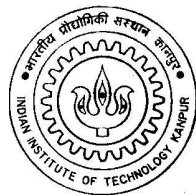


Attack Scenario Construction and Automated Report Generation in *Sachet* Intrusion Detection System

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

by
Puneet Kaur



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

June, 2005

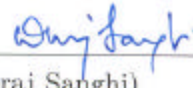
Certificate

This is to certify that the work contained in the thesis entitled "*Attack Scenario Construction and Automated Report Generation in Sachet Intrusion Detection System*", by *Puneet Kaur*, has been carried out under our supervision and that this work has not been submitted elsewhere for a degree.

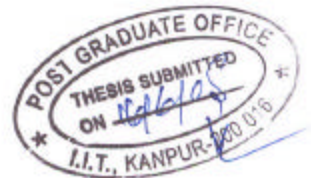
June, 2005



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



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



Abstract

Increased connectivity and the use of the Internet have exposed the organizations to subversion, thereby necessitating the use of intrusion detection systems to protect information systems and communication networks from malicious attacks and unauthorized access. An Intrusion Detection System (IDS) is a security system that monitors computer systems and network traffic, analyzes that traffic to identify possible security breaches, and raises alerts. An IDS triggers thousands of alerts per day making it difficult for human users to analyze them and take appropriate actions. It is therefore important to reduce the redundancy of alerts, intelligently integrate and correlate them, and to present high-level view of the detected security issues to the administrator.

In this thesis, we describe the design and implementation of attack scenario construction and automated report generation modules for Sachet - A distributed, real-time, network-based IDS developed at IIT Kanpur. The aim of attack scenario construction is to identify logical relations among low level alerts, correlate them, and to provide the system administrator with a condensed view of reported security issues known as attack scenarios. The alerts are correlated on the assumption that most intrusions are not isolated but related as different stages of a series of attacks, with the early stages preparing for the later ones. The module was successfully tested on a benchmark 2000 DARPA data set. Automated report generation takes the alerts produced by Sachet and generates reports which provide the system administrator with an overall picture of the status of the network under surveillance.

Acknowledgements

I take this opportunity to express my sincere thanks to my thesis supervisors, Dr. Deepak Gupta and Dr. Dheeraj Sanghi, for their support and guidance. They provided me with many valuable ideas throughout the thesis period. I also thank Prabhu Goel Research Center for partially supporting my thesis. I would also like to thank my project partner, Bharat Jain, for his co-operation and innovative suggestions.

Finally, I would like to thank my parents for their constant support and encouragement in all my endeavours.

Contents

1	Introduction	1
1.1	Problem Statement and Approach	3
1.2	Organization of Report	5
2	Related Work	6
2.1	Similarity Based Alert Correlation	6
2.2	Data Mining Based Alert Correlation	7
2.3	Prerequisite-Consequence Based Attack Scenario Construction	8
2.4	Other Methods	9
2.5	Alert Correlation in IDS Products	9
2.5.1	IBM Tivoli Intrusion Manager	9
2.5.2	Emerald	11
2.5.3	Symantec Incident Manager	11
3	Architecture of Sachet	13
3.1	The Sachet Protocol	14
3.2	The Sachet Server	16
3.3	The Sachet Agent	16
3.4	The Sachet Learning Agent	17
3.5	The Sachet Console	17
3.6	Incorporating Attack Scenario Construction module in the Sachet Architecture	18

4	Attack Scenario Construction	19
4.1	Approach Used and Justification	19
4.2	Framework for Attack Scenario Construction	20
4.3	Implementation	21
4.3.1	Knowledge Base	21
4.3.2	Alert Correlation	23
4.3.3	Transitive Edge Removal	26
4.3.4	Separating Correlation Graphs	27
4.3.5	Incremental Correlation	28
4.4	Changes made in Sachet	31
4.4.1	Correlation Agent	31
4.4.2	Sachet Protocol	32
4.4.3	Sachet Server	32
4.4.4	Sachet Console	33
5	Experimental Results	35
6	Automated Report Generation	42
6.1	Periodic Summary Report	43
6.2	Customized Summary Report	45
7	Conclusions and Future Work	47
	References	49
A	Predicates	52
B	New Messages Included in the Sachet Protocol	55

List of Tables

4.1	List of Alerts for Example Scenario	24
4.2	Signature Details for Example Scenario	25
5.1	Hyper-Alert Details for Correlation Graph 1	38
5.2	Hyper-Alert Details for Correlation Graph 2	40
5.3	Hyper-Alert Details for Correlation Graph 3	41
5.4	Signature Details for DARPA Data Set	41

List of Figures

3.1	Architecture of Sachet IDS	14
3.2	Packet Format for Sachet Server-Agent Protocol	15
3.3	Packet Format for Sachet Server-Console Protocol	16
4.1	An Example Attack Scenario Correlation Graph	26
4.2	An Example to demonstrate Incremental Correlation	29
4.3	Details of Sachet Architecture	31
4.4	Alert Correlation Screen	34
5.1	Correlation Graph 1	37
5.2	Correlation Graph 2	39
5.3	Correlation Graph 3	40
6.1	A screenshot of Summary Report	44
6.2	Template for configuring periodic Summary Report	45
6.3	Template for creating Customized Summary Report	46

Chapter 1

Introduction

Most organizations today depend on networked computer systems as an essential infrastructure for doing business. Billions of dollars are transferred around the world over computer networks. Increased connectivity and the use of the Internet have exposed the organizations to subversion. The loss to an organization due to lack of availability of critical computing resources or theft of intellectual property because of malicious actions can be significant. It has therefore become critical to protect an organization's information systems and communication networks from malicious attacks and unauthorized access.

An Intrusion Detection System (IDS) is used to inspect data for malicious or anomalous activities and detect attacks or unauthorized use of systems, networks, and related resources. It seeks to increase the security and hence the availability, integrity, and confidentiality of computer systems by providing information that would enable the system administrator to take necessary actions.

There are broadly two types of Intrusion Detection Systems. These are host-based IDS and network-based IDS. Host-based IDS uses system and audit logs as its data source, while network-based IDS uses network traffic as its data source. A host-based Intrusion Detection System consists of an agent on a host which identifies intrusions by analyzing system calls, application logs, file-system modifications (binaries, password files, etc.), and other host activities. In a network-based Intrusion Detection System, sensors are placed at strategic points within the network to

capture all network traffic flows and analyze the content of individual packets for malicious activities such as denial of service attacks, buffer overflow attacks, etc. Each approach has its respective strengths and weaknesses. Some of the attacks can be detected only by host-based or only by network-based IDS. For example, certain types of encryption present challenges to network-based IDS. Depending on where the encryption resides within the protocol stack, it may leave a network-based system blind to certain attacks which a host-based IDS can detect. Similarly, there are some attacks which can only be detected by examining packet headers for sign of malicious and suspicious activities. Host-based IDS do not see the packet header, so they cannot detect these type of attacks while network-based IDS can detect them.

The two main techniques used by Intrusion Detection Systems for detecting attacks are Misuse Detection and Anomaly Detection. In a Misuse Detection system, also known as signature-based system, well known attacks are represented by signatures. A signature is a pattern of activity which corresponds to the intrusion it represents. The IDS identifies intrusions by looking for these patterns in the data being analyzed. The accuracy of such a system depends on its signature database. Misuse Detection systems cannot detect novel attacks as well as slight variations of known attacks.

An anomaly-based IDS examines ongoing traffic, activity, transactions, or behavior for anomalies on networks or systems that may indicate attack. The underlying principle is the notion that attack behavior differs enough from normal user behavior that it can be detected by cataloging and identifying the differences involved. By creating baselines of normal behavior, anomaly-based IDS systems can observe when current behavior deviates statistically from the norm. This capability theoretically gives anomaly-based IDS ability to detect new attacks for which the signatures have not been created. The disadvantage of this approach is that there is no clear cut method for defining normal behavior. Therefore, such an IDS can report an intrusion, even when the activity is legitimate.

Intrusion Detection Systems trigger thousands of alarms per day [7]. Evaluating intrusion detection alarms and conceiving an appropriate response is a challenging task. From a security administrator's point of view, it is important to reduce the

redundancy of alarms, intelligently integrate and correlate security alerts, construct attack scenarios (defined as a sequence of related attack steps) and present high-level aggregated information. Correlating alerts to identify an attack scenario can also help forensic analysis, response and recovery and even prediction of forthcoming attacks. One of the current areas of research in Intrusion Detection Systems is to develop methodologies to give a succinct and high level view of attempted intrusions to the system administrator. Various approaches have been developed to correlate and aggregate alerts.

1.1 Problem Statement and Approach

Traditional Intrusion Detection Systems focus on low level attacks or anomalies and raise alerts. In situations where there are intensive attacks, the amount of alerts become unmanageable. As a result, it is difficult for human users or intrusion response systems to understand the alerts and take appropriate actions. Moreover, most of the alerts are not isolated. They are usually steps in multi-stage attacks which try to compromise the security of a system.

The approaches to correlate and aggregate alerts fall into three broad categories: Alert Fusion, Attack Scenario Construction, and Alert Clustering. In Alert Fusion, alerts that are generated by different IDS in response to a single attack are identified and merged. The aim of Attack Scenario Construction is to identify multi-step attacks that represent a sequence of actions performed by the same attacker. Alert Clustering groups alerts in a cluster based on similarities between alert attributes and constructs a generalized attribute from the cluster, where 'similarity' can be defined in various ways.

In this thesis, we describe the design and implementation of Attack Scenario Construction scheme and Automated Report Generation for Sachet - A distributed, real-time, network-based intrusion detection system with centralized control, developed at IIT Kanpur [14, 11]. Sachet IDS employs both misuse detection and anomaly detection. The architecture of Sachet IDS is explained in Chapter 3.

The aim of Attack Scenario Construction is to identify logical relations among low

level alerts, correlate them and to provide the system administrator with a condensed view of reported security issues known as Attack Scenarios. Most intrusions are not isolated, but related as different stages of a series of attacks, with the early stages preparing for the later ones. For example, attackers need to know what vulnerable services are running on a host before they can take advantage of these services. Thus, they typically scan for vulnerable services before they break into the system. As another example, in the Distributed Denial of Service (DDOS) attacks, the attacker has to install the DDOS daemon programs in vulnerable hosts before he instructs the daemons to launch an attack against another system. Therefore, in a series of attacks, one or more previous attacks usually prepare for the following attacks, and the success of the previous steps affects the success of the following ones. In other words, there are often logical steps or strategies behind a series of attacks.

Our approach for Attack Scenario Construction is as follows: We have developed a Knowledge Base which contains information about pre-requisites and consequences of attacks. The pre-requisite of an attack is the necessary condition for the attack to be successful. The consequence of an attack is the possible outcome of the attack. For example, the existence of a vulnerable FTP service is the pre-requisite of an FTP buffer overflow attack against this service. Consequence of this attack may be gaining local access as root from a remote machine. We correlate two alerts if the consequence of the attack corresponding to an earlier Sachet alert satisfies the pre-requisite of the attack corresponding to a later alert. This approach helps in discovering the causal relationship among related alerts and reveals the logical steps behind a sequence of intrusions. The implemented Attack Scenario Construction scheme has been tested on a benchmark 2000 DARPA Intrusion Detection Scenario Specific Data Set. The related alerts were successfully correlated into scenarios.

We have also developed automated report generation that takes the alerts produced by Sachet IDS and generates reports which give a condensed view of the reported security issues. The reports provide the system administrator with an overall picture of the status of the network under surveillance. The reports are generated periodically as well as on demand. The level of detail to be given in a report is configurable.

1.2 Organization of Report

Chapter 2 presents a brief overview of some of the approaches that have been used for alert aggregation and correlation. Chapter 3 presents the architecture of Sachet and its components. Chapter 4 presents the design and implementation of Attack Scenario Construction module in Sachet. Chapter 5 presents the results of testing the Attack Scenario Construction module using the benchmark 2000 DARPA Intrusion Detection Scenario Specific Data Set. Chapter 6 presents the design and implementation of Automated Report Generation scheme in Sachet. Chapter 7 presents conclusions and future work.

Chapter 2

Related Work

Various approaches have been used for alert aggregation and correlation. They fall into three major categories: Similarity based Alert Correlation, Data Mining based Alert Correlation, and Prerequisite-Consequence based Attack Scenario Construction. The first three sections discuss related work in these three categories. In Section 2.4, an outline of other approaches for alert correlation is given. In Section 2.5, a brief description of a few IDS products, which provide alert correlation or aggregation features, is given.

