since this protocol can be used to forward
alert messages irrespective of the content of the alert, it is highly re-usable. In fact, it is used in some of the protocols discussed in the following sections.

6.2 Denial Of Service Alert Protocol

A denial of service attack [Chr97] is an incident in which a user or organization is deprived of the services of a resource they would normally expect to have. The most common way of achieving it is through SYN flooding [Kum95]. It works by an attacker sending many TCP (Transmission Control Protocol) connection requests with spoofed source addresses to a victim’s machine. Each request causes the targeted host to instantiate data structures out of a limited pool of resources. Once the target host’s resources are exhausted, no more incoming TCP connections can be established, thus denying further legitimate access. It is important to note that neither outgoing connection attempts nor connections already established are affected by this attack. This means that

Figure 6.1. Alert Protocol. Uses FIPA performative ‘Inform’.
an agent at that host will still be able to send outgoing messages to other agents, even if it is experiencing denial of service attack.

An example of such an attack is the Mitnick Attack [Ste02]. The Mitnick attack uses SYN flooding to disable the capability of a computer to respond to connection requests so the attacker can then masquerade as the attacked computer (if the actual computer responded then he could not pretend to be that computer). For example, suppose there are two systems X and Y which trust each other. An attacker disables system X. He sends a forged SYN packet to system Y supposedly from X. System Y sends SYN/ACK to system X but X cannot respond because of the SYN attack. To then forge system X’s acknowledgement of the SYN/ACK sent from Y and establish the connection, the attacker must know the sequence number generated by Y’s TCP sequence number generator. This sequence number is included in the SYN/ACK send from Y but an attacker will not see this since it was sent to X. The attacker can get around this problem because many sequence number generators are predictable in that they always increment the number by the same amount. This allows the attacker to guess the sequence number, forge an acknowledgement of the SYN/ACK and establish a connection to the system Y posing as trusted system X.

This protocol allows Agent X(agent at system X) to send an alert to all the known agents if it is experiencing denial of service attack. The alert message contains all its current outgoing connections to other hosts. This is particularly useful when we consider the Mitnick attack case where the attacker spoofs the IP address of disabled system. Using
this protocol, the attacked system will send an alert to all the agents including the one at system Y. Upon receiving the alert Agent Y(agent at system Y) realizes that X is under denial of service, and checks whether its current connection from system X correlates with any of the connections in the connections list received from Agent X. If it does not find any match, it resets the connections seem to originate from Agent X. The Denial Of Service Alert Protocol makes use of Alert protocol, discussed in Section 6.1, to send Denial of service alert message to all the known hosts.

6.3 Query Suspicious Login Protocol

A single suspect behavior on a single host in a network may not warrant any serious action. However, repeated suspect behavior across several hosts in a network may indeed suggest an attack, with a response definitely warranted. This would be very difficult for a human to recognize. An example of such an attack is the DoorKnob attack [Lan85] where an attacker gains illegitimate access to one of our systems. He then tries to gain access to several different computers at the external site. Since the intruder tries only a few logins on each machine, the intrusion detection systems on those hosts may not flag the user attack. This suspicious login protocol helps to flag the user by gathering or sharing data among all the involved hosts. The protocol helps to contact the source and destination machines to identify all the different user accounts in the “chain” to find the initial launching point of the attack. The AUML diagram of this interaction protocol is shown in Figure 6.2.
The initiator of the protocol, say Agent A, sends a query to the recipient (source machine of the connection), say Agent B to inform it whether the username of that connection is authorized or not using ‘Query-If’ performative.

If Agent B does not trust or authenticate Agent A, it responds with ‘Refuse’ performative. If Agent B does not recognize the protocol, it sends a default response using ‘Not Understood’ performative. If it finds the user as authorized or unauthorized, it conveys the same with the ‘Inform’ performative. If it cannot find any information about the username in its log files, it sends that user is unknown with the ‘Inform’ performative. At the same time it contacts the involved host agents about the username through the **Suspicious Login Protocol**. So the same protocol now repeats between Agent B and involved hosts.

On the other side, if Agent A receives that user is authorized, it compares the received username against the one in the Suspicious Login Event it is handling. If it finds incompatibility in usernames then it flags the user and sends an alert to the source host of the connection i.e. Agent B through **User Alert Protocol**. It checks in its log files if the user has made any connections to other hosts from its system after this event. If so, it sends an alert to all those host agents through **User Alert Protocol**. In this case the alert message will be ‘User(Username) is flagged’. The systems which receive this alert realize that connections made by the user are illegal and take appropriate actions. Also, they send an alert to the machines to which the user got connected from their systems and
so on. The *User Alert Protocol* is an *Alert Protocol* and sends an alert message that the user “username” is flagged. The path of such an alert message can be depicted as below:

```
User login path :                         Alert path :
Agent A --> Agent B --> Agent C --> Agent F --> Agent G
```

If no compatibility or mismatch is found, it simply updates the log files, noting that the username associated with the event being handled is authorized. If Agent A receives that user is unauthorized, it flags the user and sends an alert to all the machines to which the user got connected after his login. Upon receiving the alert, the agents at those machines send an alert to the machines to which the user got connected after getting connected to their systems and so on. The path of such an alert message can be depicted as below:

```
User login path :                         Alert path :
Agent A --> Agent B --> Agent C --> Agent E --> Agent F --> Agent G
```

46
If it receives that the user in unknown, it updates the event correspondingly and hopes that Agent B will send an alert if it comes to know more information about the user. If it receives that the connection is unknown, it stores the message with the event in the log files.

6.4 Query Echo Storm Protocol

An Echo Storm attack is a denial of service attack that causes network congestion and shutdown. If there is continuous stream of packets transmitted between two UDP services, the output produced by these two services can result in a denial of service on the machines(s) where the services are offered. For example, by connecting a host’s chargen service (a service at the chargen port that generates a number of characters when queried)
to the echo service (a service at the echo port that echoes the request) of another host, all involved hosts may be effectively taken out of service because of the excessively high number of packets generated.

An example of such an attack is the Fraggle / Udpstorm attack where an attacker is able to create a never-ending stream of packets between the echo ports of two victims by sending a single spoofed packet. Initially, the attacker forges a single packet that has been spoofed to appear as if it has originated from the echo port on the second victim machine and sends it to the first victim. The echo service blindly responds to any request it receives by simply echoing the data of the request back to the machine and port that sent the echo request. So when the victim receives this spoofed packet it sends a response to the echo port of the second victim. This second victim responds in a similar fashion, and the loop of traffic continues until it is interrupted by an external source. This attack is illustrated in Figure 6.3.

This protocol will enable the first victim or second victim whoever has identified a continuous stream of packets at its echo service from a single host, query the other victim about the echo storm it is experiencing. The query message should contain the port number at which it is receiving the echo packets and also the first echo request it received in that stream. Upon receiving the query, the other victim checks if that packet has been sent by it.
If it is the second victim in this attack, it will not find any such echo request originating from its system because the first echo request was generated by the attacker spoofing the IP address of second victim. If the echo request was not found it realizes that it is under attack. It alerts the system administrator, so that he can take appropriate measures to ignore any more echo requests from the first victim. It also sends a request to the first victim to ignore the echo request through Ignore Request Protocol. The Ignore Request Protocol is discussed in section 6.5.

If it is the first victim, then it might have an echo request seeming to be from the second victim which occurred (in time) before the echo request sent as part of the query received from the second victim. Then it uses the same protocol to query the second victim host, from which it received the first query, about this first echo request.
request it found in its log files. Then the other host, which is obviously the second victim in this attack, responds to the query accordingly.

In either case, an agent does not wait for the reply to act. Instead it hopes to get the reply after the other agent finds more information. Hence the agent communication in this protocol is highly asynchronous. The AUML diagrams for *Query Echo Storm* and *Ignore Request* protocols are given in Figure 6.4 and Figure 6.5.

![Figure 6.4. Query Echo Storm Protocol](image-url)
6.5 Ignore Request Protocol

This protocol is generally used when an agent realizes that its IP address has been spoofed to launch an attack against it. An attacker spoofs the IP address of the victim to send a command to multiple hosts, to which those hosts reply continuously. This causes a continuous stream of incoming packets at the victim host causing denial-of-service attack. Using this protocol, the host agent requests specific host(s) to stop responding to the command which was originally not sent by it but by an attacker spoofing the IP address of its host. The AUML diagram of this protocol is given in Figure 6.5.

![Figure 6.5. Ignore Request Protocol](image)

6.6 Quit Request Protocol

Some protocols are developed for communicating with agents in the same network, which are referred to as internal protocols. One such protocol developed for the sake of running test cases was Quit Request protocol. As part of the testing, a simulated
distributed environment was created by deploying one or more agents on each machine. To control those agents from a single machine, there needs to be some sort of protocol to communicate with all the agents. If we want to shut down any agent, then request is sent asking it to quit using the *Quit Request Protocol*. Upon receiving the request, the agent checks to see if the sender AID (unique agent ID) matches with the system agent. The shutdown request is ignored when the AID does not match. The AUML diagram of this protocol is shown in Figure 6.6.

![Figure 6.6. Quit Request Protocol](image)

---

### 6.7 Query Illegal Process Protocol

In general most of the attacks include a suspicious process event indicating that an illegal process was started by an unauthorized user. If there is any abnormal system or network activity raised by the log analyzer out of a connection, then agent has to find if the other end of that connection has any illegal process. For example if Host A is receiving
abnormal length of packets or abnormal sequence of packets out of a connection from Host B, Agent A at that host queries Agent B, about any illegal process at its end.

If Agent B does not authenticate Agent A or if it does not recognize the protocol, it responds in the same way as explained in the *Suspicious Login Protocol*. Otherwise, upon receiving the query, the Agent B checks to see if the user of that connection is authorized and all its system activity is normal. Any occurrence of illegal process in informed. In case it does not have information about the connection event i.e., connection has been originated from some other machine, it informs Agent A that there is insufficient information about the connection. At the same time it contacts the involved host agents about any illegal process through *Illegal Process Protocol*. So the same protocol now repeats between Agent B and the involved hosts.

On the other side, if Agent A receives that “process is not legal”, then it simply updates the log files noting that process associated with the connection event being handled is authorized. If it receives that there is an insufficient information, it updates the event appropriately and hopes that Agent B will send an alert when it gathers sufficient information about that connection. If it receives that “process is illegal” it flags the user associated with that connection. It checks in its log files if the user has made any connections to any other hosts from its system after this event. It sends an alert to all those host agents through *Illegal Connection Alert Protocol*. The *Illegal Connection Alert Protocol* is an Alert Protocol that sends an “Illegal Connection” alert message. The AUML diagram of this protocol is given in Figure 6.7.
Figure 6.7. Query Illegal Process Protocol
CHAPTER 7

Tests and Results

The current framework and the protocols have been tested by simulating various attacks to derive experimental results. These results were useful in focusing and highlighting the current capabilities and future efforts to extend the project. The system was tested on Sun Solaris machines. The log files were stored and maintained in an ORACLE database. Since the main design goal of the system is to gather information about coordinated attacks distributed system environment was simulated by deploying one or more Agents on each Solaris machine. The computer attacks were created by entering series of suspicious events in the respective Agent’s log files. Many test cases were developed for testing the interaction protocols in various attack scenarios but only few test cases are presented in this chapter. The design, development and refinement of interaction protocols, to deal with a variety of existing and new intrusion scenarios needs to be addressed in future efforts.

7.1 Test Cases

For each test case, the events existing in the respective Agents database before running the test and the events or alerts which have been modified or newly added after running the test are presented. The data and fields presented in the following sections are relevant
to only the test cases being discussed in those sections. They are just a subset of the actual data and fields in the actual Agent’s database. The “Agents List Table” in Table 9 is used in all the test cases discussed in this chapter and contains the Agent names corresponding to the host names of those Agents. This table is used by the Agents to find the Agent names of the respective hosts whom they want to communicate with.

**Agents List Table**:

<table>
<thead>
<tr>
<th>HOST NAME</th>
<th>AGENT NAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>paluk.che.sc.edu</td>
<td>ms1</td>
</tr>
<tr>
<td><a href="http://www.reddy.net">www.reddy.net</a></td>
<td>ms6</td>
</tr>
<tr>
<td><a href="http://www.dty.com">www.dty.com</a></td>
<td>ms4</td>
</tr>
<tr>
<td><a href="http://www.lily.com">www.lily.com</a></td>
<td>ms5</td>
</tr>
<tr>
<td>chite.tr.ttu.com</td>
<td>ms3</td>
</tr>
<tr>
<td>rose.oo.org</td>
<td>ms2</td>
</tr>
</tbody>
</table>

**7.2 Testing Denial of Service Alert Protocol**

**7.2.1 Test case 01**

**Test case Scenario**:

Agent “ms1” experiences Denial of Service attack due to SYN flooding as explained in Section 6.2. It sends an alert to all the known Agents using Denial Of Service Alert Protocol.
Data before running the test:

EventsTable for “ms1”:

<table>
<thead>
<tr>
<th>EVENTID</th>
<th>TYPEOF EVENT</th>
<th>SOURCE HOSTNAME</th>
<th>SOURCE PORT</th>
<th>DEST HOSTNAME</th>
<th>DEST PORT</th>
</tr>
</thead>
<tbody>
<tr>
<td>E102175411245</td>
<td>1</td>
<td>-</td>
<td>-</td>
<td>paluk.che.sc.edu</td>
<td>3290</td>
</tr>
<tr>
<td>E1027363339510</td>
<td>4</td>
<td>chite.tr.ttu.com</td>
<td>6758</td>
<td>paluk.che.sc.edu</td>
<td>9474</td>
</tr>
<tr>
<td>E1027365839107</td>
<td>4</td>
<td>paluk.che.sc.edu</td>
<td>3734</td>
<td><a href="http://www.lily.com">www.lily.com</a></td>
<td>3484</td>
</tr>
<tr>
<td>E1031895895241</td>
<td>4</td>
<td>paluk.che.sc.edu</td>
<td>4383</td>
<td><a href="http://www.reddy.net">www.reddy.net</a></td>
<td>6793</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>EVENTDATE</th>
<th>EVENTTIME</th>
<th>USER NAME</th>
<th>DONE FLAG</th>
<th>COMMENTS</th>
<th>AGENTS CHAIN</th>
</tr>
</thead>
<tbody>
<tr>
<td>09/07/2002</td>
<td>102175411220</td>
<td>takur</td>
<td>0</td>
<td>null</td>
<td>null</td>
</tr>
<tr>
<td>09/07/2002</td>
<td>1027363338445</td>
<td>rahul</td>
<td>1</td>
<td>null</td>
<td>null</td>
</tr>
<tr>
<td>09/07/2002</td>
<td>1027365735156</td>
<td>rahul</td>
<td>1</td>
<td>null</td>
<td>null</td>
</tr>
<tr>
<td>09/07/2002</td>
<td>1031895894331</td>
<td>kimi</td>
<td>1</td>
<td>null</td>
<td>null</td>
</tr>
</tbody>
</table>

EventsTable for “ms4*”:

<table>
<thead>
<tr>
<th>EVENTID</th>
<th>TYPEOF EVENT</th>
<th>SOURCE HOSTNAME</th>
<th>SOURCE PORT</th>
<th>DEST HOSTNAME</th>
<th>DEST PORT</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>EVENTDATE</th>
<th>EVENTTIME</th>
<th>USER NAME</th>
<th>DONE</th>
<th>COMMENTS</th>
<th>AGENTS CHAIN</th>
</tr>
</thead>
</table>

TooManyPacketsEventsTable for “ms4*”:

<table>
<thead>
<tr>
<th>EVENTID</th>
<th>ALERT NAME</th>
<th>ALERT DETAIL</th>
</tr>
</thead>
</table>

EventsTable for “ms5*”:

<table>
<thead>
<tr>
<th>EVENTID</th>
<th>TYPEOF EVENT</th>
<th>SOURCE HOSTNAME</th>
<th>SOURCE PORT</th>
<th>DEST HOSTNAME</th>
<th>DEST PORT</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>EVENTDATE</th>
<th>EVENTTIME</th>
<th>USER NAME</th>
<th>DONE</th>
<th>COMMENTS</th>
<th>AGENTS CHAIN</th>
</tr>
</thead>
</table>

TooManyPacketsEventsTable for “ms5*”:

<table>
<thead>
<tr>
<th>EVENTID</th>
<th>ALERT NAME</th>
<th>ALERT DETAIL</th>
</tr>
</thead>
</table>
### EventsTable1 for “ms6”:

<table>
<thead>
<tr>
<th>EVENTID</th>
<th>TYPEOF EVENT</th>
<th>SOURCE HOSTNAME</th>
<th>SOURCE PORT</th>
<th>DEST HOSTNAME</th>
<th>DEST PORT</th>
</tr>
</thead>
<tbody>
<tr>
<td>E1031704190789</td>
<td>3</td>
<td>huda.hyd.com</td>
<td>7893</td>
<td><a href="http://www.reddy.net">www.reddy.net</a></td>
<td>4673</td>
</tr>
<tr>
<td>E1031898795235</td>
<td>3</td>
<td>rose.oo.org</td>
<td>14831</td>
<td><a href="http://www.reddy.net">www.reddy.net</a></td>
<td>7143</td>
</tr>
<tr>
<td>E1131898795236</td>
<td>0</td>
<td>paluk.che.sc.edu</td>
<td>1038</td>
<td><a href="http://www.reddy.net">www.reddy.net</a></td>
<td>7887</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>EVENTDATE</th>
<th>EVENTTIME</th>
<th>USER NAME</th>
<th>DONE FLAG</th>
<th>COMMENTS</th>
<th>AGENTS CHAIN</th>
</tr>
</thead>
<tbody>
<tr>
<td>09/07/2002</td>
<td>1031704189546</td>
<td>samsud</td>
<td>4</td>
<td>null</td>
<td>null</td>
</tr>
<tr>
<td>09/07/2002</td>
<td>1031898794334</td>
<td>nimmi</td>
<td>1</td>
<td>null</td>
<td>null</td>
</tr>
<tr>
<td>09/07/2002</td>
<td>1131898795236</td>
<td>heena</td>
<td>1</td>
<td>null</td>
<td>null</td>
</tr>
</tbody>
</table>

### TooManyPacketsEventsTable for “ms6*”:

<table>
<thead>
<tr>
<th>EVENTID</th>
</tr>
</thead>
</table>

### Results :

The Agent “ms1” reads the event (EventID “E102175411245”) and starts sending alerts to known Agents “ms4*”, “ms5*” and “ms6”. The Agents “ms4*” and “ms5*” are the replicas of “ms4” and “ms5” Agents started on different machines. Many “ms4” and “ms5” Agents were started as part of testing the system. The alert message contains all its current outgoing connections (EventIDs “E1027365839107”, “E1031895895241”).

