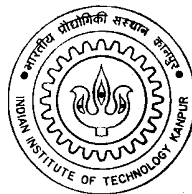
Sachet - A distributed real-time network based intrusion detection system

A Thesis Submitted
in Partial Fulfillment of the Requirements
for the Degree of
Master of Technology

by
Sachin Goel



to the
Department of Computer Science & Engineering
Indian Institute of Technology, Kanpur

June, 2004

Certificate

This is to certify that the work contained in the thesis entitled "*Sachet - A distributed real-time network based intrusion detection system*", by *Sachin Goel*, has been carried out under our supervision and that this work has not been submitted elsewhere for a degree.

June, 2004

(Dr. Dheeraj Sanghi)
Department of Computer Science & Engineering,
Indian Institute of Technology,
Kanpur.

(Dr. Deepak Gupta)
Department of Computer Science & Engineering,
Indian Institute of Technology,
Kanpur.

Abstract

While the increased inter-connectivity of the computer networks has brought a lot of benefits to the people, it also rendered networked systems vulnerable to malicious attacks from the hackers. The failure of intrusion prevention techniques to adequately secure computer systems has led to the growth of the Intrusion Detection System. In this thesis, we have designed and implemented a distributed, network-based intrusion detection system -*Sachet*. The *Sachet* word is a hindi word which means - Alert. The system uses an existing open source network based misuse detection system - *snort*. We have built upon snort to develop a heterogeneous, scalable, distributed IDS that is completely controllable from a central location. Sachet comprises of multiple agents that use snort for misuse detection, a central server that stores all alerts and controls the agents, and a console for monitoring and viewing the activities of entire Sachet system by the system administrator. The agents and server communicates using a Sachet protocol that ensures reliability, mutual authentication, confidentiality, integrity and provides tolerance from agent and server crashes.

Acknowledgments

I would like to express my deep gratitude to my thesis supervisors Dr. Dheeraj Sanghi and Dr. Deepak Gupta for their guidance and invaluable suggestions throughout the year of this thesis. It was their enthusiasm, motivation and guidance that saw the timely completion of this thesis. I would also like to thank my other team member involved with the development of Sachet - JVR Murthy for his cooperation and support throughout the year. We really had a great time together. I am also thankful to members of Prabhu Goel Research Centre for all their cheerful support. It was a great time working in Prabhu Goel Research Centre as it is equipped with all the facilities the researcher can dreamt of. Finally I would like to thank my parents for their blessings, support and encouragement.

Contents

1 Introduction **1**

1.1 What is an Intrusion Detection System? 2

1.2 Desirable characteristics of an intrusion detection system 4

1.3 Scope of Thesis 5

1.4 Organization of the Report 5

2 Related Work **6**

2.1 IDES 7

2.2 DIDS 8

2.3 AAFID 9

2.4 BlackICE/ICEcap 10

3 Architecture of Sachet **12**

3.1 **Sachet**: Architectural Goals 12

3.2 **Sachet**: Architecture 13

3.3 The **Sachet** protocol 15

3.3.1 General Packet Structure 16

3.3.2 Reliability 17

3.3.3 Authentication 18

3.3.4 Commands 20

3.3.5 Alerts 21

3.3.6 Graceful Degradation 21

3.4 The **Sachet** Server-Console Protocol 26

4 Implementation of the Sachet **28**
4.1 The Server 28
4.2 The Agent 29
4.3 The Console 31
4.4 Addition of New Signatures 32
4.5 Maintenance of Alert id 33
4.6 Public Key Management 33
4.7 Private Key Storage 34

5 Conclusion and Future Work **40**

A Formats of Messages in Sachet Protocol **42**
A.1 Authentication Messages 43
A.2 Data Messages 43

B Formats of Messages in Sachet server-console Protocol **49**
B.1 Authentication Message 49
B.2 Command Messages 50
B.3 Request Messages 53

List of Tables

A.1 Messages exchanged during authentication phase 43
A.2 Message excahnges for Key-reset command 44
A.3 Messages for enabling signatures 44
A.4 Messages for disabling signatures 44
A.5 Messages for adding new signature 45
A.6 Messages for enabling signature files 45
A.7 Messages for disabling signature files 45
A.8 Messages for starting misuse detector 46
A.9 Messages for stopping misuse detector 46
A.10 Messages for probing the agent 46
A.11 Messages for sending alerts to the server 47
A.12 Messages for sending only packets to the server 47
A.13 Messages for the failure of the Misuse Detector 48
A.14 Messages for probing the server 48
B.1 Messages for authenticating the user 50
B.2 Messages for changing the password 50
B.3 Messages for adding new agent 51
B.4 Messages for deleting the agent 51
B.5 Messages for enabling signatures 51
B.6 Messages for disabling signatures 52
B.7 Messages for enabling signature files 52
B.8 Messages for disabling signature files 52
B.9 Messages for starting misuse detector 53
B.10 Messages for stopping misuse detector 53

B.11 Messages for adding a new signature to all agents 53
B.12 Message for requesting information about the agent 54

List of Figures

3.1 Architecture of **SACHET** 14

3.2 Packet Structure of **SACHET**Protocol 17

3.3 Key Reset Implementation 20

3.4 Server crashes but has not been restarted 22

3.5 Server crashes and recovers quickly 23

3.6 Agent crashes but has not recovered 24

3.7 Agent crashes and recovers quickly 25

3.8 Packet Structure of **SACHET** server-console protocol 26

4.1 State diagram of the server (with respect to a specific agent) 29

4.2 State diagram of the agent 31

4.3 Top-Level agent screen of Console 35

4.4 Agent screen of Console 36

4.5 Alert Reporting screen of Console 37

4.6 Screen depicting list of attack signatures in Console 38

4.7 Template for creating new attack signature in Console 39

A.1 Packet Structure of **SACHET**Protocol 42

B.1 Packet Structure of **SACHET** server-console protocol 49

Chapter 1

Introduction

The widespread proliferation of computer networks has resulted in the increase of attacks on information systems. These attacks are used for illegaly gaining access to unauthorized information, misuse of information or to reduce the availiability of the information to authorized users. This results in huge financial losses to companies besides losing their goodwill to customers as their informative services are severely disrupted. These attacks are increasing at a staggering rate and so is their complexity. Thus there is a need for complete protection of organizational computing resources which is driving the attention of people towards intrusion prevention and detection systems.

We can effectively protect the computer systems, if we use three fundamental techniques against intrusions: prevention, detection and response. Earlier, intrusion prevention was widely considered as a complete and sufficient protection against the intrusions. Such preventive measures include user authentication (using passwords or biometrics), fencing around the network using firewalls, very tight access control mechanisms, avoiding programming errors etc. But, unfortunately these measures are not sufficient in adequately protecting the computer system due to many reasons. There will always be unknown programming flaws, design and architectural weaknesses in application programs, protocols and operating systems which can always be exploited by the attacker. The abuse of privileges by insiders (usually disgruntled

employees) to gain unauthorized access, the failure of firewall to prevent many attacks such as dictionary attacks and probes, the cracking of paswords are some of the other reasons that make preventive measures insufficient to protect computer systems. Hence, intrusion prevention is not a complete solution. If there are inveitable attacks on a system, we would like to detect them as soon as possible (preferably in real time) and take appropriate action. Moreover it should be possible to trace an attack to its source, and assess the extent of damage. The capability that provides these special features is known as intrusion detection. Intrusion detection tools are not preventive devices but they should be used as a second line of defense. Hence, they complement the protective mechanisms to improve system security.

1.1 What is an Intrusion Detection System?

An intrusion is defined as "any set of actions that attempt to compromise the integrity, confidentiality, or availability of a computer resource" [6]. The definition disregards the success or failure of those actions, so it corresponds to attacks against the computer systems. Accordingly, intrusion detection is defined as "the problem of identifying actions that attempts to compromise the integrity, confidentiality, or availability of a computer resource" [6]. Hence, an intrusion detection system (IDS) is a piece of software that monitors a computer system to detect any intrusions, and alerts a designated authority.

Intrusion Detection systems can be classified in several ways. Depending on the source of data, the intrusion detection systems are categorized into host-based or network-based systems. The network-based intrusion detection systems process the data that originates on the network, such as TCP/IP traffic. Malformed packets, packet flooding, probes are some of the attacks which can be detected by such systems. The host-based intrusion detection systems analyzes the data that originates on computers (hosts), such as application and operating system event logs, system call traces. Such systems are effective for insider threats. Abuse of privileges by insiders, accesses of critical data are some of the attacks which can be detected by these systems.

Intrusion detection systems can also be classified, depending on the detection model used, into misuse or anomaly detection models. Misuse detection systems look for well-defined patterns of known attacks. The known attacks are represented as *patterns* or *signatures*. Misuse Detection is therefore, simply a problem of matching patterns of attack in the given source of data. Such systems detect patterns of known attacks quite accurately and efficiently, and generate very few false alarms. The limitation of misuse detection is that it cannot detect novel, unknown attacks or variations of known attacks. In addition, misuse detection requires the nature of attacks to be well understood. This implies that human experts must work on the analysis and representation of attacks, which is usually time consuming and error prone. Anomaly detection is based on the normal behavior of the subject (*e.g.*, a user, program or a system). Any action that significantly deviates from the normal behavior is considered as intrusive. Such systems build a statistical or machine learning model of normal behavior of the subject. The model is basically a list of metrics or patterns that capture the normal profile. The system flags an intrusion if any observed metrics or patterns of given behavior significantly deviate from the model. Such systems can detect previously unknown patterns of attacks but usually generate many false positives (normal behavior classified as intrusive). Another common problem is that since a subject's normal behavior is modeled on the basis of the audit data over the period of normal operation and if undiscovered intrusive activities occur during this period, they will be consider as normal activities.

Intrusion detection systems can also be classified by their mode of operation: real-time or off-line. A real-time IDS monitors the system continuously and reports intrusions as soon as they are detected. Such systems can substantially reduce the damage to the system, if the system administrator can be notified as early as possible. Moreover, there is a great chance of stopping the attack currently in progress and catching the intruder as intruder would not get much time to delete his trail (*e.g.*, by erasing logs). An off-line IDS inspects system logs at periodic intervals and then discovers any suspicious activity that was recorded. Such systems are very effective in corelating attacks that span multiple hosts, slow probing attacks that span over hours and days, and for forensic analysis. An offline IDS typically reduces

system overhead but gives much less timely notification of intrusions.

Lastly, intrusion detection systems can be categorized based on their architecture. The most common IDS architectures are: centralized, hierarchical or distributed systems. In centralized IDS, the data may be collected from various sources (hosts or networks) but is sent to a centralized location where it is analyzed. Such systems limit the system scalability as it could become bottleneck on increasing number of sources and also represent a single point of vulnerability. In hierarchical IDS, some of the data collected from multiple hosts or a single host is passed up through the layers and is analyzed to varying degree at each level. In Distributed IDS, the data is collected and analyzed across the entire network being monitored and results are then sent to a centralized location. Such systems are scalable and not subject to a single point of failure.

1.2 Desirable characteristics of an intrusion detection system

Crosbie and Spafford [3] define the following desirable characteristics of an intrusion detection systems:

- It must run continually with minimal human supervision.
- It must be fault tolerant by being able to recover from accidental system crashes and re-initializations.
- It must resist subversion. The intrusion detection system must be able to monitor itself and detect if it has been attacked or modified by an attacker.
- It must impose a minimal overhead on the system where it is running, to avoid interfering with the system's normal operation.
- It must be scalable to monitor a large number of hosts while providing results accurately and without degradation of performance.

- It must provide graceful degradation of service. The failure of any component of the intrusion detection system should not immediately fail the entire system.
- It must allow dynamic reconfiguration, allowing the system administrator to make changes in it's configuration without restarting the whole intrusion detection system.

While building a new intrusion detection system, these above characteristics of IDS should always be kept in mind. It would not be easy to include all the characteristics as there will always exist some trade-offs between these characteristics.

1.3 Scope of Thesis

In this thesis we describe the design and implementation of a distributed, network based intrusion detection system - *sachet*. The *sachet* is a hindi word which means - alert. The system uses an existing open source network based misuse detection system - *snort* [17]. We have built upon snort to develop a full-fledged scalable, distributed, gracefully degrading IDS that is completely controllable from a central location. SACHET comprises of the following components: multiple sachet agents that use snort for misuse detection, a central sachet server that stores all alerts and controls the agents, and a sachet console that interacts with the server to provide a centralized control facility and alert information to the network administrator. The sachet server communicates with the agents using a protocol that provides mutual authentication, confidentiality, and integrity of all messages, and toleration of server and agent crashes.

1.4 Organization of the Report

In Chapter 2, we briefly review some existing intrusion detection systems.Chapter 3 describes the overall architecture of Project IDS. Chapter 4 deals with the implementation details related to the Project IDS. Chapter 5 concludes our work with the limitations and future work.

