Intrusion Prevention and Vulnerability Assessment in *Sachet* Intrusion Detection System

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

by

Bharat Jain

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 "*Intrusion Prevention and Vulnerability Assessment in Sachet Intrusion Detection System*", by *Bharat Kumar Jain*, has been carried out under our supervision and that this work has not been submitted elsewhere for a degree.

June, 2005

[illegible]

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

[illegible]

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

Abstract

An Intrusion Detection Systems (IDS) is a passive system which relies on the system administrator to take action when an attack is detected. The latency between an attack detection and corrective action taken by the administrator is usually high and therefore, by the time the administrator notices an attack and takes an action, the damage is already done. This necessitates the need for an Intrusion Prevention System which can not only detect attacks but can also actively respond to them. Intrusion prevention is a preemptive approach to system security which is used to identify potential threats and respond to them swiftly.

Vulnerabilty Assessment would provide a clear picture of all hosts on the network, the services that they provide and also information on the known vulnerabilities. This information would help the administrator in configuring the IDS and can also be used to assign priority to an alert.

In this thesis, we describe the design and implementation of Intrusion Prevention and Vulnerabilty Assessment schemes for Sachet IDS. Sachet is a distributed, real-time network-based Intrusion Detection System with centralized control developed at IIT Kanpur. Sachet uses an open source software, Snort, for signature-based detection. Recently, a new version of Snort, snort-inline, has been released for Linux which has intrusion prevention capability. The aim of Intrusion Prevention for Sachet is to provide this capability for Windows operating system. The aim of Vulnerability Assessment is to determine the vulnerabilities of machines monitored by Sachet at regular intervals and to use this information to assign priority to alerts generated by Snort.

Acknowledgements

I take this oppurtunity 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, Puneet Kaur, for her co-operation and innovative suggestions. I would also like to thank all the faculty members of the Department of Computer Science and Engineering, IIT Kanpur for enhancing my knowledge.

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

Contents

List of Figures

Chapter 1

Introduction

In today's world, where computer networks are essential for an organization's success, it is necessary to protect an organization's sensitive data and resources from intrusions. An intrusion can be defined as any activity that violates the confidentiality, integrity and availability of a system. Many preventive measures such as user authentication, tight access control mechanisms, or firewalls are employed by an organization to protect itself from intrusions. These preventive measures do not provide complete security because they are incapable of detecting attacks from disgruntled employees and network attacks like buffer overflow attacks which exploit the weaknesses in application programs. Therefore, Intrusion Detection Systems have come into existence as a second line of defence.

The aim of an Intrusion Detection System, IDS, is to detect illegal and improper use of system resources by unauthorized persons by monitoring network traffic and audit data. The techniques employed by an Intrusion Detection System fall into two broad categories: Signature-Based detection and Anomaly-Based detection. A signature-based IDS uses a signature database for detecting malicious activities. Each signature represents a pattern of activity which corresponds to a known attack. The signature-based IDS examines ongoing traffic, activity, transactions, or behavior, and tries to fina a match with these known patterns of predefined attacks. The strength of these systems lies in their signature database and therefore the database needs to be continuously updated to incorporate information about new attacks.

An anomaly-based IDS constructs a profile that represents normal usage and then uses current behavior data to detect deviations from this profile to recognize possible attack attempts. The profile can be constructed using various data mining and machine learning techniques and should be updated at regular intervals. The advantage of this approach over signature-based IDS is that it is able to detect novel attacks. However, it is non-trivial to define what constitutes a 'normal' behaviour and therefore systems based on this approach have a high tendency to generate false alarms.

Intrusion prevention is a preemptive approach to system security which is used to identify potential threats and respond to them swiftly. Like an IDS, an intrusion prevention system (IPS) monitors network traffic or audit data. However, an IPS also has the ability to take immediate action, based on a set of rules specified by the network administrator. For example, an IPS might drop a packet that it determines to be malicious and block all further traffic from that IP address. Legitimate traffic, meanwhile, is forwarded to the recipient with no apparent disruption or delay of service.

Intrusion Prevention Systems, IPS, can be classified into two main categories: Host-Based IPS and Network-Based IPS. A host-based IPS is installed on the system to be protected. It works in co-ordination with the operating system kernel to block abnormal application or user behavior. For example, it may monitor system calls or APIs invoked by applications in order to detect attacks. A host-based IPS requires tight integration with the operating system which implies that future operating system upgrades might cause problems.

A network-based IPS, also known as Inline IPS or Gateway IPS, is deployed to monitor a single host or an entire network segment. It analyzes the incoming network traffic for malicious activities. It may drop the malicious packets, reset the network session, or block traffic from particular hosts depending on user-specified policy. It should work in inline-mode, that is, every incoming packet should pass through it before reaching the target application. Therefore, it is essential that its impact on overall network performance is minimal. It should also have a high detection accuracy because it is an active system and inaccurate detection would lead to loss of legitimate traffic.

An IDS generates thousands of alerts per day. Therefore, it becomes critical to prioritize the alerts so that the administrator can focus on major threats. This is usually done through vulnerability assessment. In this scheme, information about the systems to be protected is maintained so that the attacks to which the system is known to be vulnerable are given higher priority.

1.1 Problem Statement and Our Approach

Intrusion detection systems are passive systems which rely on system administrator to take action when an attack is detected. The latency between an attack detection and corrective action taken by the administrator is usually high. By the time the system administrator notices an attack and takes any action, the damage is already done. This necessitates the need for an intrusion prevention system which can not only detect attacks but can also actively respond to them.

Vulnerability Assessment would provide a clear picture of all hosts on the network, the services that they provide and also information on the known vulnerabilities. This information would help the administrator in configuring the IDS. For example, he can disable signatures for which a host is not vulnerable. This information can also be used to assign severity level to an alert.

In this thesis, we describe the design and implementation of Intrusion Prevention and Vulnerabilty Assessment schemes for Sachet IDS [24, 23]. Sachet is a distributed, real-time network-based Intrusion Detection System with centralized control developed at IIT Kanpur. It uses both signature-based and anomaly-based techniques for detecting attacks. The architecture consists of Sachet Agents deployed at various strategic points of an enterprise network for attack detection. The detected alerts are sent to the Sachet Server which is a central command authority for controlling and managing multiple Agents. The detailed architecture of Sachet is given in Chapter 3.