2.1 Similarity Based Alert Correlation

In this approach, alert correlation is done based on similarity between alert features. Alert features include the source of the attack, the target, the class of attack, and the time at which the alert was generated. Similarity can be determined in various ways. Valdes et al [19] use a probabilistic approach in which a similarity function is defined for each alert feature. The overall similarity between two alerts is the weighted average of the feature similarities, with expectation of similarity as the normalizing weights. Expectation of similarity is the prior expectation that the feature should match if the two alerts are related and is situation specific. A minimum similarity may also be specified for features which would be given priority over overall similarity.

In the alert clustering method by Cuppens [7], an "expert system" approach is used to define similarity between two alerts. Similarity between alert features is defined using expert rules which can be classification based, or source and target similarity based. Classification based rules are used to specify common attack names for different attack signatures. Two alerts generated by same IDS or different IDS, are type classification similar if they have a common attack name. Source and target similarity based rules are used to specify in which cases two sources or two targets may be considered similar. Similar alerts are grouped together and a new alert that is representative of the information contained in the various alerts belonging to this group is created.

The approach by Klaus Julisch [12, 13] uses taxonomies on alert attributes to define similarity between alerts. A taxonomy is a generalized hierarchy on these attributes. The closer the attributes are related by way of their taxonomies, the similar are the alerts. Similar alerts are grouped into clusters satisfying a minimum size requirement.

2.2 Data Mining Based Alert Correlation

Dain et al [9] have used data mining to combine alerts into scenarios. The algorithm generates scenarios by estimating the probability that a new alert belongs to a given scenario. Alerts are then added to the most likely scenario. Standard data mining algorithms were applied to training data for probability estimation. For each existing scenario, the probability that a new alert belongs to this scenario is calculated and the new alert is added to the scenario that produced the highest probability score. If all the scores fall below a threshold, the new alert is not added to any existing scenario and it instead starts a new scenario. The assignment of an alert to a scenario is fixed which might cause errors in assembled scenarios.

2.3 Prerequisite-Consequence Based Attack Scenario Construction

The basis of this strategy is the belief that attacks usually do not happen in isolation but are related as different stages of the attack sequence. In this approach, causal relationships among intrusion alerts are uncovered leading to detection of multi-stage attack scenarios.

In the JIGSAW correlation method [18], a "requires-provides" model is used to capture the causal relationship among alerts. Capabilities and concepts are used to detect attack scenarios. Concepts define the abstract situations that form sub-tasks in an attack scenario and capabilities are the information required or the situation that must exist for a particular aspect of an attack to occur. With each concept, *required capabilities* and *provided capabilities* are associated. Concepts can be thought of as individual intrusion alerts. And if the *required capabilities* of a concept are satisfied by the *provided capabilities* of an earlier occurring concept, then the two concepts are part of an attack scenario.

Peng Ning et al [15] and Cuppens et al [8] use a slight variation of the JIGSAW correlation method [18]. They use the concept of attack pre-requisites and consequences to determine attack scenarios. Pre-requisites specify the logical conditions to be satisfied for the attack to succeed and Consequences specify the effects of the attack when this attack succeeds. Two alerts are correlated if the pre-requisites of the later alert are satisfied by the consequences of the earlier one. These two approaches allow partial satisfaction of pre-requisites allowing for undetected attacks and that of attackers gaining information through non-intrusive ways, while the JIGSAW correlation method [18] requires all required capabilities to be satisfied. The methods proposed by Peng Ning et al [15] and Cuppens et al [8] use different formalism. The latter approach allows aggregation and merging of alerts before doing alert correlation.

Dong Yu et al [20] also use prerequisite and consequence information for alert correlation, but it presents the compromised resources as well as the probability that a specified resource has been compromised by a specific intruder, instead of alerts

to the administrator.

2.4 Other Methods

In the approach by Debar et al [10], explicit rules specified in the configuration file are used for alert correlation. The concept of consequence chains is used. A consequence chain is a set of alerts linked in a given order, which must occur within a given time interval. In the configuration file, information about alert-class of the original alert and the alert-class of an alert that might be considered the consequence of the original alert, and the time to wait for the consequence alert, is specified. When an alert occurs, the consequence mechanism is triggered which correlates two alerts if the alert-class of the later occurring alert is specified as a possible consequence of the alert-class of the earlier alert in the configuration file. It requires the alerts to occur in a fixed order and within the time limit.

Qin and W. Lee [17] have developed a novel approach to discover new patterns of alert relationships without depending on prior knowledge of attack scenarios. The premise of their approach is that attack steps that do not have well-known patterns or obvious relationships may nonetheless have some statistical correlations in the alert data. This approach uses causal analysis based on a statistical test, Granger Causality Test, to identify correlation between two attacks. Two attacks are correlated if the two have a high probability of occurring together.

2.5 Alert Correlation in IDS Products

In this section, a brief outline of aggregation features provided by three IDS products is given.

2.5.1 IBM Tivoli Intrusion Manager

IBM Tivoli Intrusion Manager [10, 6] is the IBM Software solution for managing network intrusions, including both external and internal threats and attacks. It

provides centralized management console, advanced event correlation, and reporting/analysis. It consolidates events, then utilizes rules and event action plans to process the events. To minimize event traffic, with minimal loss of information, it provides Event Summarization by grouping together similar events. For example, some sensors present a single port scan as one event per port scanned. All these events represent the same activity and hence can be grouped together. The product also provides Alert Correlation. It supports two correlation techniques: Linked Events and Incident Based Correlation that is based on source, destination, or attack classification.

In the *Linked Events* correlation approach, a configuration file is used to specify when two events that are produced by a sensor might be linked together. The information specifies the temporal ordering of the events, the time limit within which the second event should occur once the first event has been observed, and the attributes of the events which should be matched. This approach allows detection of a chain of linked events. When the second linked event is received, the correlation engine might raise the severity of the second linked event. For example, when Web IDS reports a suspicious Common Gateway Interface (CGI) request, the event is associated with an appropriate severity level. However, if the suspicious CGI request is actually successful, then it is appropriate to significantly raise the severity of the second event.

In the *Incident Based* correlation approach, the correlation process groups sensor events based on a subset of the following attributes:

- Classification, or category, of the received events
- Source host of the suspicious activity
- Destination host of the suspicious activity (or the resource being accessed)

Incidents represent bursts of suspicious activity, as detected over a relatively short period of time. An additional level of aggregation is performed on incident events. This level of aggregation is implemented as a set of customizable correlation rules

that run inside Tivoli Enterprise Console correlation engine. Incident groups represent aggregations of incident events, and thus are representative of sustained suspicious activity.

2.5.2 Emerald

The EMERALD (Event Monitoring Enabling Responses to Anomalous Live Disturbances) [2] environment is a distributed scalable tool for tracking malicious activity through and across large networks. The EMERALD project is being done at Stanford Research Institute and is sponsored by the Defence Advanced Research Projects Agency (DARPA). A mission-impact based approach for alarm correlation proposed by Porras et al [16], has led to the development of a prototype system called the Mission Impact Intrusion Report Correlation System, M-Correlator, for EMERALD

In the M-Correlator, alerts produced by heterogeneous information security devices are correlated through a strategy of topology analysis, alert prioritization, and common attribute based alert aggregation. Alert prioritization is done on the basis of importance of system assets so that attention can be focussed on critical assets. A knowledge database is maintained which provides information necessary to understand which alerts can be merged and dependencies of alerts to different versions of operating systems and application softwares.

2.5.3 Symantec Incident Manager

Symantec Incident Manager [4] is built on Symantec Enterprise Security Architecture (SESA), a standards-based infrastructure that enables Symantec and third-party security solutions to work together to provide secure, manageable, and scalable enterprise security. It examines event feeds from disparate security devices across the enterprise, normalizes them and automatically identifies incidents in real time to help improve incident detection and reduce false positives.

It includes the Rules Engine (RE) which provides event correlation and incident creation. The RE analyzes the events that are logged into the SESA console by

different security products and correlates related events into security incidents by using an internal knowledge base that consists of rule files and tables. Incidents are any pattern of alerts or events that meet a specified set of criteria. The RE categorizes events using a heuristic function that judges events to be related if they have certain similarities such as time, type, location, source and destination. When a sensor detects a suspicious event, the RE groups the event to an incident containing related events.

Chapter 3

Architecture of Sachet

In this chapter, the architecture of Sachet Intrusion Detection System is briefly described. In the sub-sections, the components of the system, and the interactions between them are described. Detailed information can be found in references [14, 11]. In section 3.5, the changes made to the Sachet architecture and the Sachet protocol for incorporating Attack Scenario Construction Module are described.

The architecture of Sachet is as shown in Figure 3.1. Sachet comprises of the following components: multiple Sachet Agents, the Sachet Server, the Sachet Learning Agent, and the Sachet Console. The components communicate with each other using the Sachet protocol. The protocol provides reliability, mutual authentication, confidentiality and integrity of all messages.

Sachet Agents monitor a host or a network segment for attack events in the network traffic that is incoming to the host or that network segment. Agents can be deployed at different penetration points in an organization or enterprise network. They use misuse detection and anomaly detection for detecting attacks. Learning Agent is used to learn the normal user behavior (profile), which would be used by Agents for anomaly based attack detection. The central Server runs on a dedicated machine. It controls Agents and the Learning Agent, stores all alerts and configuration information in a database, and interacts with the Console. The Console (GUI) interacts with the Server to provide a centralized control facility and alert information to the network administrator.

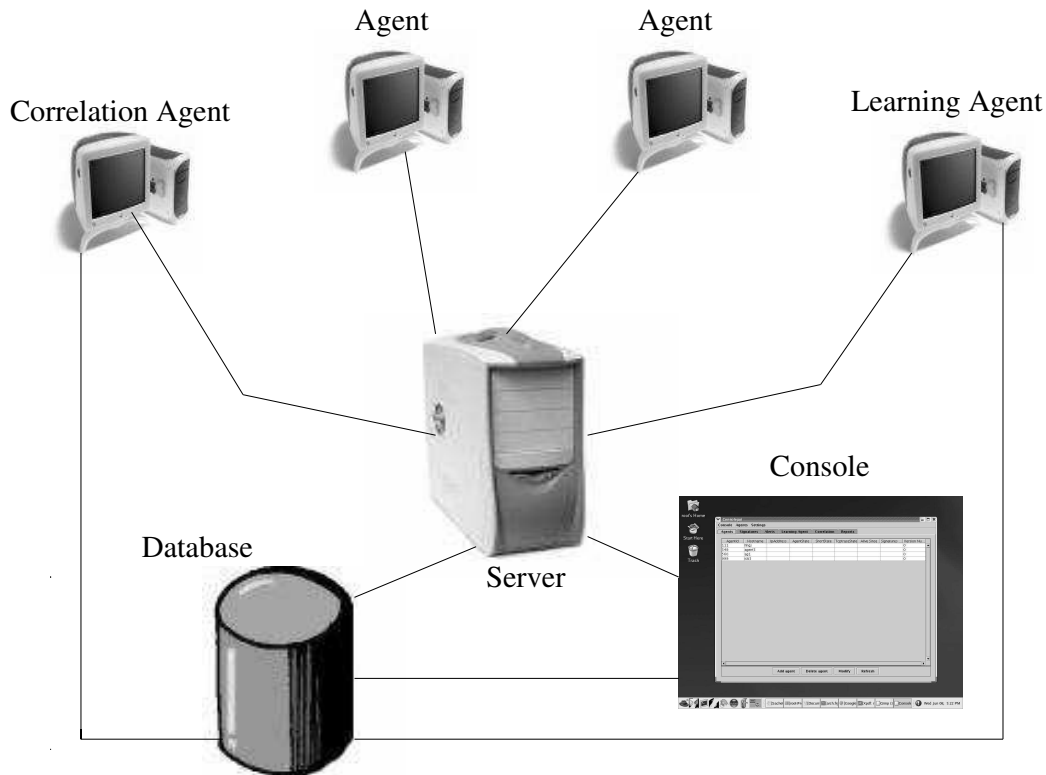


Figure 3.1: Architecture of Sachet IDS

3.1 The Sachet Protocol

Sachet protocol is used for communication between the system components. It addresses issues of security, reliability and scalability. It provides graceful degradation service to the system. The entire system is not disabled or brought into an inconsistent state if any component crashes or restarts.