Chapter 2

Related Work

A lot of work has been done in the field of intrusion detection systems. Denning proposed a first intrusion detection model [4] which was based on anomaly detection. The paper presented the idea that model of the behavior of a particular individual could be constructed by the intrusion detection system, and that subsequent behavior of that individual could be compared against the model. Intrusion detection could then be performed by identifying behavior that deviated sufficiently from the normal. Several models based on the use of statistics, time-series, and other methods were mentioned. Another important idea introduced by Denning was that intrusion detection could be performed in real-time, or near real-time.

In the area of host-based intrusion detection there has been substantial work using different methods for analyzing data generated by the host. One of the first host-based intrusion detection systems implemented was IDES [5], which used statistical detection engine based on Dennings anomaly detection model [4]. The other host-based system is Haystack [12] and its successor Stalker [13] which perform off-line misuse detection using a centralized monitoring station. Many real-time, centralized host based intrusion detection systems have also been developed such as the Next-generation Intrusion Detection Expert System (NIDES) [1],and the Computer Misuse Detection System (CMDS) [9] . Due to problems with centralized approach, some distributed host-based systems were also developed. Centralization can severely limit the scalability of the system, and introduces a single point of

failure. Distributed host based intrusion detection systems avoid these problems. The Cooperating Security Monitor (CSM) [22] and Autonomous Agents for Intrusion Detection [23] are examples of such systems. Commercially-available real-time host-based systems include SecureCom [21], Intruder Alert (ITA) [16] and Symantec Host IDS [15].

The area of network-based intrusion detection has also seen a good amount of work. One of the first implemented network-based intrusion detection system was the Network Security Monitor (NSM) [7] that was designed to capture TCP/IP packets and detect anomalous activity in a heterogeneous network. NSM used both statistical models and rule-based detection to detect anomolous network connections. Graph based Intrusion Detection System (GrIDS) [2] is one of the example of distributed network based intrusion detection systems. Distributed Intrusion Detection System (DIDS) [14] is distributed hybrid system i.e. both host-based and network-based intrusion detection system. Commercially available network based systems includes BlackICE [19], Network Flight Recorder [11] and Cisco IDS [18]. Dragon [8] and Realsecure [20] are both commercially available hybrid systems.

The following sections briefly describe some of the intrusion detection systems. They include both host-based and network-based systems.

2.1 IDES

The Intrusion Detection Expert System (IDES) [5] is one of the earliest intrusion detection systems. It is a host-based real-time system that performs anomaly detection. It is based on Dorothy statistical anomaly model [4]. The basic motivation behind IDES is that users behave in a consistent manner from time to time when performing their activities on a computer system, and that the manner in which they behave can be described by calculating various statistics for the users behavior. A users current behavior can then be compared to his or her normal profile and deviations can be flagged as possible intrusions. IDES monitors three types of subjects: users, remote hosts, and target systems. In total, 36 different parameters,

called *measures*, are monitored for the subjects, 25 for users, 6 for remote hosts, and 5 for target systems. For example, some of the measures that the system monitors for a user are: CPU usage, command usage, and network activity. These measures are kept in a real valued vector as summarized statistics for the session. These statistical profiles are typically updated to reflect new user behavior once a day, after the original profile has been "aged". This aging process ensures that newer behavior plays a larger part in the detection of anomalies than older behavior.

The anomaly detection is performed by processing each new audit record as it enters the system, and verified against the known profile for the subject. IDES also compares each session against known profiles when the session completes. In case the user is a new user, not yet known to the system, IDES uses a default profile, to start the monitoring of that user. When an anomaly is detected, IDES reports the measures that contributed the most to the classification and the site security officer can make a judgment regarding validity of the reported anomaly. IDES also has a GUI that provides the user (site security officer) with plots of anomaly data and text based reports explaining the anomalous activity. The IDES project eventually evolved into the Next-Generation Intrusion Detection Expert System, NIDES [1].

2.2 DIDS

The Distributed Intrusion Detection System (DIDS) [14] was developed at the University of Calofornia, Davis. It is a distributed, real-time hybrid intrusion detection system. DIDS monitors a heterogeneous network of computers and combines distributed monitoring and data reduction with centralized data analysis. DIDS correlates information about individual monitored users using the notion of *Network Identifier* (NID) concept, where each user is tracked as (s)he moves across the network.

The components of DIDS are the DIDS director, a single host monitor per host, and a single LAN monitor for each LAN segment in the monitored network. On

each host, a host monitor collects and analyzes audit records from the host's operating system. The detected intrusion events are subsequently communicated to the director for further analysis. The host monitor also tracks user sessions and reports anomalous behavior to the director. Haystack [12], a host based intrusion detection system, can be easily integrated into DIDS to perform the functionalities of the host monitor. The LAN monitor observes all the network traffic on its segment of LAN and monitors host-to-host communications, services used, and the volume of traffic. The LAN monitor reports to the DIDS director if it finds any suspicious activity in connections or in traffic pattern. The Network Security Monitor (NSM) [7] is typically used as the LAN monitor. The DIDS director consists of three major components: the communication manager, an expert system and the user interface. The communication manager is responsible for collecting the data sent to it from host and LAN monitors. It communicates this data to the expert system for futher processing. The expert system is responsible for evaluating and reporting on the security state of the monitored system to the System administrator. The user interface allows the System administrator to administer and configure the entire DIDS system.

2.3 AAFID

Autonomous Agents for Intrusion Detection (AAFID) [23], developed at Purdue University's Coast Laboratory is a distributed, host-based, real-time intrusion detection system. It basically addresses the shortcomings of those IDS architectures that are normally built around a single monolith that does most of data collection and processing. Hence, the architecture of AAFID is based on multiple independent entities working collectively. These entities are called Autonomous agents. The architecture uses the agents as the lowest-level elements for data collection and analysis, and employs a hierarchial structure to allow for scalability. AAFID consists of three main components: agents, transceivers, and monitors.

An agent is an independently-running entity that monitors host events for suspicious events, and reports such events to the appropriate transceiver. Each host

can contain any number of agents and all the agents in a host report their findings to a single transceiver. The agents and the corresponding transceiver runs on the same host. The agent does not have the authority to directly generate an alarm on the occurence of any suspicious events. Moreover, agents do not communicate with each other in the AAFID architecture.

Transceivers are per-host entities that oversee the operation of all the agents running on their respective hosts. A transceiver has the ability to start and stop execution of any agent, and to send configuration commands to the agents. It may also perform data reduction on the data received from the agents. Finally, the transceiver reports its results to one or more monitors.

Monitors are the highest-level entities in the AAFID architecture. Each monitor oversees the operation of several transceivers. It receives the reduced information from all the transceivers it controls and thus can do higher-level correlation and detect events that involve several hosts. Monitors can be organized in a hierarchical fashion such that a monitor may in turn report to higher-level monitor. Also, a transceiver may report to more than one monitor to provide redundancy and resistance to the failure of one of the monitors. Monitors communicate with a user interface that acts as the access point for the whole AAFID system.

2.4 BlackICE/ICEcap

BlackICE and ICEcap [19] are products from Network ICE that together perform network intrusion detection. BlackICE is the software agent that gathers the network traffic locally on each host and ICEcap is the console. BlackICE can work in both promiscuous mode and network mode and can do packet reassembly. BlackICE can also act as a personal firewall by blocking packets from threatening networks.

ICEcap is the central console that allows consolidation of alerts and centralized configuration. Using ICECap, one can deploy BlackICE at the critical points of

an enterprise network. BlackICE also has a feature called BackTrace that gathers information on hostile machines by launching NetBIOS and DNS reverse queries.

Chapter 3

Architecture of Sachet

In this chapter, we describe the architecture of **Sachet**. **Sachet** is responsible for passively monitoring the network and detecting known attacks in real-time. It generates alerts when it detects attacks. These alerts are then sent to common central location where they are stored in the database. System Administrator can view these alerts through Graphical User Interface and take further action.

3.1 Sachet: Architectural Goals

Sachet has a client-server architecture consisting of a central monitoring station (the server) and agents that monitor hosts or network segments. It is a network-based intrusion detection system designed to be used in a distributed network environment. Following are the design goals of the **Sachet** system.

Distributed architecture Multiple monitoring agents can be deployed at different penetration points in an organization or enterprise network.

Centralized control The central monitoring station (server) can independently control and manage each agent. It can stop/start each agent, change the configuration policies like enabling/disabling of specific attack signatures, etc., at each agent.

Secure and reliable communication Agents and the server communication such

as alerts should be authenticated, encrypted and checked for integrity. The information should not be lost, and should arrive in order.

Centralized storage Alerts generated from multiple agents are stored at a central location, usually in a database. Centralized storage of alerts facilitates co-relating alerts to detect distributed attacks.

User Interface A Graphical user interface (GUI) should be provided to monitor and view state of all components of **Sachet**. It forms the most important tool for the system administrator as it provides a clear picture of the complete system.

Heterogeneous environment The system should be independent of operating system. Agent and server should work on most common operating systems.

Scalability The system should be scalable, to accommodate deployment of a large number of agents at several penetration points in an organization. This should not compromise performance and accuracy.

Graceful Degradation Failure of any component should not cause failure of the whole system. Some reduction in functionality is acceptable.

3.2 Sachet: Architecture

The overall architecture of **Sachet** is shown in Figure 3.1. The figure shows the essential components of the architecture: **Sachet** agents, **Sachet** server and **Sachet** console. The agent further comprises of two components - misuse detector and the control module. **Sachet** system can be distributed over any number of hosts or sub-networks in a network. An agent monitors a host or a network segment for attack events in the network traffic that is incoming to the host or on network segment. The misuse detector analyses the network packets for patterns of attacks and generates alerts, and forwards it to the control module through UDP communication on localhost. The Control module subsequently sends all the generated alerts to the server over secure and encrypted communication channel. The Control

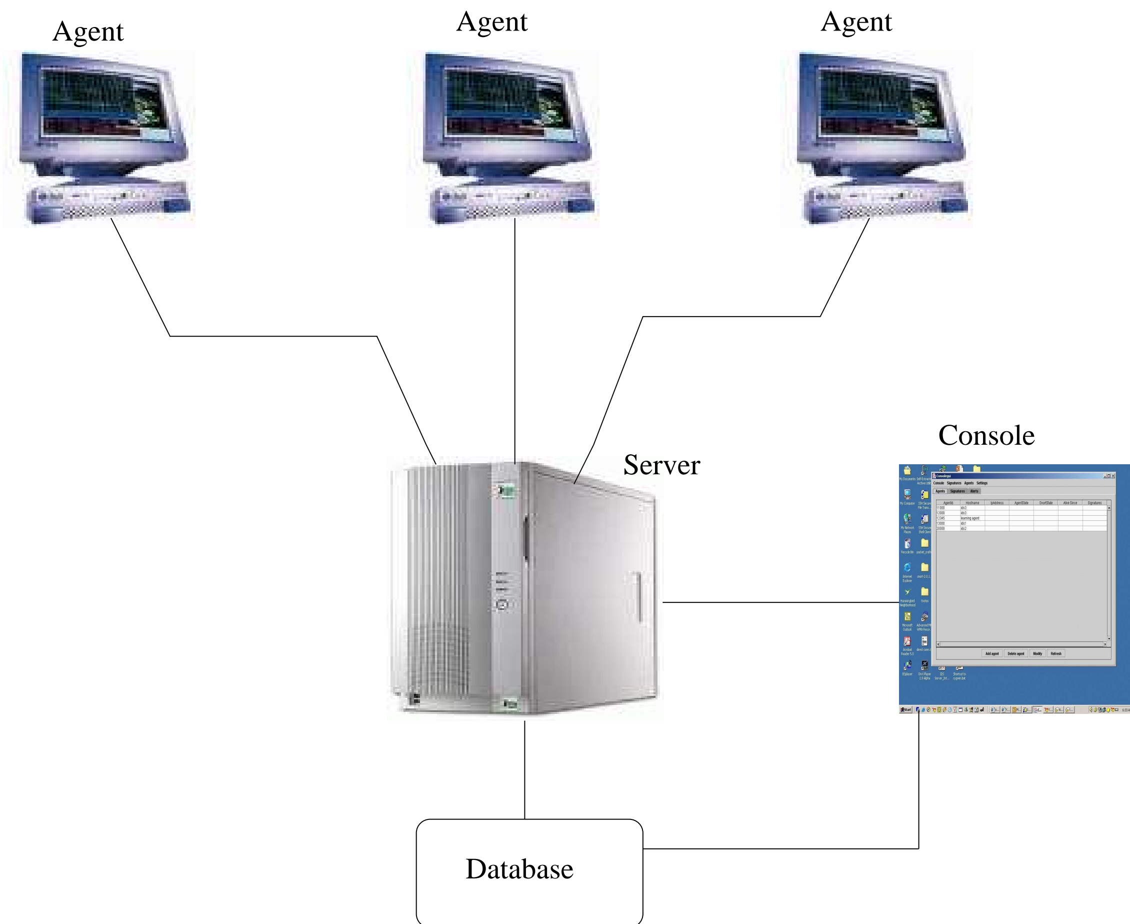


Figure 3.1: Architecture of **Sachet**

module starts and controls the misuse detector. It periodically monitors health of both the server and the misuse detector and takes appropriate action if any of the components fails. The agents and the server communicate to each other using the **Sachet** protocol. The protocol provides reliability, mutual authentication, confidentiality, and integrity of all messages. The server aggregates alerts from multiple agents and stores them in a log in a database. The server oversees the working of the agents and controls them by issuing commands to them. It also accepts requests and instructions from the console. The console is a graphical user interface provided to user to configure, control and manage **Sachet**. The console provides powerful

display capability to view alert information and detailed information of each agent. The console also provides capability of creating new signatures and then communicating them to all agents. A console has to authenticate to the server before establishing communication with it. Communication between the console and the server is provided using SACHET Server Console (SSC) protocol which has been described later in the chapter.