Sachet uses an open source software, Snort [16] for signature-based detection.

Recently, a new version of Snort, snort-inline [15] has been released for Linux which has intrusion prevention capability. The aim of Intrusion Prevention for Sachet was to provide this capability for Windows operating system.

Our approach for intrusion prevention is as follows: We have developed a kernel mode filter-hook driver for Windows. The driver registers a call-back function with the Windows IP filter-driver. For each incoming/outgoing packet of the host machine, this callback function is called by the Windows IP filter-driver. The call-back function can decide whether to drop or forward the packet. In our call-back function, if the packet is coming from the loopback interface or is an outgoing packet it is forwarded, otherwise it is queued and the Windows IP-filter is notified to drop the packet.

The snort application is waiting in an infinite loop for an event which would be signalled by the call-back function whenever a packet comes. Snort would read the packet from the queue and check the packet for attack patterns. If the packet is found to be malicious then it might be dropped or the connection reset, based on user-specified policy. Otherwise, the packet is injected into the loopback interface with ethernet MAC address of the Microsoft Loopback adapter using Libnet [9]. This packet would again come to our call-back function but would now be simply forwarded to the destined application.

Our approach for implementing vulnerability assessment in Sachet is as follows: We use an open source vulnerability scanner software, Nessus [12]. Nessus is used by an Agent to determine the vulnerabilities of the host machines being monitored by it. Nessus provides a list of the vulnerabilities. When an alert is generated by the Agent, then this list is used to determine whether the host is vulnerable to this attack and the priority of the alert is set accordingly. The vulnerability information about each monitored machine is updated at regular intervals.

1.2 Organization of Report

In Chapter 2, a brief overview of the commercial intrusion preventions systems is given. In Chapter 3, we present the architecuture of Sachet and the changes made to it to incorporate intrusion prevention and vulnerability assessment. In Chapter 4, the design and implementation of intrusion prevention scheme for Sachet is presented. Chapter 5 presents the design and implementation of vulnerability assessment in Sachet. Chapter 6 presents conclusions and future work.

Chapter 2

Related Work

In this chapter we present a brief description of intrusion prevention and vulnerability assessment products.

2.1 Intrusion Prevention Systems

Intrusion Prevention Systems (IPS) are proactive defence mechanisms designed to detect attacks and prevent them from being successful. An IPS can respond to attacks rather than simply raising alerts. Typical responses of a network-based IPS on detecting a malicious packet might be to drop that packet, reset the connection, or block all traffic from the source IP address. A host-based IPS may respond to alerts by terminating the offending application, ending the user-session or disabling the user account. In the following sub-sections, we briefly describe some commercially available IPS and the techniques which they employ to detect and prevent attacks.

2.1.1 Juniper Networks Intrusion Detection and Prevention System

Juniper Networks Intrusion Detection and Prevention System (IDP) [7] is a network-based IPS. It has a three-tier architecture and consists of three components: IDP sensors, central server, and a graphical user interface. It uses a rule-based policy management in which individual rules make up the security policy. Each rule defines the way in which Juniper Networks IDP examines the network traffic and responds to detected intrusions.

The IDP sensors can be installed at different penetration points in an organization or enterprise network. Policies are loaded on individual IDP sensors. The central management server collects logs from all the sensors, and maintains state and policy information of each sensor in the database. The graphical user interface allows the system administrator to quickly view and investigate alerts, and to update security policies.

Juniper uses signature-based detection, protocol anomaly detection, and traffic anomaly detection techniques. Signature-based rules represent an attack pattern to be searched in each packet passing through the IDP sensor. Protocol anomaly detection is used to look for traffic which violates network or application-level protocols. Traffic anomaly detection compares the incoming traffic with normal traffic patterns and identifies deviations.

The IDP can respond in a number of ways when an attack is detected. It can drop/close the connection, send email alarms, or log the connection for future forensic investigation, depending on security policy. If a connection has to be dropped or closed, the sensor needs to be deployed in-line as an active device monitoring all network traffic. The connection is dropped by sending a TCP reset packet to both the source and destination IP addresses.

2.1.2 Attack Mitigator Intrusion Prevention System

Top Layer has developed the Attack Mitigator Intrusion Prevention System [18] which provides real-time proactive defence from network and application-based attacks. It provides both content-based and rate-based intrusion prevention. Rate-based IPS blocks traffic based on network load. This functionality is useful for stopping denial of service (DoS) attacks. The Attack Mitigator selectively blocks traffic based on connect rate, connection count and bandwidth consumption. Content-based IPS blocks traffic using IDS-like signatures, searching for specific contents in the network packets.

The Attack Mitigator uses six different real-time protection mechanisms to provide security for an organization: Protocol Validation filters, Attack Signatures, Advanced Firewall filters, intelligent Rate-based filters, Packet filters, and patented DDOS algorithms. Protocol Validation filters stop most of the application-based attacks by blocking sessions that violate application protocol rules. To construct the entire session, dedicated Application Specific Integrated Circuits (ASIC) are used. The entire session is then forwarded to its 'Deep Packet Inspection Engine' where dedicated Field Programmable Gate Array (FPGA) looks for protocol violation and actively blocks the transmission when protocol violation is detected. To prevent attackers from using evasion techniques like slow attacks, the Attack Mitigator uses session-aware 'Application Inspection Engine' which can maintain real-time intelligence of over two million IP addresses. Future packets originating from an IP address, from which malicious traffic had been observed before, are closely scrutinized.

Attack Signatures are similar to the rules used in Juniper Network IDP. Each signature detects known attacks. The signature database is updated at regular intervals to detect newly discovered attacks. Advanced-Firewall filters provide protection against undesired access by allowing the system administrator to specify filtering criteria based on origination or destination of a packet. Intelligent Rate-based filters along with patented DDoS algorithms provide protection against Denial of Service attacks. Packet filters examine packet headers and drop malformed packets.

2.1.3 Symantec Host-based Intrusion Prevention System

Symantec Host-based IPS [17] monitors activities of the host on which it is installed. It has the following four components which monitor various resources for detecting system access, resource usage and changes to files and configurations.

- Event Log Collector: It monitors applications, system and security event logs.
- Audit Collector: It detects changes to the Local Audit Policy on Windows systems.