Sachet Server-Agent protocol is used for communication between the Server and Agents (and the Learning Agent). Sachet uses a public-key cryptography algorithm, RSA, for authentication between the Server and Agents. The Server and Agents know each others' authentic public keys. The authentication mechanism is based on challenge-response method and allows Agents and the Server to authenticate with each other and negotiate on symmetric cryptographic key before transmitting any

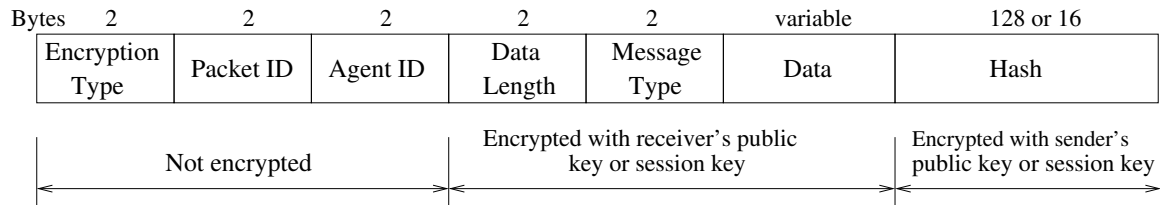


Figure 3.2: Packet Format for Sachet Server-Agent Protocol

application data.

The packet structure is as shown in Figure 3.2. *Encryption Type* field is used to indicate the method used for encrypting the packet. It can take three different values which indicate that packet is either encrypted with public key or with symmetric key or not at all encrypted. *PacketID* field contains a number that identifies each unique packet sent or received and can be used for detecting duplicates. Each Agent is recognized by a fixed and unique number called agent ID. *Agent ID* field contains the agent ID of the Agent which sent the packet. Agent ID value of the Server is zero so as to distinguish it from the Agents. *Data Length* field gives the length of data in bytes. *Message Type* field describes the type of message such as, an alert message, probe message, command message etc. *Data* field is interpreted based on the value of Message type. Data is encrypted with public key during authentication phase, and afterwards with the session key. *Hash* field contains the encrypted hash (MD5) for the entire packet. It provides packet integrity and ensures that the packet has not been modified or damaged while on its way. The hash is encrypted with private key during authentication phase and with session key afterwards. Here, session key refers to the shared secret key that is exchanged during the authentication phase. The communication between the Server and Agents is done using UDP.

Sachet Server-Console protocol is mainly designed for local communication between the Server and the Console. The Server and Console should be installed on the same host. The Console must authenticate to the Server before issuing any instructions or requests. This protocol is implemented over TCP. The Sachet Server-Console protocol packet format is as shown in Figure 3.3. *Packet Length* is the size of the complete packet in bytes. *Message type* indicates the type of packet. The

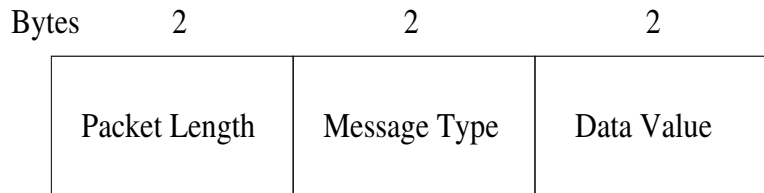


Figure 3.3: Packet Format for Sachet Server-Console Protocol

packet can be a command message, a request-message, or a response-message. The *Value* field contains data that is specific to the message type.

3.2 The Sachet 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 Sachet that allows consolidation of alerts from multiple Agents and stores these alerts in the database. It usually runs in the 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. The Console is used for controlling the Server. The Server communicates with the Console, which is a separate 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 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.

3.3 The Sachet Agent

The Agent is an application which can run in background and does not interact with the user. It comprises of three sub-components: Misuse Detector, Anomaly 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 and searches for pre-defined patterns or signatures in network traffic and generates alerts. Then it passes on the alert and the corresponding packet (that triggered the alert), to the Control Module. Open source software called Snort [3] is used for misuse detection. The Anomaly Detector also runs as a child process of the Control Module. It checks for anomalous behavior in network traffic. It uses the normal profile provided by the Learning Agent to detect deviations from the normal behavior. Control Module receives alerts from the Anomaly Detector as well as from the Misuse Detector and sends these alerts to the Server. Misuse Detector and Anomaly Detector can be stopped/started/restarted as desired by the Control Module. Control Module also periodically monitors them and reports their status to the Server.

3.4 The Sachet Learning Agent

The Learning Agent constructs normal profile by using certain metrics or features extracted from network traffic by Agents. Reservoir Sampling, which is a stream handling technique, is used to deal with the continuous stream of feature vectors and Support Vector Clustering, which is an unsupervised learning technique, is used for learning the profile. The profile generated by the Learning Agent is used by Anomaly Detector in Agents for detecting deviations in observed patterns of network activity.

3.5 The Sachet Console

The Sachet Console is a Java based GUI application using which the administrator interacts with the system. The administrator uses the Console to configure, monitor and control the system from a central location. For example, the administrator can add, modify or delete an Agent. It also interacts with the database for extracting information requested by the administrator. The strength of Misuse Detector in Agents lies in its signature database. The Console provides a mechanism by which the signature database can be remotely updated for every Agent. It also provides alert filtering based on source and destination IP, port, timestamp, severity level

etc.

3.6 Incorporating Attack Scenario Construction module in the Sachet Architecture

In this section, we describe the changes made to the architecture and components of Sachet to incorporate Attack Scenario Construction. Attack Scenario Construction is done by a special Agent known as Correlation Agent. The major change to the architecture was the addition of the Correlation Agent to Sachet. The Server starts the correlation process every 'x' hours by requesting the Correlation Agent to correlate alerts received in the past 'y' hours, where both 'x' and 'y' are configurable. The Server may also start the correlation process on request from the Console.

The Sachet Correlation Agent uses Sachet Server-Agent protocol for communicating with the Server. The main task of the Correlation Agent is to correlate alerts from various Agents. It authenticates with the Server like other Agents, and waits for commands from the Server. Upon receiving the 'start correlation' command from the Server, it fetches the relevant alerts from database and applies the correlation algorithm on this data. When correlation is completed, it sends the correlated alerts as Attack Scenarios to the Server.

The Server maintains the current state (running/idle) of the Correlation Agent. If the correlation process is running when a request for correlation comes from Console, the user is asked to try again at a later time. If the request for periodic correlation comes when the Correlation Agent is not idle, then the request is queued and is executed as soon as the Correlation Agent is done with the current job.

Console was modified so that it gives information about the Correlation Agent in a separate screen. The periodic correlation results are shown by default. The user can explicitly start the correlation process from Console where he needs to specify the time interval or number of hours of alerts to correlate. Many new messages were added to the Sachet protocol to incorporate the Correlation Agent.

Chapter 4

Attack Scenario Construction

In this section, we describe the construction of attack scenarios from detected alerts. In section 4.1, we describe the approach we have used. In Section 4.2, framework for attack scenario construction is given. The implementation details are given in Section 4.3. In Section 4.4, the changes made in Sachet to incorporate Correlation Agent are presented.

4.1 Approach Used and Justification

An attack scenario is the sequence of events the intruder performs to compromise the security of a system. Similarity Based Alert Correlation approach is able to group similar alerts together but is unable to determine the causal relationships among alerts and the choice of similarity criterion requires expert system knowledge or is domain specific. Data Mining based Techniques are able to aggregate the attacks into scenarios but the detection is constrained to known scenarios and requires training data. Moreover, in both these approaches, though one is able to group related alerts together, the exact nature of relationship between alerts is not revealed, that is, one is not able to determine the sequence of actions which the intruder undertook to compromise the system security.

The disadvantage of using approach by Debar et al [7], in which concept of consequence chains is used to link alerts, is that the addition of a new misuse detection

signature requires scanning of the entire knowledge base for possible updation since the new signature might be a possible consequence of many other signatures. This makes the knowledge base updation a non-trivial task since new attacks are being created at a frequent pace thereby requiring new signatures to be written.

The discovery of statistically correlated patterns in the approach by Qin and W. Lee[14] requires the same scenarios to occur many times. In real-life situations, this criterion might not be satisfied. This approach can be used as a complementary approach to other scenario detection approaches. The new patterns discovered can be used for knowledge base evaluation and possible updation.

We use the approach "Pre-requisite-Consequence Based Attack Scenario Construction" for construction of attack scenarios. This approach allows to detect causal relationship among related alerts. The addition of a new signature requires information about its pre-requisites and consequences and does not require specification of the new signature's relation to existing signatures, making knowledge base updation a comparatively much easier task. This approach allows detection of new attack scenarios and does not require training data.

A Knowledge Base which contains information about pre-requisites and consequences of attacks is developed and is used to construct attack scenarios from detected attacks. Two attacks are part of an attack scenario if the conditions implied by the consequences of the earlier attack satisfy the pre-requisites of the later attack, that is, if the earlier attack lays the foundation necessary for the success of the later attack. This approach reveals the high level strategies or logical steps behind a sequence of intrusions.

4.2 Framework for Attack Scenario Construction

Predicates are the semantic objects/constructs which represent the information required or the situation that must exist for an attack to succeed. The pre-requisites and consequences of an attack are semantic objects that encapsulate the system state. Pre-requisites of an attack are a set of predicates that must be present for a particular instance of an attack to be entailed. Consequences of an attack are a

set of predicates which would be satisfied if the instance of that particular attack is successful.

By specifying the pre-requisites of an attack, we are able to express all possible conditions which must hold for an attack to be successful without explicitly stating how these conditions would be met. By specifying the consequences of an attack, we are able to express the conditions which would hold in case the attack is successful without explicitly stating as to how this information would be used.

Multi-Stage attacks involve carrying out various intermediate attacks which would lay the foundation for later attacks to be successful. Using this as the basis for construction of attack scenarios, we correlate two alerts if the consequences of earlier attack satisfy the pre-requisites of the later one. For example, to carry out a buffer overflow attack against a vulnerable service, the knowledge that the particular service exists must be gained. The root access to the victim host that might be gained if the buffer overflow attack is successful, might then be used to carry out another attack against the victim host which, for instance, might involve installation of a trojan program.

The strength of this approach lies in the fact that we do not require knowledge of known attack scenarios in order to correlate alerts. It provides the capability to add new attacks to the system, without explicitly stating with whom they might be related, but just how their concepts would be used locally. We allow partial satisfaction of pre-requisites [8, 15] recognizing the possibility of undetected attacks and of attackers gaining information through non-intrusive ways.

4.3 Implementation

4.3.1 Knowledge Base

Sachet uses open source software Snort for misuse detection. Snort is a lightweight network intrusion detection system, capable of performing real-time traffic analysis and packet logging on IP networks. It uses a signature database for intrusion detection. Each signature corresponds to a possible attack on the system and has a unique Signature-Id.

The aim of attack scenario construction is to correlate the Sachet Misuse Detection Alerts and identify the high level strategy behind multi-stage attacks. Each Sachet Alert contains signature id of the Snort signature which caused the alert to be raised. In order to correlate Sachet IDS alerts, we have studied the Snort signature database which consists of about 2500 signatures and created a knowledge base. The knowledge base consists of a set of predicates and pre-requisites-consequences information for each Snort signature id.

Each predicate in the pre-requisites-consequences set for a signature has one or more arguments associated with them. The arguments are: source IP address, source port number, destination IP address, destination port number. In many cases, the presence of some information/situation necessarily implies the presence of another. For example, if an attacker has root access to a victim machine, the fact that he has access to the victim machine is trivially satisfied. This implication information is also included in the knowledge base as Implications.

For each snort signature, we identified the pre-requisites and consequences. The list of predicates used in the knowledge base is given in Appendix A. As an example, consider the case of Snort signature "FTP EXPLOIT wu-ftpd 2.6.0 site exec format string overflow Solaris 2.8". The Washington University FTP daemon (wu-ftpd) for Solaris is vulnerable to this attack. The daemon allows user input to be sent directly to printf function allowing an attacker to overwrite important data, such as a return address, on the stack. When this is accomplished, the function can jump into shell code pointed to by the overwritten instruction pointer and execute arbitrary commands as root. This can be done both with anonymous and real user logins.

For this attack, we identified the following pre-requisites: The victim machine running vulnerable FTP service on Solaris, the attacker having access to a user account for the FTP service on the victim machine. The possible consequence of this attack is attacker gaining root access to the victim machine. The predicates and their arguments to encode the above attack information are as follows:

Pre-requisites:

- ExistService (Destination IP address x, Destination Port y): It represents the

information that a service is running on port y on machine x .

- OSSolaris (Destination IP address x): It represents the information that machine x is running on Solaris.
- VulnerableFTPServer (Destination IP address x): It represents the information that FTP server running on machine x is vulnerable
- GainAccess (destination IP address x): It represents the information that access has been gained on machine x .