Upon receiving the alert, the Agents “ms4*” and “ms5*” and “ms6” check in their logged events if their current incoming connections from “ms1” matches with any in the list of current outgoing connections of “ms1” sent as part of the alert message. The Agents “ms4*” and “ms5*” does not find any connections from “ms1”. The Agent “ms6” finds an incoming connection (EventID “E1131898795236”) from “ms1” which is not listed in the outgoing connections of “ms1”. The Agent “ms6” updates that incoming connection
event with doneFlag = 2 and comments = “Reset the Connection, IPSpoofed”, which indicates that it needs attention from system administrator. It is assumed that system administrator takes appropriate actions to reset the connection.

**Data after running the test:**

**EventsTable for “ms1”:**

<table>
<thead>
<tr>
<th>EVENTID</th>
<th>TYPEOF EVENT</th>
<th>SOURCE HOSTNAME</th>
<th>SOURCE PORT</th>
<th>DEST HOSTNAME</th>
<th>DEST PORT</th>
</tr>
</thead>
<tbody>
<tr>
<td>E102175411245</td>
<td>1</td>
<td>-</td>
<td>-</td>
<td>paluk.che.sc.edu</td>
<td>3290</td>
</tr>
<tr>
<td>E1027363339510</td>
<td>4</td>
<td>chite.tr.ttu.com</td>
<td>6758</td>
<td>paluk.che.sc.edu</td>
<td>9474</td>
</tr>
<tr>
<td>E1027365839107</td>
<td>4</td>
<td>paluk.che.sc.edu</td>
<td>3734</td>
<td><a href="http://www.lily.com">www.lily.com</a></td>
<td>3484</td>
</tr>
<tr>
<td>E1031895895241</td>
<td>4</td>
<td>paluk.che.sc.edu</td>
<td>4383</td>
<td><a href="http://www.reddy.net">www.reddy.net</a></td>
<td>6793</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>EVENTDATE</th>
<th>EVENTTIME</th>
<th>USERNAME</th>
<th>DONE FLAG</th>
<th>COMMENTS</th>
<th>AGENTS CHAIN</th>
</tr>
</thead>
<tbody>
<tr>
<td>09/07/2002</td>
<td>102175411220</td>
<td>takur</td>
<td>1</td>
<td>null</td>
<td>null</td>
</tr>
<tr>
<td>09/07/2002</td>
<td>1027363338445</td>
<td>rahul</td>
<td>1</td>
<td>null</td>
<td>null</td>
</tr>
<tr>
<td>09/07/2002</td>
<td>1027365735156</td>
<td>rahul</td>
<td>1</td>
<td>null</td>
<td>null</td>
</tr>
<tr>
<td>09/07/2002</td>
<td>1031895894331</td>
<td>kimi</td>
<td>1</td>
<td>null</td>
<td>null</td>
</tr>
</tbody>
</table>

**EventsTable for “ms4*”:**

<table>
<thead>
<tr>
<th>EVENTID</th>
<th>TYPEOF EVENT</th>
<th>SOURCE HOSTNAME</th>
<th>SOURCE PORT</th>
<th>DEST HOSTNAME</th>
<th>DEST PORT</th>
</tr>
</thead>
<tbody>
<tr>
<td>E1036270575849</td>
<td>1</td>
<td>-</td>
<td>-</td>
<td>paluk.che.sc.edu</td>
<td>3290</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>EVENTDATE</th>
<th>EVENTTIME</th>
<th>USERNAME</th>
<th>DONE FLAG</th>
<th>COMMENTS</th>
<th>AGENTS CHAIN</th>
</tr>
</thead>
<tbody>
<tr>
<td>09/07/2002</td>
<td>102175411220</td>
<td>takur</td>
<td>1</td>
<td>null</td>
<td>ms1</td>
</tr>
</tbody>
</table>

**TooManyPacketsEventsTable for “ms4*”:**

<table>
<thead>
<tr>
<th>EVENTID</th>
<th>ALERT NAME</th>
<th>ALERT DETAIL</th>
</tr>
</thead>
<tbody>
<tr>
<td>E1036270575849</td>
<td>Denial Of Service</td>
<td>Too many connections</td>
</tr>
</tbody>
</table>
EventsTable for “ms5*”:

<table>
<thead>
<tr>
<th>EVENTID</th>
<th>TYPEOF EVENT</th>
<th>SOURCE HOSTNAME</th>
<th>SOURCE PORT</th>
<th>DEST HOSTNAME</th>
<th>DEST PORT</th>
</tr>
</thead>
<tbody>
<tr>
<td>E1036270575796</td>
<td>1</td>
<td>-</td>
<td>-</td>
<td>paluk.che.sc.edu</td>
<td>3290</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>EVENTDATE</th>
<th>EVENTTIME</th>
<th>USER NAME</th>
<th>DONE FLAG</th>
<th>COMMENTS</th>
<th>AGENTS CHAIN</th>
</tr>
</thead>
<tbody>
<tr>
<td>09/07/2002</td>
<td>102175411220</td>
<td>takur</td>
<td>1</td>
<td>null</td>
<td>ms1</td>
</tr>
</tbody>
</table>

TooManyPacketsEventsTable for “ms5*”:

<table>
<thead>
<tr>
<th>EVENTID</th>
<th>ALERT NAME</th>
<th>ALERT DETAIL</th>
</tr>
</thead>
<tbody>
<tr>
<td>E1036270575796</td>
<td>Denial Of Service</td>
<td>Too many connections</td>
</tr>
</tbody>
</table>

EventsTable1 for “ms6”:

<table>
<thead>
<tr>
<th>EVENTID</th>
<th>TYPEOF EVENT</th>
<th>SOURCE HOSTNAME</th>
<th>SOURCE PORT</th>
<th>DEST HOSTNAME</th>
<th>DEST PORT</th>
</tr>
</thead>
<tbody>
<tr>
<td>E1031704190789</td>
<td>3</td>
<td>huda.hyd.com</td>
<td>7893</td>
<td><a href="http://www.reddy.net">www.reddy.net</a></td>
<td>4673</td>
</tr>
<tr>
<td>E1031898795235</td>
<td>3</td>
<td>rose.oo.org</td>
<td>14831</td>
<td><a href="http://www.reddy.net">www.reddy.net</a></td>
<td>7143</td>
</tr>
<tr>
<td>E1131898795236</td>
<td>0</td>
<td>paluk.che.sc.edu</td>
<td>1038</td>
<td><a href="http://www.reddy.net">www.reddy.net</a></td>
<td>7887</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>EVENTDATE</th>
<th>EVENTTIME</th>
<th>USER NAME</th>
<th>DONE FLAG</th>
<th>COMMENTS</th>
<th>AGENTS CHAIN</th>
</tr>
</thead>
<tbody>
<tr>
<td>09/07/2002</td>
<td>1031704189546</td>
<td>samsud</td>
<td>4</td>
<td>null</td>
<td>null</td>
</tr>
<tr>
<td>09/07/2002</td>
<td>1031898794334</td>
<td>nimmi</td>
<td>1</td>
<td>null</td>
<td>null</td>
</tr>
<tr>
<td>09/07/2002</td>
<td>1131898795236</td>
<td>heena</td>
<td>2</td>
<td>Reset Connection, Ip Spoofed</td>
<td>null</td>
</tr>
</tbody>
</table>

TooManyPacketsEventsTable for “ms6”:

<table>
<thead>
<tr>
<th>EVENTID</th>
<th>ALERT NAME</th>
<th>ALERT DETAIL</th>
</tr>
</thead>
<tbody>
<tr>
<td>E1131898795236</td>
<td>Denial Of Service</td>
<td>Too many connections</td>
</tr>
</tbody>
</table>

Messages Exchanged:

Alert Message sent from “ms1” to “ms4*”, “ms5*” and “ms6” Agents

```xml
<?xml version="1.0"?>
<mnss-denialofservice-alert>
  <Alert>Denial Of Service</Alert>
  <AlertDesc>Toomany connections</AlertDesc>
</mnss-denialofservice-alert>
```
<no_of_outgoing_connections>2</no_of_outgoing_connections>

<connections>
  <connection1>
    <connection1:SourceHostName>paluk.che.sc.edu</connection1:SourceHostName>
    <connection1:SourcePort>4383</connection1:SourcePort>
    <connection1:DestHostName>www.reddy.net</connection1:DestHostName>
    <connection1:DestPort>6793</connection1:DestPort>
    <connection1:EventDate>09/07/2002</connection1:EventDate>
    <connection1:EventTime>1031895894331</connection1:EventTime>
    <connection1:UserName>kimi</connection1:UserName>
  </connection1>
  <connection2>
    <connection2:SourceHostName>paluk.che.sc.edu</connection2:SourceHostName>
    <connection2:SourcePort>3734</connection2:SourcePort>
    <connection2:DestHostName>www.lily.com</connection2:DestHostName>
    <connection2:DestPort>3484</connection2:DestPort>
    <connection2:EventDate>09/07/2002</connection2:EventDate>
    <connection2:EventTime>1027365735156</connection2:EventTime>
    <connection2:UserName>rahul</connection2:UserName>
  </connection2>
</connections>

<event>
  <EventDate>09/06/2002</EventDate>
  <EventTime>102175411220</EventTime>
  <DestHostName>paluk.che.sc.edu</DestHostName>
  <DestPort>3290</DestPort>
  <UserName>takur</UserName>
  <SourcePort>no</SourcePort>
  <SourceHostName>no</SourceHostName>
</event>

<AgentChain>ms1</AgentChain>
<mnss-denialofservice-alert>
7.3 Testing Query Suspicious Login Protocol

7.3.1 Test case 01

Test case Scenario:

Agent “ms1” senses a suspicious login connection event as explained in Section 6.3. It sends a query to the source machine of that connection using Query Suspicious Login Protocol. The source machine Agent replies that the user is unauthorized. Upon receiving the reply, Agent “ms1” sends an alert to all the machines to whom the user got connected after his login to its host.

Data before running the test:

EventsTable for “ms1”:

<table>
<thead>
<tr>
<th>EVENTID</th>
<th>TYPEOF EVENT</th>
<th>SOURCE HOSTNAME</th>
<th>SOURCE PORT</th>
<th>DEST HOSTNAME</th>
<th>DEST PORT</th>
</tr>
</thead>
<tbody>
<tr>
<td>E102175478273</td>
<td>4</td>
<td>rose.oo.org</td>
<td>4733</td>
<td>paluk.che.sc.edu</td>
<td>3738</td>
</tr>
<tr>
<td>E102175315736</td>
<td>4</td>
<td>paluk.che.sc.edu</td>
<td>4125</td>
<td><a href="http://www.dty.com">www.dty.com</a></td>
<td>4322</td>
</tr>
<tr>
<td>E1021754564744</td>
<td>4</td>
<td>paluk.che.sc.edu</td>
<td>3478</td>
<td><a href="http://www.lily.com">www.lily.com</a></td>
<td>4394</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>EVENTDATE</th>
<th>EVENTTIME</th>
<th>USERNAME</th>
<th>DONE FLAG</th>
<th>COMMENTS</th>
<th>AGENTS CHAIN</th>
</tr>
</thead>
<tbody>
<tr>
<td>09/06/2001</td>
<td>1021754278560</td>
<td>srikar</td>
<td>0</td>
<td>null</td>
<td>null</td>
</tr>
<tr>
<td>09/06/2001</td>
<td>1021754345383</td>
<td>srikar</td>
<td>1</td>
<td>null</td>
<td>null</td>
</tr>
<tr>
<td>09/06/2001</td>
<td>1021754456383</td>
<td>srikar</td>
<td>1</td>
<td>null</td>
<td>null</td>
</tr>
</tbody>
</table>

LoginEventsTable for “ms1”:

<table>
<thead>
<tr>
<th>EVENTID</th>
<th>USERNAME</th>
<th>LIST OF COMMANDS</th>
</tr>
</thead>
<tbody>
<tr>
<td>E102175478273</td>
<td>srikar</td>
<td>null</td>
</tr>
<tr>
<td>E102175315736</td>
<td>srikar</td>
<td>null</td>
</tr>
<tr>
<td>E1021754564744</td>
<td>Srikar</td>
<td>null</td>
</tr>
</tbody>
</table>

UsersTable for “ms1”:

<table>
<thead>
<tr>
<th>USERNAME</th>
<th>AUTHORIZED</th>
<th>FLAGGED</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### EventsTable for “ms2”:

<table>
<thead>
<tr>
<th>EVENTID</th>
<th>TYPEOF EVENT</th>
<th>SOURCE HOSTNAME</th>
<th>SOURCE PORT</th>
<th>DEST HOSTNAME</th>
<th>DEST PORT</th>
</tr>
</thead>
<tbody>
<tr>
<td>E1021754278456</td>
<td>4</td>
<td>rose.oo.org</td>
<td>4733</td>
<td>paluk.che.sc.edu</td>
<td>3738</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>EVENTDATE</th>
<th>EVENTTIME</th>
<th>USERNAME</th>
<th>DONE</th>
<th>FLAG</th>
<th>COMMENTS</th>
<th>AGENTS CHAIN</th>
</tr>
</thead>
<tbody>
<tr>
<td>09/06/2001</td>
<td>1021754277345</td>
<td>srikar</td>
<td>1</td>
<td>null</td>
<td>null</td>
<td>null</td>
</tr>
</tbody>
</table>

### LoginEventsTable for “ms2”:

<table>
<thead>
<tr>
<th>EVENTID</th>
<th>USERNAME</th>
<th>LIST OF COMMANDS</th>
</tr>
</thead>
<tbody>
<tr>
<td>E1021754278456</td>
<td>srikar</td>
<td>null</td>
</tr>
</tbody>
</table>

### UsersTable for “ms2”:

<table>
<thead>
<tr>
<th>USERNAME</th>
<th>AUTHORIZED</th>
<th>FLAGGED</th>
</tr>
</thead>
<tbody>
<tr>
<td>srikar</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

### EventsTable for “ms4”:

<table>
<thead>
<tr>
<th>EVENTID</th>
<th>TYPEOF EVENT</th>
<th>SOURCE HOSTNAME</th>
<th>SOURCE PORT</th>
<th>DEST HOSTNAME</th>
<th>DEST PORT</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>EVENTDATE</th>
<th>EVENTTIME</th>
<th>USERNAME</th>
<th>DONE</th>
<th>FLAG</th>
<th>COMMENTS</th>
<th>AGENTS CHAIN</th>
</tr>
</thead>
</table>

### UsersTable for “ms4”:

<table>
<thead>
<tr>
<th>USERNAME</th>
<th>AUTHORIZED</th>
<th>FLAGGED</th>
</tr>
</thead>
</table>

### EventsTable for “ms5”:

<table>
<thead>
<tr>
<th>EVENTID</th>
<th>TYPEOF EVENT</th>
<th>SOURCE HOSTNAME</th>
<th>SOURCE PORT</th>
<th>DEST HOSTNAME</th>
<th>DEST PORT</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>EVENTDATE</th>
<th>EVENTTIME</th>
<th>USERNAME</th>
<th>DONE</th>
<th>FLAG</th>
<th>COMMENTS</th>
<th>AGENTS CHAIN</th>
</tr>
</thead>
</table>
UsersTable for “ms5”:

<table>
<thead>
<tr>
<th>USERNAME</th>
<th>AUTHORIZED</th>
<th>FLAGGED</th>
</tr>
</thead>
</table>

Results:

Agent “ms1” reads the event (EventID “E102175478273”) and sends a query to the Agent “ms2” running on the source machine of the connection. The Agent “ms2” checks in its UsersTable and finds that user “srikar” is flagged. So it replies that the user is not authorized. Upon receiving this message, Agent “ms1” checks in its Events Table for any other connections made by the user after his login event to its host. It finds two connections (EventIDs “E102175315736”, “E1021754564744”) and sends an alert about the user to respective host Agents “ms4” and “ms5”. The Agents “ms4” and “ms5” store the alert in the respective tables (EventsTable and UsersTable).