- File Collector: It monitors the files and directories specified by the administrator. It detects and prevents tampering.

- Registry collector: It monitors changes in the Windows registry.

The Symantec Host-based IPS has a policy-based Processing Engine that processes events received from the above components to check for security violations and takes actions as specified in policies. A policy is a collection of rules that are configured to detect and respond to specific events. Actions fall in two broad categories: passive and preventive. Passive actions raise an alert on attack detection and provide details about the event which caused the alert. Preventive actions take intrusion prevention measures which can be one of the following:

- Kill Process Action : This action ends a Windows process based on detected changes to registry key values or on receiving events from the Event Log Collector.

- Disconnect Session Action : The connections that have the same user name or process ID as the process that generated the event are disconnected. However, it does not disconnect a session that is associated with an administrator account. This action does not prevent a user from logging into the system again.

- Disable User Action : This action is used to disable a user account.

2.1.4 Internet Security Systems Proventia IPS

Proventia Intrusion Prevention System [6] is a signature-based network IPS. The sensors use signatures to detect malicious traffic. An event is generated when an attack signature is detected. A configuration file known as 'Response File' is used to determine the actions to be taken on event detection. The actions can be one of the following:

- User-Specified : The administrator can specify an executable to be run when a specified event is detected.

- Block: The attack is blocked by dropping the packets and sending TCP resets to both source and destination IP addresses.
- Quarantine: The appliance creates quarantine rules in response to events and stores the rules in the 'Quarantine Rules' table. The following fields are available in the quarantine rules table : source IP, destination IP, source port, destination port, protocol, and expiration time. These rules are used to determine packets to block and the length of time to block them.
- Log-Events: The events are simply logged. The administrator can also specify an email-address to which a log of an event should be sent.

2.2 Vulnerability Assessment Systems

Vulnerability assessment is systematic examination of a system to identify components that may be at risk from an attack and the determination of appropriate procedures that can be implemented to reduce that risk. In the following sub-sections, we briefly describe two commercially available vulnerability assessment tools.

2.2.1 SAINT Scanning Engine

The Security Administrator's Integrated Network Tool [14], SAINT, detects security vulnerabilities of the target hosts. The sequence of steps followed by SAINT to identify vulnerabilities are:

- It scans every live target within the target list or range for TCP and UDP services.
- For each service it finds running, it launches a set of probes designed to detect anything that could allow an attacker to gain unauthorized access, create a denial-of-service, or gain sensitive information about the network.
- The data from the probes is used by its 'Inference Engine' to schedule further probes and to infer vulnerabilities and other information.

- The data analysis and reporting modules categorize the results in several ways, allowing customers to view the results conveniently.

SAINT can group vulnerabilities according to severity, type, or count. It provides description of each of the detected vulnerabilities and also provides suggestions to correct them. It references Common Vulnerabilities and Exposures (CVE) [2], and Information Assurance Vulnerability Alerts (IAVA) [5]. CVE provides a list of standardized names for vulnerabilities and other information security exposures. It aims to standardize the names for all publicly known vulnerabilities and security exposures. The IAVA system is used in US Department of Defence organizations to standardize the announcement and remediation of critical vulnerabilites.

2.2.2 Retina Network Security Scanner

The eEye Digital Security's Retina Network Security Scanner [3] scans a host or range of IP addresses. It first finds which all hosts are alive and then launches scans against them. The results give detailed information about the vulnerabilities found and possible fixes. It may even fix the vulnerabilities by downloading the patches depending on user configuration. The vulnerabilty signature database is automatically synchronized with eEye's Server on startup.

One of the unique feature of Retina is its proprietary CHAM (Common Hacking Attack Methods) technology. When this functionality is enabled, Retina takes on two roles. First, it performs a normal scan to identify all vulnerabilites. Then it switches to CHAM mode and becomes a confidential 'hacking-consultant'. In this mode, Retina will attempt to discover buffer-overflows, format string attacks by sending malformed data to the target. CHAM will also look for deviations from published RFCs for each service audited.

Chapter 3

Architecture of Sachet

In this chapter we briefly describe the architecture of Sachet. The essential components of Sachet are multiple Agents, Learning Agent, Correlation Agent, Server and Console. The components communicate with each other using Sachet protocol. Agents can be deployed at various strategic positions in an organization. An Agent monitors a host or a network segment for attacks in the network traffic and generates alerts. It forwards the detected alerts to the Server. The Server aggregates alerts from multiple Agents and stores them in a database. It controls Agents, Learning Agent, and Correlation Agent and interacts with the Console. The Console provides a graphical user interface to the administrator to monitor, configure and manage the Sachet IDS. It also provides detailed alert information. The architecture of Sachet is as shown in Figure 3.1.

In the following sections, we describe the components of Sachet and the Sachet protocol. In the final section, we describe the changes made to Sachet for incorporating intrusion prevention and vulnerability assessment modules.

3.1 The Sachet Protocol

Sachet protocol is used to provide secure and reliable communication between Sachet components. It provides authentication and encryption to the communicating parties. Sachet uses a public-key cryptography algorithm for authentication between

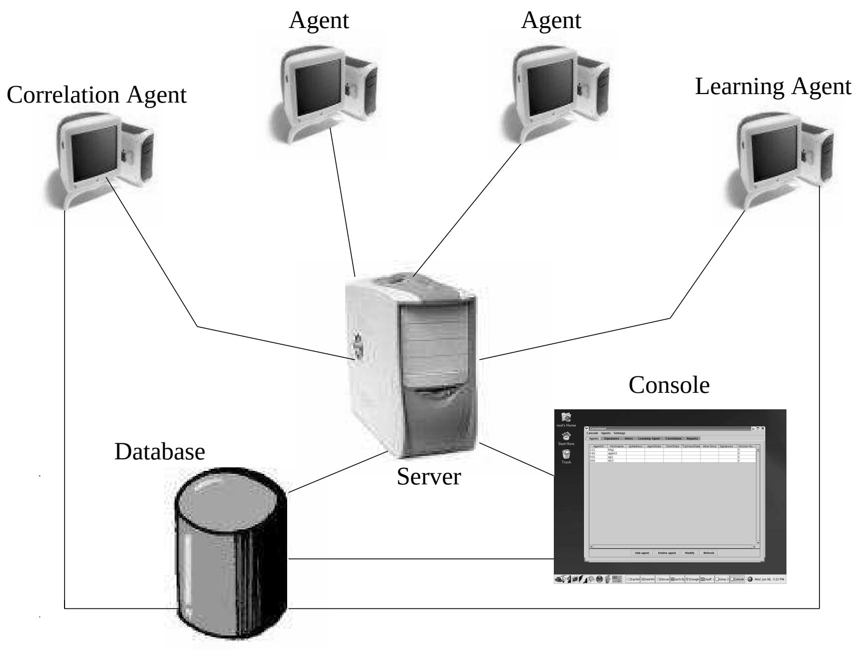

Figure 3.1: Architecture of Sachet IDS

the Server and Agents.