In the following sections we discuss the two communicating protocols: SACHET protocol between the agent and the server, and SSC protocol between the server and the console.

3.3 The SACHET protocol

SACHET protocol is used for communication between the server and agents. It is designed to primarily address the issues of security and scalability. If we do not use security features of the Protocol, the whole system could be attacked and rendered ineffective. Possible attacks on the system could be:

- **Deception attack** An attacker may pose as a valid agent and send false alerts to the server. This corrupts our history of attacks. Similarly, it may also pose as the server and try to stop the misuse detector on some machine so that attacks are not detected.

- **Usurpation attack** The packets containing valid alerts may be modified while they are in transit from a agent to the server.

- **Disruption attack** Communication protocol used may also be subject to denial-of-service attacks in which an attacker makes it impossible or difficult for messages to get delivered.

In view of above problems the SACHET protocol should serve the following purposes:

Reliability For reason given later in the section, we cannot implement protocol over TCP. Since UDP does not provide reliability in data-delivery, the **Sachet** protocol must recover from data that is damaged, lost, duplicated or delivered out of order.

Connection Security **Sachet** Protocol should provide privacy and data integrity between two communicating peers to prevent eavesdropping, tampering or message forgery. It can be achieved using symmetric cryptography for data encryption and providing message integrity check using message digest functions. This security service acts as upper layer in the protocol and works over the first layer (reliability service) that is discussed above.

Mutual Authentication There should be an initial handshake protocol which permit Two communicating host can authenticate each other and negotiate on shared cryptographic key. Mutual Authentication should be implemented in such a way so as to provide both entity and key authentication.

Graceful Degradation To provide graceful degradation capability such that **Sachet** should be able to tolerate from agent and server crashes.

One may note that protocol cannot be implemented over TCP, since then server will have to open TCP sockets for maintaining connections with agents. This situation limits scalability on **Sachet** because the operating system puts a limit on the number of sockets that can be created and hence a limit on the number of agents that can be deployed.

3.3.1 General Packet Structure

The **Sachet** protocol packet format is shown in Figure 3.2. The 'EncryptionType' field is used to indicate the encryption method used for encrypting the packet. It has three different values which indicate that packet is either encrypted with public key or with symmetric key or not at all encrypted. 'EncryptionType' field contains fixed values and packets which do not have any of this values are just discarded without any further processing. The 'PacketID' field contains a number that identifies each

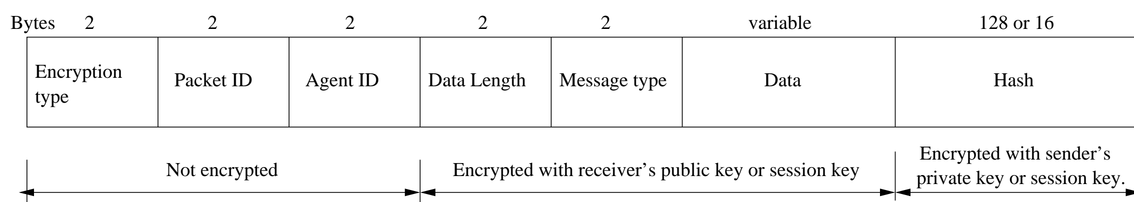


Figure 3.2: Packet Structure of **Sachet**Protocol

unique packet sent or received and can be used for detecting duplicates. Each agent is recognized by the fixed and unique number called agent ID. The 'AgentID' field contains the agent ID of the agent which sent the packet. AgentID value of the server is zero so as to distinguish it from the agents. The 'Data Length' field gives the length of the data portion of the packet in bytes. The maximum length is the maximum amount of data that can be sent by an UDP packet minus the sum of the sizes of all other fields. The 'Message type' field describes the type of message such as if it is an alert, probe, command message etc. The 'data' field contains the value associated with the Message type. For example, the authentication messages contain random numbers in their 'Data' field. The data is encrypted with public key during authentication phase and afterwards with the session key. The 'Hash' field contains the encrypted hash (MD5) for the entire packet. It provides packet integrity and ensures that packet has not been modified or damaged while on its way. The hash is encrypted with private key during authentication and with session key after authentication phase. Here session key refers to shared secret key that is exchanged during the authentication phase. Please refer to Appendix A for complete description of message formats.

3.3.2 Reliability

The **Sachet** protocol is based on the 'Stop and Wait' protocol in which the sender sends one packet and then waits for an acknowledgement before sending the next packet. It starts a timer whenever it sends the packet. If the sender does not receive the acknowledgement within the time out period, it retransmits the packet. If the

packet is not acknowledged even after the transmitting it for MAXRETRYCOUNT of times, then the packet is discarded, and the application is informed. Each packet is identified by the unique packet id assigned by the sender. This needed to detect duplicate packets. The sender maintains a variable 'RTT' which is the current estimate of the round-trip time to the destination. The RTT is used to decide the timeout period and is the exponential average of the time taken for the packets to be acknowledged. In the **Sachet** protocol, every packet has a corresponding response message. Hence, the response message acts as an acknowledgement for the packet.

On the receiving side, the receiver buffers the response message before sending it. This is necessary in the event of receiving duplicate packets. The receiver judge the duplicate or delayed packet by looking at the packet id of the incoming packet. If the packet id of the incoming packet is same as the packet id of the buffered response then the incoming packet is a duplicate packet. In that case, the receiver retransmits the buffered response message.

3.3.3 Authentication

The Authentication mechanism of the **Sachet** protocol allows the client and server to authenticate with each other and negotiate on symmetric cryptographic key before transmitting any application data. The mechanism provides entity and key authentication, key confirmation, and key freshness guarantees for the agreed session key.

Authentication algorithm: We have used RSA as the public key cryptography algorithm. Each communicating host will have a pair of keys (public key and private key). In this case the communicating hosts are: the agent and the server. The authentication protocol is based on the challenge-response method. The authentication messages are as follows.

A $\longrightarrow$ B : A
B $\longrightarrow$ A : $P_B(R_1)$.
A $\longrightarrow$ B : $P_A(R_1,R_2)$.

B $\longrightarrow$ A : $P_B(R_2,K_S,\text{last_alert_id})$.
A $\longrightarrow$ B : Ack, some information to the server.

It is assumed that the agent and the server already know each other's authentic public key. 'A' is the Agent and 'B' is the server. P_S indicates encryption done with the public key of sender S. R_1, R_2 are random numbers and K_S is the session key. The messages 2, 3 and 4 are signed with senders private key while acknowledgment message carries hash encrypted with session key K_S. Message 1 is a plain-text message and does not carry any hash.

The *last_alert_id* contains the alert_id of the last alert that the server had received from the agent. Its need has been discussed in the next chapter. The Acknowledgement message includes some other information relevant to the server such as status of misuse detector, largest value of signature id, etc.

▪ *Session Key Management*

The shared secret key that is negotiated during the authentication phase is also called session key because it is valid only for that session till the agent re-authenticates itself. Every time agent authenticates with server it gets a new session key. If the session lasts for a very long time, then there is a need of changing the session key periodically. It is important because an attacker can otherwise accumulate large amount of encrypted data, making it easier to crack the session key used for communication. We change session key after 1 hour of session or if more than 200 MB of data has been exchanged. The server keeps track of the above parameters for each agent and initiates a key reset after the expiry of the current session key.

The server initiates a 'key reset' command to the agent. This message also contains the new key to be used and is encrypted using the existing key. The server will not send any other commands to the agent until it receives a reply for this message from the agent. Also, if at this point of time, it receives any message from the agent encrypted with key used previously, it will discard that message. The agent, on

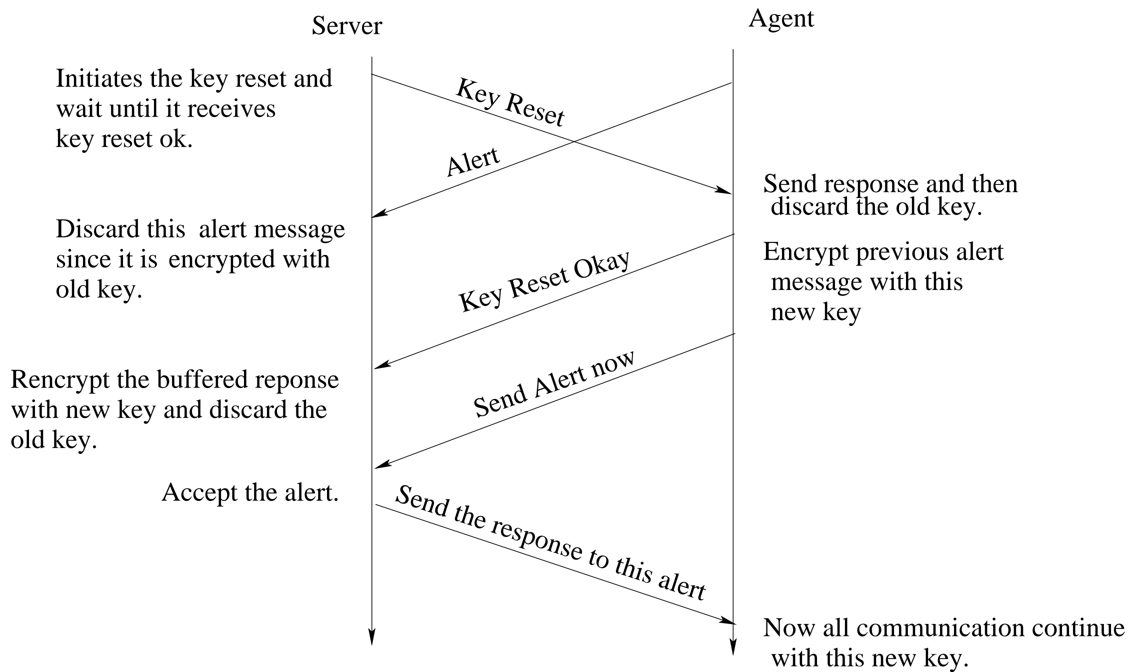


Figure 3.3: Key Reset Implementation

receiving the key reset command, sends a 'key reset ok' reply which also contains the new key encrypted with previous session key. The agent also discards the existing key and starts using the new key for further communication with the server. If the agent had previously sent any packet to the server, encrypted with the old key for which acknowledgement has not yet been received, it encrypts that packet again with the new key, and sends it to the server. The server, on receiving 'key reset ok' message, immediately discards the existing key and starts using new key. The following procedure is shown in figure 3.3.

3.3.4 Commands

Command messages are sent by the server for controlling and configuring agents. The commands to a agent include: starting/stopping misuse detector, enabling/disabling attack signatures, adding/deleting attack signatures, requesting a list of attack signatures, 'key reset' for changing key, *etc.* The agent acts on these commands and

replies to the server along with the status of command execution, *i.e.*, success or failure, and if possible , the reason for success or failure. For detailed format of command and reply messages, please see the Appendix A.

3.3.5 Alerts

Alerts describe the network attacks detected by misuse detector by analyzing the network-traffic. An alert is first generated by the misuse detector and contains information like type of attack, attack description, signature id of attack signature that matches with this attack, timestamp, source and destination IP address, and port numbers. The misuse detector passes alert to the agent which assigns a unique alert_id to each alert. These alerts are subsequently sent to the server by the agent. Usually many alerts are communicated in a single packet along with their alert_ids for efficiency. The server accepts all alerts and sends reply back to the agent containing list of alert_ids received and logs these alerts to the database. Refer Appendix A for packet format of alert messages.

3.3.6 Graceful Degradation

The SACHET protocol helps in providing a graceful degradation service to SACHET. If any component crashes or restarts, it should not disable the entire system, nor should it bring the system to an inconsistent state. We will discuss some scenarios now that illustrate how system detects failures and recovers from them.