Data after running the test:

EventsTable for “ms1”:

<table>
<thead>
<tr>
<th>EVENTID</th>
<th>TYPEOF EVENT</th>
<th>SOURCE HOSTNAME</th>
<th>SOURCE PORT</th>
<th>DEST HOSTNAME</th>
<th>DEST PORT</th>
</tr>
</thead>
<tbody>
<tr>
<td>E102175478273</td>
<td>4</td>
<td>rose.oo.org</td>
<td>4733</td>
<td>paluk.che.sc.edu</td>
<td>3738</td>
</tr>
<tr>
<td>E102175315736</td>
<td>4</td>
<td>paluk.che.sc.edu</td>
<td>4125</td>
<td><a href="http://www.dty.com">www.dty.com</a></td>
<td>4322</td>
</tr>
<tr>
<td>E1021754564744</td>
<td>4</td>
<td>paluk.che.sc.edu</td>
<td>3478</td>
<td><a href="http://www.lily.com">www.lily.com</a></td>
<td>4394</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>EVENTDATE</th>
<th>EVENTTIME</th>
<th>USER NAME</th>
<th>DONE</th>
<th>FLAG</th>
<th>COMMENTS</th>
<th>AGENTS CHAIN</th>
</tr>
</thead>
<tbody>
<tr>
<td>09/06/2001</td>
<td>1021754278560</td>
<td>srikar</td>
<td>1</td>
<td></td>
<td>null</td>
<td>null</td>
</tr>
<tr>
<td>09/06/2001</td>
<td>1021754345383</td>
<td>srikar</td>
<td>1</td>
<td></td>
<td>null</td>
<td>ms1-ms4</td>
</tr>
<tr>
<td>09/06/2001</td>
<td>1021754456383</td>
<td>srikar</td>
<td>1</td>
<td></td>
<td>null</td>
<td>ms1-ms5</td>
</tr>
</tbody>
</table>
LoginEventsTable for “ms1”:

<table>
<thead>
<tr>
<th>EVENTID</th>
<th>USERNAME</th>
<th>LIST OF COMMANDS</th>
</tr>
</thead>
<tbody>
<tr>
<td>E102175478273</td>
<td>srikar</td>
<td>null</td>
</tr>
<tr>
<td>E102175315736</td>
<td>srikar</td>
<td>null</td>
</tr>
<tr>
<td>E1021754564744</td>
<td>srikar</td>
<td>null</td>
</tr>
</tbody>
</table>

UsersTable for “ms1”:

<table>
<thead>
<tr>
<th>USERNAME</th>
<th>AUTHORIZED</th>
<th>FLAGGED</th>
</tr>
</thead>
<tbody>
<tr>
<td>srikar</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

EventsTable for “ms4”:

<table>
<thead>
<tr>
<th>EVENTID</th>
<th>TYPEOF EVENT</th>
<th>SOURCE HOSTNAME</th>
<th>SOURCE PORT</th>
<th>DEST HOSTNAME</th>
<th>DEST PORT</th>
</tr>
</thead>
<tbody>
<tr>
<td>E1036283771701</td>
<td>4</td>
<td>paluk.che.sc.edu</td>
<td>4125</td>
<td><a href="http://www.dty.com">www.dty.com</a></td>
<td>4322</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>EVENTDATE</th>
<th>EVENTTIME</th>
<th>USERNAME</th>
<th>DONE FLAG</th>
<th>COMMENTS</th>
<th>AGENTS CHAIN</th>
</tr>
</thead>
<tbody>
<tr>
<td>09/06/2001</td>
<td>1021754345383</td>
<td>srikar</td>
<td>1</td>
<td>null</td>
<td>ms1</td>
</tr>
</tbody>
</table>

LoginEventsTable for “ms4”:

<table>
<thead>
<tr>
<th>EVENTID</th>
<th>USERNAME</th>
<th>LIST OF COMMANDS</th>
</tr>
</thead>
<tbody>
<tr>
<td>E1036283771701</td>
<td>srikar</td>
<td>null</td>
</tr>
</tbody>
</table>

UsersTable for “ms4”:

<table>
<thead>
<tr>
<th>USERNAME</th>
<th>AUTHORIZED</th>
<th>FLAGGED</th>
</tr>
</thead>
<tbody>
<tr>
<td>srikar</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

EventsTable for “ms5”:

<table>
<thead>
<tr>
<th>EVENTID</th>
<th>TYPEOF EVENT</th>
<th>SOURCE HOSTNAME</th>
<th>SOURCE PORT</th>
<th>DEST HOSTNAME</th>
<th>DEST PORT</th>
</tr>
</thead>
<tbody>
<tr>
<td>E1036283771771</td>
<td>4</td>
<td>paluk.che.sc.edu</td>
<td>3478</td>
<td><a href="http://www.lily.com">www.lily.com</a></td>
<td>4394</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>EVENTDATE</th>
<th>EVENTTIME</th>
<th>USERNAME</th>
<th>DONE FLAG</th>
<th>COMMENTS</th>
<th>AGENTS CHAIN</th>
</tr>
</thead>
<tbody>
<tr>
<td>09/06/2001</td>
<td>1021754456383</td>
<td>srikar</td>
<td>1</td>
<td>null</td>
<td>ms1</td>
</tr>
</tbody>
</table>
LoginEventsTable for “ms5”:

<table>
<thead>
<tr>
<th>EVENTID</th>
<th>USERNAME</th>
<th>LIST OF COMMANDS</th>
</tr>
</thead>
<tbody>
<tr>
<td>E1036283771771</td>
<td>srikar</td>
<td>null</td>
</tr>
</tbody>
</table>

UsersTable for “ms5”:

<table>
<thead>
<tr>
<th>USERNAME</th>
<th>AUTHORIZED</th>
<th>FLAGGED</th>
</tr>
</thead>
<tbody>
<tr>
<td>srikar</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

Messages Exchanged:

Query Message sent from “ms1” to “ms2” Agent:

```xml
<?xml version="1.0"?>
<mnss-susp-login>
  <event>
    <SourceHostName>rose.oo.org</SourceHostName>
    <SourcePort>4733</SourcePort>
    <DestHostName>paluk.che.sc.edu</DestHostName>
    <DestPort>3738</DestPort>
    <EventDate>09/06/2001</EventDate>
    <EventTime>1021754278560</EventTime>
    <UserName>srikar</UserName>
  </event>
</mnss-susp-login>
```

Reply Message sent from “ms2” to “ms1” Agent:

```xml
<?xml version="1.0"?>
<mnss-susp-login>
  <event>
    <SourceHostName>rose.oo.org</SourceHostName>
    <SourcePort>4733</SourcePort>
    <DestHostName>paluk.che.sc.edu</DestHostName>
    <DestPort>3738</DestPort>
    <EventDate>09/06/2002</EventDate>
    <EventTime>1021754277345</EventTime>
    <UserName>srikar</UserName>
    <Flagged>yes</Flagged>
  </event>
</mnss-susp-login>
Alert Message sent from “ms1” to “ms4” Agent :

```xml
<?xml version="1.0"?>
<mnss-alert-abt-user>
<event>
  <SourceHostName>paluk.che.sc.edu</SourceHostName>
  <SourcePort>4125</SourcePort>
  <DestHostName>www.dty.com</DestHostName>
  <DestPort>4322</DestPort>
  <EventDate>09/06/2001</EventDate>
  <EventTime>1021754345383</EventTime>
  <UserName>srikar</UserName>
</event>
<AgentChain>ms1</AgentChain>
<Flagged>yes</Flagged>
<UserNames>[srikar]</UserNames>
</mnss-alert-abt-user>
```

Alert Message sent from “ms1” to “ms5” Agent :