Sachet Server-Agent protocol is used for communication between the Server and Agents. The packet structure is as shown in Figure 3.2. *Encryption Type* field is used to indicate the method used for encrypting the packet. *PacketID* field contains a number that uniquely identifies each packet sent or received. *Agent ID* field contains the agent ID, which uniquely identifies an Agent. *Data Length* field gives the length of data in bytes. *Message Type* field describes the type of message such as, an alert

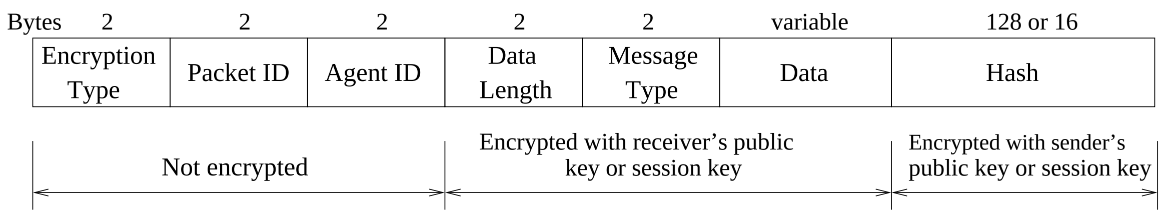

Figure 3.2: Packet Format for Sachet Server-Agent Protocol

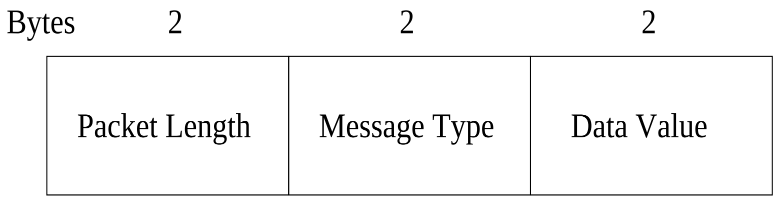

Figure 3.3: Packet Format for Sachet Server-Console Protocol

message, probe message, command message, etc. *Data* field contains data. *Hash* field contains the encrypted MD5 hash for the entire packet.

Sachet Server-Console protocol is used for communication between the Server and the Console. It is used for local communication between them and is implemented over TCP. The 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 such as, command message, request message, etc. *Value* field contains data.

3.2 Sachet Server

Sachet Server is installed on a dedicated machine and usually runs in background as a daemon in Linux or as a service in Windows. It maintains information about each Agent in the database and retrieves this information at the beginning of its execution. It oversees the working of the Agents and controls them by issuing commands to them. It periodically monitors the health of each Agent. It receives alerts from all Agents and stores them in a database. The Console is used by the administrator to interact with the Server because the Server by itself does not have a user interface. It uses a simple request-response protocol. The Console needs to authenticate itself with the Server.

3.3 Sachet Agent

Agents are deployed at strategic locations of an enterprise network to monitor the network traffic for malicious activites. An Agent uses both signature-based and anomaly-based intrusion detection techniques. It runs as a daemon in Linux or as a service in Windows and does not interact with the user. It consists of three components: Misuse Detector, Anomaly Detector, and Control Module, which run as separate processes on the host machine. Misuse Detector uses an open source software, Snort [16] for signature-based intrusion detection. Anomaly Detector uses the normal profile generated by the Learning Agent for anomaly-based intrusion detection. The Misuse Detector and Anomaly Detector are controlled by the Control Module. They send the detected alerts to the Control Module which in turn sends them to the Server over an encrypted channel. The Server and Agents need to mutually authenticate with each other and use Sachet Server-Agent Protocol for communication.

3.4 Sachet Learning Agent

The Learning Agent is used to build the normal profile of the network which is used by the Anomaly Detector in an Agent for anomaly-based intrusion detection. It learns on the connection features extracted from network traffic by Agents by using machine learning techniques. The learning is done on request from the administrator. As in the case of a normal Agent, the Learning Agent and the Server need to mutually authenticate with each other using public key cryptography.

3.5 Sachet Correlation Agent

The Correlation Agent is used to correlate the alerts received from the Misuse Detector and usually runs on a dedicated machine. The alerts are correlated based on the assumption that most intrusions are not isolated but related as different stages of a series of attacks, with the earlier stages preparing for the later ones. The correlation process generates attack scenarios. The scenarios are displayed as graphs in the Console. The Server periodically requests the Correlation Agent to correlate the last ‘x’ hours of alerts, where ‘x’ is configurable. The Console can also request the Correlation Agent through the Server to correlate a set of alerts. The Correlation

Agent and Server also need to mutually authenticate with each other.

3.6 The Sachet Console

The Console provides a GUI to the system administrator for monitoring, configuring and controlling the Sachet IDS. The Console and the Server should be run on the same host. The Console needs to authenticate itself to the Server and uses a simple request-response protocol for communication with the Server. The system administrator can use the Console to request the Server to issue commands to Agents (for example, disable/enable signatures, start/stop misuse detector and/or anomaly detector, update signature database etc.) and report the Server responses. It provides the means to add, modify and delete Agents. The Console periodically requests the Server to provide information about the entire system and displays the same. It periodically retrieves the alerts, received by the Server, from the database.