- *Server crashes*

When a agent starts, it sends probe messages to find out the state of the server. If the server is alive, the agent receives probe reply message 'SERVER_ALIVE' from the server. Only when the agent receives a reply to its probe message, it starts the authentication process with the server. After successful authentication, the agent stops probe packets to the server. Now, it may happen that server crashes or restarts after the authentication phase. This may give rise the following two scenarios:

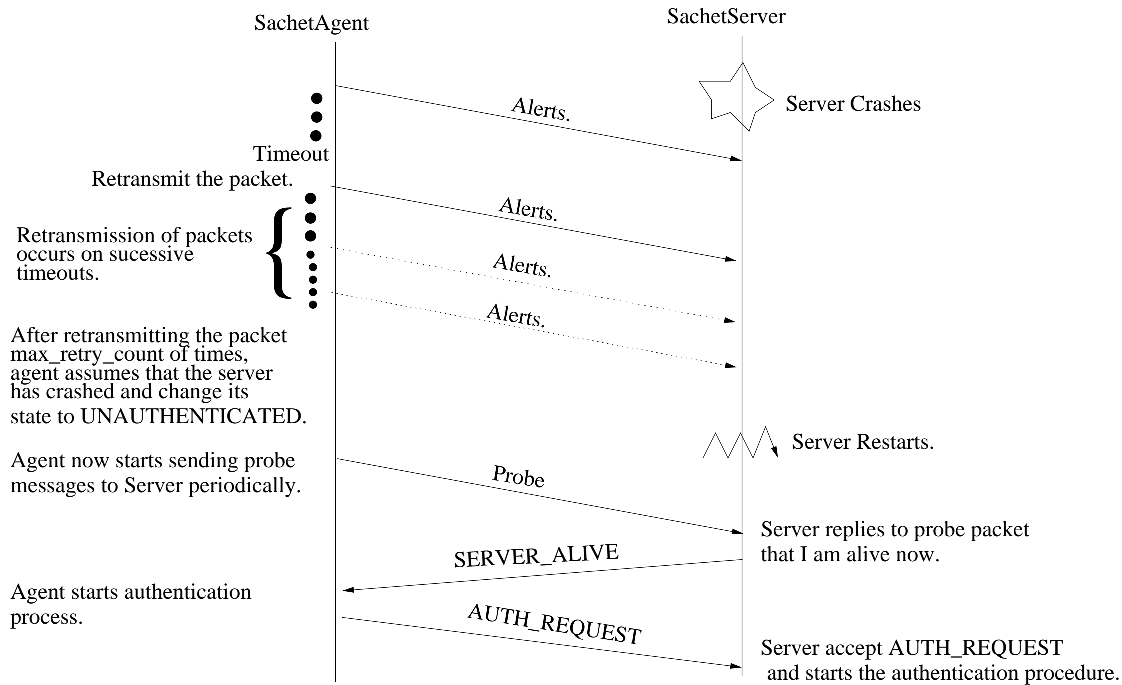


Figure 3.4: Server crashes but has not been restarted

- **Server crashes but has not been restarted** The agent, initially, will have not know that the server has crashed. The agent will continue to send alerts to the server assuming that the server is alive, and wait for an acknowledgement. The agent will not receive any response to the alert packet, and will retransmit the packet again. It will retransmit the packet MAXRETRYCOUNT number of times and then save the packet for future transmission, and change its state to unauthenticated. Then, it will start sending periodic probe messages to the server until it receives reply from the server. After receiving reply, it will again start authentication process with the server. This scenario is shown in Figure 3.4.

- **Server crashes and recovers quickly** The agent will continue to send alerts to the server since it does not know about the server failure. When the server recovers, it assumes every agent to be in unauthenticated state. The server will

continue to discard alert packets received from an agent until the agent authenticates with it. The Agent will retransmit alert packet MAXRETRYCOUNT number of times and it will change its state to unauthenticated. The agent will start sending probes to the server. It will receive reply from the server immediately and hence will start the authentication process. This scenario is shown in Figure 3.5.

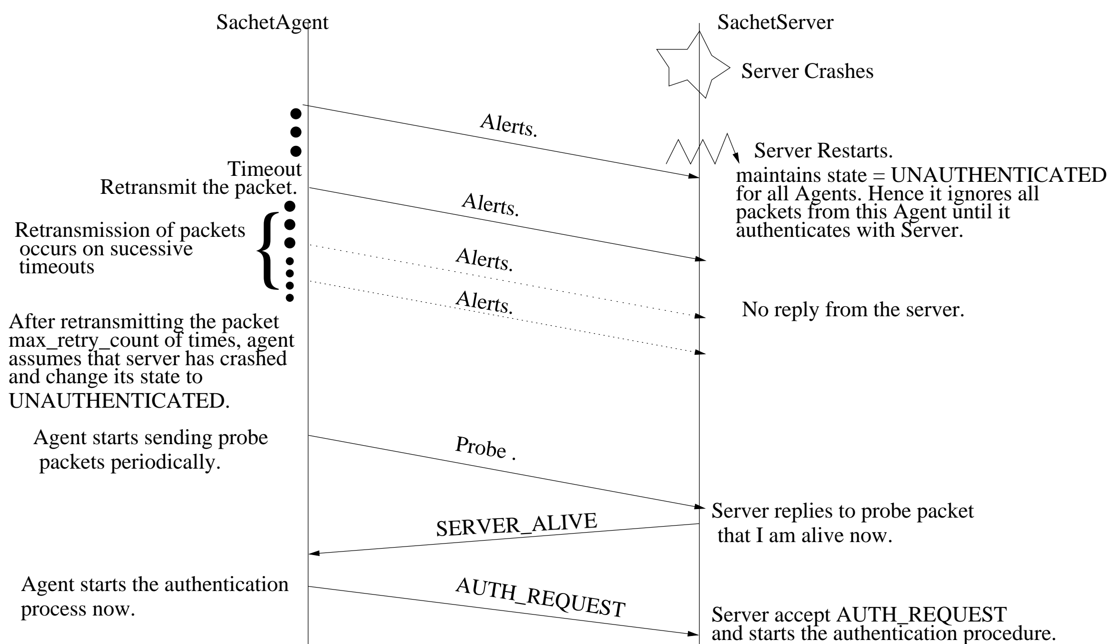


Figure 3.5: Server crashes and recovers quickly

In both scenarios, alerts received from the misuse detector will be buffered in the agent's memory. There is a limit to the size of memory buffer and currently it can store a maximum of 10000 alerts. As soon as the server recovers, the agent will send all these alerts.

- *Agent crashes*

The server periodically sends probe messages to all authenticated agents to know about their state. Here the probe messages are encrypted with session key specific

to that agent. Here also we are assuming that the failure takes place after the authentication is complete. It does not matter whether the server is waiting for a reply of a probe messages or a command message. The situation is similar for both message types.

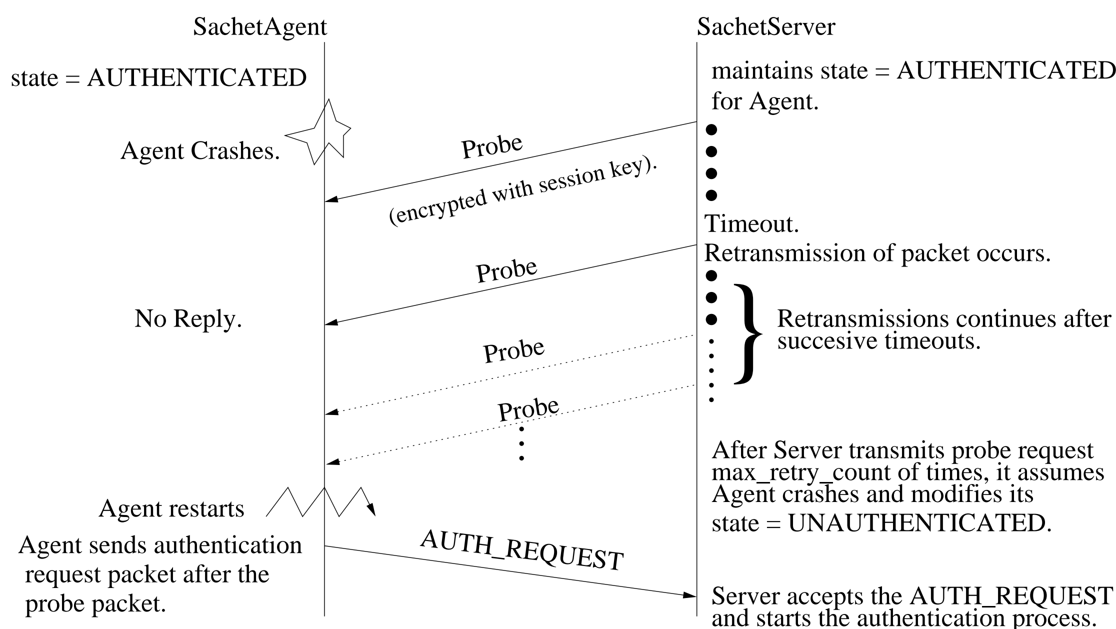


Figure 3.6: Agent crashes but has not recovered

- **Agent crashes but has not recovered** If the server does not receive reply to a probe or command message from an agent, it retransmits the message. Even after retransmitting the message for MAXRETRYCOUNT number of times, if the server does not get a reply, it assumes that the agent has failed and hence changes the state of this agent to unauthenticated. This scenario is shown in Figure 3.6.

- **Agent agent crashes and recovers quickly** When the agent recovers, it is in unauthenticated state, and does not know the previous session key. Therefore it cannot reply or acknowledge server messages. It will start the authentication phase. It may happen that while the server is retransmitting messages,

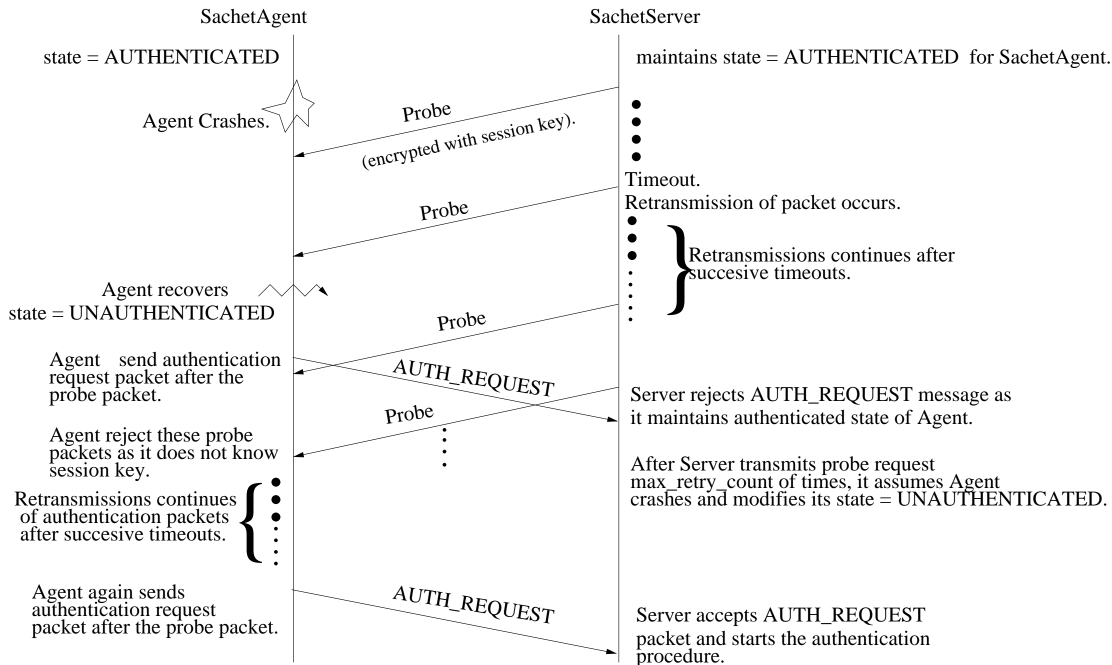


Figure 3.7: Agent crashes and recovers quickly

it receives a packet from the agent, which has recovered quickly, requesting the server to authenticate it. The server rejects authentication request from an agent if that agent is already in authenticated state so as to avoid denial of service attacks. In this case also the server rejects authentication request from the agent since it does not know that agent had failed. The server only comes to know about failure of the agent when it does not get reply for probe or command message until it has retransmitted a message MAXRETRYCOUNT number of times. Finally, it recognizes authentication request message from the agent and starts the authentication procedure. This scenario is shown in Figure 3.7.

The server finally displays the status of the agent in main screen of the console. If an agent is not alive for a long time, the administrator can take appropriate action.

- *Misuse Detector crashes*

The Misuse Detector is started by the control module as its child process during startup. Hence Misuse Detector runs as a separate process but is controlled by the control module. The control module periodically checks whether it is running or not. If control module finds that the misuse detector has failed, it first tries to restart it. If it fails to restart the misuse detector it immediately sends a message 'MISUSE_DETECTOR_FAILED' message to the server. The server communicates the status of the misuse detector to the console which displays it on the agent screen. The system administrator can then take appropriate action.