Consequence:

- GainRootAccess (Destination IP Address): It represents the information that root access has been gained on machine x .

4.3.2 Alert Correlation

■ *Preprocessing*

A Sachet agent is uniquely identified by Agent-Id. A Sachet alert contains Agent-Id, Alert-Id (which is unique for each Agent), Snort Signature Id, source IP address and port number, and destination IP address and port number. We have associated another attribute *hyper-id* with each alert. In Sachet, alerts from all the Agents are sent to the Server. The Server assigns a hyper-id to each alert which uniquely identifies each alert and is monotonically increasing with time, that is, a smaller value of hyper-id indicates an alert occurring earlier in time. The alerts are stored in a database.

The set of alerts which have to be correlated are instantiated as a set of *hyper-alerts*. A *hyper-alert* is a 4-tuple hyper-id, Sachet Alert, Pre-requisite, Consequence. The pre-requisite and consequence information for each alert is obtained from the knowledge base using Snort signature Id. Each hyper-alert encodes the knowledge about a type of attack. The arguments of the predicates associated with the pre-requisite set of a hyper-alert h , denoted by $P(h)$ and the consequence set, denoted by $C(h)$, are replaced with the corresponding attribute values of the Sachet alert in h .

Hyper-Id	Signature ID	Src IP	Dest IP	Src Port	Dest Port
1	469	A	B	40	21
2	619	A	B	50	60
3	342	A	B	50	21
4	469	A	C	40	200
5	619	A	C	1026	1230
6	316	A	C	890	200
7	235	C	B	40	100

Table 4.1: List of Alerts for Example Scenario

■ Correlation Process

Hyper-alert h_1 *prepares-for* hyper-alert h_2 , if there exists $p \in P(h_2)$ which is satisfied by some $c \in C(h_1)$ and $h_1.\text{hyper-id} < h_2.\text{hyper-id}$. Since the arguments of the predicates have been replaced with the corresponding alert attributes, the first condition can be checked by string matching. The first constraint allows partial satisfaction of pre-requisites. The second constraint restricts the alert correlation to meaningful ones because hyper-ids implicitly maintain the temporal ordering.

A hyper-alert h_1 is correlated with hyper-alert h_2 if h_1 *prepares-for* h_2 . We identify all the correlation relationships. Hyper-alert h_1 is known as preparing hyper-id and h_2 is known as prepared hyper-id. A correlated hyper-alert is a hyper-alert which occurs in some correlation relationship. The *prepares-for* relationship captures the causal relationship among alerts. Hyper-alert correlation graphs are used to represent the attack scenarios, which can be obtained on the basis of *textitprepares-for* relationships.

A hyper-alert correlation graph is a connected, directed acyclic graph, where the set N of nodes is the set of hyper-alerts and for each $n_1, n_2 \in N$, there is an edge from n_1 to n_2 if and only if n_1 *prepares-for* n_2 .

■ An Example

An example of a contrived attack scenario is presented below. In Table 4.1, a set of 7 alerts and their hyper-ids is given. In Table 4.2, information about pre-requisites and consequences of the corresponding Snort signatures is given.

Sig-ID	Pre-requisites	Consequences
235	<i>SystemCompromised(SrcIP, DestIP)</i>	<i>ReadyToDoS(SrcIP, DestIP)</i>
316	<i>GainAccess(DestIP), ExistService(DestIP, DestPort), OSLinux(DestIP)</i>	<i>GainRootAccess(DestIP)</i>
342	<i>ExistService(DestIP, DestPort), OSSolaris(DestIP), GainAccess(DestIP), VulnerableFTPServer(DestIP)</i>	<i>GainRootAccess(DestIP)</i>
469	–	<i>ExistService(DestIP, DestPort)</i>
619	–	<i>GainOSInfo(DestIP) VulnerabilityScanner(DestIP)</i>

Table 4.2: Signature Details for Example Scenario

A brief description of the Snort signature-ids present in the example scenario is given below:

- 235 : Trinoo is a distributed denial of service attack. This event is generated when communication between a Trinoo attacker and a Trinoo compromised machine is detected.
- 316 : This event is generated when an attempt is made to escalate privileges remotely using a vulnerability in mountd daemon on Linux.
- 342 : This event is generated when an attack attempt is made against an FTP server, possibly running a vulnerable FTP daemon on Solaris, to gain root access.
- 469 : This event is generated when an ICMP ping typically generated by nmap (port scanner software) is detected. This could indicate a full scan by nmap.
- 619 : This event indicates that an attempt has been made to scan a host.

In the example scenario, attacker "A" scans machines "B" and "C" to gather information about vulnerable services. "A" exploits vulnerable FTP service on Solaris machine "B" and vulnerable mountd daemon on Linux machine "C" to gain root access to both machines. Then "A" installs Trinoo attacker on "C" and Trinoo

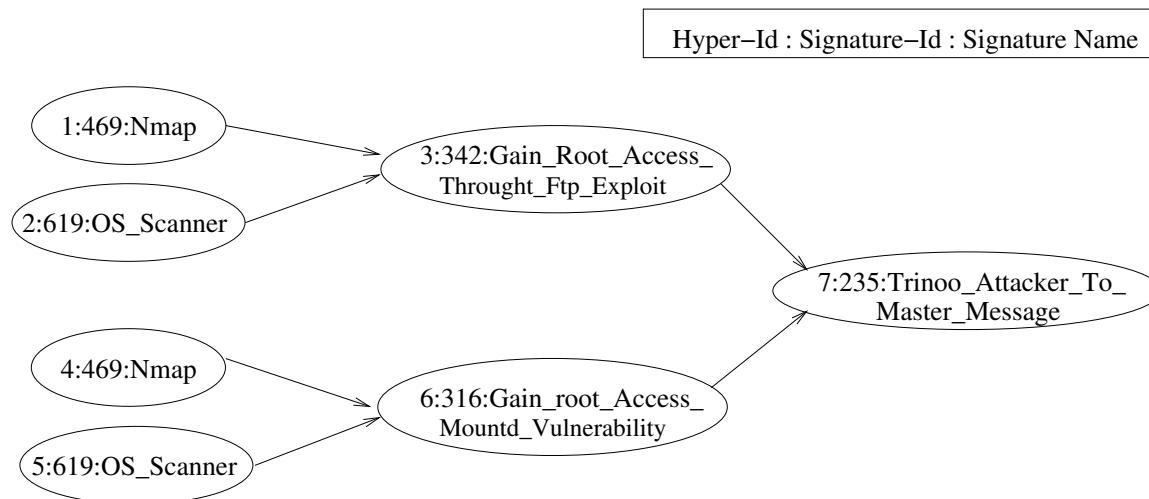


Figure 4.1: An Example Attack Scenario Correlation Graph

master on "B". Our correlation process correlates the alerts for the above scenario and the resultant correlation graph is given in Figure 4.1.

4.3.3 Transitive Edge Removal

The correlation graphs are represented by a correlation matrix, denoted by CM. The correlation matrix is a $N \times N$ matrix, where N is the number of correlated hyper-alerts. If there is an edge from some vertex x to some vertex y , then the element $CM_{x,y}$ is 1, otherwise it is 0.

Hyper-alert correlation graphs might contain some transitive edges. We say that edge (v,w) belonging to correlation graph is a transitive edge if and only if, there is a directed path $v=v_1, v_2, \dots, v_k = w$ in G , with $k \geq 3$.

Transitive edges are unnecessary and affect easy understanding of attack scenarios. Therefore, we do transitive edge reduction. We first find a Path matrix, denoted by PM, from the correlation matrix. A Path matrix is an $N \times N$ matrix, where N denotes the number of correlated hyper-alerts. If there is a directed path in the hyper-alert correlation graphs from some vertex x to some vertex y , then $PM_{x,y} = 1$, otherwise it is 0. Path matrix is the transitive closure of correlation matrix, and

is based on the assumption that if an alert A *prepares-for* an alert B and the alert B *prepares-for* an alert C, then alert A transitively *prepares-for* alert C. Warshall's algorithm is used to find the Path Matrix, with some optimizations done taking into account the fact that there cannot exist a path from a hyper-alert with a larger value of hyper-id to a hyper-alert with a lower value of hyper-id, that is, if $CM_{i,j} = 1$ then i is definitely less than j . The basic idea behind the algorithm is that if there is an edge between nodes i and j , then there would be a path from i to all the nodes to which j is connected. The algorithm is as follows:

1. Initialize Path Matrix, PM with correlation matrix, CM
2. For $i = (N-1)$ to 1, decrement by 1
 - For $j = (i+1)$ to N , increment by 1
 - if $PM_{i,j} = 1$
 - For $k=j+1$ to N , increment by 1

$$PM_{i,k} = PM_{i,k} \vee PM_{j,k}$$

Transitive edges are determined and removed from the correlation matrix, using the algorithm given below. The idea behind the algorithm is that if there is a path from hyper-alerts A to B and from A to C, where $A < B < C$, then correlation between A and C is removed if B is correlated to C.

1. For $i=1$ to N , increment by 1
 2. For $j=N$ to $i-1$, decrement by 1
 - if $PM_{i,j} = 1$
 - For $k=i+1$ to $j-1$, increment by 1
 - if $PM_{i,k} \wedge CM_{k,j}$

$$CM_{i,j} = 0$$

4.3.4 Separating Correlation Graphs

The current Correlation Matrix contains information about all *prepare-for* relationships. We assign a unique graph-id to each connected component. Each connected

component represents an attack scenario as a directed acyclic graph. We use graph-id as the indicator of different graphs. We associate a graph-id with each hyper-alert. All the hyper-alerts with the same graph-id are in the same correlation graph. The algorithm to assign graph-ids to hyper-alerts is given below. The graph-ids of all hyper-alerts are initialized to 0 and a variable ‘current-graph-id’, which at any point represents the number of connected components found till now, is maintained. A hyper-alert ‘h’ (if not already assigned a graph-id) is assigned ‘current-graph-id’ as its graph-id. All the hyper-alerts, say set s, which are reachable from ‘h’ are assigned the same graph-id. Then all the hyper-alerts which have a path to hyper-alerts in set ‘s’ are also assigned the same graph-id.

1. For $i = 1$ to N , increment by 1
 $graphid_i = 0$
2. current-graph-id = 1
3. For $i = 1$ to N , increment by 1
 if $graphid_i = 0$
 $graphid_i = \text{current-graph-id}$
 current-graph-id = current-graph-id + 1
 For $k = i$ to N , increment by 1
 if $PM_{i,k} = 1$
 $graphid_k = graphid_i$
 For $j = i$ to N , increment by 1
 if $PM_{j,k} = 1$
 $graphid_j = graphid_i$;

4.3.5 Incremental Correlation

The design of our correlation process allows us to use knowledge about already correlated alerts in future correlation requests. We maintain two tables in database: Correlation-Table and Correlation-Range-Table. Correlation Table has two columns: Preparing-Hyper-Id and Prepared-Hyper-Id. Each row in this table represents a *prepares-for* relationship between the hyper-alerts corresponding to the hyper-ids.

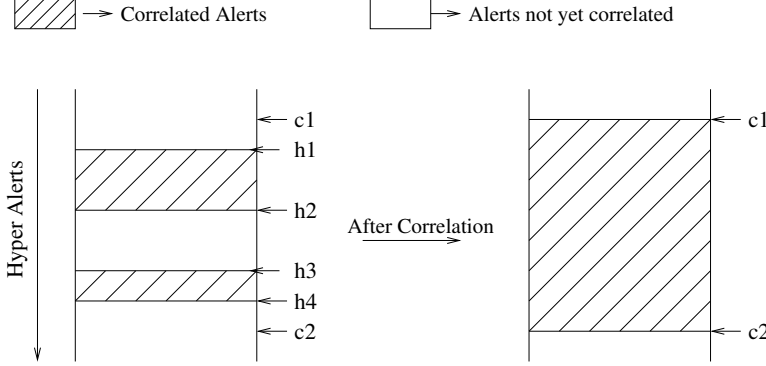


Figure 4.2: An Example to demonstrate Incremental Correlation

Correlation-Range-Table has two columns: Start-Hid and End-Hid. Each row of this table represents a range of hyper-alerts (corresponding to the hyper-ids) which have already been correlated and the *prepares-for* relationships stored in the Correlation-Table.