3.7 Changes made to Sachet

In this section we describe the changes made to Sachet to incorporate intrusion prevention and vulnerability assessment. The Misuse Detector in an Agent uses an open source software, Snort [16] for signature-based intrusion detection. A recently released version of Snort, Snort-Inline [15], provides intrusion prevention capability for Linux. We extended the Windows version of Snort to provide this functionality. The Misuse Detector can now operate in two modes: Passive-Mode and Inline-Mode. In Passive-Mode, it simply alerts the administrator on detecting malicious activities whereas in Inline-Mode it can also respond to them. The administrator can configure the Misuse Detector to drop a packet, or reset a connection when a malicious packet is detected. The Misuse Detector reports information about packets dropped and any connection that it has reset to the Control Module. The Control Module sends this information along with the alerts to the Server and this information is also displayed in the GUI. The administrator can configure the mode of operation of Misuse Detector from the Console.

An open source software, Nessus [12] is used to assess the network vulnerabilities. The Control Module of an Agent periodically runs Nessus on the machines monitored by the Misuse Detector and stores information about the detected vulnerabilities in a file. When an alert is raised by the Misuse Detector and reported to the Control Module, the gathered vulnerability information is used to inform the system administrator whether the victim machine is vulnerable to the attack corresponding to that alert. This information is also logged in the database. When Nessus is run by the Control Module, it causes alerts to be generated. We ignore alerts generated for packets whose source IP address is the Agent's IP address while assessment by Nessus is underway. Nessus uses a set of plugins for vulnerability detection which are regularly updated. A capability has been provided to the administrator to remotely update the plugins from the Console. Some new messages have been added to the Sachet Protocol to provide this functionality. The periodicity of Nessus scans, and the machines to be scanned can be configured through an Agent's configuration file.

Chapter 4

Intrusion Prevention System

We have implemented a kernel-mode driver for Windows to incorporate intrusion prevention capability in Sachet. In Section 4.1, we describe the working of Snort-Inline for Linux. In Section 4.2, we give a brief description of Windows network architecture. In Section 4.3, we briefly discuss various mechanisms by which packets can be filtered in Windows. In Section 4.4, the basic concepts required to write a kernel-mode driver such as synchronization mechanisms and interrupt request levels, are presented. In Section 4.5, we present the design and implementation of intrusion prevention capabilities in Snort for Windows.

4.1 Snort-Inline for Linux

Snort-Inline is a modified version of Snort that includes intrusion prevention capabilities. It can run in two modes:

- Network Intrusion Detection System (NIDS) mode: In this mode, Snort sniffs network packets using Libpcap [10], searches for attack patterns in packets, and generates alerts on attack detection. It is a passive mode of operation.

- Inline Mode: In this mode, Snort actively responds to detected attacks. It works in inline-mode, that is, every incoming packet passes through it before reaching the target application. It may respond to attack packets by dropping them or resetting the connection, depending on configuration.

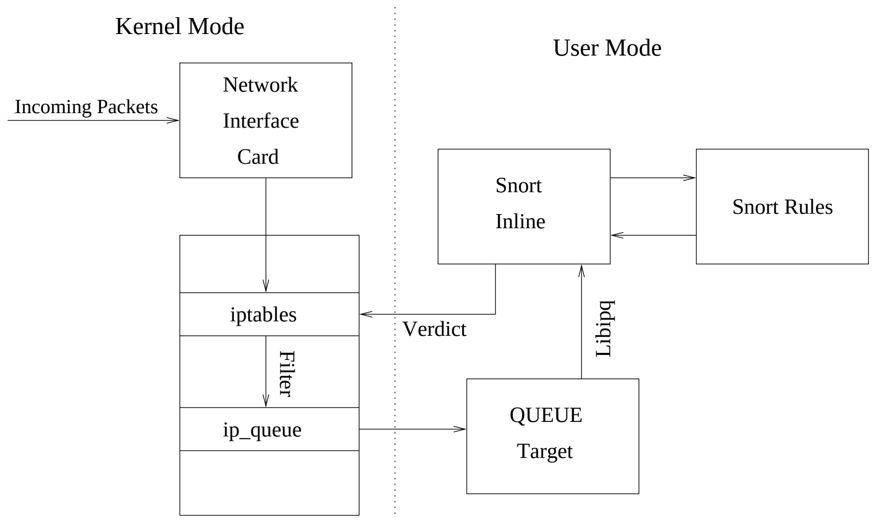

Figure 4.1: Snort-Inline for Linux

We briefly describe here the working of Snort in inline-mode (refer Fig. 4.1). In the inline-mode, Snort-Inline obtains packets from 'iptables', the default firewall package for Linux, and then causes 'iptables' to drop or pass packets based on Snort rules. A mechanism is needed to read the packets from network stack, which is in kernel space, into user space to allow Snort-Inline to process the packet, and then to receive these packets back into the network stack with a verdict specifying what to do with the packet, that is whether to drop the packet or forward the packet to the intended application. In Linux, this mechanism is provided by Netfilter [13]. Netfilter can be used to get packets from the network stack and a kernel module, called a queue handler, may be registered with Netfilter to pass packets to and from the kernel space to user space. Snort-Inline uses Netfilter to pass the packets out of the network stack, then these packets are copied to and from userspace using ip_queue, a standard kernel module which we register with the Netfilter.

The ip_queue module allows all incoming packets to be queued for user-space processing on adding the following rule to the iptables:

```
iptables -A INPUT -j QUEUE
```

The above rule causes all incoming traffic to be queued via the QUEUE target. Snort-Inline receives and processes these packets via Libipq [8]. Libipq provides an API for communicating with ip_queue. The 'ipq_set_verdict' function issues a verdict on a packet previously obtained with ipq_read, specifying the intended disposition of the packet, and optionally supplying a modified version of the payload data. Now, Snort-inline reads packets from the QUEUE target using Libipq and if the packet is found to be malicious, then a verdict to drop the packet is issued.

Following are the responses that can be configured once an attack packet is detected.

- alert - generates an alert and then logs the packet
- log - logs the packet
- pass - no action is taken
- drop - The drop rule-type will tell iptables to drop the packet and log it via usual Snort means.
- reject - The reject rule-type will tell iptables to drop the packet, and log it. A TCP reset, if the protocol is TCP, or an ICMP port unreachable, if the protocol is UDP, is sent using Libnet [9], an open source packet construction library.
- sdrop - The sdrop rule-type will tell iptables to drop the packet. Nothing is logged.