3.4 The Sachet Server-Console Protocol

The **Sachet** Server-Console (SSC) protocol is mainly designed for local communication between the server and the console. The console can control and manage the sachet server only through this protocol. The console must authenticate to the server before issuing any instructions or requests. This protocol is implemented over TCP so that the console need not authenticate every time to server before sending any instruction or request to it. The server and console should be installed on the same host and the server must accept connection requests from the console from the localhost only. After accepting a connection from the console, the server first checks for the user name and password received from the console, and verifies it. If verification fails it immediately terminates the connection, otherwise it is ready to accept packets from the console. The **Sachet** server-console protocol packet format is shown as below:

2 bytes	2 bytes	Variable
Packet Length	Message Type	Data Value

Figure 3.8: Packet Structure of **Sachet** server-console protocol

The 'Packet Length' is the size of the complete packet in bytes. The 'Message Type' indicates the type of packet. The packet can be either a command-message/request-message/response-message. The 'Value' field contains the meaningful data that is communicated and is specific to the message type. For detailed description of the message types and the format of the **Sachet** Server-Console protocol, packets, please refer to Appendix B.

Chapter 4

Implementation of the SACHET

In this chapter we discusses issues in the implementation of the **SACHET** system. The **SACHET** system has been implemented on three major platforms: Linux, Windows 2000 and Solaris. The server and agent are implemented in C language while the console is implemented using Java.

In section 4.1, 4.2 and 4.3 we briefly describe the functionality of the server, agent and console respectively. In rest of the chapter we discuss issues in the implementation of **SACHET** Protocol.

4.1 The Server

The server is a central command authority for controlling and managing multiple agents which are deployed at critical points of an enterprise network. It is the nerve centre of the intrusion detection system that allows consolidation of alerts from multiple agents and stores these alerts in the database. It usually runs in background as a daemon or service and is installed on a dedicated machine. The server does not have its own user interface and hence cannot directly interact with the user. But it can be accessed through various other interfaces: web interface, command line interface or graphical user interface. We have implemented GUI console for controlling the server, although the other two interfaces can be easily incorporated into it. The server communicates with the console, which is a separated process,

using a simple request-response protocol in which the console sends a request for some information and the server responds by providing appropriate information or result. The user (system administrator) needs to authenticate himself to server before using the interface. The server periodically monitors the health of each agent and reports it to the console. It maintains information about agents in a database and retrieves it at the beginning of its execution. The server maintains the state of each agent and follows the state engine as shown below:

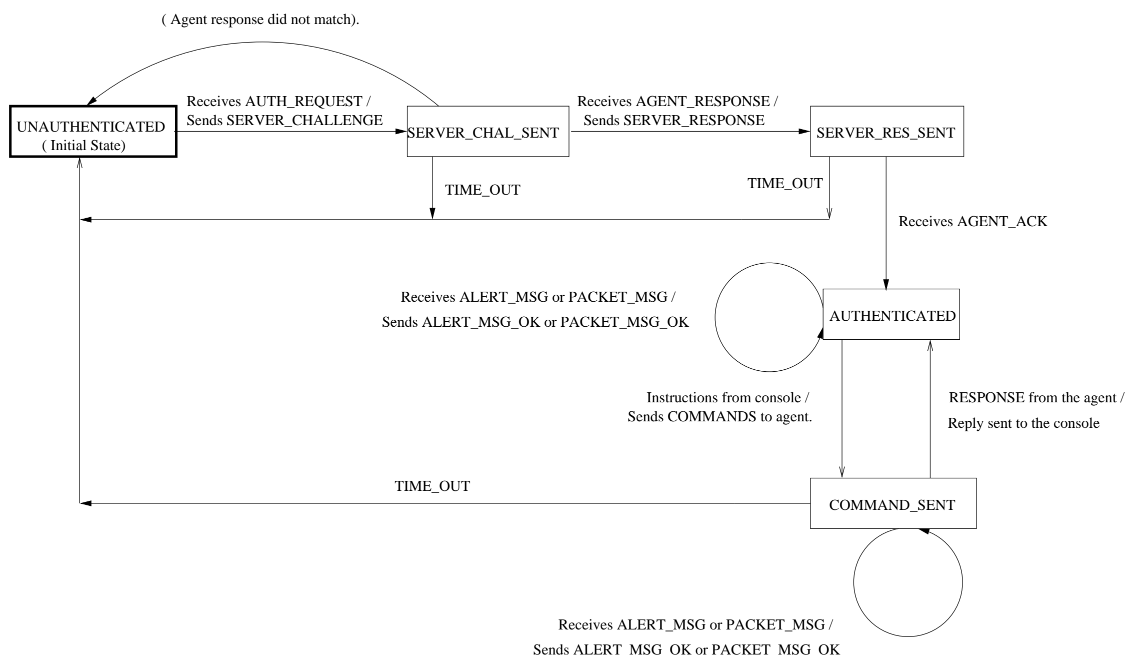


Figure 4.1: State diagram of the server (with respect to a specific agent)

4.2 The Agent

The agent passively monitors either the entire network traffic on a LAN segment, or only the network traffic received by a host. It reports any suspicious activity as alerts to the server over a secure channel using the protocol. It is a console based application which can run in background and does not interact with the user. It needs to authenticate itself before it can communicate with the server. After it has

been authenticated, it sends all the alerts generated by it to the server and accepts commands from the server and execute them locally. The agent comprises of two sub-components: Misuse detector and control module. These sub-components run as separate processes on the target host.

The misuse Detector runs as a child process of the control module. The misuse detector monitors the network-traffic, searches for pre-defined patterns or signatures of misuse and generates alerts. Then it passes on the alert and the corresponding packet (that triggered the alerts), to the control module. In this project we have used *snort* as the misuse detector.

Snort is an open-source network intrusion detection system, capable of performing real-time traffic analysis and packet logging on IP networks. It features rule-based logging and can perform protocol analysis and content searching/matching in order to detect variety of attacks and probes, such as stealth port scans, CGI attacks, SMB probes, OS fingerprinting attempts etc. Snort has a Plugin architecture that facilitates in extending its detection and reporting subsystems. It provides the facility of writing output modules which utilizes this plugin architecture and allow Snort to be much more flexible in the formatting and presentation of output to its users. The output modules are run when the alert or logging subsystems of Snort are called. Multiple output plugins may be specified in the Snort Configuration file. When multiple plugins of the same type (log, alert) are specified, they are stacked and called in sequence when an event occurs. Output modules are loaded at runtime and specified as a rule in Snort Configuration file. In our case, we have written an output module which communicates alerts and packets generated by snort to the control agent through a UDP socket.

The control module controls the Snort process by sending it appropriate signals. For example, SIGHUP signal is sent for restarting snort. Hence, Snort can be stopped/started/restarted as desired by the control module. The control module also periodically monitors the Snort and report its status to the server. The state diagram of the agent is as shown below:

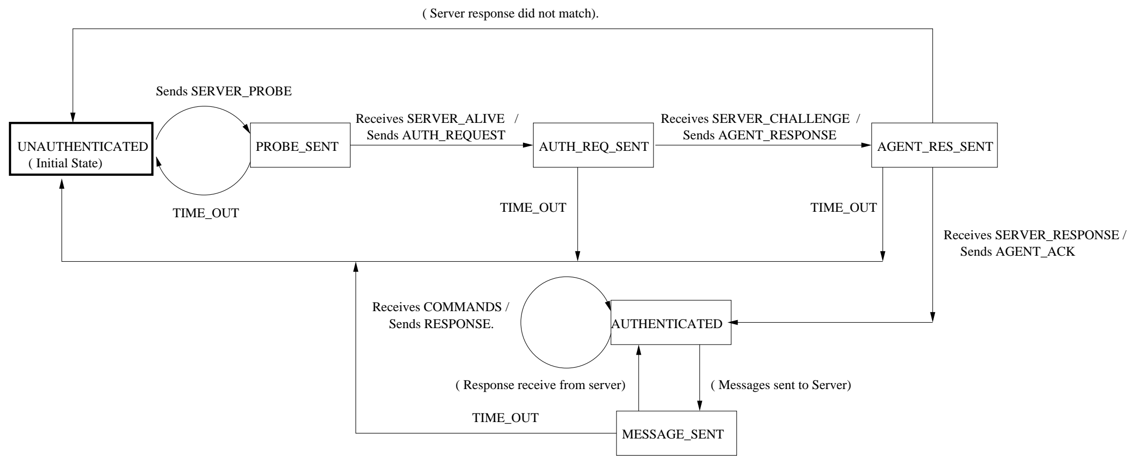


Figure 4.2: State diagram of the agent

4.3 The Console

The console provides a GUI to the system administrator for controlling the entire system. It forms the most important operational component from the point of view of the system administrator since one can monitor and view the activities of entire SACHET IDS using this GUI. More importantly, it is used to present the information in such a manner such that it can be used in the context of surveillance and decision support of the system. For example, the system administrator, by viewing the number of alerts generated at each penetration point, can find out which hosts are mainly targeted by attackers. Then system administrator can decide to take certain actions such as increasing the surveillance on those systems, or reconfiguring the router or firewall to block all incoming data from the IP address of the machine that caused Snort to generate alerts.

The console interacts with the server using the SACHET server-console (SSC) Protocol. On behalf of the system administrator, it instructs the server to issue commands (disabling/enabling of signatures or classes of signatures, adding new signatures etc.) to the agent and report responses. It provides the means to add and delete agents without disrupting the server. Moreover, the console periodically

requests the server to provide information about the entire system. The console and server should be run on the same host and system administrator needs to authenticate with the server before interacting with it.

The console shows the status information of each agent in a grid as a top-level screen. Figure 4.3 shows the top-level screen. The screen focuses on displaying the basic information about agents, such as condition of the agent (alive or dead), agent-id, IP address of agent, etc. Double clicking any agent shows the advanced agent screen (Figure 4.4). This screen provides detailed information about agent such as the alerts generated with their descriptions, last time the agent authenticated with the server, list of classes of signatures that are enabled etc. This screen also facilitates the system administrator to configure the agent. The console also has an alert reporting screen (Figure 4.5) which displays the alerts received from all the agents. The system administrator can select the time period (in days and hours) to view the alerts that were generated during this period. By default, the screen displays alerts generated in the last 30 minutes. The console retrieves alert information from the database.

The console can also display the list of all attack signatures (Figure 4.6), used by the misuse detector for detecting network attacks, with their complete description (signature id, functionality of the attack, class of attack it belongs to, its references and URL links). It retrieves all this information from the database. The most significant capability of console is that of allowing creation of new attack signatures via a template and sending it to all the agents through the server. Figure 4.7 shows the template for creating new signatures.

4.4 Addition of New Signatures

The console allows the system administrator to create new attack signatures through a template. These special user-created signatures are assigned signature-ids starting from 2,00,000 onwards. This restriction is imposed by Snort itself to distinguish standard signatures from the user created ones. The server and the agent maintain

a variable for storing the largest signature-id among all the user-created signatures. This is to ensure that all all the agents have consistent information about these new signatures and must add these signatures to the database of the misuse detector. At any given time, it is possible that the server is not communicating with all the agents. Therefore, when the signature is created, the server will not be able to propagate this new signature information to those agents which are currently not authenticated to the server. When any of these agents start up, it sends its maximum signature-id to the server in the acknowledgement message of the authentication phase. The server compares its maximum signature-id to that of agent. If there is any difference it sends the remaining user-created signatures to the agent.

On receiving new signatures from the server, the agent adds them to a particular file, which only stores new signatures for misuse detector, and restarts the snort.

4.5 Maintenance of Alert id

The server stores alert-id of the last alert it received from the agent in the database. It maintains last alert-ids of all the agents and retrieves this information when it starts up. This helps in maintaining the information about alerts consistent in the event of agent or the server failure. When the server restarts, it uses the last alert-id of the particular agent to accept only those alerts having theirs ids greater than the last alert-id of this agent. This allows the server to discard alerts. The server sends this last alert-id to the agent when it authenticates with the server. When the agent restarts, it accepts this last alert-id from the server so that it can assign alert-ids to the new alerts it receives from the misuse detector.

4.6 Public Key Management

As mentioned earlier, the **Sachet** protocol assumes that both communicating hosts have authentic copies of each other's public key. Therefore, the server needs to maintain public keys of all agents. Also, an agent should know the public key of the server. Regarding the server acquiring the public key of the agent, we have adopted

the following approach. During the installation of the agent, a public and private key pair for the agent is generated. The public key is then manually transferred (through CD, for example) by system administrator to the machine where the server is running. Through the console we add detailed information of that particular agent (Agent id, public key, IP address) to the database and inform the server about the new agent. Thus the server stores the public key of each agent in the database.

A agent can acquire the public key of the server either in a similar manner as described above, or it can simply send a plain-text UDP message to the server requesting for its public key. All of this will happen during the installation of agent. Note though, that the latter method is not very secure.

4.7 Private Key Storage

A common problem is any subsystem that uses cryptography is the secure storage of private keys. In the SACHET system, the server and the agents need to store their respective private RSA keys. Storing the private key unencrypted on disk is clearly insecure because if anyone is able to gain access to these keys then (s)he can corrupt the entire intrusion detection system by introducing false agent or server in the system. However the alternative of encrypting it with the key derived from a passphrase implies user intervention at system startup time. In our implementation, therefore, the use of a passphrase to encrypt the private key is optional, in both the server and the agent. If a passphrase is used, the private key is encrypted using 3DES with the MD5 checksum of the passphrase as the key.