When a request for correlation comes, Correlation-Range-Table is used to determine the set of alerts to be correlated. For example, suppose sets of hyper-alerts h_1-h_2 and h_3-h_4 have already been correlated. Then this information would be present in Correlation-Range-Table and the causal relationships between them would be stored in Correlation-Table. When a request to correlate hyper-alerts from c_1-c_2 comes, then we only need to correlate alerts from c_1-h_1 with $c_1 - c_2$, from h_1-h_3 with h_2-h_4 and from h_1-c_2 with h_4-c_2 . We then update the Correlation-Range-Table by deleting the two entries for h_1-h_2 and h_3-h_4 and entering the row c_1-c_2 .

The algorithm for Incremental Correlation for hyper-alerts with hyper-ids in the range Min-Hid - Max-Hid is presented below:

- Correlate (Hyper-Id h_1 , Hyper-Id h_2 , Hyper-Id h_3 , Hyper-Id h_4) : Correlation function which correlates the set of hyper-alerts corresponding to hyper-id range h_1-h_2 with the set of hyper-alerts corresponding to the hyper-id range h_3-h_4 .
- CR_i : i^{th} row in Correlation-Range-Table. A value of -1 implies an empty row.

- n : Number of rows in Correlation-Range-Table

1. Low-Index = 'x' where $CR_x.Start-Hid < 'x' < CR_{x+1}.Start-Hid$
 = -1 if $Min-Hid < CR_1.Start-Hid$
 = -2 if $Min-Hid > CR_n.End-Hid$
2. High-Index = 'x' where $CR_x.Start-Hid < 'x' < CR_{x+1}.Start-Hid$
 = -1 if $Max-Hid < CR_1.Start-Hid$
 = -2 if $Max-Hid > CR_n.End-Hid$
3. If $((Low-Index = -1 \wedge High-Index = -1) \vee (Low-Index = -2 \wedge High-Index = -2))$
 Correlate(Min-Hid, Max-Hid, Min-Hid, Max-Hid)
4. Else If $(Low-Index = -1 \wedge High-Index = -2)$
 i = 1
 Correlate(Min-Hid, $CR_1.Start-Hid$, Min-Hid, Max-Hid)
 while($CR_{i+1}.Start-Hid \neq -1$), Do
 Correlate($CR_1.Start-Hid$, $CR_{i+1}.Start-Hid$, $CR_i.End-Hid$, $CR_{i+1}.End-Hid$)
 i = i + 1
 Correlate($CR_1.Start-Hid$, Max-Hid, $CR_{i-1}.End-Hid$, Max-Hid)
5. Else if $(Low-Index = -1 \wedge High-Index \geq 0)$
 i = 1
 if ($Max-Hid > CR_{High-Index}.End-Hid$)
 High-Index = High-Index + 1
 Correlate(Min-Hid, $CR_1.Start-Hid$, Min-Hid, $CR_{High-Index}.End-Hid$)
 while(i < High-Index), Do
 Correlate($CR_1.Start-Hid$, $CR_{i+1}.Start-Hid$, $CR_i.End-Hid$, $CR_{i+1}.End-Hid$)
 i = i + 1
6. Else
 if($Max-Hid > CR_{High-Index}.End-Hid$)
 High-Index = High-Index + 1
 i = Low-Index
 while(i < High-Index), Do
 Correlate($CR_{LowIndex}.Start-Hid$, $CR_{i+1}.Start-Hid$, $CR_i.End-Hid$, $CR_{i+1}.End-Hid$)
 i = i + 1

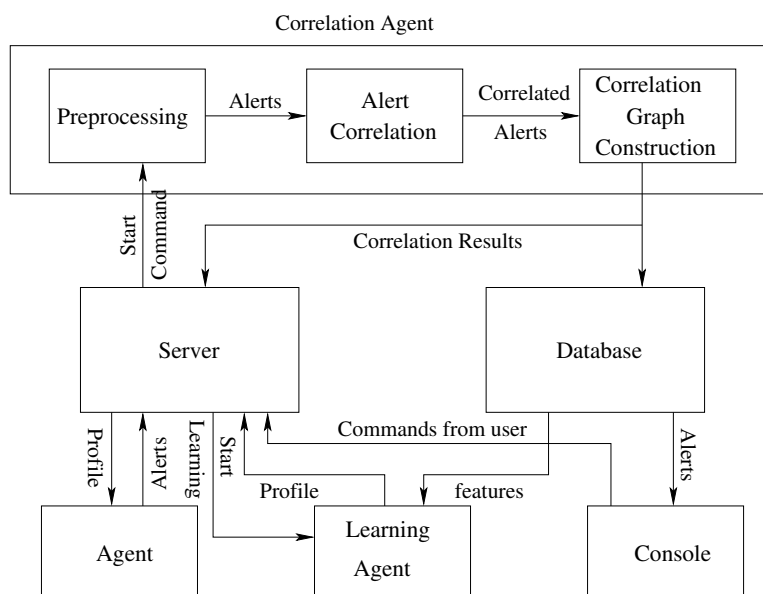


Figure 4.3: Details of Sachet Architecture

7. Update the Correlation-Range-Table

4.4 Changes made in Sachet

In this section, we describe the changes made to Sachet architecture and its components to include Attack Scenario Construction module. A special agent, known as Correlation Agent, has been added.

4.4.1 Correlation Agent

The Correlation Agent starts the Correlation process in a separate thread upon request from the Server. This thread connects to the database and fetches the relevant alerts using the ODBC interface. The data source name, username and password required for this purpose are provided to the Correlation Agent through a configuration file. After fetching the required data from the database, it starts the correlation process. It sends the correlation results to the server. If an error occurs

at any point during this process, e.g connection to the database may fail, a message indicating the type of failure is sent to the server.

4.4.2 Sachet Protocol

Many new messages have been added to the protocol to implement the Attack Scenario Construction module in Sachet. These messages are required to start/stop correlation process command from the Server to Correlation Agent and to transfer the correlation results from Correlation agent to the Server. The new messages included in the Sachet protocol are given in Appendix B.

In the case of the transfer of results, a single message cannot hold the entire result as there is an upper limit on the size of a UDP datagram. To overcome this drawback, the correlation result is sent in multiple messages with a 'more' flag in the data part of the message. A value of 1 for this flag indicates that at least one more message containing the remaining part of the result can be expected and a value of 0 indicates that the results are completely transferred.

As mentioned before, the Console periodically probes the Server for status of each agent. In case of Correlation agent, the Server sends "Periodic-Correlation" and "Result-Valid" flags in the data part of the message to the Console. A value of 1 for "Result-Valid" flag implies that new correlation results have been obtained from Correlation Agent and the GUI needs to be refreshed, a value of 2 implies that the correlation process has failed and a value of 0 implies that there are no new results. A value of 1 for "Periodic-Correlation" flag indicates that the new correlation results are for the periodic correlation and a value of 0 indicates that the results are for the explicit request made by the administrator through the Console.

4.4.3 Sachet Server

The server identifies the Correlation Agent among all Agents using the 'Type' field present in the information saved for each agent in the database. The Server starts the correlation process every 'x' hours by requesting the Correlation Agent to correlate alerts received in the past 'y' hours where both 'x' and 'y' are configurable and stored

in the Server configuration file. The Server also start the correlation process on request from the Console. The Server also maintains the current state (running/idle) of the Correlation Agent so that it makes new requests for correlation only when the Agent is idle.

4.4.4 Sachet Console

The Console has been enhanced with a separate screen for the Correlation Agent in which it shows correlation results and some basic information about the Correlation Agent including its current status. The same screen also has a button through which the user can explicitly start the correlation process. When user presses the 'Start' button for correlation, the Console prompts the user for correlation parameters, which can either be time-interval or last number of hours of alerts to be correlated. The administrator can view either the default correlation results obtained from periodic-correlation requested by Server, or administrator-requested correlation results.

The Console provides the capability to view the attack scenarios correlation graphs as a tree or as a group of correlated alerts. Figure 4.4 gives a snapshot of the Alert Correlation screen.

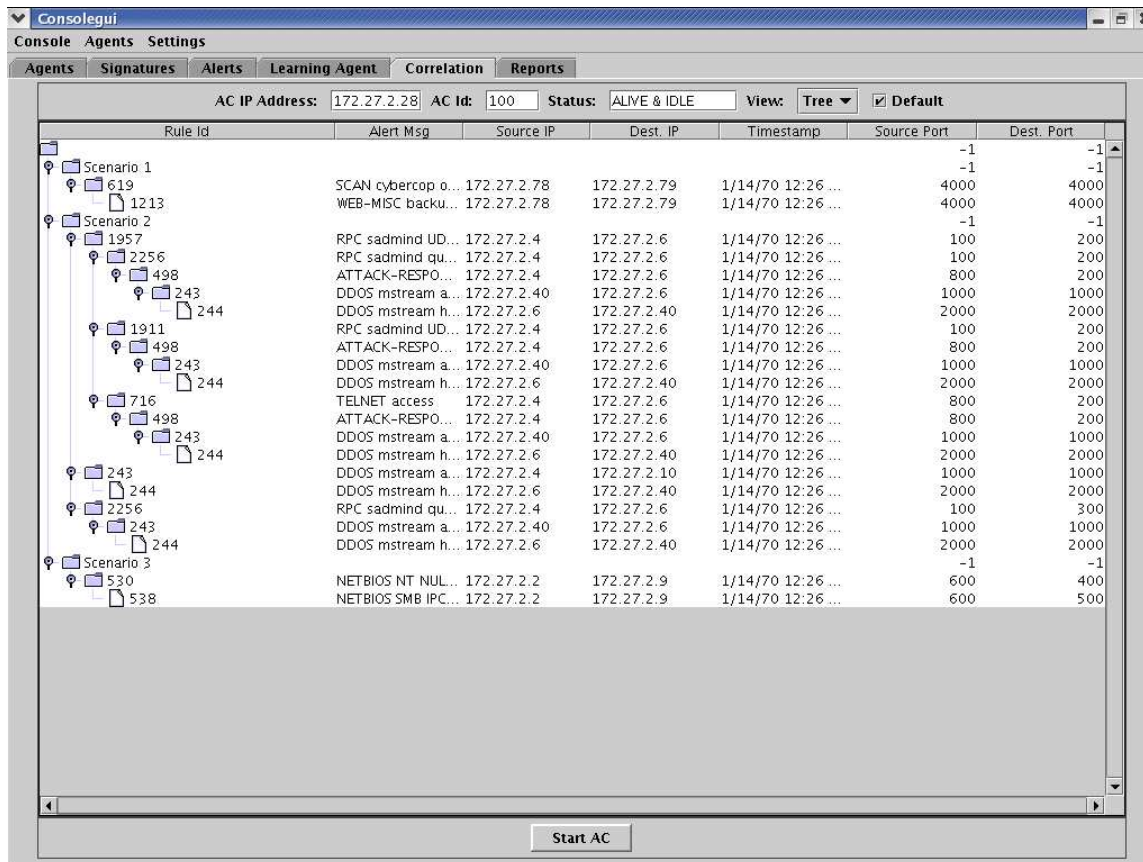


Figure 4.4: Alert Correlation Screen

Chapter 5

Experimental Results

In this chapter we present the results of evaluating the attack scenario detection in Satchet on 2000 DARPA Intrusion Detection Scenario Specific Data Set [1], a benchmark dataset generated for the purpose of evaluating alert correlation. We first give a brief description of the 2000 DARPA data. We then describe the experimental methodology, and finally present the results.

Lincoln Laboratory Scenario (DDoS) 1.0 data, LLDDOS1.0 [1] was generated for DARPA by MIT Lincoln Laboratory, using a test bed that simulated an existing military network. It includes a distributed denial of service (DDoS) scripted attack run by a novice attacker. The attacker tries to use the vulnerability of ‘Sadmin’ RPC service to break into vulnerable hosts and launch ‘mstream’ DDoS from the compromised hosts. Mstream is a DDoS tool. The mstream architecture is a standard 3-tier design used by most DDoS tools. It includes a *client*, a *master controller* and a *zombie*. The *client* is the machine that an attacker uses to launch the attack. The *client* co-ordinates the attack through a telnet connection to the *master*. A *master* controls all of the *zombies*. The *zombies* perform the denial of service attack on the specified victim machine. Each *master* can control any number of *zombies*, and each *zombie* can have any number of *masters* controlling it.

The attack scenarios in LLDDOS1.0 has been split into 5 attack phases which are briefly described below:

- Phase 1 : The attacker probes the network to determine live hosts.