4.2 Windows Network Architecture

The Windows network architecture is as shown in Figures 4.2 and 4.3 [19] (the figures are taken from http://www.ndis.com/papers/winpktfilter.htm). The Network Driver Interface Specification (NDIS) library abstracts the network hardware from network drivers. NDIS supports the following types of network drivers:

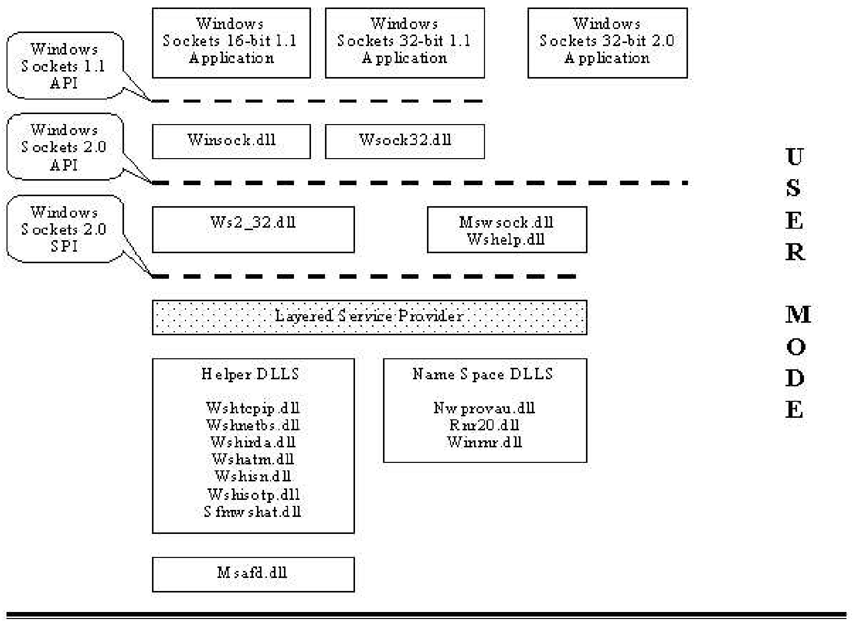

Figure 4.2: Windows User-mode Network Architecture

- Miniport drivers: An NDIS miniport driver is used to manage a network interface card (NIC) and interface with higher-level drivers, such as intermediate drivers and transport protocol drivers.

- Intermediate drivers: An intermediate driver communicates with both overlying protocol drivers and underlying miniport drivers. It is typically used to translate packets between network media (from ethernet packets to ATM packets), filter packets, and balance packet transmission across more than one NIC.

- Protocol drivers: Protocol driver is the highest level driver in the NDIS hierarchy of drivers. It copies data from the sending application into a packet, and sends these packets to the lower level driver. It also provides a protocol interface to receive incoming packets from the next lower-level driver.

The Transport Driver Interface (TDI) is a common interface for drivers and is used to communicate with the various network transport protocols. It provides standard methods for protocol addressing, sending and receiving datagrams, writing and reading on streams, detecting disconnects, etc. This allows services to remain independent of transport protocols.

Windows sockets provide an application programming interface (API) through which a user can transmit and receive application data across the wire, independent of the network protocol being used. It interfaces with the underlying TDI interface for sending and receiving application packets.

4.3 Windows Packet Filtering Mechanisms

The packet filtering mechanisms provided by Windows can be broadly classified into two categories: user-mode traffic filtering and kernel-mode traffic filtering. In this section, we briefly describe the two categories.

4.3.1 User-mode Traffic Filtering

In Windows, traffic filtering in user-mode can be done in the the following two ways (refer Fig. 4.2):

- Windows 2000 Packet Filtering Interface: An API is provided by Windows 2000 using which a user-mode application can specify a set of filtering rules such as, pass/drop based on IP addresses and port numbers, which would be used by TCP-IP for packet filtering.

- Winsock Layered Service Provider (LSP) : All the packets which are generated by user applications using Winsock API pass through this layer. Winsock LSP relies on services provided by the TDI interface for transmission of packets. Before transmitting/receiving a packet to/from the TDI interface, it can be modified, encrypted, or filtered.