AgentId	Hostname	IpAddress	AgentState	SnortState	Alive Since	Signatures
1000	ids2		NOT_ALIVE			
2000	ids1		NOT_ALIVE			
3000	ids3		NOT_ALIVE			
4000	pp4		NOT_ALIVE			
5000	pp3		NOT_ALIVE			
6000	pp1		NOT_ALIVE			
7000	cluster1	172.27.1.3	ALIVE	RUNNING	11/15/03 11:45 PM	1779
8000	cluster2	172.27.1.5	ALIVE	RUNNING	11/15/03 11:49 PM	1779
9000	cluster3	172.27.1.7	ALIVE	RUNNING	11/15/03 11:52 PM	1779
10000	cluster4		NOT_ALIVE			
12000	ids3	172.27.2.8	ALIVE	RUNNING	11/16/03 12:11 AM	1714

Figure 4.3: Top-Level agent screen of Console

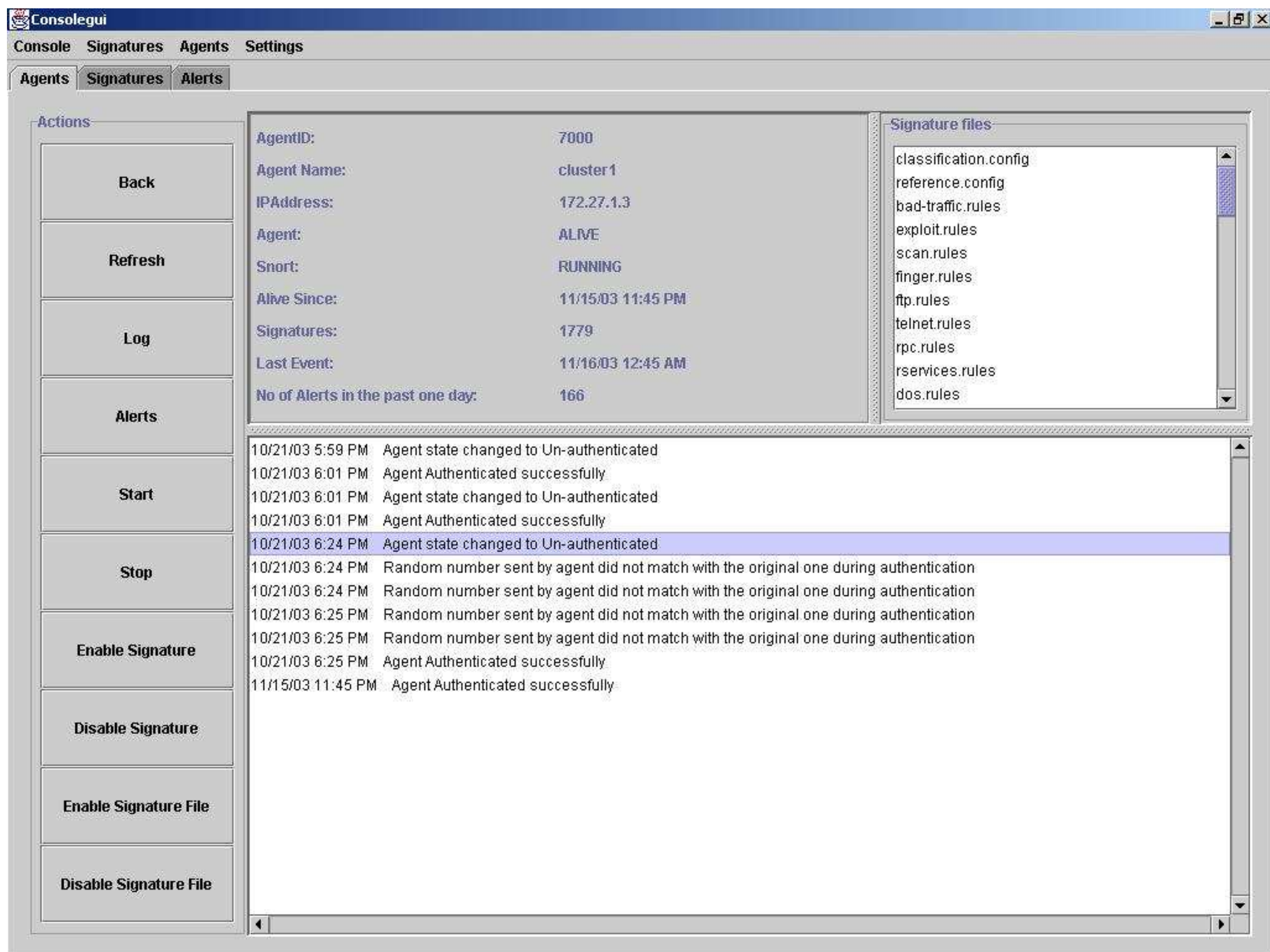


Figure 4.4: Agent screen of Console

Consolegui

Console Signatures Agents Settings

Agents | Signatures | Alerts

Options for viewing the Alerts

View the Alerts of last : 7 Days 12 Hours Go

AgentId	AlertId	TimeStamp	Rule Sid	Alert Message
12000	1	6/9/04 11:16 PM	1465	WEB-CGI auktion.cgi access
12000	2	6/9/04 11:16 PM	1364	WEB-ATTACKS lsof command attempt
12000	3	6/9/04 11:16 PM	1361	WEB-ATTACKS nmap command attempt
12000	4	6/9/04 11:16 PM	1870	WEB-CGI siteUserMod.cgi access
12000	5	6/9/04 11:16 PM	1363	WEB-ATTACKS X application to remote host attempt
12000	6	6/9/04 11:16 PM	1868	WEB-CGI story.pl arbitrary file read attempt
12000	7	6/9/04 11:16 PM	1823	WEB-CGI AlienForm af.cgi directory traversal attempt
12000	8	6/9/04 11:16 PM	1465	WEB-CGI auktion.cgi access
12000	9	6/9/04 11:16 PM	1361	WEB-ATTACKS nmap command attempt
12000	10	6/9/04 11:16 PM	1870	WEB-CGI siteUserMod.cgi access
12000	11	6/9/04 11:16 PM	1868	WEB-CGI story.pl arbitrary file read attempt
12000	12	6/9/04 11:16 PM	1364	WEB-ATTACKS lsof command attempt
12000	13	6/9/04 11:16 PM	1823	WEB-CGI AlienForm af.cgi directory traversal attempt
12000	14	6/9/04 11:16 PM	1363	WEB-ATTACKS X application to remote host attempt
12000	15	6/11/04 2:57 AM	1364	WEB-ATTACKS lsof command attempt
12000	16	6/11/04 2:57 AM	1823	WEB-CGI AlienForm af.cgi directory traversal attempt
12000	17	6/11/04 2:57 AM	1361	WEB-ATTACKS nmap command attempt
12000	18	6/11/04 2:57 AM	1868	WEB-CGI story.pl arbitrary file read attempt
13000	8	6/10/04 12:35 AM	1361	WEB-ATTACKS nmap command attempt
13000	7	6/10/04 12:35 AM	1542	WEB-CGI cgimail access
13000	6	6/10/04 12:35 AM	1868	WEB-CGI story.pl arbitrary file read attempt
13000	5	6/10/04 12:35 AM	1465	WEB-CGI auktion.cgi access
13000	3	6/9/04 11:12 PM	1823	WEB-CGI AlienForm af.cgi directory traversal attempt
13000	2	6/9/04 11:12 PM	1465	WEB-CGI auktion.cgi access
13000	1	6/9/04 11:12 PM	1868	WEB-CGI story.pl arbitrary file read attempt
13000	4	6/9/04 11:16 PM	1364	WEB-ATTACKS lsof command attempt
1000	1	6/11/04 2:58 AM	1542	WEB-CGI cgimail access
1000	2	6/9/04 11:16 PM	1465	WEB-CGI auktion.cgi access
1000	3	6/9/04 11:16 PM	1364	WEB-ATTACKS lsof command attempt

Figure 4.5: Alert Reporting screen of Console

Consolegui

Console Signatures Agents Settings

Agents | Signatures | Alerts

Rule Id	Rule Description	Class Name	Class Description	Class Priority	References
304	EXPLOIT sco calserver overflow	attempted-admin	Attempted Administrator Privileg...	1	http://www.securityfocus.com/bid/235
305	EXPLOIT delegate proxy overflow	attempted-admin	Attempted Administrator Privileg...	1	http://cve.mitre.org/cgi-bin/cvename.c
306	EXPLOIT VQServer admin	attempted-admin	Attempted Administrator Privileg...	1	http://cve.mitre.org/cgi-bin/cvename.c
307	EXPLOIT CHAT IRC topic overflow	attempted-user	Attempted User Privilege Gain	1	http://www.securityfocus.com/bid/573
308	EXPLOIT NextFTP client overflow	attempted-user	Attempted User Privilege Gain	1	http://cve.mitre.org/cgi-bin/cvename.c
309	EXPLOIT sniffit overflow	attempted-admin	Attempted Administrator Privileg...	1	http://www.whitehats.com/info/IDS27
310	EXPLOIT x86 windows MailMax overflow	attempted-admin	Attempted Administrator Privileg...	1	http://cve.mitre.org/cgi-bin/cvename.c
311	EXPLOIT netscape 4.7 unsucessful overflow	unsuccessful-user	Unsuccessful User Privilege Gain	1	http://www.whitehats.com/info/IDS21
312	EXPLOIT ntpdx overflow attempt	attempted-admin	Attempted Administrator Privileg...	1	http://www.securityfocus.com/bid/254
313	EXPLOIT ntalkd x86 linux overflow	attempted-admin	Attempted Administrator Privileg...	1	http://www.securityfocus.com/bid/210
314	DNS EXPLOIT named tsig overflow attempt	attempted-admin	Attempted Administrator Privileg...	1	http://www.securityfocus.com/bid/230
315	EXPLOIT x86 linux mountd overflow	attempted-admin	Attempted Administrator Privileg...	1	http://www.securityfocus.com/bid/121
316	EXPLOIT x86 linux mountd overflow	attempted-admin	Attempted Administrator Privileg...	1	http://www.securityfocus.com/bid/121
317	EXPLOIT x86 linux mountd overflow	attempted-admin	Attempted Administrator Privileg...	1	http://www.securityfocus.com/bid/121
320	FINGER cmd_rootsh backdoor attempt	attempted-admin	Attempted Administrator Privileg...	1	http://www.sans.org/y2k/fingerd.htm
321	FINGER account enumeration attempt	attempted-recon	Attempted Information Leak	2	http://cgi.nessus.org/plugins/dump.p
322	FINGER search query	attempted-recon	Attempted Information Leak	2	http://www.whitehats.com/info/IDS37
323	FINGER root query	attempted-recon	Attempted Information Leak	2	http://www.whitehats.com/info/IDS37
324	FINGER null request	attempted-recon	Attempted Information Leak	2	http://www.whitehats.com/info/IDS37
326	FINGER remote command ; execution attempt	attempted-user	Attempted User Privilege Gain	1	http://www.whitehats.com/info/IDS37
327	FINGER remote command pipe execution att...	attempted-user	Attempted User Privilege Gain	1	http://www.whitehats.com/info/IDS38
328	FINGER bomb attempt	attempted-dos	Attempted Denial of Service	2	http://cve.mitre.org/cgi-bin/cvename.c
330	FINGER redirection attempt	attempted-recon	Attempted Information Leak	2	http://cve.mitre.org/cgi-bin/cvename.c
331	FINGER cybercop query	attempted-recon	Attempted Information Leak	2	http://cve.mitre.org/cgi-bin/cvename.c
332	FINGER 0 query	attempted-recon	Attempted Information Leak	2	http://cve.mitre.org/cgi-bin/cvename.c
333	FINGER . query	attempted-recon	Attempted Information Leak	2	http://cve.mitre.org/cgi-bin/cvename.c
334	FTP .forward	suspicious-filename-det...	A suspicious filename was dete...	2	http://www.whitehats.com/info/IDS31
335	FTP .rhosts	suspicious-filename-det...	A suspicious filename was dete...	2	http://www.whitehats.com/info/IDS32
336	FTP CWD ~root attempt	bad-unknown	Potentially Bad Traffic	2	http://www.whitehats.com/info/IDS31
337	FTP CEL overflow attempt	attempted-admin	Attempted Administrator Privileg...	1	http://www.whitehats.com/info/IDS25
353	FTP adm scan	suspicious-login	An attempted login using a susp...	2	http://www.whitehats.com/info/IDS33
354	FTP iss scan	suspicious-login	An attempted login using a susp...	2	http://www.whitehats.com/info/IDS33
355	FTP pass wh00t	suspicious-login	An attempted login using a susp...	2	http://www.whitehats.com/info/IDS32
356	FTP passwd retreval attempt	suspicious-filename-det...	A suspicious filename was dete...	2	http://www.whitehats.com/info/IDS21
357	FTP piss scan	suspicious-login	An attempted login using a susp...	2	-
358	FTP saint scan	suspicious-login	An attempted login using a susp...	2	http://www.whitehats.com/info/IDS33