- Phase 2 : The attacker probes the live hosts to determine which hosts are running the ‘sadmind’ remote administration tool.
- Phase 3 : The attacker tries to break into the vulnerable hosts using buffer overflow attack and create a new user account having root privileges. The attack script determines the success of the overflow attack by attempting a telnet login.
- Phase 4 : The attack script has built a list of those hosts on which it has successfully created a user account. For each host on this list, the script installs trojan mstream DDoS software using telnet and rsh.
- Phase 5 : The attacker launches a DDoS attack at an off site server from the compromised hosts. During this phase, the compromised hosts communicate with each other using telnet.

In our experiments, we used Tcpreplay [5] to replay the network traffic (tcpdump file) collected from the inside part of the DARPA evaluation network to a Sachet Agent. Snort has signatures for detecting mstream attacks but it was unable to detect attacks in the dump file because Snort was looking for attack patterns in UDP packets while the mstream attacks in DARPA attack scenario used TCP connections. We analyzed the dump file and modified Snort signatures so that it would detect mstream attacks. The attack construction module was then used to process the alerts to discover the correlation graphs. Figures 5.1, 5.2 and 5.3 show the correlation graphs we obtained from the DARPA data. The graphs show the hyper-ids of the hyper-alerts. The details of the alerts are given in Tables 5.1, 5.2 and 5.3 respectively. Table 5.4 shows the pre-requisite and consequence information for the signature-ids of the correlated hyper-alerts.

There are 37 alerts in the correlation graph of Figure 5.1. The hyper-alerts can be divided into four stages. In the first stage is the hyper-alert (234) received when attacker (202.77.162.213) probes for ‘sadmind’ vulnerable hosts. The alerts received when the attacker tries to do buffer-overflow attacks (Root Credential Attempt, Overflow Attempt), against the destination host 172.16.115.20, fall in second stage.

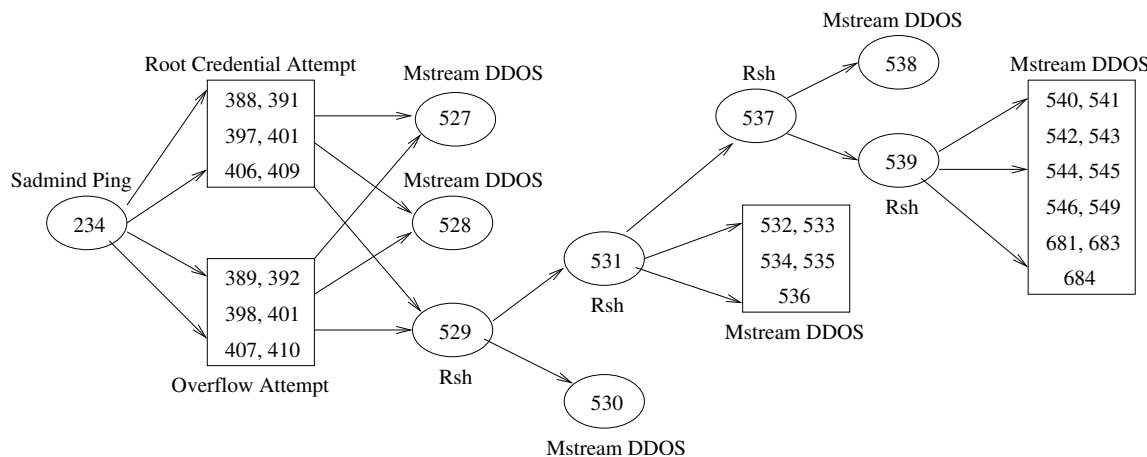


Figure 5.1: Correlation Graph 1

The hyper-alerts corresponding to the Snort alerts, which are generated when attacker tries to do rsh into the machine 172.16.115.20 to install trojan mstream, fall in third stage. The hyper-alerts in the fourth stage correspond to the alerts generated by Snort when it detects communication between attacker and the victim host.

In the graph (Figure 5.1), two hyper-alerts, 527 and 528, are correlated with the second stage and not with the third stage. This is because in ‘Phase 4’ of DARPA data, the attacker first creates ‘/tmp/.mstream/’ directory in the compromised host, copies the mstream DDoS software in it, and then starts the DDoS software using rsh. Snort does content-matching on the keyword ‘stream/’ for signature-id 244 and hence generates "Mstream DDoS" alerts corresponding to hyper-ids 527 and 528. The rsh hyper-alerts are generated after these alerts and hence will not be correlated because of temporal constraint.

The hyper-alerts for correlation graphs in Figures 5.2 and 5.3 can also be divided into four stages. The above stated reason for out-of-sequence mstream hyper-alerts also holds for them. The correlation graphs of Figures 5.1, 5.2 and 5.3 represent installation of mstream DDoS software on victim machines 172.16.115.20, 172.16.112.10, and 172.16.112.50 respectively. Snort does not detect the DDoS attack against the off-site server. If it had, we would have obtained a single correlation graph instead of three, because the hyper-alerts of the last stage in each correlation

Hyper Id	Sig-ID	Signature Name	Src IP	Src Port	Dst IP	Dst Port
234	1957	<i>SadminPing</i>	202.77.162.213	54792	172.16.115.20	32773
388	2256	<i>RootCr.Attempt</i>	202.77.162.213	60251	172.16.115.20	32773
389	1911	<i>OverflowAttempt</i>	202.77.162.213	60251	172.16.115.20	32773
391	2256	<i>RootCr.Attempt</i>	202.77.162.213	60255	172.16.115.20	32773
392	1911	<i>OverflowAttempt</i>	202.77.162.213	60255	172.16.115.20	32773
397	2256	<i>RootCr.Attempt</i>	202.77.162.213	60269	172.16.115.20	32773
398	1911	<i>OverflowAttempt</i>	202.77.162.213	60269	172.16.115.20	32773
400	2256	<i>RootCr.Attempt</i>	202.77.162.213	60276	172.16.115.20	32773
401	1911	<i>OverflowAttempt</i>	202.77.162.213	60276	172.16.115.20	32773
406	2256	<i>RootCr.Attempt</i>	202.77.162.213	60289	172.16.115.20	32773
407	1911	<i>OverflowAttempt</i>	202.77.162.213	60289	172.16.115.20	32773
409	2256	<i>RootCr.Attempt</i>	202.77.162.213	60300	172.16.115.20	32773
410	1911	<i>OverflowAttempt</i>	202.77.162.213	60300	172.16.115.20	32773
527	244	<i>MStreamDDOS</i>	202.77.162.213	47496	172.16.115.20	23
528	244	<i>MStreamDDOS</i>	202.77.162.213	23	172.16.115.20	47496
529	610	<i>Rsh</i>	172.16.115.20	1023	202.77.162.213	514
530	244	<i>MStreamDDOS</i>	202.77.162.213	1023	172.16.115.20	514
531	610	<i>Rsh</i>	172.16.115.20	1022	202.77.162.213	514
532	244	<i>MStreamDDOS</i>	202.77.162.213	1022	172.16.115.20	514
533	243	<i>MStreamDDOS</i>	172.16.115.20	514	202.77.162.213	1022
534	246	<i>MStreamDDOS</i>	172.16.115.20	514	202.77.162.213	1022
535	244	<i>MStreamDDOS</i>	202.77.162.213	47496	172.16.115.20	23
536	244	<i>MStreamDDOS</i>	202.77.162.213	23	172.16.115.20	47496
537	610	<i>Rsh</i>	172.16.115.20	1022	202.77.162.213	514
538	244	<i>MStreamDDOS</i>	202.77.162.213	1022	172.16.115.20	514
539	610	<i>Rsh</i>	172.16.115.20	1021	202.77.162.213	514
540	244	<i>MStreamDDOS</i>	202.77.162.213	1021	172.16.115.20	514
541	243	<i>MStreamDDOS</i>	172.16.115.20	514	202.77.162.213	1021
542	244	<i>MStreamDDOS</i>	202.77.162.213	514	172.16.115.20	1021
543	246	<i>MStreamDDOS</i>	172.16.115.20	514	202.77.162.213	1021
544	244	<i>MStreamDDOS</i>	202.77.162.213	514	172.16.115.20	1021
545	244	<i>MStreamDDOS</i>	202.77.162.213	514	172.16.115.20	1021
546	244	<i>MStreamDDOS</i>	202.77.162.213	1023	172.16.115.20	514
549	244	<i>MStreamDDOS</i>	202.77.162.213	1023	172.16.115.20	514
681	244	<i>MStreamDDOS</i>	202.77.162.213	23	172.16.115.20	49212
683	244	<i>MStreamDDOS</i>	202.77.162.213	49212	172.16.115.20	23
684	244	<i>MStreamDDOS</i>	202.77.162.213	23	172.16.115.20	49212

Table 5.1: Hyper-Alert Details for Correlation Graph 1

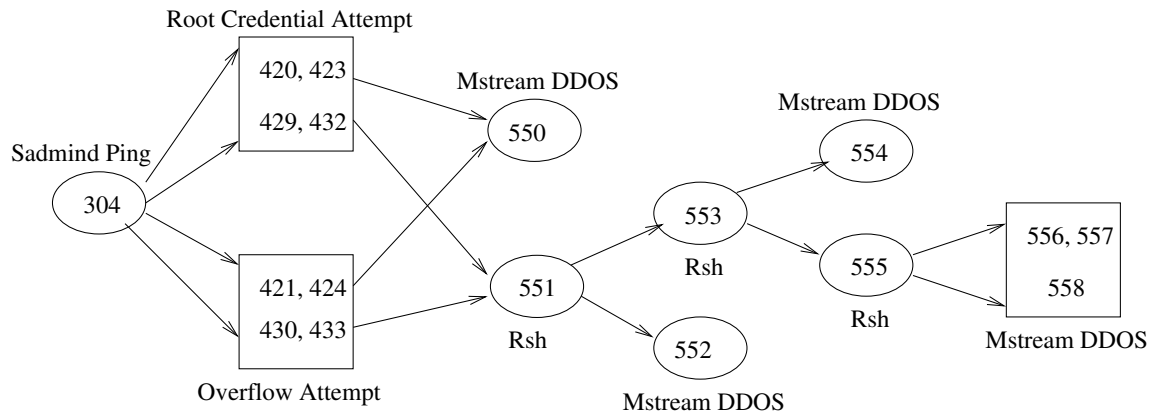


Figure 5.2: Correlation Graph 2

graph would have got correlated to the hyper-alert corresponding to DDoS attack.

Snort generated 959 alerts for the DARPA data. The attack scenario construction module generated 73 correlated hyper-alerts. Thus, the module is able to reduce the number of alerts to be evaluated by the administrator by a large amount and also gives a high-level picture of the attacker's strategy.

Hyper Id	Sig-ID	Signature Name	Src IP	Src Port	Dst IP	Dst Port
304	1957	<i>SadminPing</i>	202.77.162.213	56256	172.16.112.10	32774
420	2256	<i>RootCr.Attempt</i>	202.77.162.213	60519	172.16.112.10	32774
421	1911	<i>OverflowAttempt</i>	202.77.162.213	60519	172.16.112.10	32774
423	2256	<i>RootCr.Attempt</i>	202.77.162.213	60524	172.16.112.10	32774
424	1911	<i>OverflowAttempt</i>	202.77.162.213	60524	172.16.112.10	32774
429	2256	<i>RootCr.Attempt</i>	202.77.162.213	60542	172.16.112.10	32774
430	1911	<i>OverflowAttempt</i>	202.77.162.213	60542	172.16.112.10	32774
432	2256	<i>RootCr.Attempt</i>	202.77.162.213	60549	172.16.112.10	32774
433	1911	<i>OverflowAttempt</i>	202.77.162.213	60549	172.16.112.10	32774
550	244	<i>MStreamDDOS</i>	202.77.162.213	23	172.16.112.10	47502
551	610	<i>Rsh</i>	172.16.112.10	1023	202.77.162.213	514
552	244	<i>MStreamDDOS</i>	202.77.162.213	1023	172.16.112.10	514
553	610	<i>Rsh</i>	172.16.112.10	1022	202.77.162.213	514
554	244	<i>MStreamDDOS</i>	202.77.162.213	1022	172.16.112.10	514
555	610	<i>Rsh</i>	172.16.112.10	1022	202.77.162.213	514
556	244	<i>MStreamDDOS</i>	202.77.162.213	1022	172.16.112.10	514
557	243	<i>MStreamDDOS</i>	172.16.112.10	514	202.77.162.213	1022
558	246	<i>MStreamDDOS</i>	172.16.112.10	514	202.77.162.213	1022