```xml
<?xml version="1.0"?>
<mnss-alert-abt-user>
<event>
  <SourceHostName>paluk.che.sc.edu</SourceHostName>
  <SourcePort>3478</SourcePort>
  <DestHostName>www.lily.com</DestHostName>
  <DestPort>4394</DestPort>
  <EventDate>09/06/2001</EventDate>
  <EventTime>1021754456383</EventTime>
  <UserName>srikar</UserName>
</event>
<AgentChain>ms1</AgentChain>
<Flagged>yes</Flagged>
<UserNames>[srikar]</UserNames>
</mnss-alert-abt-user>
```

7.3.2 Test case 02

Test case Scenario :

Agent “ms1” senses a suspicious login connection event as explained in Section 6.3. It sends a query to the source machine of that connection using Query Suspicious Login
Protocol. The source machine Agent replies that the user is authorized. Upon receiving the reply, Agent “ms1” finds an incompatibility in usernames, it flags the user and sends an alert to the source machine of the connection. At the same time it checks for any connections made by the user after his login event to its host. If it finds any, it sends an alert about the user to the Agents of the respective host machines.

Data before running the test:

EventsTable for “ms1”:

<table>
<thead>
<tr>
<th>EVENTID</th>
<th>TYPEOF EVENT</th>
<th>SOURCE HOSTNAME</th>
<th>SOURCE PORT</th>
<th>DEST HOSTNAME</th>
<th>DEST PORT</th>
</tr>
</thead>
<tbody>
<tr>
<td>E1031704165153</td>
<td>4</td>
<td>chite.tr.ttu.com</td>
<td>7855</td>
<td>paluk.che.sc.edu</td>
<td>5744</td>
</tr>
<tr>
<td>E1031704178474</td>
<td>4</td>
<td>paluk.che.sc.edu</td>
<td>5744</td>
<td><a href="http://www.reddy.net">www.reddy.net</a></td>
<td>5785</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>EVENTDATE</th>
<th>EVENTTIME</th>
<th>USERNAME</th>
<th>DONE FLAG</th>
<th>COMMENTS</th>
<th>AGENTS CHAIN</th>
</tr>
</thead>
<tbody>
<tr>
<td>09/06/2001</td>
<td>1031704165153</td>
<td>mitch</td>
<td>0</td>
<td>null</td>
<td>null</td>
</tr>
<tr>
<td>09/06/2001</td>
<td>1031704179474</td>
<td>mitch</td>
<td>1</td>
<td>null</td>
<td>null</td>
</tr>
</tbody>
</table>

LoginEventsTable for “ms1”:

<table>
<thead>
<tr>
<th>EVENTID</th>
<th>USERNAME</th>
<th>LIST OF COMMANDS</th>
</tr>
</thead>
<tbody>
<tr>
<td>E1031704165153</td>
<td>mitch</td>
<td>null</td>
</tr>
<tr>
<td>E1031704178474</td>
<td>mitch</td>
<td>null</td>
</tr>
</tbody>
</table>

UsersTable for “ms1”:

<table>
<thead>
<tr>
<th>USERNAME</th>
<th>AUTHORIZED</th>
<th>FLAGGED</th>
</tr>
</thead>
<tbody>
<tr>
<td>---------</td>
<td>------------</td>
<td>---------</td>
</tr>
</tbody>
</table>
### EventsTable for “ms3”:

<table>
<thead>
<tr>
<th>EVENTID</th>
<th>TYPEOF EVENT</th>
<th>SOURCE HOSTNAME</th>
<th>SOURCE PORT</th>
<th>DEST HOSTNAME</th>
<th>DEST PORT</th>
</tr>
</thead>
<tbody>
<tr>
<td>E1031704145383</td>
<td>4</td>
<td>chite.tr.ttu.com</td>
<td>7855</td>
<td>paluk.che.sc.edu</td>
<td>5744</td>
</tr>
<tr>
<td>E1031704137894</td>
<td>4</td>
<td>rose.oo.org</td>
<td>4674</td>
<td>chite.tr.ttu.com</td>
<td>7855</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>EVENTDATE</th>
<th>EVENTTIME</th>
<th>USER NAME</th>
<th>DONE FLAG</th>
<th>COMMENTS</th>
<th>AGENTS CHAIN</th>
</tr>
</thead>
<tbody>
<tr>
<td>09/06/2001</td>
<td>1031704140234</td>
<td>mike</td>
<td>1</td>
<td>null</td>
<td>null</td>
</tr>
<tr>
<td>09/06/2001</td>
<td>1031704136748</td>
<td>mike</td>
<td>1</td>
<td>null</td>
<td>null</td>
</tr>
</tbody>
</table>

### LoginEventsTable for “ms3”:

<table>
<thead>
<tr>
<th>EVENTID</th>
<th>USERNAME</th>
<th>LIST OF COMMANDS</th>
</tr>
</thead>
<tbody>
<tr>
<td>E1031704145383</td>
<td>mike</td>
<td>null</td>
</tr>
<tr>
<td>E1031704137894</td>
<td>mike</td>
<td>null</td>
</tr>
</tbody>
</table>

### UsersTable for “ms3”:

<table>
<thead>
<tr>
<th>USERNAME</th>
<th>AUTHORIZED</th>
<th>FLAGGED</th>
</tr>
</thead>
<tbody>
<tr>
<td>mike</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

### EventsTable for “ms2”:

<table>
<thead>
<tr>
<th>EVENTID</th>
<th>TYPEOF EVENT</th>
<th>SOURCE HOSTNAME</th>
<th>SOURCE PORT</th>
<th>DEST HOSTNAME</th>
<th>DEST PORT</th>
</tr>
</thead>
<tbody>
<tr>
<td>E1031704123743</td>
<td>4</td>
<td><a href="http://www.dty.com">www.dty.com</a></td>
<td>5678</td>
<td>rose.oo.org</td>
<td>4674</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>EVENTDATE</th>
<th>EVENTTIME</th>
<th>USER NAME</th>
<th>DONE FLAG</th>
<th>COMMENTS</th>
<th>AGENTS CHAIN</th>
</tr>
</thead>
<tbody>
<tr>
<td>09/05/2002</td>
<td>1031704122456</td>
<td>mike</td>
<td>1</td>
<td>null</td>
<td>null</td>
</tr>
</tbody>
</table>

### LoginEventsTable for “ms2”:

<table>
<thead>
<tr>
<th>EVENTID</th>
<th>USERNAME</th>
<th>LIST OF COMMANDS</th>
</tr>
</thead>
<tbody>
<tr>
<td>E1031704123743</td>
<td>mike</td>
<td>null</td>
</tr>
</tbody>
</table>

### UsersTable for “ms2”:

<table>
<thead>
<tr>
<th>USERNAME</th>
<th>AUTHORIZED</th>
<th>FLAGGED</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
EventsTable for “ms6”:

<table>
<thead>
<tr>
<th>EVENTID</th>
<th>TYPEOF EVENT</th>
<th>SOURCE HOSTNAME</th>
<th>SOURCE PORT</th>
<th>DEST HOSTNAME</th>
<th>DEST PORT</th>
</tr>
</thead>
<tbody>
<tr>
<td>E1031704190465</td>
<td>4</td>
<td><a href="http://www.reddy.net">www.reddy.net</a></td>
<td>7849</td>
<td><a href="http://www.lily.com">www.lily.com</a></td>
<td>5785</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>EVENTDATE</th>
<th>EVENTTIME</th>
<th>USERNAME</th>
<th>DONE</th>
<th>FLAG</th>
<th>COMMENTS</th>
<th>AGENTS</th>
<th>CHAIN</th>
</tr>
</thead>
<tbody>
<tr>
<td>09/06/2002</td>
<td>1031704185647</td>
<td>mitch</td>
<td>1</td>
<td></td>
<td>null</td>
<td>null</td>
<td></td>
</tr>
</tbody>
</table>

LoginEventsTable for “ms6”:

<table>
<thead>
<tr>
<th>EVENTID</th>
<th>USERNAME</th>
<th>LIST OF COMMANDS</th>
</tr>
</thead>
<tbody>
<tr>
<td>E1031704190465</td>
<td>mitch</td>
<td>null</td>
</tr>
</tbody>
</table>

UsersTable for “ms6”:

<table>
<thead>
<tr>
<th>USERNAME</th>
<th>AUTHORIZED</th>
<th>FLAGGED</th>
</tr>
</thead>
</table>

EventsTable for “ms4”:

<table>
<thead>
<tr>
<th>EVENTID</th>
<th>TYPEOF EVENT</th>
<th>SOURCE HOSTNAME</th>
<th>SOURCE PORT</th>
<th>DEST HOSTNAME</th>
<th>DEST PORT</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>EVENTDATE</th>
<th>EVENTTIME</th>
<th>USERNAME</th>
<th>DONE</th>
<th>FLAG</th>
<th>COMMENTS</th>
<th>AGENTS</th>
<th>CHAIN</th>
</tr>
</thead>
</table>

LoginEventsTable for “ms4”:

<table>
<thead>
<tr>
<th>EVENTID</th>
<th>USERNAME</th>
<th>LIST OF COMMANDS</th>
</tr>
</thead>
</table>

UsersTable for “ms4”:

<table>
<thead>
<tr>
<th>USERNAME</th>
<th>AUTHORIZED</th>
<th>FLAGGED</th>
</tr>
</thead>
</table>

70
EventsTable for “ms5”:

<table>
<thead>
<tr>
<th>EVENTID</th>
<th>TYPEOF EVENT</th>
<th>SOURCE HOSTNAME</th>
<th>SOURCE PORT</th>
<th>DEST HOSTNAME</th>
<th>DEST PORT</th>
<th>EVENTDATE</th>
<th>EVENTTIME</th>
<th>USER NAME</th>
<th>DONE FLAG</th>
<th>COMMENTS</th>
<th>AGENTS CHAIN</th>
</tr>
</thead>
</table>

LoginEventsTable for “ms5”:

<table>
<thead>
<tr>
<th>EVENTID</th>
<th>USERNAME</th>
<th>LIST OF COMMANDS</th>
</tr>
</thead>
</table>

UsersTable for “ms5”:

<table>
<thead>
<tr>
<th>USERNAME</th>
<th>AUTHORIZED</th>
<th>FLAGGED</th>
</tr>
</thead>
</table>

Results:
The Agent “ms1” reads the event (EventID “E1031704165153”) and queries the source machine of the connection “ms3” about the username of the connection. The Agent “ms3” checks in its UsersTable and replies that user, “mike”, is authorized. Agent “ms1” finds incompatibility in usernames and sends an alert to the source machine, “ms3”, about usernames, “mitch/mike”. At the same time it finds that user got connected to host machine, “ms6”, (EventID “E1031704178474”) after his login event and sends an alert about the usernames to “ms6” Agent. Upon receiving the alert “ms6” Agent finds that user got connected to the host machine, “ms5” (EventID “E1031704190465”) and sends an alert to “ms5” Agent about the usernames. The Agent “ms3” receives an alert from “ms1” about usernames “mitch/mike”. The Agent checks in its logged events if the user got connected to any other machines after his login event. Also it checks if the user has logged on to its machine from other machine. It finds such event from host
machine,”ms2” (EventID “E1031704137894”) and sends an alert back to “ms2” Agent. The Agent “ms2” repeats the same, finds an event from “ms4” host machine (EventID “E1031704123743”) and sends an alert back to “ms4” Agent. The Agent “ms4” does not find any user involved events and so the transfer of alert message stops.

Data after running the test:

EventsTable for “ms1”:

<table>
<thead>
<tr>
<th>EVENTID</th>
<th>TYPEOF EVENT</th>
<th>SOURCE HOSTNAME</th>
<th>SOURCE PORT</th>
<th>DEST HOSTNAME</th>
<th>DEST PORT</th>
</tr>
</thead>
<tbody>
<tr>
<td>E1031704165153</td>
<td>4</td>
<td>chite.tr.ttu.com</td>
<td>7855</td>
<td>paluk.che.sc.edu</td>
<td>5744</td>
</tr>
<tr>
<td>E1031704178474</td>
<td>4</td>
<td>paluk.che.sc.edu</td>
<td>5744</td>
<td><a href="http://www.reddy.net">www.reddy.net</a></td>
<td>5785</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>EVENTDATE</th>
<th>EVENTTIME</th>
<th>USERNAME</th>
<th>DONE</th>
<th>COMMENTS</th>
<th>AGENTS CHAIN</th>
</tr>
</thead>
<tbody>
<tr>
<td>09/06/2001</td>
<td>1031704165153</td>
<td>mitch</td>
<td>1</td>
<td>null</td>
<td>ms1-ms3</td>
</tr>
<tr>
<td>09/06/2001</td>
<td>1031704179474</td>
<td>mitch</td>
<td>1</td>
<td>null</td>
<td>ms1-ms6</td>
</tr>
</tbody>
</table>

LoginEventsTable for “ms1”:

<table>
<thead>
<tr>
<th>EVENTID</th>
<th>USERNAME</th>
<th>LIST OF COMMANDS</th>
</tr>
</thead>
<tbody>
<tr>
<td>E1031704165153</td>
<td>mitch</td>
<td>null</td>
</tr>
<tr>
<td>E1031704178474</td>
<td>mitch</td>
<td>null</td>
</tr>
</tbody>
</table>

UsersTable for “ms1”:

<table>
<thead>
<tr>
<th>USERNAME</th>
<th>AUTHORIZED</th>
<th>FLAGGED</th>
</tr>
</thead>
<tbody>
<tr>
<td>mitch</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Mike</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
EventsTable for “ms3”:

<table>
<thead>
<tr>
<th>EVENTID</th>
<th>TYPEOF EVENT</th>
<th>SOURCE HOSTNAME</th>
<th>SOURCE PORT</th>
<th>DEST HOSTNAME</th>
<th>DEST PORT</th>
</tr>
</thead>
<tbody>
<tr>
<td>E1031704145383</td>
<td>4</td>
<td>chite.tr.ttu.com</td>
<td>7855</td>
<td>paluk.che.sc.edu</td>
<td>5744</td>
</tr>
<tr>
<td>E1031704137894</td>
<td>4</td>
<td>rose.oo.org</td>
<td>4674</td>
<td>chite.tr.ttu.com</td>
<td>7855</td>
</tr>
<tr>
<td>E1036289135377</td>
<td>4</td>
<td>chite.tr.ttu.com</td>
<td>7855</td>
<td>paluk.che.sc.edu</td>
<td>5744</td>
</tr>
<tr>
<td>E1036289136264</td>
<td>4</td>
<td>chite.tr.ttu.com</td>
<td>7855</td>
<td>paluk.che.sc.edu</td>
<td>5744</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>EVENTDATE</th>
<th>EVENTTIME</th>
<th>USER NAME</th>
<th>DONE FLAG</th>
<th>COMMENTS</th>
<th>AGENTS CHAIN</th>
</tr>
</thead>
<tbody>
<tr>
<td>09/06/2001</td>
<td>1031704140234</td>
<td>mike</td>
<td>1</td>
<td>null</td>
<td>null</td>
</tr>
<tr>
<td>09/06/2001</td>
<td>1031704136748</td>
<td>mike</td>
<td>1</td>
<td>null</td>
<td>ms1-ms3-ms2</td>
</tr>
<tr>
<td>09/06/2002</td>
<td>1031704165153</td>
<td>mitch</td>
<td>1</td>
<td>null</td>
<td>ms1</td>
</tr>
<tr>
<td>09/06/2002</td>
<td>1031704165153</td>
<td>mike</td>
<td>1</td>
<td>null</td>
<td>ms1</td>
</tr>
</tbody>
</table>

LoginEventsTable for “ms3”:

<table>
<thead>
<tr>
<th>EVENTID</th>
<th>USERNAME</th>
<th>LIST OF COMMANDS</th>
</tr>
</thead>
<tbody>
<tr>
<td>E1031704145383</td>
<td>mike</td>
<td>null</td>
</tr>
<tr>
<td>E1031704137894</td>
<td>mike</td>
<td>null</td>
</tr>
<tr>
<td>E1036289136264</td>
<td>mike</td>
<td>null</td>
</tr>
<tr>
<td>E1036289135377</td>
<td>mitch</td>
<td>null</td>
</tr>
</tbody>
</table>

UsersTable for “ms3”:

<table>
<thead>
<tr>
<th>USERNAME</th>
<th>AUTHORIZED</th>
<th>FLAGGED</th>
</tr>
</thead>
<tbody>
<tr>
<td>mike</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>mitch</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

EventsTable for “ms2”:

<table>
<thead>
<tr>
<th>EVENTID</th>
<th>TYPEOF EVENT</th>
<th>SOURCE HOSTNAME</th>
<th>SOURCE PORT</th>
<th>DEST HOSTNAME</th>
<th>DEST PORT</th>
</tr>
</thead>
<tbody>
<tr>
<td>E1031704123743</td>
<td>4</td>
<td><a href="http://www.dty.com">www.dty.com</a></td>
<td>5678</td>
<td>rose.oo.org</td>
<td>4674</td>
</tr>
<tr>
<td>E1036289139081</td>
<td>4</td>
<td>rose.oo.org</td>
<td>4674</td>
<td>chite.tr.ttu.com</td>
<td>7855</td>
</tr>
<tr>
<td>E1036289140291</td>
<td>4</td>
<td>rose.oo.org</td>
<td>4674</td>
<td>chite.tr.ttu.com</td>
<td>7855</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>EVENTDATE</th>
<th>EVENTTIME</th>
<th>USER NAME</th>
<th>DONE FLAG</th>
<th>COMMENTS</th>
<th>AGENTS CHAIN</th>
</tr>
</thead>
<tbody>
<tr>
<td>09/05/2002</td>
<td>1031704122456</td>
<td>mike</td>
<td>1</td>
<td>null</td>
<td>ms1-ms3-ms2-ms4</td>
</tr>
<tr>
<td>09/06/2002</td>
<td>1031704136748</td>
<td>mitch</td>
<td>1</td>
<td>null</td>
<td>ms1-ms3</td>
</tr>
<tr>
<td>09/06/2002</td>
<td>1031704136748</td>
<td>mike</td>
<td>1</td>
<td>null</td>
<td>ms1-ms3</td>
</tr>
</tbody>
</table>
LoginEventsTable for “ms2”:

<table>
<thead>
<tr>
<th>EVENTID</th>
<th>USERNAME</th>
<th>LIST OF COMMANDS</th>
</tr>
</thead>
<tbody>
<tr>
<td>E1031704123743</td>
<td>mike</td>
<td>null</td>
</tr>
<tr>
<td>E1036289139081</td>
<td>mitch</td>
<td>null</td>
</tr>
<tr>
<td>E1036289140291</td>
<td>mike</td>
<td>null</td>
</tr>
</tbody>
</table>

UsersTable for “ms2”:

<table>
<thead>
<tr>
<th>USERNAME</th>
<th>AUTHORIZED</th>
<th>FLAGGED</th>
</tr>
</thead>
<tbody>
<tr>
<td>mike</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>mitch</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

EventsTable for “ms4”:

<table>
<thead>
<tr>
<th>EVENTID</th>
<th>TYPEOF EVENT</th>
<th>SOURCE HOSTNAME</th>
<th>SOURCE PORT</th>
<th>DEST HOSTNAME</th>
<th>DEST PORT</th>
</tr>
</thead>
<tbody>
<tr>
<td>E1036289142532</td>
<td>4</td>
<td><a href="http://www.dty.com">www.dty.com</a></td>
<td>5678</td>
<td>rose.oo.org</td>
<td>4674</td>
</tr>
<tr>
<td>E1036289143621</td>
<td>4</td>
<td><a href="http://www.dty.com">www.dty.com</a></td>
<td>5678</td>
<td>rose.oo.org</td>
<td>4674</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>EVENTDATE</th>
<th>EVENTTIME</th>
<th>USERNAME</th>
<th>DONE</th>
<th>FLAG</th>
<th>COMMENTS</th>
<th>AGENTS CHAIN</th>
</tr>
</thead>
<tbody>
<tr>
<td>09/05/2002</td>
<td>1031704122456</td>
<td>mitch</td>
<td>1</td>
<td></td>
<td>null</td>
<td>ms1-ms3-ms2</td>
</tr>
<tr>
<td>09/05/2002</td>
<td>1031704122456</td>
<td>mike</td>
<td>1</td>
<td></td>
<td>null</td>
<td>ms1-ms3-ms2</td>
</tr>
</tbody>
</table>

LoginEventsTable for “ms4”:

<table>
<thead>
<tr>
<th>EVENTID</th>
<th>USERNAME</th>
<th>LIST OF COMMANDS</th>
</tr>
</thead>
<tbody>
<tr>
<td>E1036289143621</td>
<td>mike</td>
<td>null</td>
</tr>
<tr>
<td>E1036289142532</td>
<td>mitch</td>
<td>null</td>
</tr>
</tbody>
</table>

UsersTable for “ms4”:

<table>
<thead>
<tr>
<th>USERNAME</th>
<th>AUTHORIZED</th>
<th>FLAGGED</th>
</tr>
</thead>
<tbody>
<tr>
<td>mike</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>mitch</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

EventsTable for “ms5”:

<table>
<thead>
<tr>
<th>EVENTID</th>
<th>TYPEOF EVENT</th>
<th>SOURCE HOSTNAME</th>
<th>SOURCE PORT</th>
<th>DEST HOSTNAME</th>
<th>DEST PORT</th>
</tr>
</thead>
<tbody>
<tr>
<td>E1036289140919</td>
<td>4</td>
<td><a href="http://www.reddy.net">www.reddy.net</a></td>
<td>7849</td>
<td><a href="http://www.lily.com">www.lily.com</a></td>
<td>5785</td>
</tr>
<tr>
<td>EVENTID</td>
<td>USER NAME</td>
<td>DONE FLAG</td>
<td>COMMENTS</td>
<td>AGENTS CHAIN</td>
<td></td>
</tr>
<tr>
<td>---------------</td>
<td>-----------</td>
<td>-----------</td>
<td>----------</td>
<td>--------------</td>
<td></td>
</tr>
<tr>
<td>E1036289142173</td>
<td>mitch</td>
<td>1</td>
<td>null</td>
<td>ms1-ms6</td>
<td></td>
</tr>
<tr>
<td>E1036289140919</td>
<td>mike</td>
<td>1</td>
<td>null</td>
<td>null</td>
<td></td>
</tr>
<tr>
<td>E1036289142173</td>
<td>mitch</td>
<td>1</td>
<td>null</td>
<td>ms1-ms6</td>
<td></td>
</tr>
</tbody>
</table>

**LoginEventsTable for “ms5”:**

| EVENTID       | USERNAME | LIST OF COMMANDS | |
|---------------|----------|------------------||
| E1036289140919| mike     | null             | |
| E1036289142173| mitch    | null             | |

**UsersTable for “ms5”:**

<table>
<thead>
<tr>
<th>USERNAME</th>
<th>AUTHORIZED</th>
<th>FLAGGED</th>
</tr>
</thead>
<tbody>
<tr>
<td>mike</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>mitch</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

**EventsTable for “ms6”:**

<table>
<thead>
<tr>
<th>EVENTID</th>
<th>TYPE OF EVENT</th>
<th>SOURCE HOSTNAME</th>
<th>SOURCE PORT</th>
<th>DEST HOSTNAME</th>
<th>DEST PORT</th>
</tr>
</thead>
<tbody>
<tr>
<td>E1031704190465</td>
<td>4</td>
<td><a href="http://www.reddy.net">www.reddy.net</a></td>
<td>7849</td>
<td><a href="http://www.lily.com">www.lily.com</a></td>
<td>5785</td>
</tr>
<tr>
<td>E1036289137517</td>
<td>4</td>
<td>paluk.che.sc.edu</td>
<td>5744</td>
<td><a href="http://www.reddy.net">www.reddy.net</a></td>
<td>5785</td>
</tr>
<tr>
<td>E1036289138648</td>
<td>4</td>
<td>paluk.che.sc.edu</td>
<td>5744</td>
<td><a href="http://www.reddy.net">www.reddy.net</a></td>
<td>5785</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>EVENTDATE</th>
<th>EVENTTIME</th>
<th>USER NAME</th>
<th>DONE FLAG</th>
<th>COMMENTS</th>
<th>AGENTS CHAIN</th>
</tr>
</thead>
<tbody>
<tr>
<td>09/06/2002</td>
<td>1031704185647</td>
<td>mitch</td>
<td>1</td>
<td>null</td>
<td>ms1-ms6</td>
</tr>
<tr>
<td>09/06/2002</td>
<td>1031704185647</td>
<td>mike</td>
<td>1</td>
<td>null</td>
<td>ms1-ms6</td>
</tr>
<tr>
<td>09/06/2002</td>
<td>1031704179474</td>
<td>mitch</td>
<td>1</td>
<td>null</td>
<td>ms1</td>
</tr>
</tbody>
</table>

**LoginEventsTable for “ms6”:**

| EVENTID       | USERNAME | LIST OF COMMANDS | |
|---------------|----------|------------------||
| E1031704190465| mitch    | null             | |
| E1036289137517|          | null             | |
| E1036289138648|          | null             | |
UsersTable for “ms6”:

<table>
<thead>
<tr>
<th>USERNAME</th>
<th>AUTHORIZED</th>
<th>FLAGGED</th>
</tr>
</thead>
<tbody>
<tr>
<td>mike</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>mitch</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

Messages Exchanged: The messages exchanged are similar to the Test Case 01 except the connection event details. So only crucial messages among Agents are shown below.

Query Message sent from “ms1” to “ms3” Agent:

```xml
<?xml version="1.0"?>
<mnss-susp-login>
<event>
  <SourceHostName>chite.tr.ttu.com</SourceHostName>
  <SourcePort>7855</SourcePort>
  <DestHostName>paluk.che.sc.edu</DestHostName>
  <DestPort>5744</DestPort>
  <EventDate>09/06/2002</EventDate>
  <EventTime>1031704165153</EventTime>
</event>
<UserName>mitch</UserName>
</mnss-susp-login>
```

Reply Message sent from “ms3” to “ms1” Agent:

```xml
<?xml version="1.0"?>
<mnss-susp-login>
<event>
  <SourceHostName>chite.tr.ttu.com</SourceHostName>
  <SourcePort>7855</SourcePort>
  <EventDate>09/06/2002</EventDate>
  <EventTime>1031704140234</EventTime>
  <DestHostName>paluk.che.sc.edu</DestHostName>
  <DestPort>5744</DestPort>
</event>
<UserName>mike</UserName>
<Authorized>yes</Authorized>
</mnss-susp-login>
```
Alert Message sent from ms1 to all the involved Agents:

```xml
<?xml version="1.0"?>
<mnss-alert-abt-user>
  <event>
    <SourceHostName>chite.tr.ttu.com</SourceHostName>
    <SourcePort>7855</SourcePort>
    <EventDate>09/06/2002</EventDate>
    <EventTime>1031704165153</EventTime>
    <DestHostName>paluk.che.sc.edu</DestHostName>
    <DestPort>5744</DestPort>
    <UserName>mitch</UserName>
  </event>
  <AgentChain>ms1</AgentChain>
  <Flagged>yes</Flagged>
  <UserNames>[mitch, mike]</UserNames>
</mnss-alert-abt-user>
```

The remaining alert messages exchanged among other Agents are similar to this alert message except the connection event details.

7.3.3 Test case 03

Test case Scenario:

Agent “ms1” senses a suspicious login connection event as explained in Section 6.3. It sends a query to the source machine of that connection using Query Suspicious Login Protocol. The source machine Agent cannot find information about the user in its logged events. So it sends a failed response as it is unable to provide the information. But it tries to find the user information from the source machine of the connection from which the user got connected to its machine. Upon receiving the reply that user is unauthorized, Agent checks if the user got connected to any other machine after his login event. Obviously it finds at least a single connection to the host machine “ms1”. So ultimately Agent “ms1” received the information about the user without waiting in a loop until the
queried Agent finds more information about the user. In that way, the messaging is highly asynchronous.

**Data before running the test:**

**EventsTable for “ms1”:**

<table>
<thead>
<tr>
<th>EVENTID</th>
<th>TYPEOF EVENT</th>
<th>SOURCE HOSTNAME</th>
<th>SOURCE PORT</th>
<th>DEST HOSTNAME</th>
<th>DEST PORT</th>
</tr>
</thead>
<tbody>
<tr>
<td>E1031893143243</td>
<td>4</td>
<td>rose.oo.org</td>
<td>2398</td>
<td>paluk.che.sc.edu</td>
<td>2383</td>
</tr>
<tr>
<td>E1031895895241</td>
<td>4</td>
<td>paluk.che.sc.edu</td>
<td>4383</td>
<td><a href="http://www.reddy.net">www.reddy.net</a></td>
<td>6793</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>EVENTDATE</th>
<th>EVENTTIME</th>
<th>USER NAME</th>
<th>DONE FLAG</th>
<th>COMMENTS</th>
<th>AGENTS CHAIN</th>
</tr>
</thead>
<tbody>
<tr>
<td>09/07/2002</td>
<td>1031893134332</td>
<td>kimi</td>
<td>0</td>
<td>null</td>
<td>null</td>
</tr>
<tr>
<td>09/07/2002</td>
<td>1031895894331</td>
<td>kimi</td>
<td>1</td>
<td>null</td>
<td>null</td>
</tr>
</tbody>
</table>

**LoginEventsTable for “ms1”:**

<table>
<thead>
<tr>
<th>EVENTID</th>
<th>USERNAME</th>
<th>LIST OF COMMANDS</th>
</tr>
</thead>
<tbody>
<tr>
<td>E1031893143243</td>
<td>kimi</td>
<td>null</td>
</tr>
<tr>
<td>E1031895895241</td>
<td>kimi</td>
<td>null</td>
</tr>
</tbody>
</table>

**UsersTable for “ms1”:**

<table>
<thead>
<tr>
<th>USERNAME</th>
<th>AUTHORIZED</th>
<th>FLAGGED</th>
</tr>
</thead>
</table>

**EventsTable for “ms2”:**

<table>
<thead>
<tr>
<th>EVENTID</th>
<th>TYPEOF EVENT</th>
<th>SOURCE HOSTNAME</th>
<th>SOURCE PORT</th>
<th>DEST HOSTNAME</th>
<th>DEST PORT</th>
</tr>
</thead>
<tbody>
<tr>
<td>E1031892095244</td>
<td>4</td>
<td>rose.oo.org</td>
<td>3483</td>
<td><a href="http://www.dty.com">www.dty.com</a></td>
<td>4143</td>
</tr>
<tr>
<td>E1031892143243</td>
<td>4</td>
<td>rose.oo.org</td>
<td>2398</td>
<td>paluk.che.sc.edu</td>
<td>2383</td>
</tr>
<tr>
<td>E1031892095243</td>
<td>4</td>
<td>chite.tr.ttu.com</td>
<td>9878</td>
<td>rose.oo.org</td>
<td>2343</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>EVENTDATE</th>
<th>EVENTTIME</th>
<th>USER NAME</th>
<th>DONE FLAG</th>
<th>COMMENTS</th>
<th>AGENTS CHAIN</th>
</tr>
</thead>
<tbody>
<tr>
<td>09/07/2002</td>
<td>1031892594331</td>
<td>kimi</td>
<td>1</td>
<td>null</td>
<td>null</td>
</tr>
<tr>
<td>09/07/2002</td>
<td>1031892134332</td>
<td>kimi</td>
<td>1</td>
<td>null</td>
<td>null</td>
</tr>
<tr>
<td>09/07/2002</td>
<td>1031892094332</td>
<td>kimi</td>
<td>1</td>
<td>null</td>
<td>null</td>
</tr>
</tbody>
</table>
LoginEventsTable for “ms2”:

<table>
<thead>
<tr>
<th>EVENTID</th>
<th>USERNAME</th>
<th>LIST OF COMMANDS</th>
</tr>
</thead>
<tbody>
<tr>
<td>E1031892095244</td>
<td>kimi</td>
<td>null</td>
</tr>
<tr>
<td>E1031892143243</td>
<td>kimi</td>
<td>null</td>
</tr>
<tr>
<td>E1031892095243</td>
<td>kimi</td>
<td>null</td>
</tr>
</tbody>
</table>

UsersTable for “ms2”:

<table>
<thead>
<tr>
<th>USERNAME</th>
<th>AUTHORIZED</th>
<th>FLAGGED</th>
</tr>
</thead>
</table>

EventsTable for “ms3”:

<table>
<thead>
<tr>
<th>EVENTID</th>
<th>TYPEOF EVENT</th>
<th>SOURCE HOSTNAME</th>
<th>SOURCE PORT</th>
<th>DEST HOSTNAME</th>
<th>DEST PORT</th>
<th>EVENTDATE</th>
<th>EVENTTIME</th>
<th>USERNAME</th>
<th>DONE</th>
<th>FLAG</th>
<th>COMMENTS</th>
<th>AGENTS CHAIN</th>
</tr>
</thead>
<tbody>
<tr>
<td>E1031891595241</td>
<td>4</td>
<td>chite.tr.ttu.com</td>
<td>9878</td>
<td>rose.oo.org</td>
<td>2343</td>
<td>09/07/2002</td>
<td>1031891594331</td>
<td>kimi</td>
<td>1</td>
<td></td>
<td>null</td>
<td>null</td>
</tr>
</tbody>
</table>

LoginEventsTable for “ms3”:

<table>
<thead>
<tr>
<th>EVENTID</th>
<th>USERNAME</th>
<th>LIST OF COMMANDS</th>
</tr>
</thead>
<tbody>
<tr>
<td>E1031891595241</td>
<td>kimi</td>
<td>null</td>
</tr>
</tbody>
</table>

UsersTable for “ms3”:

<table>
<thead>
<tr>
<th>USERNAME</th>
<th>AUTHORIZED</th>
<th>FLAGGED</th>
</tr>
</thead>
</table>

EventsTable for “ms4”:

<table>
<thead>
<tr>
<th>EVENTID</th>
<th>TYPEOF EVENT</th>
<th>SOURCE HOSTNAME</th>
<th>SOURCE PORT</th>
<th>DEST HOSTNAME</th>
<th>DEST PORT</th>
<th>EVENTDATE</th>
<th>EVENTTIME</th>
<th>USERNAME</th>
<th>DONE</th>
<th>FLAG</th>
<th>COMMENTS</th>
<th>AGENTS CHAIN</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
LoginEventsTable for “ms4”:

<table>
<thead>
<tr>
<th>EVENTID</th>
<th>USERNAME</th>
<th>LIST OF COMMANDS</th>
</tr>
</thead>
</table>

UsersTable for “ms4”:

<table>
<thead>
<tr>
<th>USERNAME</th>
<th>AUTHORIZED</th>
<th>FLAGGED</th>
</tr>
</thead>
</table>

EventsTable for “ms6”:

<table>
<thead>
<tr>
<th>EVENTID</th>
<th>TYPEOF EVENT</th>
<th>SOURCE HOSTNAME</th>
<th>SOURCE PORT</th>
<th>DEST HOSTNAME</th>
<th>DEST PORT</th>
<th>EVENTDATE</th>
<th>EVENTTIME</th>
<th>USERNAME</th>
<th>DONE</th>
<th>FLAG</th>
<th>COMMENTS</th>
<th>AGENTS</th>
<th>CHAIN</th>
</tr>
</thead>
</table>

LoginEventsTable for “ms6”:

<table>
<thead>
<tr>
<th>EVENTID</th>
<th>USERNAME</th>
<th>LIST OF COMMANDS</th>
</tr>
</thead>
</table>

UsersTable for “ms6”:

<table>
<thead>
<tr>
<th>USERNAME</th>
<th>AUTHORIZED</th>
<th>FLAGGED</th>
</tr>
</thead>
</table>

Results:

The Agent “ms1” reads the event (EventID “E1031893143243”) and sends a query about the suspicious login under username “kimi”, to Agent “ms2”. The Agent “ms2” checks in its EventsTable and UsersTable for relevant information. Though it could find the connection, it does not have any information about the user in its UsersTable. It replies that it cannot find much information. At the same time it checks in its logged events the source machine of the connection through which the user has logged on to its host. It finds the connection from host machine, “ms3” (EventID “E1031892095243”) and
queries the Agent “ms3”. The Agent “ms3” checks in its UsersTable and replies that “user is not authorized”. Upon receiving the reply, the Agent ms2 checks in its logged events if the user got connected to other machines after his login event to its host. It finds connections to the host machines, “ms1”, “ms4” (EventIDs “E1031892143243”, “E1031892095244”) and sends an alert about the user to “ms1” and “ms4”. Upon receiving the alerts, the Agents “ms1” and “ms4” checks in their logged events for the user involved connections. The Agent “ms1” finds a connection to host machine “ms6” (EventID “E1031895895241”) and sends a user alert to that Agent. The Agent “ms6” repeats the same but cannot find any connection and the transfer of alert message stops.

**Data after running the test:**

**EventsTable for “ms1”:**

<table>
<thead>
<tr>
<th>EVENTID</th>
<th>TYPEOF EVENT</th>
<th>SOURCE HOSTNAME</th>
<th>SOURCE PORT</th>
<th>DEST HOSTNAME</th>
<th>DEST PORT</th>
</tr>
</thead>
<tbody>
<tr>
<td>E1031893143243</td>
<td>4</td>
<td>rose.oo.org</td>
<td>2398</td>
<td>paluk.che.sc.edu</td>
<td>2383</td>
</tr>
<tr>
<td>E1031895895241</td>
<td>4</td>
<td>paluk.che.sc.edu</td>
<td>4383</td>
<td><a href="http://www.reddy.net">www.reddy.net</a></td>
<td>6793</td>
</tr>
<tr>
<td>E1036302563387</td>
<td>4</td>
<td>rose.oo.org</td>
<td>2398</td>
<td>paluk.che.sc.edu</td>
<td>2383</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>EVENTDATE</th>
<th>EVENTTIME</th>
<th>USERNAME</th>
<th>DONE FLAG</th>
<th>COMMENTS</th>
<th>AGENTS CHAIN</th>
</tr>
</thead>
<tbody>
<tr>
<td>09/07/2002</td>
<td>1031893134332</td>
<td>kimi</td>
<td>5</td>
<td>null</td>
<td>null</td>
</tr>
<tr>
<td>09/07/2002</td>
<td>1031895894331</td>
<td>kimi</td>
<td>1</td>
<td>null</td>
<td>ms2-ms6-ms1</td>
</tr>
<tr>
<td>09/07/2002</td>
<td>1031893134332</td>
<td>kimi</td>
<td>1</td>
<td>null</td>
<td>ms2</td>
</tr>
</tbody>
</table>

**LoginEventsTable for “ms1”:**

<table>
<thead>
<tr>
<th>EVENTID</th>
<th>USERNAME</th>
<th>LIST OF COMMANDS</th>
</tr>
</thead>
<tbody>
<tr>
<td>E1031893143243</td>
<td>kimi</td>
<td>null</td>
</tr>
<tr>
<td>E1031895895241</td>
<td>kimi</td>
<td>null</td>
</tr>
<tr>
<td>E1036302563387</td>
<td>kimi</td>
<td>null</td>
</tr>
</tbody>
</table>
### UsersTable for “ms1”:

<table>
<thead>
<tr>
<th>USERNAME</th>
<th>AUTHORIZED</th>
<th>FLAGGED</th>
</tr>
</thead>
<tbody>
<tr>
<td>kimi</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

### EventsTable for “ms2”:

<table>
<thead>
<tr>
<th>EVENTID</th>
<th>TYPEOF EVENT</th>
<th>SOURCE HOSTNAME</th>
<th>SOURCE PORT</th>
<th>DEST HOSTNAME</th>
<th>DEST PORT</th>
</tr>
</thead>
<tbody>
<tr>
<td>E1031892095244</td>
<td>4</td>
<td>rose.oo.org</td>
<td>3483</td>
<td><a href="http://www.dty.com">www.dty.com</a></td>
<td>4143</td>
</tr>
<tr>
<td>E1031892143243</td>
<td>4</td>
<td>rose.oo.org</td>
<td>2398</td>
<td>paluk.che.sc.edu</td>
<td>2383</td>
</tr>
<tr>
<td>E1031892095243</td>
<td>4</td>
<td>chite.tr.ttu.com</td>
<td>9878</td>
<td>rose.oo.org</td>
<td>2343</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>EVENTDATE</th>
<th>EVENTTIME</th>
<th>USER NAME</th>
<th>DONE FLAG</th>
<th>COMMENTS</th>
<th>AGENTS CHAIN</th>
</tr>
</thead>
<tbody>
<tr>
<td>09/07/2002</td>
<td>1031892594331</td>
<td>kimi</td>
<td>1</td>
<td>null</td>
<td>ms2-ms4</td>
</tr>
<tr>
<td>09/07/2002</td>
<td>1031892134332</td>
<td>kimi</td>
<td>1</td>
<td>null</td>
<td>ms2-ms1</td>
</tr>
<tr>
<td>09/07/2002</td>
<td>1031892094332</td>
<td>kimi</td>
<td>1</td>
<td>null</td>
<td>null</td>
</tr>
</tbody>
</table>

### LoginEventsTable for “ms2”:

<table>
<thead>
<tr>
<th>EVENTID</th>
<th>USERNAME</th>
<th>LIST OF COMMANDS</th>
</tr>
</thead>
<tbody>
<tr>
<td>E1031892095244</td>
<td>kimi</td>
<td>null</td>
</tr>
<tr>
<td>E1031892143243</td>
<td>kimi</td>
<td>null</td>
</tr>
<tr>
<td>E1031892095243</td>
<td>kimi</td>
<td>null</td>
</tr>
</tbody>
</table>

### UsersTable for “ms2”:

<table>
<thead>
<tr>
<th>USERNAME</th>
<th>AUTHORIZED</th>
<th>FLAGGED</th>
</tr>
</thead>
<tbody>
<tr>
<td>kimi</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

### EventsTable for “ms4”:

<table>
<thead>
<tr>
<th>EVENTID</th>
<th>TYPEOF EVENT</th>
<th>SOURCE HOSTNAME</th>
<th>SOURCE PORT</th>
<th>DEST HOSTNAME</th>
<th>DEST PORT</th>
</tr>
</thead>
<tbody>
<tr>
<td>E1036302560211</td>
<td>4</td>
<td>rose.oo.org</td>
<td>3483</td>
<td><a href="http://www.dty.com">www.dty.com</a></td>
<td>4143</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>EVENTDATE</th>
<th>EVENTTIME</th>
<th>USER NAME</th>
<th>DONE FLAG</th>
<th>COMMENTS</th>
<th>AGENTS CHAIN</th>
</tr>
</thead>
<tbody>
<tr>
<td>09/07/2002</td>
<td>1031892594331</td>
<td>kimi</td>
<td>1</td>
<td>null</td>
<td>ms2</td>
</tr>
</tbody>
</table>
LoginEventsTable for “ms4”:

<table>
<thead>
<tr>
<th>EVENTID</th>
<th>USERNAME</th>
<th>LIST OF COMMANDS</th>
</tr>
</thead>
<tbody>
<tr>
<td>E1036302560211</td>
<td>kimi</td>
<td>null</td>
</tr>
</tbody>
</table>

UsersTable for “ms4”:

<table>
<thead>
<tr>
<th>USERNAME</th>
<th>AUTHORIZED</th>
<th>FLAGGED</th>
</tr>
</thead>
<tbody>
<tr>
<td>Kimi</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

EventsTable for “ms6”:

<table>
<thead>
<tr>
<th>EVENTID</th>
<th>TYPEOF EVENT</th>
<th>SOURCE HOSTNAME</th>
<th>SOURCE PORT</th>
<th>DEST HOSTNAME</th>
<th>DEST PORT</th>
</tr>
</thead>
<tbody>
<tr>
<td>E1036302565001</td>
<td>4</td>
<td>paluk.che.sc.edu</td>
<td>4383</td>
<td><a href="http://www.reddy.net">www.reddy.net</a></td>
<td>6793</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>EVENTDATE</th>
<th>EVENTTIME</th>
<th>USERNAME</th>
<th>DONE</th>
<th>FLAG</th>
<th>COMMENTS</th>
<th>AGENTS CHAIN</th>
</tr>
</thead>
<tbody>
<tr>
<td>09/07/2002</td>
<td>1031895894331</td>
<td>kimi</td>
<td>1</td>
<td></td>
<td>null</td>
<td>ms2-ms1</td>
</tr>
</tbody>
</table>

LoginEventsTable for “ms6”:

<table>
<thead>
<tr>
<th>EVENTID</th>
<th>USERNAME</th>
<th>LIST OF COMMANDS</th>
</tr>
</thead>
<tbody>
<tr>
<td>E1036302565001</td>
<td>kimi</td>
<td>null</td>
</tr>
</tbody>
</table>

UsersTable for “ms4”:

<table>
<thead>
<tr>
<th>USERNAME</th>
<th>AUTHORIZED</th>
<th>FLAGGED</th>
</tr>
</thead>
<tbody>
<tr>
<td>Kimi</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

Messages Exchanged: The messages exchanged are similar to the Test Case 01 except the connection details. So only crucial messages exchanged among Agents are shown below.

Query Message sent from “ms1” to “ms2” Agent:

```xml
<?xml version="1.0"?>
<mnss-susp-login>
```
<event>
  <SourceHostName>rose.oo.org</SourceHostName>
  <SourcePort>2398</SourcePort>
  <DestHostName>paluk.che.sc.edu</DestHostName>
  <DestPort>2383</DestPort>
  <EventDate>09/07/2002</EventDate>
  <EventTime>1031893134332</EventTime>
</event>
<UserName>kimi</UserName>
</mnss-susp-login>

**Reply Message sent from “ms2” to “ms1” Agent :**

No username is sent to imply that it does not have enough information about the user to reply.

<?xml version="1.0"?>
<mnss-susp-login>
  <event>
    <SourceHostName>rose.oo.org</SourceHostName>
    <SourcePort>2398</SourcePort>
    <DestHostName>paluk.che.sc.edu</DestHostName>
    <DestPort>2383</DestPort>
    <EventDate>09/07/2002</EventDate>
    <EventTime>1031893134332</EventTime>
  </event>
</mnss-susp-login>

**Query Message sent from “ms2” to “ms3” Agent :**

<?xml version="1.0"?>
<mnss-susp-login>
  <event>
    <SourceHostName>chite.tr.ttu.com</SourceHostName>
    <SourcePort>9878</SourcePort>
    <DestHostName>rose.oo.org</DestHostName>
    <DestPort>2343</DestPort>
    <EventDate>09/07/2002</EventDate>
    <EventTime>1031892094332</EventTime>
  </event>
  <UserName>kimi</UserName>
</mnss-susp-login>
Reply Message sent from “ms3” to “ms2” Agent:

<?xml version="1.0"?>
<mnss-susp-login>
  <event>
    <SourceHostName>chite.tr.ttu.com</SourceHostName>
    <SourcePort>9878</SourcePort>
    <DestHostName>rose.oo.org</DestHostName>
    <DestPort>2343</DestPort>
    <EventDate>09/07/2002</EventDate>
    <EventTime>1031891594331</EventTime>
  </event>
  <UserName>kimi</UserName>
  <Flagged>yes</Flagged>
</mnss-susp-login>

Alert Message sent from “ms2” to “ms1” Agent:

<?xml version="1.0"?>
<mnss-alert-abt-user>
  <event>
    <SourceHostName>rose.oo.org</SourceHostName>
    <SourcePort>2398</SourcePort>
    <DestHostName>paluk.che.sc.edu</DestHostName>
    <DestPort>2383</DestPort>
    <UserName>kimi</UserName>
    <EventDate>09/07/2002</EventDate>
    <EventTime>1031892134332</EventTime>
    <AgentChain>ms2</AgentChain>
    <Flagged>yes</Flagged>
    <UserNames>[kimi]</UserNames>
  </event>
</mnss-alert-abt-user>

Alert Message sent from “ms2” to “ms4” Agent:

Similar to the above alert message except the connection event details

Alert Message sent from “ms1” to “ms6” Agent:

<?xml version="1.0"?>
<mnss-alert-abt-user>
  <event>
    <EventDate>09/07/2002</EventDate>
    <EventTime>1031895894331</EventTime>
    <DestHostName>www.reddy.net</DestHostName>
    <DestPort>6793</DestPort>
  </event>
</mnss-alert-abt-user>
7.3.4 Test case 04

Test case Scenario:

This test case checks the possibility of **loops** during the propagation of an alert. Agent “ms2” senses a suspicious login connection event as explained in Section 6.3. It sends a query to the source machine of that connection using Query Suspicious Login Protocol. The source machine Agent replies that the user is authorized. Upon receiving the reply, Agent “ms2” finds an incompatibility in usernames, it flags the user and sends an alert to the source machine of the connection. At the same time it checks for any connections made by the user after his login event to its host. If it finds any, it sends an alert about the user to the Agents of the respective host machines. Upon receiving the alerts, those Agents find connections to “ms2” and may send alerts back to “ms2”. Any agent that receives the user alert checks for the connections made by the user after his login event to its host. So again “ms2” may send an alert to the machines from which it received the alert. This will result in alert loops and unnecessary communications.

For example, in the figure 7.1, the attacker logs from ms4 -> ms3 -> ms2 -> ms5 -> ms2. After some time he logs again from ms3 to ms2 again. The attacker login path is shown in dashed lines and the alert path is shown in solid lines. The Agent “ms2” flags the user...
and generates an alert to the source machine “ms3” and destination machine “ms5”. The Agent “ms3” propagates the alert to the source machine “ms4”. At the same time it finds a connection to “ms2” in its log files. But it should not send an alert back to “ms2”, as “ms2” is already aware of that alert. Similarly, the Agent “ms5” finds a connection to Agent “ms2” but should not send an alert back to “ms2”. If it sends an alert to “ms2”, “ms2” checks in its log files and finds a connection to “ms5”. So it will send an alert again to “ms5” and “ms5” will send an alert back to “ms2”. This may continue forever causing a loop of alerts.

Figure 7.1. Illustration of Alert Loops

To avoid such alert loops, the following measures are taken by every Agent:

1. Before propagating an alert to some receiver Agent, the sender Agent checks if the agent name to which it is going to send an alert, exists in the AgentsChain of the alert. If it exists, then it implies that the alert has already been propagated by that receiver agent and so the receiver agent is already aware of the alert. In that case the sender Agent no longer sends that alert to the receiver Agent.
2. After propagating an alert about some connection Event, the sender Agent updates the AgentsChain of that connection Event to the agent names of receiver Agents. In that way, the sender Agent will be careful not generating or propagating the alert about the same connection Event to same receiver Agents more than once.

Data before running the test:

EventsTable for “ms2”:

<table>
<thead>
<tr>
<th>EVENTID</th>
<th>TYPEOF EVENT</th>
<th>SOURCE HOSTNAME</th>
<th>SOURCE PORT</th>
<th>DEST HOSTNAME</th>
<th>DEST PORT</th>
</tr>
</thead>
<tbody>
<tr>
<td>E1032492143243</td>
<td>4</td>
<td>chite.tr.ttu.com</td>
<td>3743</td>
<td>rose.oo.org</td>
<td>3899</td>
</tr>
<tr>
<td>E1032492145673</td>
<td>4</td>
<td>rose.oo.org</td>
<td>8391</td>
<td><a href="http://www.lily.com">www.lily.com</a></td>
<td>3843</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>EVENTDATE</th>
<th>EVENTTIME</th>
<th>USERNAME</th>
<th>DONE</th>
<th>COMMENTS</th>
<th>AGENTS CHAIN</th>
</tr>
</thead>
<tbody>
<tr>
<td>09/08/2002</td>
<td>1032492143256</td>
<td>pran</td>
<td>0</td>
<td>null</td>
<td>null</td>
</tr>
<tr>
<td>09/08/2002</td>
<td>1032492145673</td>
<td>pran</td>
<td>1</td>
<td>null</td>
<td>null</td>
</tr>
</tbody>
</table>

LoginEventsTable for “ms2”:

<table>
<thead>
<tr>
<th>EVENTID</th>
<th>USERNAME</th>
<th>LIST OF COMMANDS</th>
</tr>
</thead>
<tbody>
<tr>
<td>E1032492145673</td>
<td>pran</td>
<td>null</td>
</tr>
<tr>
<td>E1032492143243</td>
<td>pran</td>
<td>null</td>
</tr>
</tbody>
</table>

UsersTable for “ms2”:

<table>
<thead>
<tr>
<th>USERNAME</th>
<th>AUTHORIZED</th>
<th>FLAGGED</th>
</tr>
</thead>
</table>

EventsTable for “ms3”:

<table>
<thead>
<tr>
<th>EVENTID</th>
<th>TYPEOF EVENT</th>
<th>SOURCE HOSTNAME</th>
<th>SOURCE PORT</th>
<th>DEST HOSTNAME</th>
<th>DEST PORT</th>
</tr>
</thead>
<tbody>
<tr>
<td>E1032492142101</td>
<td>4</td>
<td>chite.tr.ttu.com</td>
<td>3743</td>
<td>rose.oo.org</td>
<td>3899</td>
</tr>
<tr>
<td>E1032492142564</td>
<td>4</td>
<td>chite.tr.ttu.com</td>
<td>4374</td>
<td>rose.oo.org</td>
<td>4493</td>
</tr>
<tr>
<td>E1032492141101</td>
<td>4</td>
<td><a href="http://www.dty.com">www.dty.com</a></td>
<td>6784</td>
<td>chite.tr.ttu.com</td>
<td>3833</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>EVENTDATE</th>
<th>EVENTTIME</th>
<th>USERNAME</th>
<th>DONE</th>
<th>COMMENTS</th>
<th>AGENTS CHAIN</th>
</tr>
</thead>
</table>
LoginEventsTable for “ms3”:

<table>
<thead>
<tr>
<th>EVENTID</th>
<th>USERNAME</th>
<th>LIST OF COMMANDS</th>
</tr>
</thead>
<tbody>
<tr>
<td>E1032492142564</td>
<td>cheta</td>
<td>null</td>
</tr>
<tr>
<td>E1032492142101</td>
<td>cheta</td>
<td>null</td>
</tr>
</tbody>
</table>

UsersTable for “ms3”:

<table>
<thead>
<tr>
<th>USERNAME</th>
<th>AUTHORIZED</th>
<th>FLAGGED</th>
</tr>
</thead>
<tbody>
<tr>
<td>cheta</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

EventsTable for “ms5”:

<table>
<thead>
<tr>
<th>EVENTID</th>
<th>TYPEOF EVENT</th>
<th>SOURCE HOSTNAME</th>
<th>SOURCE PORT</th>
<th>DEST HOSTNAME</th>
<th>DEST PORT</th>
</tr>
</thead>
<tbody>
<tr>
<td>E1032492145987</td>
<td>4</td>
<td><a href="http://www.lily.com">www.lily.com</a></td>
<td>4743</td>
<td>rose.oo.org</td>
<td>3485</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>EVENTDATE</th>
<th>EVENTTIME</th>
<th>USERNAME</th>
<th>DONE FLAG</th>
<th>COMMENTS</th>
<th>AGENTS CHAIN</th>
</tr>
</thead>
<tbody>
<tr>
<td>09/08/2002</td>
<td>1032492145987</td>
<td>pran</td>
<td>1</td>
<td>null</td>
<td>null</td>
</tr>
</tbody>
</table>

LoginEventsTable for “ms5”:

<table>
<thead>
<tr>
<th>EVENTID</th>
<th>USERNAME</th>
<th>LIST OF COMMANDS</th>
</tr>
</thead>
<tbody>
<tr>
<td>E1032492145987</td>
<td>pran</td>
<td>null</td>
</tr>
</tbody>
</table>

UsersTable for “ms5”:

<table>
<thead>
<tr>
<th>USERNAME</th>
<th>AUTHORIZED</th>
<th>FLAGGED</th>
</tr>
</thead>
</table>

EventsTable for “ms4”:

<table>
<thead>
<tr>
<th>EVENTID</th>
<th>TYPEOF EVENT</th>
<th>SOURCE HOSTNAME</th>
<th>SOURCE PORT</th>
<th>DEST HOSTNAME</th>
<th>DEST PORT</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>EVENTDATE</th>
<th>EVENTTIME</th>
<th>USERNAME</th>
<th>DONE FLAG</th>
<th>COMMENTS</th>
<th>AGENTS CHAIN</th>
</tr>
</thead>
</table>
LoginEventsTable for “ms4”:

<table>
<thead>
<tr>
<th>EVENTID</th>
<th>USERNAME</th>
<th>LIST OF COMMANDS</th>
</tr>
</thead>
</table>

UsersTable for “ms4”:

<table>
<thead>
<tr>
<th>USERNAME</th>
<th>AUTHORIZED</th>
<th>FLAGGED</th>
</tr>
</thead>
</table>

Results:

The Agent “ms2” reads the event (EventID “E1032492143243”) and queries the source machine of the connection “ms3” about the username of the connection. The Agent “ms3” checks in its UsersTable and replies that user, “cheta”, is authorized. Agent “ms2” finds incompatibility in usernames and sends an alert to the source machine, “ms2”, about usernames, “pran/cheta”. At the same time it finds that user got connected to host machine, “ms5”, (EventID “E1032492145673”) after his login event and sends an alert about the usernames to “ms5” Agent. Upon receiving the alert, “ms5” Agent finds that user got connected to the host machine “ms2” (EventID “E1032492145987”). But it checks in its AgentsChain and finds that “ms2” exists in the AgentsChain of the alert. So it no longer propagates the alert to “ms2”. The Agent “ms3” receives an alert from “ms2” about usernames “pran/cheta”. The Agent checks in its logged events if the user got connected to any other machines after his login event. Also it checks if the user has logged on to its machine from other machine. It finds such event from host machine “ms4” (EventID “E1032492141101”) and sends an alert back to “ms4” Agent. At the same time the Agent “ms3” finds a connection to “ms2” (EventID “E1032492142564”). But it checks the AgentsChain of the alert and finds that “ms2” exists among the agentnames in the AgentsChain. So it no longer propagates the alert to “ms2”. The
Agent “ms4” does not find any user involved events and so the transfer of alert message stops.

Data after running the test:

EventsTable for “ms2”:

<table>
<thead>
<tr>
<th>EVENTID</th>
<th>TYPEOF EVENT</th>
<th>SOURCE HOSTNAME</th>
<th>SOURCE PORT</th>
<th>DEST HOSTNAME</th>
<th>DEST PORT</th>
</tr>
</thead>
<tbody>
<tr>
<td>E1032492143243</td>
<td>4</td>
<td>chite.tr.ttu.com</td>
<td>3743</td>
<td>rose.oo.org</td>
<td>3899</td>
</tr>
<tr>
<td>E1032492145673</td>
<td>4</td>
<td>rose.oo.org</td>
<td>8391</td>
<td><a href="http://www.lily.com">www.lily.com</a></td>
<td>3843</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>EVENTDATE</th>
<th>EVENTTIME</th>
<th>USER NAME</th>
<th>DONE FLAG</th>
<th>COMMENTS</th>
<th>AGENTS CHAIN</th>
</tr>
</thead>
<tbody>
<tr>
<td>09/08/2002</td>
<td>1032492143256</td>
<td>pran</td>
<td>1</td>
<td>null</td>
<td>ms2-ms3</td>
</tr>
<tr>
<td>09/08/2002</td>
<td>1032492145673</td>
<td>pran</td>
<td>1</td>
<td>null</td>
<td>ms2-ms5</td>
</tr>
</tbody>
</table>

LoginEventsTable for “ms2”:

<table>
<thead>
<tr>
<th>EVENTID</th>
<th>USERNAME</th>
<th>LIST OF COMMANDS</th>
</tr>
</thead>
<tbody>
<tr>
<td>E1032492145673</td>
<td>pran</td>
<td>null</td>
</tr>
<tr>
<td>E1032492143243</td>
<td>pran</td>
<td>null</td>
</tr>
</tbody>
</table>

UsersTable for “ms2”:

<table>
<thead>
<tr>
<th>USERNAME</th>
<th>AUTHORIZED</th>
<th>FLAGGED</th>
</tr>
</thead>
<tbody>
<tr>
<td>pran</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>cheta</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

EventsTable for “ms3”:

<table>
<thead>
<tr>
<th>EVENTID</th>
<th>TYPEOF EVENT</th>
<th>SOURCE HOSTNAME</th>
<th>SOURCE PORT</th>
<th>DEST HOSTNAME</th>
<th>DEST PORT</th>
</tr>
</thead>
<tbody>
<tr>
<td>E1032492141101</td>
<td>4</td>
<td><a href="http://www.dty.com">www.dty.com</a></td>
<td>6784</td>
<td>chite.tr.ttu.com</td>
<td>3833</td>
</tr>
<tr>
<td>E1037844814391</td>
<td>4</td>
<td>chite.tr.ttu.com</td>
<td>3743</td>
<td>rose.oo.org</td>
<td>3899</td>
</tr>