Add signature to the database

Figure 4.6: Screen depicting list of attack signatures in Console

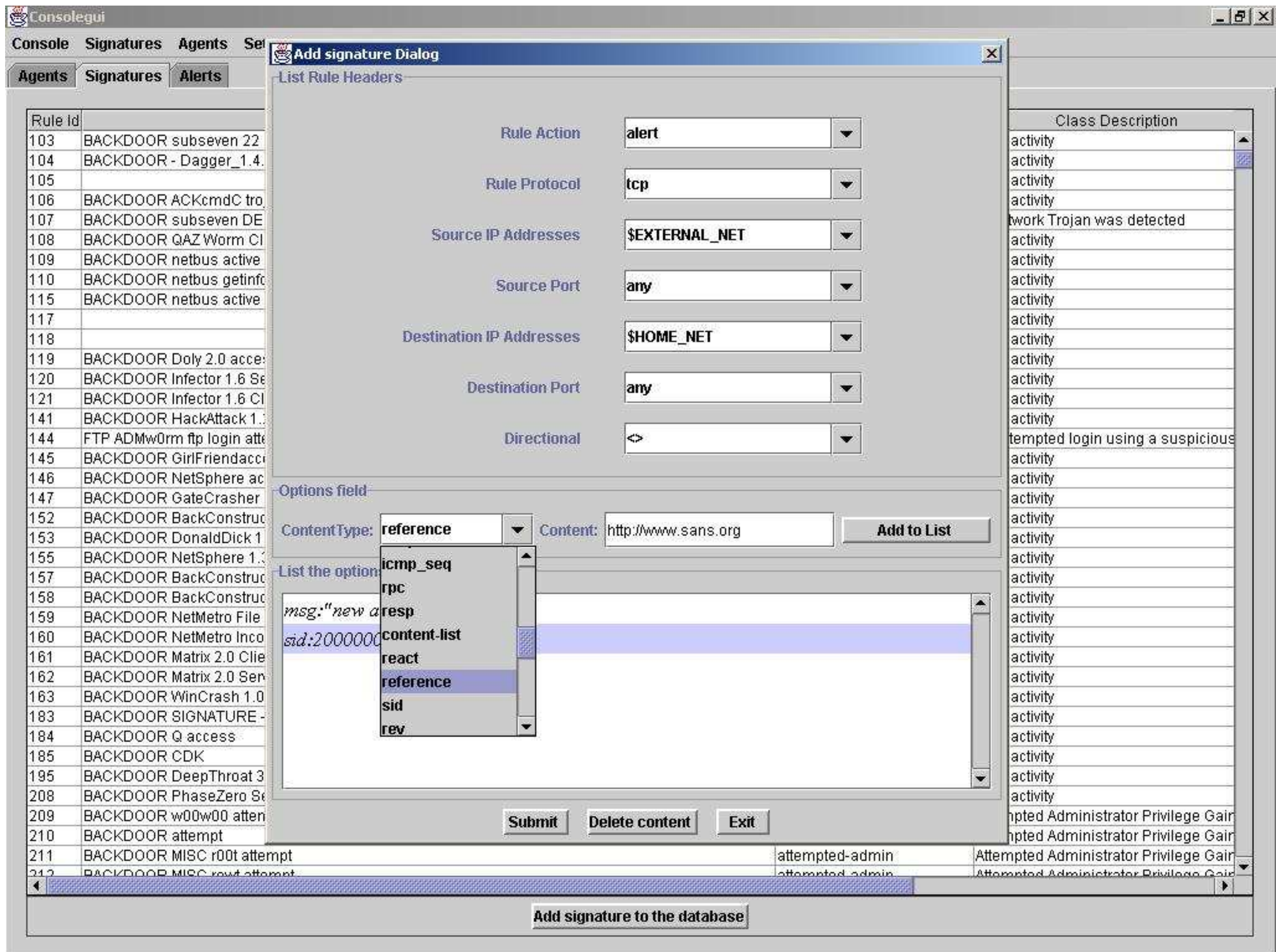


Figure 4.7: Template for creating new attack signature in Console

Chapter 5

Conclusion and Future Work

We have designed and implemented a distributed architecture for Intrusion Detection System called Sachet, that employs independent entities called *agents* for performing monitoring and analysis of network traffic at various penetration points of the organization. Each agent uses Snort as the misuse detector to detect attacks and report these attacks to a centralized server where they are stored in database for further analysis. The agents communicate with the server using the Sachet Protocol that provides reliability, mutual authentication, security and graceful degradation feature. A GUI is provided as an interface for accessing by the system administrator for monitoring the entire Sachet system.

This work can be extended in several ways mentioned below:

- It is possible that the alerts generated across multiple agents are all related to the same attack. Distributed Denial of Service attacks and stealth probes are examples of such attacks. The facility should be provided to correlate alerts from multiple agents to detect these type of attacks.

- The other issue is the large number of false positives generated by Snort. This happens partly beacuse Snort does not reconstruct higher layers in the protocol stack (such as HTTP, SMTP, etc.). For example, if a particular exploit involves finding a certain string in the URL of an HTTP GET request, Snort will alert even if the string appears innocuously in the cookies that accompany the GET request. Although, this contributes to its speed, false

alarms may overwhelm the system administrator giving them no opportunity to focus on relatively few events of real interest.

- GUI needs further improvement. It is because an attacker can directly target the user interface. An attacker can deliberately generate large number of spurious packets purely for the purpose of triggering the intrusion detection system. In this way, she can overflow the console with alerts and prevent the analyst from noticing some small number of more serious intrusions, which represent the attacker attempting her true goal. Hence, the console needs to be carefully designed to foil this decoy attack from succeeding. One of the solution lies in providing multiple levels of alert views such as viewing of alerts by categorization of attacks, source IP address etc.

- There are some attacks which results in generation of very large number of alerts by Snort. For exmple, Probes, Denial of Service attacks are such attacks. One needs to apply some 'Data Reduction' techniques either at the agent or at the server to reduce these large number of alerts, all of which refer to the same attack, to a single alert that solely represent the attack.

Appendix A

Formats of Messages in Sachet Protocol

In this appendix, we describe formats of the messages exchanged between the agent and the server using the SACHET Protocol. The Messages can be of following type: authentication, command, response, alert, probe. We have only shown the 'Meesage Type' and 'data' field of the packet of various messages. The packet format of SACHET Protocol is shown below:

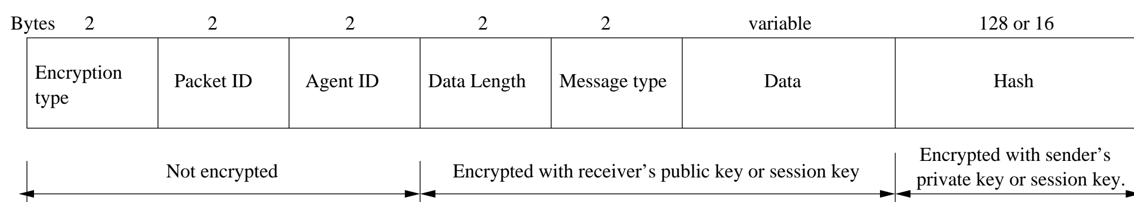


Figure A.1: Packet Structure of SACHETProtocol

The 'Encryption type' field describe the properties of the packet and can attain any of the four values. Only NO_HASH value can be used as in combination with other values.

NO_ENCRYPT The packet contains plain-text message and data.

NO_HASH The hash has not been computed over the packet.

SYMMETRIC_ENCRYPT The contents of the packet are encrypted with session key using symmetric cipher algorithm.

RSA_ENCRYPT The contents of the packet are encrypted with receiver's public key.

A.1 Authentication Messages

The following sequence of messages are exchanged during the authentication phase. The first message is sent by the agent to the server to start the mutual authentication mechanism.

Message Type (code)	Data (bytes)
AUTH_REQUEST (20)	Empty (0)
CONSOLE_CHALLENGE (21)	Rand1 (16)
AGENT_RESPONSE (22)	Rand1 (16), Rand2(16)
CONSOLE_RESPONSE (23)	Rand2 (16), Secret Key (14), last alert ID (2)
AGENT_ACK (24)	Snort_status (2), Number of signatures enabled (4), Max. signature id value (4)

Table A.1: Messages exchanged during authentication phase

Where Rand1, Rand2 are 16 byte random numbers.

A.2 Data Messages

Data exchange takes place only after the success of authentication phase. Data messages received before the completion of authentication phase are ignored by the receiver. Data can be alerts from agent to console or commands from console to agent. Every data message has two possible replies - one indicating success and the other indicating failure.

Messages from the server to a agent include the following:

Key reset command: This command message tells the agent to use new session key for further communication. The message exchanges are:

Message Type (code)	Data (bytes)
KEY_RESET (30)	New Key (14)
KEY_RESET_OK (518)	New Key (14)

Table A.2: Message excahnges for Key-reset command

Enabling and disabling signatures: Each standard signature has a unique SID. Thus it is sufficient to just mention the SID in the message instead of the entire signature. More than one SID can be mentioned in this message. The server expects a reply from the agent which indicates either success or failure or that the signature is already enabled or disabled. The message exchange sequence is shown below.

Message Type (code)	Data (bytes)
ENABLE_SIGNATURE (31)	SID (4), SID (4) ....
ENABLE_SIGNATURE_REPLY (502)	SID (4), reply_code (2), SID (4), reply_code (2) ...

Table A.3: Messages for enabling signatures

Where reply-code can be one of the following: ENABLE_SIGNATURE_OK (503), ENABLE_SIGNATURE_FAILED (504), ENABLE_SIGNATURE_ALREADY(505).

Message Type (code)	Data (bytes)
DISABLE_SIGNATURE (32)	SID (4), SID (4) ....
DISABLE_SIGNATURE_REPLY (506)	SID (4), reply_code (2), SID (4), reply_code (2) ...

Table A.4: Messages for disabling signatures

Where reply-code can be one of the following: DISABLE_SIGNATURE_OK (507), DISABLE_SIGNATURE_FAILED (508), DISABLE_SIGNATURE_ALREADY (509).

Adding new signatures: Server can add new signatures to the misuse detector engine of Snort. It assigns a new SID to the signature and sends the message to the agent. The agent responds by indicating whether the command was successful or

not. The signature to be added is also present in the message. Many signatures can be sent in a message. The following message exchange sequence takes place.

Message Type (code)	**Data (bytes)**
ADD_NEW_SIGNATURE (33)	SID (4), Signature (String), SID (4), Signature (String),...
ADD_NEW_SIGNATURE_OK (510)	SID (4), SID (4), SID (4),...

Table A.5: Messages for adding new signature

Where the signature is a NULL terminated string.

Enabling and disabling sigfiles: A sigfile houses a particular class of signatures. Signatures are classified on type of attacks or vulnerabilities in services. For eg. there are many different signatures for detecting Denial of service attacks. If there is a need that a particular agent should detect particular class of attacks rather than all attacks then this message specifies the list of sigfiles that should be disabled/enabled on that agent. The sigfiles are represented as sequence of NULL terminated string.

Message Type (code)	**Data (bytes)**
ENABLE_SIGFILE (37)	List of sigfiles each separated by NULL character...
ENABLE_SIGFILE_REPLY (524)	SIGFILE_1 (String), reply_code (2), SIGFILE_2 ...

Table A.6: Messages for enabling signature files

Where reply-code can be one of the following: ENABLE_SIGFILE_OK (525), ENABLE_SIGFILE_FAILED (526), ENABLE_SIGFILE_ALREADY (527).

Message Type (code)	**Data (bytes)**
DISABLE_SIGFILE (38)	List of sigfiles each separated by NULL character...
DISABLE_SIGFILE_REPLY (519)	SIGFILE_1 (String), reply_code (2), SIGFILE_2 ...

Table A.7: Messages for disabling signature files

Where reply_code can be one of the following: DISABLE_SIGFILE_OK (520), DISABLE_SIGFILE_FAILED (521), DISABLE_SIGFILE_ALREADY (522).

Starting and stopping the misuse detector: Server can ask the agent to start or stop the misuse detector. The start message also contains the options with which the misuse detector program should be started. The reply to start contains any error message generated if start fails.

Message Type (code)	Data (bytes)
START_DETECTOR (34)	Options
START_DETECTOR_OK (513) or START_DETECTOR_FAILED (514)	Message (String, if any) Error Message (String)

Table A.8: Messages for starting misuse detector

Message Type (code)	Data (bytes)
STOP_DETECTOR (35)	Empty (0)
STOP_DETECTOR_OK (516)	Empty (0)

Table A.9: Messages for stopping misuse detector

Where 'Options' field depends on the misuse detector being used. We can also include a message which indicates failure but generally stopping will not fail.
Heartbeat: The server periodically sends a probe to all the agents which are in authenticated state. The agents should send a reply. The format of the probe and reply are as follows. The periodicity of this probe is tunable.