Table 5.2: Hyper-Alert Details for Correlation Graph 2

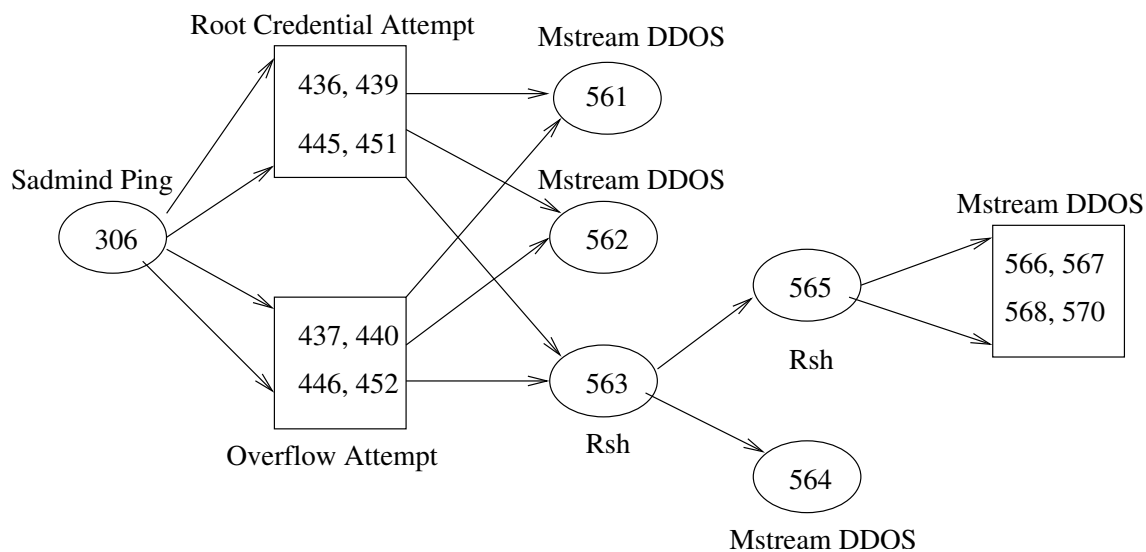


Figure 5.3: Correlation Graph 3

Hyper Id	Sig-ID	Signature Name	Src IP	Src Port	Dst IP	Dst Port
306	1957	<i>SadminPing</i>	202.77.162.213	56262	172.16.112.50	32773
436	2256	<i>RootCr.Attempt</i>	202.77.162.213	60569	172.16.112.50	32773
437	1911	<i>Over flowAttempt</i>	202.77.162.213	60569	172.16.112.50	32773
439	2256	<i>RootCr.Attempt</i>	202.77.162.213	60578	172.16.112.50	32773
440	1911	<i>Over flowAttempt</i>	202.77.162.213	60578	172.16.112.50	32773
445	2256	<i>RootCr.Attempt</i>	202.77.162.213	60605	172.16.112.50	32773
446	1911	<i>Over flowAttempt</i>	202.77.162.213	60605	172.16.112.50	32773
451	2256	<i>RootCr.Attempt</i>	202.77.162.213	60619	172.16.112.50	32773
452	1911	<i>Over flowAttempt</i>	202.77.162.213	60619	172.16.112.50	32773
560	244	<i>MStreamDDoS</i>	202.77.162.213	1023	172.16.112.10	514
561	244	<i>MStreamDDoS</i>	202.77.162.213	47512	172.16.112.50	23
562	244	<i>MStreamDDoS</i>	202.77.162.213	23	172.16.112.50	47512
563	610	<i>Rsh</i>	172.16.112.50	1023	202.77.162.213	514
564	244	<i>MStreamDDoS</i>	202.77.162.213	1023	172.16.112.50	514
565	610	<i>Rsh</i>	172.16.112.50	1023	202.77.162.213	514
566	244	<i>MStreamDDoS</i>	202.77.162.213	1023	172.16.112.50	514
567	243	<i>MStreamDDoS</i>	172.16.112.50	514	202.77.162.213	1023
568	246	<i>MStreamDDoS</i>	172.16.112.50	514	202.77.162.213	1023
570	244	<i>MStreamDDoS</i>	202.77.162.213	1023	172.16.112.50	514

Table 5.3: Hyper-Alert Details for Correlation Graph 3

Sig-ID	Signature Name	Pre-requisites	Consequences
243	<i>MStreamDDoS</i>	<i>SystemCompromised</i> (<i>SrcIP, DestIP</i>)	<i>ReadyToDoS</i> (<i>SrcIP, DestIP</i>)
244	<i>MStreamDDoS</i>	<i>SystemCompromised</i> (<i>SrcIP, DestIP</i>)	<i>ReadyToDoS</i> (<i>SrcIP, DestIP</i>)
246	<i>MStreamDDoS</i>	<i>SystemCompromised</i> (<i>SrcIP, DestIP</i>)	<i>ReadyToDoS</i> (<i>SrcIP, DestIP</i>)
610	<i>Rsh</i>	<i>OSLinux</i> (<i>SrcIP</i>), <i>ExistService</i> (<i>SrcIP, SrcPort</i>), <i>GainRootAccess</i> (<i>SrcIP</i>)	<i>SystemCompromised</i> (<i>SrcIP</i>)
1911	<i>Over flowAttempt</i>	<i>ExistService</i> (<i>SrcIP, DestIP</i>), <i>OSSolaris</i> (<i>DestIP</i>)	<i>GainRootAccess</i> (<i>DestIP</i>)
1957	<i>SadminPing</i>	<i>OSSolaris</i> (<i>DestIP</i>)	<i>ExistService</i> (<i>DestIP, DestPort</i>)
2256	<i>RootCr.Attempt</i>	<i>ExistService</i> (<i>SrcIP, DestIP</i>), <i>OSSolaris</i> (<i>DestIP</i>),	<i>GainRootAccess</i> (<i>DestIP</i>)

Table 5.4: Signature Details for DARPA Data Set

Chapter 6

Automated Report Generation

Sachet has been enhanced to provide summary reports to the administrator. The Console takes the alerts produced by Sachet IDS and generates reports which give an aggregated view of the reported security issues. The reports provide an overall picture of the status of the network under surveillance. The reports are generated periodically as well as on demand. The level of detail to be given in a report is configurable.

The summary reports include the following information (N is configurable):

- Top N Services: Reports the top N port numbers attacked.
- Top N Attack Classes: Snort signatures are grouped into attack-classes. This category reports the top N attack-classes of detected alerts
- Top N Signatures: Reports information about Top N Misuse Detector signatures for which the alerts were generated. The information includes signature-id, number of alerts, signature name, alert priority, and attack-class of the signature.
- Top N Victims: Reports the top N destination IP addresses (victims) that have been targeted for attack during a specified time period.
- Top N Attacker-Victim Pairs: Reports the top N source-destination pairs (that is, connections or sessions) that have generated the most alerts during a specified time period.

- **Top N Attackers:** Reports top N source IP addresses (attackers) that have generated the most alerts during a specified time period. It also gives the following per-attacker details:
 - Top N services attacked and the corresponding number of attacks.
 - Information about Top N Misuse Detector signature-ids for which the alerts were generated.
 - Top N attack-classes and the number of alerts.

The above information is provided for the complete system as well as for each agent. If the agent is monitoring more than one machine, then the information per monitored machine is also provided. The reports are stored as html files. A separate screen is provided in the Console to view the generated reports. A screenshot of a generated report is shown in Figure 6.1.

There are two types of reports: Periodic Summary Report and Customized Summary Report. They are described in detail in the following sections.

6.1 Periodic Summary Report

Periodic daily, weekly, and fortnightly summary reports for the complete system are generated. In the configuration file of the Console, timestamps for the last generated daily, weekly and fortnightly reports are stored. When the Console is started, this information is used to generate pending reports, if any. Configuration files for daily, weekly, and fortnightly reports are maintained. These files control the amount of detail which the corresponding report would contain. The administrator can modify these configuration files through GUI. As shown in Figure 6.2, the parameters that can be configured are:

- Whether to generate attacker-wise summary of alerts.
- Whether to generate agent-wise detailed Summary:
- Whether to generate per monitored machine summary

Consolegui

Console Agents Settings

Agents Signatures Alerts Learning Agent Correlation Reports

Summary Templates

Select a Template: Select a Report:

Top Attacker-Victim Pairs			Top Signatures				
Attacker IP Address	Victim IP Address	# Attacks	Sig Id	Sig Name	#Attacks	Attack Class	Priority
172.16.114.1	172.16.114.50	416	472	ICMP redirect host	425	bad-unknown	2
172.16.112.100	172.16.116.20	56	585	RPC portmap sadmind request UDP	199	rpc-portmap-decode	2
172.16.116.20	172.16.112.100	32	528	BAD-TRAFFIC loopback traffic	142	bad-unknown	2
202.77.162.213	172.16.112.100	24	716	TELNET access	139	not-suspicious	3
202.77.162.213	172.16.113.148	24	530	NETBIOS NT NULL session	44	attempted-recon	2
202.77.162.213	172.16.115.87	24	538	NETBIOS SMB IPC\$ share unicode access	44	protocol-command-decode	3
202.77.162.213	172.16.113.105	24	480	ICMP PING speedera	34	misc-activity	3
172.16.112.100	194.7.248.153	18	718	TELNET login incorrect	22	bad-unknown	2
172.16.115.20	172.16.112.105	18	1292	ATTACK-RESPONSES directory listing	20	bad-unknown	2

Per Attacker Summary

- Attacker IP: 172.16.114.1 #Attacks: 425

Top Services		Top Attack Classes			Top Signatures				
Service	#Attacks	Attack Class	#Attacks	Priority	Sig Id	Sig Name	#Attacks	Attack Class	Priority
0	425	bad-unknown	425	2	472	ICMP redirect host	425	bad-unknown	2

Create a Template Configure Template Generate Report Delete Template Delete Report

Figure 6.1: A screenshot of Summary Report

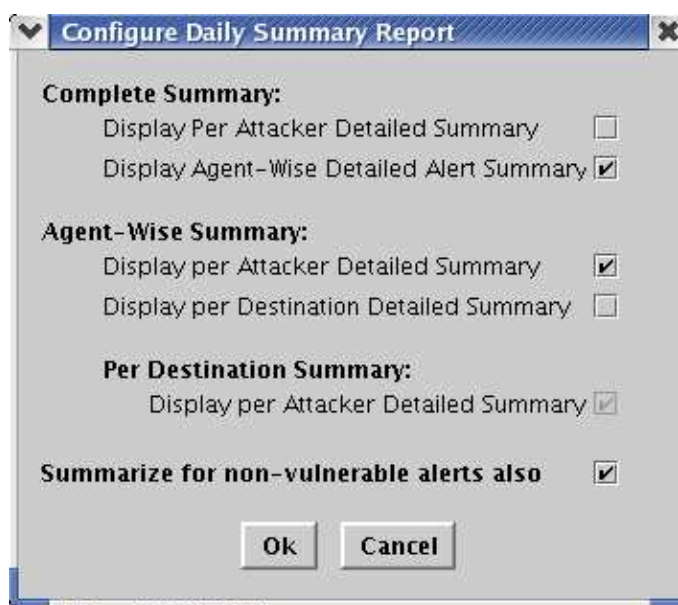


Figure 6.2: Template for configuring periodic Summary Report

- Whether or not to generate attacker-wise summary for each monitored machine.

6.2 Customized Summary Report

The administrator can generate customized summary reports. He can specify the Agent(/s) for which the summary is to be generated, the time interval for which the report should be generated, and the same configurable options as provided for the periodic summary report, in a configuration file. A template for creating a customized summary report is shown in Figure 6.3. Reports corresponding to the template files can be generated at any time. An option is also provided to modify / delete the configuration files. The reports are stored as html files. They can also be viewed through the Console.