<tr>
<td>E1037844815404</td>
<td>4</td>
<td>chite.tr.ttu.com</td>
<td>3743</td>
<td>rose.oo.org</td>
<td>3899</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>EVENTDATE</th>
<th>EVENTTIME</th>
<th>USER NAME</th>
<th>DONE FLAG</th>
<th>COMMENTS</th>
<th>AGENTS CHAIN</th>
</tr>
</thead>
<tbody>
<tr>
<td>09/08/2002</td>
<td>1032492141101</td>
<td>cheta</td>
<td>1</td>
<td>null</td>
<td>ms2-ms3-ms4</td>
</tr>
<tr>
<td>09/08/2002</td>
<td>1032492143256</td>
<td>pran</td>
<td>1</td>
<td>null</td>
<td>ms2</td>
</tr>
</tbody>
</table>
### LoginEventsTable for “ms3”:

<table>
<thead>
<tr>
<th>EVENTID</th>
<th>USERNAME</th>
<th>LIST OF COMMANDS</th>
</tr>
</thead>
<tbody>
<tr>
<td>E103784814391</td>
<td>cheta</td>
<td>null</td>
</tr>
<tr>
<td>E103784815404</td>
<td>pran</td>
<td>null</td>
</tr>
</tbody>
</table>

### UsersTable for “ms3”:

<table>
<thead>
<tr>
<th>USERNAME</th>
<th>AUTHORIZED</th>
<th>FLAGGED</th>
</tr>
</thead>
<tbody>
<tr>
<td>cheta</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>pran</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

### EventsTable for “ms5”:

<table>
<thead>
<tr>
<th>EVENTID</th>
<th>TYPEOF EVENT</th>
<th>SOURCE HOSTNAME</th>
<th>SOURCE PORT</th>
<th>DEST HOSTNAME</th>
<th>DEST PORT</th>
</tr>
</thead>
<tbody>
<tr>
<td>E1032492145987</td>
<td>4</td>
<td><a href="http://www.lily.com">www.lily.com</a></td>
<td>4743</td>
<td>rose.oo.org</td>
<td>3485</td>
</tr>
<tr>
<td>E1037844818569</td>
<td>4</td>
<td>rose.oo.org</td>
<td>8391</td>
<td><a href="http://www.lily.com">www.lily.com</a></td>
<td>3843</td>
</tr>
<tr>
<td>E1037844817584</td>
<td>4</td>
<td>rose.oo.org</td>
<td>8391</td>
<td><a href="http://www.lily.com">www.lily.com</a></td>
<td>3843</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>EVENTDATE</th>
<th>EVENTTIME</th>
<th>USERNAME</th>
<th>DONE</th>
<th>FLAG</th>
<th>COMMENTS</th>
<th>AGENTS CHAIN</th>
</tr>
</thead>
<tbody>
<tr>
<td>09/08/2002</td>
<td>1032492145987</td>
<td>pran</td>
<td>1</td>
<td></td>
<td>null</td>
<td>ms5-ms2</td>
</tr>
<tr>
<td>09/08/2002</td>
<td>1032492145673</td>
<td>pran</td>
<td>1</td>
<td></td>
<td>null</td>
<td>ms2</td>
</tr>
<tr>
<td>09/08/2002</td>
<td>1032492145673</td>
<td>pran</td>
<td>1</td>
<td></td>
<td>null</td>
<td>ms2</td>
</tr>
</tbody>
</table>

### LoginEventsTable for “ms5”:

<table>
<thead>
<tr>
<th>EVENTID</th>
<th>USERNAME</th>
<th>LIST OF COMMANDS</th>
</tr>
</thead>
<tbody>
<tr>
<td>E1032492145987</td>
<td>pran</td>
<td>null</td>
</tr>
<tr>
<td>E1037844818569</td>
<td>cheta</td>
<td>null</td>
</tr>
<tr>
<td>E1037844817584</td>
<td>pran</td>
<td>null</td>
</tr>
</tbody>
</table>

### UsersTable for “ms5”:

<table>
<thead>
<tr>
<th>USERNAME</th>
<th>AUTHORIZED</th>
<th>FLAGGED</th>
</tr>
</thead>
<tbody>
<tr>
<td>pran</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>cheta</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
EventsTable for “ms4” :

<table>
<thead>
<tr>
<th>EVENTID</th>
<th>TYPEOF EVENT</th>
<th>SOURCE HOSTNAME</th>
<th>SOURCE PORT</th>
<th>DEST HOSTNAME</th>
<th>DEST PORT</th>
</tr>
</thead>
<tbody>
<tr>
<td>E1037844818152</td>
<td>4</td>
<td><a href="http://www.dty.com">www.dty.com</a></td>
<td>6784</td>
<td>chite.tr.ttu.com</td>
<td>3833</td>
</tr>
<tr>
<td>E1037844819028</td>
<td>4</td>
<td><a href="http://www.dty.com">www.dty.com</a></td>
<td>6784</td>
<td>chite.tr.ttu.com</td>
<td>3833</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>EVENTDATE</th>
<th>EVENTTIME</th>
<th>USER NAME</th>
<th>DONE FLAG</th>
<th>COMMENTS</th>
<th>AGENTS CHAIN</th>
</tr>
</thead>
<tbody>
<tr>
<td>09/08/2002</td>
<td>1032492141101</td>
<td>pran</td>
<td>1</td>
<td>null</td>
<td>ms2-ms3</td>
</tr>
<tr>
<td>09/08/2002</td>
<td>1032492141101</td>
<td>cheta</td>
<td>1</td>
<td>null</td>
<td>ms2-ms3</td>
</tr>
</tbody>
</table>

LoginEventsTable for “ms4” :

<table>
<thead>
<tr>
<th>EVENTID</th>
<th>USERNAME</th>
<th>LIST OF COMMANDS</th>
</tr>
</thead>
<tbody>
<tr>
<td>E1037844818152</td>
<td>pran</td>
<td>null</td>
</tr>
<tr>
<td>E1037844819028</td>
<td>cheta</td>
<td>null</td>
</tr>
</tbody>
</table>

UsersTable for “ms4” :

<table>
<thead>
<tr>
<th>USERNAME</th>
<th>AUTHORIZED</th>
<th>FLAGGED</th>
</tr>
</thead>
<tbody>
<tr>
<td>pran</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>cheta</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

Messages Exchanged : The messages exchanged are similar to the Test Case 02 except the connection event details .

7.4 Testing Query Illegal Process Protocol

7.4.1 Test case 01

Test case Scenario :
The Agent “ms2” experiences an Echo Storm attack as described in section 6.7. The Agent queries the source machine of the connection about any illegal process using Query Echo storm protocol. The source machine confirms that there is an illegal process.
associated with the event. The Agent “ms2” checks if the user associated with that event connected to other machines after his login event. If it finds any, it sends an alert about the illegal connection to those machines.

Data before running the test:

EventsTable for “ms2”:

<table>
<thead>
<tr>
<th>EVENTID</th>
<th>TYPEOF EVENT</th>
<th>SOURCE HOSTNAME</th>
<th>SOURCE PORT</th>
<th>DEST HOSTNAME</th>
<th>DEST PORT</th>
</tr>
</thead>
<tbody>
<tr>
<td>E1021754456747</td>
<td>3</td>
<td>chite.tr.ttu.com</td>
<td>5743</td>
<td>rose.oo.org</td>
<td>4894</td>
</tr>
<tr>
<td>E1021754456483</td>
<td>7</td>
<td>rose.oo.org</td>
<td>4674</td>
<td><a href="http://www.dty.com">www.dty.com</a></td>
<td>3484</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>EVENTDATE</th>
<th>EVENTTIME</th>
<th>USER NAME</th>
<th>DONE FLAG</th>
<th>COMMENTS</th>
<th>AGENTS CHAIN</th>
</tr>
</thead>
<tbody>
<tr>
<td>09/07/2002</td>
<td>1021754412839</td>
<td>shally</td>
<td>0</td>
<td>null</td>
<td>null</td>
</tr>
<tr>
<td>09/07/2002</td>
<td>1021754454393</td>
<td>shally</td>
<td>1</td>
<td>null</td>
<td>null</td>
</tr>
</tbody>
</table>

ProcessEventsTable for “ms2”:

<table>
<thead>
<tr>
<th>EVENTID</th>
<th>ISILLEGALPROCESS</th>
</tr>
</thead>
<tbody>
<tr>
<td>E1021754456747</td>
<td>0</td>
</tr>
</tbody>
</table>

IllegalProcessEventsTable for “ms2”:

UsersTable for “ms2”:

<table>
<thead>
<tr>
<th>USERNAME</th>
<th>AUTHORIZED</th>
<th>FLAGGED</th>
</tr>
</thead>
</table>

EventsTable for “ms3”:

<table>
<thead>
<tr>
<th>EVENTID</th>
<th>TYPEOF EVENT</th>
<th>SOURCE HOSTNAME</th>
<th>SOURCE PORT</th>
<th>DEST HOSTNAME</th>
<th>DEST PORT</th>
</tr>
</thead>
<tbody>
<tr>
<td>E1021751235634</td>
<td>3</td>
<td>chite.tr.ttu.com</td>
<td>5743</td>
<td>rose.oo.org</td>
<td>4894</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>EVENTDATE</th>
<th>EVENTTIME</th>
<th>USER NAME</th>
<th>DONE FLAG</th>
<th>COMMENTS</th>
<th>AGENTS CHAIN</th>
</tr>
</thead>
</table>

09/07/2002 1021751234654 shally 1 null null

**ProcessEventsTable for “ms3”:**

<table>
<thead>
<tr>
<th>EVENTID</th>
<th>ISILLEGALPROCESS</th>
</tr>
</thead>
<tbody>
<tr>
<td>E1021751235634</td>
<td>1</td>
</tr>
</tbody>
</table>

**IllegalProcessEventsTable for “ms3”:**

<table>
<thead>
<tr>
<th>EVENTID</th>
<th>ALERTFIELD</th>
<th>ALERTMESSAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>E1021751235634</td>
<td>null</td>
<td>null</td>
</tr>
</tbody>
</table>

**UsersTable for “ms3”:**

<table>
<thead>
<tr>
<th>USERNAME</th>
<th>AUTHORIZED</th>
<th>FLAGGED</th>
</tr>
</thead>
</table>

**EventsTable for “ms4”:**

<table>
<thead>
<tr>
<th>EVENTID</th>
<th>TYPEOF EVENT</th>
<th>SOURCE HOSTNAME</th>
<th>SOURCE PORT</th>
<th>DEST HOSTNAME</th>
<th>DEST PORT</th>
</tr>
</thead>
<tbody>
<tr>
<td>E1021754478439</td>
<td>7</td>
<td><a href="http://www.dty.com">www.dty.com</a></td>
<td>3478</td>
<td><a href="http://www.lily.com">www.lily.com</a></td>
<td>3844</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>EVENTDATE</th>
<th>EVENTTIME</th>
<th>USER NAME</th>
<th>DONE FLAG</th>
<th>COMMENTS</th>
<th>AGENTS CHAIN</th>
</tr>
</thead>
<tbody>
<tr>
<td>09/07/2002</td>
<td>1021754478354</td>
<td>shally</td>
<td>1</td>
<td>null</td>
<td>null</td>
</tr>
</tbody>
</table>

**IllegalConnectionEventsTable for “ms4”:**

<table>
<thead>
<tr>
<th>EVENTID</th>
</tr>
</thead>
</table>

**EventsTable for “ms5”:**

<table>
<thead>
<tr>
<th>EVENTID</th>
<th>TYPEOF EVENT</th>
<th>SOURCE HOSTNAME</th>
<th>SOURCE PORT</th>
<th>DEST HOSTNAME</th>
<th>DEST PORT</th>
</tr>
</thead>
<tbody>
<tr>
<td>E1021751235634</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>EVENTDATE</th>
<th>EVENTTIME</th>
<th>USER NAME</th>
<th>DONE FLAG</th>
<th>COMMENTS</th>
<th>AGENTS CHAIN</th>
</tr>
</thead>
</table>

95
IllegalConnectionEventsTable for “ms5”:

<table>
<thead>
<tr>
<th>EVENTID</th>
<th>ALERTFIELD</th>
<th>ALERTMESSAGE</th>
</tr>
</thead>
</table>

Results:
The Agent “ms2” reads the event (EventID “E1021754456747”) and sends a query about the illegal process to the source machine of the connection, “ms3”. The Agent “ms3” checks in its IllegalProcessEventsTable and replies that there is an illegal process. Upon receiving the reply, Agent “ms2” updates its IllegalProcessEventsTable and ProcessEventsTable. It flags the user associated with the connection and updates its UsersTable. At the same time it checks if the user has connected to any other machines after his login event. It finds a connection to the host machine, “ms4” (EventID “E1021754456483”) and sends an illegal connection alert to “ms4” Agent. Upon receiving the alert, the Agent “ms4” checks if the user has connected to any other machines after his login event. It finds a connection to host machine “ms5” (EventID “E1021754478439”) and sends an illegal connection alert to “ms5”. The Agent “ms5” repeats the same but cannot find any external connections and so the alert transfer stops.

Data after running the test:

EventsTable for “ms2”:

<table>
<thead>
<tr>
<th>EVENTID</th>
<th>TYPEOF EVENT</th>
<th>SOURCE HOSTNAME</th>
<th>SOURCE PORT</th>
<th>DEST HOSTNAME</th>
<th>DEST PORT</th>
</tr>
</thead>
<tbody>
<tr>
<td>E1021754456747</td>
<td>3</td>
<td>chite.tr.ttu.com</td>
<td>5743</td>
<td>rose.oo.org</td>
<td>4894</td>
</tr>
<tr>
<td>E1021754456483</td>
<td>7</td>
<td>rose.oo.org</td>
<td>4674</td>
<td><a href="http://www.dty.com">www.dty.com</a></td>
<td>3484</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>EVENTDATE</th>
<th>EVENTTIME</th>
<th>USER NAME</th>
<th>DONE FLAG</th>
<th>COMMENTS</th>
<th>AGENTS CHAIN</th>
</tr>
</thead>
<tbody>
<tr>
<td>09/07/2002</td>
<td>1021754412839</td>
<td>shally</td>
<td>2</td>
<td>null</td>
<td>null</td>
</tr>
</tbody>
</table>
ProcessEventsTable for “ms2”:

<table>
<thead>
<tr>
<th>EVENTID</th>
<th>ISILLEGALPROCESS</th>
</tr>
</thead>
<tbody>
<tr>
<td>E1021754456747</td>
<td>1</td>
</tr>
</tbody>
</table>

IllegalProcessEventsTable for “ms2”:

<table>
<thead>
<tr>
<th>EVENTID</th>
<th>ALERTFIELD</th>
<th>ALERTMESSAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>E1021754456747</td>
<td>Echostorm</td>
<td>Echo service on the port</td>
</tr>
</tbody>
</table>

UsersTable for “ms2”:

<table>
<thead>
<tr>
<th>USERNAME</th>
<th>AUTHORIZED</th>
<th>FLAGGED</th>
</tr>
</thead>
<tbody>
<tr>
<td>shally</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

EventsTable for “ms4”:

<table>
<thead>
<tr>
<th>EVENTID</th>
<th>TYPEOF EVENT</th>
<th>SOURCE HOSTNAME</th>
<th>SOURCE PORT</th>
<th>DEST HOSTNAME</th>
<th>DEST PORT</th>
<th>EVENTDATE</th>
<th>EVENTTIME</th>
<th>USER NAME</th>
<th>DONE</th>
<th>FLAG</th>
<th>COMMENTS</th>
<th>AGENTS CHAIN</th>
</tr>
</thead>
<tbody>
<tr>
<td>E1021754478439</td>
<td>7</td>
<td><a href="http://www.dty.com">www.dty.com</a></td>
<td>3478</td>
<td><a href="http://www.lily.com">www.lily.com</a></td>
<td>3844</td>
<td>09/07/2002</td>
<td>1021754478354</td>
<td>shally</td>
<td>1</td>
<td></td>
<td>null</td>
<td>ms2-ms4-ms5</td>
</tr>
<tr>
<td>E1036296083471</td>
<td>5</td>
<td>rose.oo.org</td>
<td>4674</td>
<td><a href="http://www.dty.com">www.dty.com</a></td>
<td>3484</td>
<td>09/07/2002</td>
<td>1021754454393</td>
<td>shally</td>
<td>1</td>
<td></td>
<td>null</td>
<td>ms2</td>
</tr>
</tbody>
</table>

IllegalConnectionEventsTable for “ms4”:

<table>
<thead>
<tr>
<th>EVENTID</th>
<th>ALERTFIELD</th>
<th>ALERTMESSAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>E1036296083471</td>
<td>IllegalConnectionByUser</td>
<td>EchoStorm Received</td>
</tr>
</tbody>
</table>

EventsTable for “ms5”:

<table>
<thead>
<tr>
<th>EVENTID</th>
<th>TYPEOF EVENT</th>
<th>SOURCE HOSTNAME</th>
<th>SOURCE PORT</th>
<th>DEST HOSTNAME</th>
<th>DEST PORT</th>
<th>EVENTDATE</th>
<th>EVENTTIME</th>
<th>USER</th>
<th>DONE</th>
<th>COMMENTS</th>
<th>AGENTS</th>
</tr>
</thead>
</table>
IllegalConnectionEventsTable for “ms5”:

<table>
<thead>
<tr>
<th>EVENTID</th>
<th>ALERTFIELD</th>
<th>ALERTMESSAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>E1036296084921</td>
<td>IllegalConnectionByUser</td>
<td>EchoStorm Received</td>
</tr>
</tbody>
</table>

Messages Exchanged:

Query Message sent from “ms2” to “ms3” Agent:

```xml
<?xml version="1.0"?>
<mnss-query-echo-storm>
  <event>
    <SourceHostName>chite.tr.ttu.com</SourceHostName>
    <SourcePort>5743</SourcePort>
    <DestHostName>rose.oo.org</DestHostName>
    <DestPort>4894</DestPort>
    <EventDate>09/07/2002</EventDate>
    <EventTime>1021754412839</EventTime>
    <UserName>shally</UserName>
  </event>
</mnss-query-echo-storm>
```

Reply Message sent from “ms3” to “ms2” Agent:

```xml
<?xml version="1.0"?>
<mnss-query-illegal-process>
  <event>
    <SourceHostName>chite.tr.ttu.com</SourceHostName>
    <SourcePort>5743</SourcePort>
    <DestHostName>rose.oo.org</DestHostName>
    <DestPort>4894</DestPort>
    <EventDate>09/07/2002</EventDate>
    <EventTime>1021751234654</EventTime>
    <UserName>shally</UserName>
    <IllegalProcess>yes</IllegalProcess>
  </event>
</mnss-query-illegal-process>
```
7.5 Testing Timeouts on Protocols

7.5.1 Test case 01

Test case Scenario:
The Agent “ms6” experiences Echo storm attack as described in section 6.7. It queries the Agent source machine of the connection about any illegal process associated with the connection using Query Echo storm protocol. Since the Agent at that host is unreachable, the task through which the Agent sends the query times out after a specified amount of time. The Agent “ms6” updates the event with doneFlag = 4 which indicates that task timed out and event is not handled completely.

**Data before running the test:**

**Agents List Table:**

<table>
<thead>
<tr>
<th>HOST NAME</th>
<th>AGENT NAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>huda.hyd.com</td>
<td>ms1huda</td>
</tr>
</tbody>
</table>

**EventsTable for “ms6”:**

<table>
<thead>
<tr>
<th>EVENTID</th>
<th>TYPE OF EVENT</th>
<th>SOURCE HOSTNAME</th>
<th>SOURCE PORT</th>
<th>DEST HOSTNAME</th>
<th>DEST PORT</th>
</tr>
</thead>
<tbody>
<tr>
<td>E1031704190789</td>
<td>3</td>
<td>huda.hyd.com</td>
<td>7893</td>
<td><a href="http://www.reddy.net">www.reddy.net</a></td>
<td>4673</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>EVENTDATE</th>
<th>EVENTTIME</th>
<th>USER NAME</th>
<th>DONE FLAG</th>
<th>COMMENTS</th>
<th>AGENTS CHAIN</th>
</tr>
</thead>
<tbody>
<tr>
<td>09/07/2002</td>
<td>1031704189546</td>
<td>samsud</td>
<td>0</td>
<td>null</td>
<td>null</td>
</tr>
</tbody>
</table>

**ProcessEventsTable for “ms6”:**

<table>
<thead>
<tr>
<th>EVENTID</th>
<th>ISILLEGALPROCESS</th>
</tr>
</thead>
<tbody>
<tr>
<td>E1031704190789</td>
<td>0</td>
</tr>
</tbody>
</table>

**IllegalProcessEventsTable for “ms6”:**

<table>
<thead>
<tr>
<th>EVENTID</th>
<th>ALERTFIELD</th>
<th>ALERTMESSAGE</th>
</tr>
</thead>
</table>
Results:

The Agent “ms6” reads the event (EventID “E1031704190789”) and sends a query to source machine of the connection, “ms5huda”. The Agent “ms5huda” is unreachable and the task handling the query times out. The Agent “ms6” updates the event with doneFlag = 4 and comments = “task timed out” in its EventsTable.

Data after running the test:

EventsTable for “ms6”:

<table>
<thead>
<tr>
<th>EVENTID</th>
<th>TYPEOF EVENT</th>
<th>SOURCE HOSTNAME</th>
<th>SOURCE PORT</th>
<th>DEST HOSTNAME</th>
<th>DEST PORT</th>
</tr>
</thead>
<tbody>
<tr>
<td>E1031704190789</td>
<td>3</td>
<td>huda.hyd.com</td>
<td>7893</td>
<td><a href="http://www.reddy.net">www.reddy.net</a></td>
<td>4673</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>EVENTDATE</th>
<th>EVENTTIME</th>
<th>USER NAME</th>
<th>DONE FLAG</th>
<th>COMMENTS</th>
<th>AGENTS CHAIN</th>
</tr>
</thead>
<tbody>
<tr>
<td>09/07/2002</td>
<td>1031704189546</td>
<td>samsud</td>
<td>4</td>
<td>“task timedout”</td>
<td>null</td>
</tr>
</tbody>
</table>

ProcessEventsTable for “ms6”:

<table>
<thead>
<tr>
<th>EVENTID</th>
<th>ISILLEGALPROCESS</th>
</tr>
</thead>
<tbody>
<tr>
<td>E1031704190789</td>
<td>0</td>
</tr>
</tbody>
</table>

IllegalProcessEventTable for “ms6”:

<table>
<thead>
<tr>
<th>EVENTID</th>
<th>ALERTFIELD</th>
<th>ALERTMESSAGE</th>
</tr>
</thead>
</table>

Messages Exchanged: The messages exchanged are similar to the Test Case 03 except the connection details.
CHAPTER 8

Conclusion and Future Work

8.1 Conclusion

The proposed architecture aids in the development of an open Internet-wide distributed security protocol that will allow systems to dynamically detect and respond to security threats that would otherwise be invisible to the individual system. Since the current framework was successful in coordinating the agents to provide additional information (in the form of events and alerts) about a suspicious entries in the log file, this prototype could help system administrators to identify new attacks, spot and defend known attacks, develop better protection and countermeasures for their systems. However, the current architecture falls short of some aspects that have been traditionally in the realm of distributed systems that needs to be improved.

Features of the Current Architecture:

- Dynamic deployment of agents – since the agents are independently-running entities, they can be added to or removed from a system without altering other components and not having to restart the system.
• Reusable tasks - the tasks used to handle the suspicious events can easily be reused to handle new suspicious events in future evaluations.

• Robust Architecture - the agents are robust to the unknown protocols and malformed messages - they send the default “Not Understood” messages to all the unhandled messages and protocols.

• Reliability - The tasks which handle the communication protocols time out if they cannot get the messages in the specified timeframe - they mark the event as incomplete and update the log file accordingly. Agent continuously reads and passes the unhandled events to all its registered tasks. So after some random amount of time the same event is handled again by the same task.

Suggested Improvements to the Current Architecture :

• Events should be made independent of the host machine. While the current implementation ties the agents to specific intrusion detection and log analysis software, the extension of this architecture should allow wrapping of the agents around any of the existing distributed security frameworks. This means that users would not need to abandon their current software systems if they wanted to migrate to this system. They would only need to implement an agent that translates their security events to work with the proposed system.

• Multiple tasks should be able to handle an event. Currently there is only one task dealing with one type of event.

• Tasks should also have the capability to handle multiple events at a time.
It should be possible to add new Tasks dynamically for handling new type of events using configuration files (may be in XML) rather than restarting the system.

The communication protocols used may also be subject to denial-of-service attacks in which case an attacker makes it impossible or difficult for messages to get delivered.

Privacy and authentication are important needs for any communication model because some of the messages generated by the agents may contain sensitive data about the hosts being monitored. Agents should handle situations where the messages are generated by unauthorized entities. Usually, cryptography is the solution to both the problems.

### 8.2 Future Work

Additional work needs to focus on extension of the current prototype by implementing all the features mentioned suggested earlier. Also, in co-operative systems trust is an important issue in most of the decision-making processes. It is feasible that hosts in distributed systems may not reveal the information about an intrusion out of fear of bad reputation or of leaking sensitive information. Future work should explore a more flexible systems that would extend this basic architecture and allow the agents to dynamically learn trust measures, form alliances and trust relationships, sign contracts etc. These types of sophisticated social contracts, when better understood, will greatly aid in reducing the amount of communications needed and protect the system from self-
interested and untrustworthy agents. This will allow the system to scale up to millions of
agents without any bottlenecks.

The proposed prototype is an initial step towards the development of open security
interaction protocols using an agent communication language among distributed intrusion
detection systems. The long term goal of this research is to extend the protocols into a
common language for Distributed Intrusion Detection systems. It is believed that these
interaction protocols and content languages will serve as enablers to distributed intrusion
detection in the same way that HTTP and HTML served to enable the web. This would
eventually lead to a common and open standard for security systems to inter-operate on
an Internet wide scale.
REFERENCES


[Web07] FIPA ACL Message Structure Specification

[Web08] FIPA Developer’s Guide

[Web09] FIPA Message Transport Service
http://www.fipa.org/specs/fipa00067/XC00067D.html


[Web08] SECURITY FOCUS ARTICLE
http://online.securityfocus.com/guest/10094, January 2002


[Web09] TCP/IP Daemon Wrapper Package

Figure A.1. Higher Level Abstraction of the System Implementation
Figure A.2. dbManager Class
Figure A.3. LogFile Class

Figure A.4. mainGUI Class
Figure A.5. mnssAgent Class
Figure A.6. invokeTasks Class

Figure A.7. MnsTask Class
Figure A.8. DenialOfServiceAlertTask Class
Figure A.9. TooManyPacketsEvent Class
Figure A.10. IgnoreRequestTask Class

```java
public class IgnoreRequestTask extends Task {
    private Vector contactList;
    private Vector agentList;

    public IgnoreRequestTask() {
        super();
    }

    public IgnoreRequestTask(Vector contactList, Vector agentList) {
        super();
        this.contactList = contactList;
        this.agentList = agentList;
    }

    public void startTask() {
        // Implementation
    }

    public void doneGetAgentIds(LogicalView view) {
        // Implementation
    }

    public void createXMLDoc(String ignoreRequestEvent, LogicalView view) {
        // Implementation
    }

    public void sendIgnoreRequests(String ignoreRequest, LogicalView view, Vector agents) {
        // Implementation
    }

    public void setAgentList(Vector agents) {
        // Implementation
    }

    public boolean canHandle(String typeOfEvent) {
        // Implementation
    }

    public void setEventToHandle(String event) {
        // Implementation
    }

    public void setDbManager(Vector dbManager) {
        // Implementation
    }

    public void getTaskName(LogicalView view) {
        // Implementation
    }

    public void setGUI(String gui) {
        // Implementation
    }

    public void setLogFile(String logfile) {
        // Implementation
    }

    public void setLogs(Vector logs) {
        // Implementation
    }

    public void close() {
        // Implementation
    }
}
```
**Figure A.11. IgnoreRequestEvent Class**

<table>
<thead>
<tr>
<th>Method</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>IgnoreRequestEvent()</td>
<td></td>
</tr>
<tr>
<td>IgnoreRequestEvent(eventID)</td>
<td>Logical View:java.lang: String</td>
</tr>
<tr>
<td>setEventID</td>
<td>Logical View:java.lang: String</td>
</tr>
<tr>
<td>getEventName</td>
<td>Logical View:java.lang: String</td>
</tr>
<tr>
<td>getEventID()</td>
<td>Logical View:java.lang: String</td>
</tr>
<tr>
<td>setEventID(eventID)</td>
<td>Logical View:java.lang: String</td>
</tr>
<tr>
<td>setEventName(requestName)</td>
<td>Logical View:java.lang: String</td>
</tr>
<tr>
<td>getEventID(eventID)</td>
<td>Logical View:java.lang: String</td>
</tr>
<tr>
<td>completed()</td>
<td>Logical View:java.lang: String</td>
</tr>
<tr>
<td>getEventName()</td>
<td>Logical View:java.lang: String</td>
</tr>
<tr>
<td>setEventName(CompletableFuture)</td>
<td>void</td>
</tr>
<tr>
<td>initialize()</td>
<td>Logical View:java.lang: String</td>
</tr>
<tr>
<td>getEventName()</td>
<td>Logical View:java.lang: String</td>
</tr>
<tr>
<td>setEventName(CompletableFuture)</td>
<td>void</td>
</tr>
<tr>
<td>getEventID()</td>
<td>Logical View:java.lang: String</td>
</tr>
<tr>
<td>setEventName(CompletableFuture)</td>
<td>void</td>
</tr>
</tbody>
</table>
Figure A.12. SuspiciousLoginTask Class
Figure A.13. LoginEvent Class
<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>AgentList : Vector</code></td>
<td></td>
</tr>
<tr>
<td><code>newVector : Vector</code></td>
<td></td>
</tr>
<tr>
<td><code>msgsUsersVector : Vector</code></td>
<td></td>
</tr>
<tr>
<td><code>msgsDestinationsByUser : Vector</code></td>
<td></td>
</tr>
<tr>
<td><code>msgsSourceConnectionsByUser : Vector</code></td>
<td></td>
</tr>
<tr>
<td><code>start()</code></td>
<td></td>
</tr>
<tr>
<td><code>handleReceivedConnection(registeredEvent : Event, connections : Vector)</code></td>
<td></td>
</tr>
<tr>
<td><code>loggingTask()</code></td>
<td></td>
</tr>
<tr>
<td><code>appAlertDestinationsTask(task : Task)</code></td>
<td></td>
</tr>
<tr>
<td><code>setEventTaskListeners(event : Event)</code></td>
<td></td>
</tr>
<tr>
<td><code>setDbManager(dbManager : dbManager)</code></td>
<td></td>
</tr>
<tr>
<td><code>setTaskName(taskName : LogicalView.java.lang.String)</code></td>
<td></td>
</tr>
<tr>
<td><code>setLayout(mainGUI : mainGUI)</code></td>
<td></td>
</tr>
<tr>
<td><code>setLogFiles(log : Logs)</code></td>
<td></td>
</tr>
<tr>
<td><code>close()</code></td>
<td></td>
</tr>
</tbody>
</table>
Figure A.15. SendUserAlertToSourceTask Class
### AlertDestinationsTask Class

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AlertDestinationsTask()</td>
<td>Constructor</td>
</tr>
<tr>
<td>startTask()</td>
<td>void</td>
</tr>
<tr>
<td>updateAgentChainOfEvent(eventVector: Event, alertTo: Event)</td>
<td>void</td>
</tr>
<tr>
<td>doneGetAgentDoTask(obj: LogicalView, lang: Object)</td>
<td>void</td>
</tr>
<tr>
<td>createXMLDoc(agent: AlertEvent, event: LogicalView, lang: String)</td>
<td>void</td>
</tr>
<tr>
<td>sendAlertsUserAlert(logicalView: LogicalView, lang: String, vAgents: Vector)</td>
<td>void</td>
</tr>
<tr>
<td>setEventToList(event: Event)</td>
<td>void</td>
</tr>
<tr>
<td>setAgentList(AgentList: Vector)</td>
<td>void</td>
</tr>
<tr>
<td>canHandle(event: Object)</td>
<td>boolean</td>
</tr>
<tr>
<td>setDBManager(dbManager: dbManager)</td>
<td>void</td>
</tr>
<tr>
<td>close()</td>
<td>void</td>
</tr>
<tr>
<td>getTaskName()</td>
<td>LogicalView, lang: String</td>
</tr>
</tbody>
</table>

**Figure A.16.** AlertDestinationsTask Class
Figure A.17. AlertSourceTask Class
<table>
<thead>
<tr>
<th>Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>QueryEchostormTask</td>
</tr>
<tr>
<td>QueryEchostormTask(event : Event, AgentList : Vector, dbManager : dbManager, mgui : mainGUI, logs : Logs)</td>
</tr>
<tr>
<td>QueryEchostormTask(event : Event, AgentList : Vector, dbManager : dbManager, mgui : mainGUI, logs : Logs, logfile : LogFile)</td>
</tr>
<tr>
<td>startTask() : void</td>
</tr>
<tr>
<td>doneWaitTask() : void</td>
</tr>
<tr>
<td>doneGetAgentID(obj : Logical View; java::lang::Object) : void</td>
</tr>
<tr>
<td>createXMLDoc(event : EchostormEvent) : Logical View; java::lang::String</td>
</tr>
<tr>
<td>sendQueryEchostorm(_echoStormStr : Logical View; java::lang::String, vAgents : Vector) : void</td>
</tr>
<tr>
<td>handleInform(conv : Conversation) : void</td>
</tr>
<tr>
<td>checkForString(checkString : Logical View; java::lang::String, elemVector : Vector) : int</td>
</tr>
<tr>
<td>setAgentList(AgentList : Vector) : void</td>
</tr>
<tr>
<td>loanHandle(TypeOfEvent : int) : boolean</td>
</tr>
<tr>
<td>setEventToHandle(event : Event) : void</td>
</tr>
<tr>
<td>setDbManager(dbManager : dbManager) : void</td>
</tr>
<tr>
<td>getTaskName() : Logical View; java::lang::String</td>
</tr>
<tr>
<td>setGUI(gui : mainGUI) : void</td>
</tr>
<tr>
<td>setLogFile(logfile : LogFile) : void</td>
</tr>
<tr>
<td>setLogSize(logs : Logs) : void</td>
</tr>
<tr>
<td>close() : void</td>
</tr>
</tbody>
</table>

Figure A.18. QueryEchostormTask Class
Figure A.19. EchostormEvent Class
Figure A.20. QueryIllegalProcessTask Class
Figure A.21. IllegalProcessEvent Class

Figure A.22. SendAlertToUserConnectedMachinesTask Class