Message Type (code)	Data (bytes)
AGENT_PROBE (103)	Empty (0)
AGENT_ALIVE (604)	Empty (0)

Table A.10: Messages for probing the agent

Messages from Agent to the server include the following:

Alerts from agent to console: Each alert is associated with packets that generated the alert. Packets are sent along with the alert. It may not be possible to send all packets in one UDP packet. So, two types of messages are required for sending alerts and packets. In the first type the full alert is included along with as many of its packets as possible. In the second type only the alert ID is included along with the packets. Since packets can be of different lengths there is a length field preceding each packet data. The alert itself contains the alert ID. If it is possible to send more than one alert, then many alerts can be sent in a single message. The message format is shown below.

Message containing alert and packets:

Message Type	**Data (bytes)**
ALERT_MSG (101)	Alert Length (2), Alert (variable), Packet length (2), Packet (variable)...
ALERT_MSG_OK (601)	Alert ID (4), Alert ID (4),...

Table A.11: Messages for sending alerts to the server

Message containing only packets:

Message Type (code)	**Data (bytes)**
PACKET_MSG (102)	Alert ID (4), Packet length (2), Packet (variable)....
PACKET_MSG_OK (602)	Empty(0)

Table A.12: Messages for sending only packets to the server

Packet is in standard binary format. Alert has the following format. The number in the parenthesis indicates the size in bytes.

```
Alert ID (4) | Signature ID(4) | Timestamp(4) | Priority(1) | Classification
| Alert message | Reference
```

Classification, Alert message and Reference are null terminated strings.

Misuse Detector Failure: The agent periodically monitors the health of the

misuse detector and report it to the server when requested by the server. While monitoring the misuse detector, if the agent finds out that the misuse detector is not running, then the agent first tries to start the misuse detector. If the agent fails to start the misuse detector it reports this information immedialtely to the server so that the concerned user can take appropriate action.

Message Type (code)	Data (bytes)
MISUSE_DETECTOR_FAILED (104)	Empty
MISUSE_DETECTOR_FAILED_REPLY (605)	Empty

Table A.13: Messages for the failure of the Misuse Detector

Heartbeat: If the agent finds that the server is down it periodically sends a probe to the server until the server is up again. The server has to reply to this probe. The periodicity of this probe is tunable.

Message Type (code)	Data (bytes)
SERVER_PROBE (23)	Empty (0)
SERVER_ALIVE (603)	Empty (0)

Table A.14: Messages for probing the server

Appendix B

Formats of Messages in Sachet server-console Protocol

In this appendix, we describe formats of the messages exchanged between the server and the console using the SACHET server-console (SSC) protocol. All the messages are initiated by the console, and the server only needs to responds them. The Messages can be of following type: command, request and response. The packet format of the SSC Protocol is shown below.

2 bytes	2 bytes	Variable
Packet Length	Message Type	Data Value

Figure B.1: Packet Structure of SACHET server-console protocol

B.1 Authentication Message

This message contains the password to be verified by the server in order to establish the correct identity of the user. This user will finally interact with the console and hence control the server. This is the first message before any interaction takes place between the console and the server. The message exchanges are as follows:

Message Type (code)	Data (bytes)
I_AUTHENTICATE (2030)	password (16)
I_AUTH_SUCESS or (2031) I_AUTH_FAILED (2032)	empty (0)

Table B.1: Messages for authenticating the user

The password is not sent in the plain-text format. First, the message digest (MD5) on the password is computed and then this hash value is sent to the server. Hence the password is of 16 bytes of length.

Change Password Message It is always advisable to chnage password periodically. The old password should be provided alongwith the new password in the message. The server accepts the new password only if the old password provided is correct. The message exchanges are as follows:

Message Type (code)	Data (bytes)
I_CHANGE_PASSWD (2037)	old password (16), new password (16).
I_CHANGE_PASSWD_OK (2531) or I_OLDPWD_INCORRECT (2532)	

Table B.2: Messages for changing the password

In this case also the hash value of the passwords are being sent, not the plain-text.

B.2 Command Messages

'Command messages' are the commands to the particular agent, on behalf of the user interacting with the console. These commands are sent to particular agent through the server. 'Command messages' can also be commands to the server to receive the information from the user and updates it on the database. The information can be like adding/deleting of agents. The following are the types of commands issued by the console.

Adding and Deleting Agent As the enterprise will keeps on growing, more number of agents will be deployed at the startegic locations of the organization. The information about new agents should be provided to the server so that it can authenticate with the new agents and starts communicating with them. Suppose if it has been decided to stop monitoring a particular host or network segment, then this infromation should also be communicated to the server. The messages involved are as follows:

Message Type (code)	Data (bytes)
I_ADD_AGENT (2010)	Agent id (2), Public key of Agent (String).
I_ADD_AGENT_OK (2522) or I_AGENTID_INUSE (2525) or I_FAILED (2523)	empty(0)

Table B.3: Messages for adding new agent

Message Type (code)	Data (bytes)
I_DELETE_AGENT (2011)	Agent id (2).
I_DELETE_AGENT_OK (2526) or I_AGENT_NOT_FOUND (2527) or I_FAILED (2523)	empty(0)

Table B.4: Messages for deleting the agent

Enabling or Disabling signatures This message directs the server to issue command to the particular agent for enabling or disabling of certain signatures. Each signature is known by its signature id. The message include the list of signature ids separated by space. The message exchanges are as follows:

Message Type (code)	Data (bytes)
I_ENABLE_SIGNATURES (2001)	Agent id (2), SID (4), SID (4), SID (4) ....
I_ENABLE_SIGNATURES_REPLY (2512)	SID (4), reply_code(2), SID (4), reply_code (2) ...

Table B.5: Messages for enabling signatures

Where reply-code can be one of the following: ENABLE_SIGNATURE_OK

(503), ENABLE_SIGNATURE_FAILED (504), ENABLE_SIGNATURE_ALREADY (505).

Message Type (code)	Data (bytes)
I_DISABLE_SIGNATURES (2002)	Agent id (2), SID (4), SID (4), SID (4) ....
I_DISABLE_SIGNATURES_REPLY (2513)	SID (4), reply_code (2), SID (4), reply_code (2)....

Table B.6: Messages for disabling signatures

where reply-code can be one of the following: DISABLE_SIGNATURE_OK (507), DISABLE_SIGNATURE_FAILED (508), DISABLE_SIGNATURE_ALREADY (509).

Enabling or Disabling signature files This message directs the server to issue command to the particular agent for enabling or disabling of certain signature files. The message may contain more than one signature files and each signature file is separated by NULL terminated string.

Message Type (code)	Data (bytes)
I_ENABLE_SIGFILES (2004)	Agent id (2), List of sigfiles each separated by NULL character..
I_ENABLE_SIGFILES_REPLY (2515)	SIGFILE_1 (String), reply_code (2), SIGFILE_2 (String)...

Table B.7: Messages for enabling signature files

Where reply-code can be one of the following: ENABLE_SIGFILE_OK (525), ENABLE_SIGFILE_FAILED (526), ENABLE_SIGFILE_ALREADY (527).

Message Type (code)	Data (bytes)
I_DISABLE_SIGFILES (2005)	Agent id (2), List of sigfiles each separated by NULL character...
I_DISABLE_SIGFILES_REPLY (2516)	SIGFILE_1 (String), reply_code (2), SIGFILE_2 (String)...

Table B.8: Messages for disabling signature files

Where reply_code can be one of the following: DISABLE_SIGFILE_OK (520), DISABLE_SIGFILE_FAILED (521), DISABLE_SIGFILE_ALREADY (522).

Starting or Stopping Misuse Detector This message instructs the server to command the agent to stop its misuse detector. The message exchnages are as follows:

Message Type (code)	**Data (bytes)**
I_START_MISUSE_DETECTOR (2008)	Agent id (2).
I_START_MISUSE_DETECTOR_REPLY (2519)	reply_code (2).

Table B.9: Messages for starting misuse detector

Message Type (code)	**Data (bytes)**
I_STOP_MISUSE_DETECTOR (2009)	Agent id (2).
I_STOP_MISUSE_DETECTOR_REPLY (2520)	reply_code (2).

Table B.10: Messages for stopping misuse detector

In both the cases, the reply_code can be any one of the following: I_AGENT_NOT_FOUND (2527), I_AGENT_NOT_ALIVE (2528), I_FAILED (2523).

Adding a new signature The console provides a simple template though which the concerned user can create a new attack signature. This new signature is updated to all the agents which are communicatig with the server.

Message Type (code)	**Data (bytes)**
I_ADD_SIGNATURE (2003)	Signature (string)
I_ADD_SIGNATURE_REPLY (2514)	empty (0).

Table B.11: Messages for adding a new signature to all agents

B.3 Request Messages

Request messages are the request to the server for providing current information about the entire sachet system to the user who is interacting with the console.

Basic information about agent This message is periodically sent to the server to know about basic information about each agent. The server replies to this message by processing the information present in its local data structures. The message exchanges are as follows:

Message Type (code)	**Data (bytes)**
I_BASIC_INFO (2035)	Agent id (2).
I_BASIC_INFO_REPLY (2529)	Agent id (2), status of agent (1), status of misuse detector (1), last authentication time of agent with server (4) Number of signature files (2), IP Address (24).

Table B.12: Message for requesting information about the agent

Bibliography

[1] D. Anderson, T. Frivold, and A. Valdes. Next-generation intrusion-detection expert system (nides). Technical Report SRI-CSL-95-07, CSL, SRI International, Computer Science Laboratory, May 1995.

[2] S. Staniforn Chen, S. Cheung, and R. Crawford. Grids-a graph based intrusion detection system for large networks. In *Proceedings of the 19th National Information Systems Security Conference*, 1996.

[3] M. Crosbie and E. Spafford. Active defense of a computer system using autonomous agents. Technical Report 95-008, COAST Group, Department of Computer Science, Purdue University, Feb 1995.

[4] D. Denning. An intrusion-detection model. *IEEE Transactions on Software Engineering SE-13*, 2:222–232, February 1987.

[5] Teresa F.Lunt, Ann Tamaru, Fred Gilham, and R. Jagannathan. A real-time intrusion-detection expert system(ides). Technical Report Project 6784, CSL, SRI International, Computer Science Laboratory, February 1992.

[6] R. Heady, G. Luger, A. Maccabe, and M. Servilla. The architecture of a network level intrusion detection system. Technical report, University of New Mexico, Department of Computer Science, August 1990.

[7] L. Heberlein, G. Dias, K. Levitt, B. Mukherjee, J. Wood, and D. Wolber. A network security monitor. In *Proceedings of the 1990 Symposium on Research in Security and Privacy*, pages 296–303. IEEE Computer Security, May 1990.

[8] Enterasys Networks. Enterasys dragon intrusion defense systems. `http://www.enterasys.com/products/ids`.

[9] P. Proctor. Audit reduction and misuse detection in hetrogeneous environments: Framework and appications. In *Proceedings of the 15th National Computer Security Conference*, pages 117–125, December 1994.

[10] Paul E. Proctor. The practical intrusion detection handbook, 2001.

[11] Network Flight Recorder. Nfr intrusion detection appliance. `http://www.nfr.com`, 1996.

[12] S. Smaha. Haystack: An intrusion detection system. In *IEEE Fourth Aerospace Computer Security Application Conference*. IEEE Computer Society Press, December 1988.

[13] S. Smaha and J. Winslow. Misuse detection tools. *Computer Security Journal 10*, pages 39–49, 1994.

[14] Steven R. Snapp, James Brentano, Gihan V. Dias, and Terrance L. Goan. Dids(distributed intrusion detection system)- motivation, architecture, and an early prototype. Technical report, Computer Security Laboratory, University of California, 1999.

[15] Symantec Enterprises Solutions. Symantec host ids. `http://enterprisesecurity.symantec.com`.

[16] Symantec Enterprises Solutions. Symantec intruder alert. `http://enterprisesecurity.symantec.com`.

[17] Snort The Open Source Network Intrusion Detection System. `http://www.snort.org`.

[18] Cisco Systems. Cisco intrusion detection system. `http://www.cisco.com/warp/public/cc/pd/sqsw/sqidsz`.

[19] Internet Security Systems. Blackice server protection 3.6. `http://blackice.iss.net/update_center/index.php`.

[20] Internet Security Systems. Realsecure network 10/100. `http://www.iss.net/products_services/enterprise_protection/rsnetwork/s%ensor.php`.

[21] Intrusion Systems. Securecom. `http://www.intrusions.com`.

[22] G. White and V. Pooch. Cooperating security managers: Distributed intrusion detection systems. *Computers and Security*, Vol. 15(No. 5):441–450, 1996.

[23] D. Zamboni, J. Bala, J. Omar Garcia-Fernandez, D. Isacoff, and E. Sppafford. An architecture for intrusion detection using autunomous agents. Technical Report 98/05, COAST Laboratory, Purdue University, June 1998.