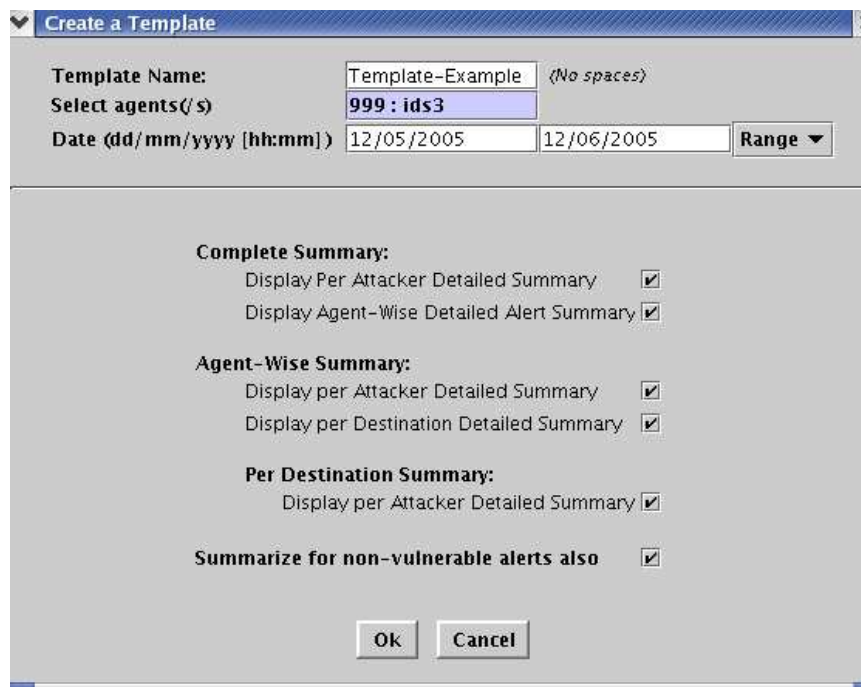


Figure 6.3: Template for creating Customized Summary Report

Chapter 7

Conclusions and Future Work

We have developed an attack scenario construction module for the Sachet Intrusion Detection System using a pre-requisite - consequence based approach. We studied various alert correlation techniques and implemented this one. We have tested this scheme on a benchmark DARPA 2000 intrusion detection scenario specific dataset. In the experiment with this dataset, we have successfully correlated the alerts and the constructed attack scenarios matched the DARPA attack description very well.

Intrusion detection systems generate a large number of attacks per day necessitating the need to distinguish between real threats and false alerts. Our correlation module generates correlation graphs which depict causal relationship among alerts and therefore have a high probability of being real threats. They help the administrator in identifying the sequence of steps taken by the attacker to compromise system security.

The correlation graphs obtained might be hard to understand if the dataset is very huge. Our alert correlation module can be extended to provide utilities like graph reduction based on criteria like source IP, destination IP etc. or grouping of hyper-alerts based on some similarity criteria, which would help in understanding the graphs.

We can also complement the approach implemented by us with the approach by Qin and W.Lee [17]. The approach by Qin et al might discover some statistical patterns of intrusive activity which may be missed by our current module. This may

happen when the knowledge base has not been updated to incorporate information about new attacks. These discovered patterns can be stored and our alert correlation module can be extended to also look for these patterns while correlating the alerts.

Other possible improvements in this alert correlation scheme is prediction of future attacks based on the current attack scenario, and correlating alerts generated by the anomaly detector.

References

- [1] 2000 darpa intrusion detection scenario specific data set. http://www.ll.mit.edu/IST/ideval/data/2000/LLS_DDOS_1.0.html.
- [2] Emerald:event monitoring enabling responses to anomalous live disturbances. <http://www.sdl.sri.com/projects/emerald/>.
- [3] Snort, open source network intrusion detection system. <http://www.snort.org>.
- [4] Symantec incident manager. <http://enterprisesecurity.symantec.com/products/-products.cfm?productid=166%20/>.
- [5] Tcpreplay, open source program to replay captured network traffic. <http://tcpreplay.sourceforge.net>.
- [6] Tivoli intrusion manager. http://publib.boulder.ibm.com/tividd/td/TRM/GC32-1323-00/en_US/HTML/admin02.htm#wq1.
- [7] CUPPENS, F. Managing alerts in a multi-intrusion detection environment. In *ACSAC '01: Proceedings of the 17th Annual Computer Security Applications Conference* (Washington, DC, USA, 2001), IEEE Computer Society, p. 22.
- [8] CUPPENS, F., AND MIEGE, A. Alert correlation in a cooperative intrusion detection framework. In *SP '02: Proceedings of the 2002 IEEE Symposium on Security and Privacy* (Washington, DC, USA, 2002), IEEE Computer Society, p. 202.

- [9] DAIN, O., AND CUNNINGHAM, R. Fusing a heterogeneous alert stream into scenarios. In *Proc. of the 2001 ACM Workshop on Data Mining for Security Applications* (Nov. 2001), pp. 1–13.
- [10] DEBAR, H., AND WESPI, A. Aggregation and correlation on intrusion-detection alerts. In *Recent Advances in Intrusion Detection, LNCS 2212* (2001), pp. 85–103.
- [11] GOEL, S. Sachet - a distributed real-time network based intrusion detection system. Master's thesis, Indian Institute of Technology, Kanpur, June 2004.
- [12] JULISCH, K. Mining alarm clusters to improve alarm handling efficiency. In *ACSAC '01: Proceedings of the 17th Annual Computer Security Applications Conference* (Washington, DC, USA, 2001), IEEE Computer Society, p. 12.
- [13] JULISCH, K. Clustering intrusion detection alarms to support root cause analysis. *ACM Trans. Inf. Syst. Secur.* 6, 4 (2003), 443–471.
- [14] MURTHY, J. V. R. Design and implementation of an anomaly detection scheme in sachet intrusion detection system. Master's thesis, Indian Institute of Technology, Kanpur, June 2004.
- [15] NING, P., CUI, Y., REEVES, D. S., AND XU, D. Techniques and tools for analyzing intrusion alerts. *ACM Trans. Inf. Syst. Secur.* 7, 2 (2004), 274–318.
- [16] PORRAS, P. A., FONG, M. W., , AND VALDES, A. A mission-impact-based approach to INFOSEC alarm correlation. In *Lecture Notes in Computer Science, Proceedings Recent Advances in Intrusion Detection* (Zurich, Switzerland, October 2002), pp. 95–114.
- [17] QIN, X., AND LEE, W. Statistical causality analysis of infosec alert data. In *Proceedings of the 6th symposium on Recent Advances in Intrusion Detection (RAID 2003)* (2003), G. Vigna, E. Jonsson, and C. Kruegel, Eds., vol. 2820 of *Lecture Notes in Computer Science*, Springer Verlag, pp. 73–93.

- [18] TEMPLETON, S. J., AND LEVITT, K. A requires/provides model for computer attacks. In *NSPW '00: Proceedings of the 2000 workshop on New security paradigms* (New York, NY, USA, 2000), ACM Press, pp. 31–38.
- [19] VALDES, A., AND SKINNER, K. Probabilistic alert correlation. In *RAID '00: Proceedings of the 4th International Symposium on Recent Advances in Intrusion Detection* (London, UK, 2001), Springer-Verlag, pp. 54–68.
- [20] YU, D., AND FRINCKE, D. A novel framework for alert correlation and understanding. In *International Conference on Applied Cryptography and Network Security(ACNS)* (2004), vol. 3089, Springer LNCS series.

Appendix A

Predicates

The predicates for the knowledge base for attack scenario construction, and their implications are given in this section.

Predicate-Id	Predicate	Implications
1	<i>OSWindowsAll</i>	–
2	<i>ExistHost</i>	–
3	<i>ReadyToLaunchDDosAttack</i>	–
4	<i>SystemCompromised</i>	12,9
5	<i>ExistService</i>	2
7	<i>GainInfo</i>	–
8	<i>OSLinux</i>	–
9	<i>GainAccess</i>	2
10	<i>GainFileInfo</i>	2,5
11	<i>OSSolaris</i>	–
12	<i>GainRootAccess</i>	9,4
13	<i>VulnerableWebServer</i>	–
14	<i>GainDBAccess</i>	2,5
15	<i>GainDBADBAccess</i>	2,5
16	<i>SystemAttacked</i>	–
17	<i>VulnerableDNSServer</i>	–
18	<i>VulnerableMailServer</i>	–

Predicate-Id	Predicate	Implications
19	<i>DenialofServiceAttack</i>	–
20	<i>OSWindows9598</i>	–
21	<i>OSWindows2K</i>	–
22	<i>GainNetworkInfo</i>	2, 5
23	<i>OSMacintosh</i>	–
24	<i>GainServiceInfo</i>	5
25	<i>LossofData</i>	–
26	<i>VulnerableFTPServer</i>	–
27	<i>VulnerableCGI</i>	–
28	<i>GainBindVersionInfo</i>	2, 5
29	<i>OSHP</i>	–
30	<i>VulnerableTCPIP</i>	–
31	<i>VulnerableBrowser</i>	–
32	<i>OSIBMAIX</i>	–
33	<i>GainUseraccounts</i>	–
34	<i>GainRootinfo</i>	2, 5
35	<i>GainDNSVersionInfo</i>	2, 5
36	<i>EmailGame</i>	–
37	<i>ExistCiscoSwitch</i>	–
38	<i>IDSEvasion</i>	–
39	<i>VulnerableIMClient</i>	–
40	<i>GainOSInfo</i>	1, 8, 11, 21, 20, 23, 32, 29
41	<i>VulnerableNNTPServer</i>	–
42	<i>SystemVulnerable</i>	–
43	<i>VulnerableSMTPServer</i>	–
44	<i>VulnerableSQLServer</i>	–
45	<i>VulnerableColdFusion</i>	–
46	<i>VulnerableFrontPage</i>	–
47	<i>VulnerabilityScanner</i>	31, 27, 45, 17, 46, 26, 48, 49, 39, 18, 41, 43, 44, 30, 52, 13
48	<i>VulnerableIISServer</i>	–

Predicate-Id	Predicate	Implications
49	<i>VulnerableIMAPServer</i>	–
50	<i>GainFileAccess</i>	2, 5
51	<i>CreateBinary</i>	2
52	<i>VulnerableTelnetServer</i>	–

Appendix B

New Messages Included in the Sachet Protocol

The new messages added to the Sachet protocol for implementing Attack Scenario Construction in Sachet IDS are described in this appendix along with their format. The packet format for these messages is shown in Figures 3.2 and 3.3. Here, we present only the format of the data part of these messages. The numbers in the brackets beside the field in the format indicate their size in bytes. Strings have variable size and are terminated by a NULL character.

- **START_CORRELATION:** The Server instructs the Correlation Agent to start the correlation process. The data part of this message contains four fields: flag, hours, time-beg, time-end. A value of 1 for 'flag' field indicates that the last 'hours' of data needs to be correlated. If the value of 'flag' is 0, it indicates a request to correlate alerts between timeinterval 'time-beg' - 'time-end'. For periodic correlation, these parameters are supplied by the Server, and for specific-correlation request from the Console, these parameters are supplied by the administrator.

START_CORRELATION | flag (2) | hours (2) | time-beg (2) | time-end (2)

The reply to this message contains the message code **START_CORRELATION_REPLY** with a reply code in the data part indicating whether the correlation has started successfully or not.

- **CORRELATION_RESULT:** This message is used to transfer the correlation results from the Correlation Agent to the Server. Generally, the results cannot be sent in a single message and require multiple such messages. The results are sent as a file and in each message consecutive lines from this file are included till the entire file has been transferred. The results are handled by the Server as a text file and the file is reconstructed at the receiving end from these messages. The format is as follows.

CORRELATION_RESULT | more (2) | correlation results (string)

where 'more' flag indicates whether more messages will follow for this transfer. The reply to this message contains the message code **CORRELATION_RESULT_REPLY** with an empty data field.

- **CORRELATION_FAILED:** The Correlation Agent sends this message to Server when it encounters any problem during the correlation process. For example, database connectivity may fail while fetching the alerts, memory allocation may fail if too many alerts are there, etc. The data part of this message contain a 2-byte code indicating the cause for the failure, if possible, otherwise it contains zero.

CORRELATION_FAILED | code indicating reason (2)

The reply to this message from the Server contains the message code **CORRELATION_FAILED_REPLY** with an empty data part.

- **I_AC_DETAILS:** The Console sends this message to the Server requesting it to give the status of Alert Correlation Agent. The data part is empty. The reply to this message from the Server contains message code **I_AC_DETAILS_REPLY**. The message format is as follows

I_AC_DETAILS_REPLY | agentId (2) | isResultValid (2) | isPeriodic (2) | isACRunning | IP Address (string).

The 'agentId' field gives the Agent Id of the Correlation Agent. A value of 1 for "isResultValid" flag implies that new correlation results have been obtained from Correlation Agent and the GUI needs to be refreshed, a value of 2 implies

that the correlation process has failed and a value of 0 implies that there are no new results. A value of 1 for "isPeriodic" flag indicates that the new correlation results are for the periodic correlation and a value of 0 indicates that the results are for the explicit request made by the administrator through the Console. A value of 1 for "isACRunning" indicates that the Alert Correlation Agent is currently processing some request and a value of zero indicates that the Alert Correlation Agent is idle. The "IP Address" contains the IP Address of the machine on which Alert Correlation Agent is running.