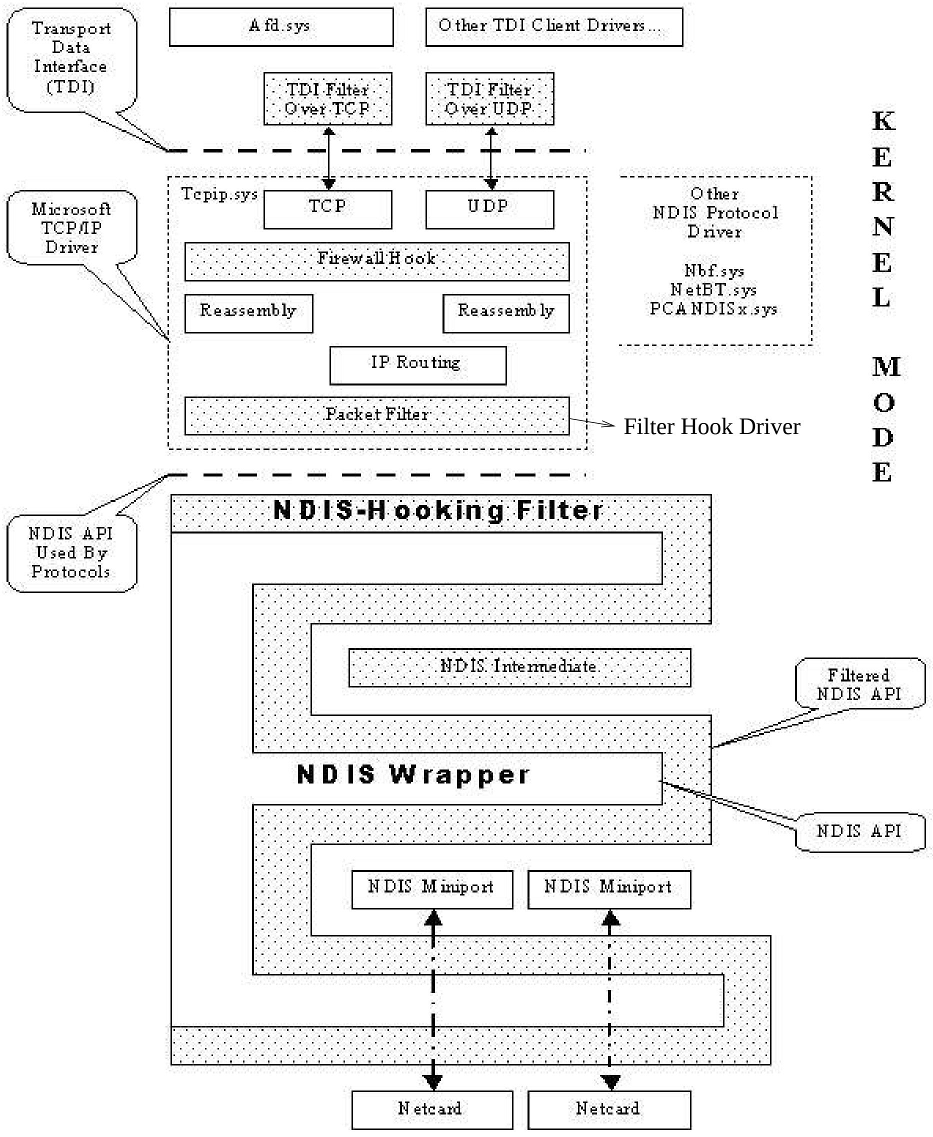

Figure 4.3: Kernel-Mode Network Architecture

4.3.2 Kernel-mode traffic filtering

In Windows kernel-mode, traffic filtering can be done in the following ways (refer Fig. 4.3):

- Kernel-mode sockets filter : It intercepts all the calls made by Windows socket DLL to the TDI interface. This method is similar to Winsock LSP.

- TDI-filter driver : In TDI filters, packets are captured by intercepting all calls directed to the devices created by tcpip.sys driver. Tcpip.sys driver implements TCP/IP stack and provides an interface to higher level drivers. TDI filters are generally used for creating personal firewalls.

- NDIS filter Intermediate Driver : It exposes a virtual miniport for each underlying miniport driver and the protocol drivers located above bind to this virtual miniport. The filter intermediate driver can modify packets received from the protocol drivers and sends them to the NDIS miniport driver. Similarly, packets received from a miniport driver can be modified and passed to protocol drivers.

- NDIS Hooking Filter Driver. This filtering technique is based on the interception of some subset of NDIS functions. The advantage of this method is ease of installation on the end user machine.

- Filter-Hook Driver. A Filter-Hook driver is a kernel-mode driver that implements a callback-function called a filter-hook and registers that callback function with the Windows IP-filter driver. The IP-filter driver then uses the filter-hook to determine how to process incoming and outgoing packets.

4.4 Kernel Mode Driver: Concepts

In this section, we give a brief overview of the concepts required to write a kernel-mode driver in Windows.

4.4.1 IRQL - Interrupt ReQuest Level

The IRQL is the priority ranking of an interrupt. An IRQL defines the hardware priority at which a processor operates at any given time. The IRQL of the processor essentially helps determine how a thread running at a specific IRQL is allowed to be interrupted. A thread running at a lower IRQL can be pre-empted to run code at a higher IRQL. Kernel APIs documented by MSDN generally specify the IRQL level at which you need to be running in order to use the API. The number of IRQLs and their specific values are processor dependent. A brief outline of some IRQLs is given below:

- PASSIVE_LEVEL : This is the lowest IRQL. No interrupts are masked off. All user threads and most of kernel-mode operations execute at this level.
- APC_LEVEL : Asynchronous procedure calls and page faults execute at this level.
- DISPATCH_LEVEL : In this level, pagable memory cannot be accessed.The APIs that can be used at this level are greatly reduced because one can only use non-paged memory. Thread scheduler executes at this level.
- DIRQL (Device IRQL) : This is a range of IRQLs, and is a method to determine which devices have priority over other devices.

4.4.2 IRP - I/O Request Packet

The IRP data-structure packages the information that a driver requires to respond to an I/O request. The requests might be from user mode or from kernel mode and are called IRP Major requests. Some of the IRP Major requests are IRP_MJ_CREATE, IRP_MJ_CLOSE, IRP_MJ_READ, IRP_MJ_WRITE, IRP_MJ_DEVICE_-CONTROL. A user mode application can use certain APIs to communicate with the driver using these IRPs. In the entry point of a driver, one can associate functions with the IRP major requests. The APIs would call these associated functions. This allows parameters to be passed to the driver. The APIs which call the functions associated with some of the IRP major requests are as follows:

- CreateFile(): This API calls the driver function associated with the IRP major request, IRP_MJ_CREATE. This API would be used by a user application to create a driver instance and would return a handle to the driver object. This handle would be used by the application for communication with the driver.

- CloseFile() : This API calls the driver function associated with the IRP major request, IRP_MJ_CLOSE. It is used to close the handle to the driver object and perform and clean-up operations such as deallocating memory.

- ReadFile() : This API calls the driver function associated with the IRP major request, IRP_MJ_READ. This API is used to read data from the driver.

- WriteFile() : This API calls the driver function associated with the IRP major request, IRP_MJ_WRITE. It is used to write data to the driver.

- DeviceIoControl() : This API calls the driver function associated with the IRP major request, IRP_MJ_DEVICE_CONTROL. It is used for requests other than read or write. An I/O control code (IOCTL) is sent as part of the IRP for requests. An IOCTL is a 32-bit control code that identifies an I/O or device operation.

4.4.3 Synchronization and Mutual Exclusion Mechanisms

Windows provides the following synchronization mechanisms:

▮ *Events*

Events are used to to synchronize between kernel-mode threads and between a kernel-mode thread and a user-mode application. An application can wait on an Event to be signalled by another application. There are two types of Events:

- Notification Events: A Notification Event wakes every waiting thread and remains in the signalled state until it is explicitly reset.

- Synchronization Events: A Synchronization Event wakes a single thread and immediately returns to the non-signalled state.

▮ *Spin-Locks*

While one thread owns a spin lock, any other threads that are waiting to acquire the lock keep waiting in a loop till the current owner-thread relinquishes the lock. Spin lock raises the IRQL to DISPATCH_LEVEL or higher and is the only synchronization mechanism that can be used at IRQL >= DISPATCH_LEVEL.

▮ *Kernel Mutexes*

A mutex is used to synchronize access to memory in pageable code or over a long period of time. It ensures that a thread has exclusive access to protected data.

4.5 Design and Implementation

4.5.1 Choice of Windows FiIltering Mechanism

We considered various Windows packet filtering mechanisms for incorporating intrusion prevention capabilities in Snort for Windows. ‘Windows 2000 Packet Filtering Interface’ does not allow filtering based on packet contents, and hence was not considered further. ‘Winsock Layered Service Provider’ and ‘Kernel-mode sockets filter’ were not used because a user applicaton may completely bypass them and communicate directly with the TDI interface. The disadvantage of a ‘TDI filter’ is that it is positioned at a high-level in the network stack, and hence filtering based on lower-level headers cannot be done. ‘NDIS-hooking drivers’ use techniques which are usually used by kernel-mode debuggers. These techniques are not well documented and hence this option was also not considered further.

Both ‘NDIS intermediate drivers’ and ‘filter-hook drivers’ were viable options for our intrusion prevention module. We decided to implement ‘Filter-Hook Driver’ because of it ease of implementation. The deployment time for an ‘NDIS intermediate driver’ is usually high and it needs to be digitally signed at Microsoft.

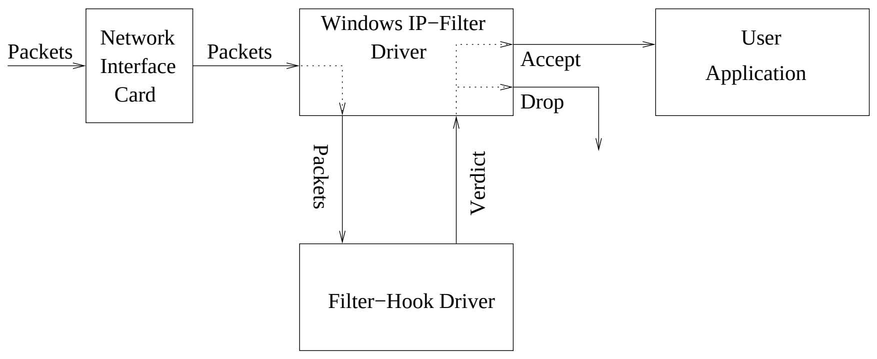

Figure 4.4: Filter-Hook Driver Mechanism

4.5.2 Filter Hook Driver

A filter-hook driver is a kernel-mode driver. It implements a callback-function, known as a filter-hook, and registers that callback-function with the Windows IP-filter driver. The Windows IP-filter driver then uses the filter-hook to determine how to process incoming and outgoing packets (refer Fig. 4.4).

The Windows IP-filter driver sends the following information to the callback-function:

- Pointer to the IP packet header
- Pointer to the IP packet
- Packet Length : Length of the IP packet, excluding the header
- Interface number of the network adapter that received the packet (-1 for outgoing packets)
- Interface number of the network adapter which will transmit the packet (-1 for incoming packets)
- IP address of the interface adapter that received the packet
- IP address of the interface adapter that will transmit the packet

The filter-hook driver processes a packet and returns one of the following response-codes to the Windows IP-filter driver which is used by the IP-filter driver to determine the fate of the packet:

- PF_FORWARD : It directs the Windows filter-driver to immediately forward the packet to the IP stack. If it is a local packet, IP forwards them up the stack. If the packet is destined for another computer, IP routes it accordingly.
- PF_DROP : It directs the Windows filter-driver to drop the packet.
- PF_PASS : It directs the Windows filter-driver to filter the packet as defined by the packet filtering API.

4.5.3 Design of Intrusion Prevention Module

We considered two design options for incorporating intrusion prevention capabilities in Sachet. In the first option, we considered integrating Snort code with the code for Filter-Hook driver, that is, the code for signature-based intrusion detection would be a part of the driver. The result of detection would be used for deciding the future of the packet and the current network-session. In the second option, we considered writing a Filter-Hook driver which would interact with the user-mode Snort application using IRPs.

The first option required major changes to Snort code because Snort uses user-mode APIs which cannot be called from kernel-mode, which is the mode in which Filter-Hook driver runs. Windows provides equivalent kernel-mode APIs for most of the user-mode APIs. This option required replacing every user-mode API in Snort with an equivalent kernel-mode API. Implementation of this option would have been non-trivial considering the size of Snort code. Therefore, this option was not implemented and the second option was considered further.

4.5.4 Implementation of Intrusion Prevention Module

We considered implementing the module using two events. Snort would wait for an event, say *PacketArrivedEvent*, which would be signalled by the callback-function whenever it would receive a packet from the Windows IP-filter driver. When this event is signalled, Snort would read the packet from the kernel mode using IRP major request, IRP_MJ_READ. After signalling the *PacketArrivedEvent* event, the callback function would wait on an event, say *PacketProcessedEvent*. The *PacketProcessedEvent* would be signalled by Snort after it has processed the packet to check for attack patterns. Snort would pass the result of attack detection to the driver using IRP major request, IRP_MJ_WRITE. The callback-function would use the result of attack detection to pass verdict about that packet to the Windows IP-filter driver. This method was not implemented because the callback function is called by the Windows IP-filter driver with the interrupt-request level DISPATCH_LEVEL. This is the level at which the thread scheduler runs, therefore this function can not be pre-empted. The above mentioned scheme would cause a deadlock because the callback-function would be waiting for an event to be signalled by the Snort application and the Snort application can not run until the callback-function has returned.

Then, we decided on a different implementation scheme for intrusion prevention module in Sachet. The architecture is as shown in Figure 4.5. A Filter-Hook driver, known as Snort-driver, was implemented. It would register a callback-function with the Windows IP-filter driver. The Snort-driver would communicate with the (modified) Snort application using IRPs. The Windows IP-filter driver would send all incoming and outgoing packets to the callback-function. The callback function would not process outgoing packets and would simply request the Windows IP-filter driver to forward the packets. For an incoming packet, it would check the interface number of the adapter on which it was received. If the interface number is that of the Loopback adapter, it would request the Windows IP-filter driver to forward the packet. Otherwise, it would queue the packet in a data structure, known as QUEUE, and would request the Windows IP-filter driver to drop the packet.

The modified Snort-application would create an instance of the Snort-driver at start-up. This would register the callback-function of the Snort-driver with the Windows IP-filter driver and would return a handle to the Snort-driver to the Snort-application. This handle would be used by the Snort application for communicating

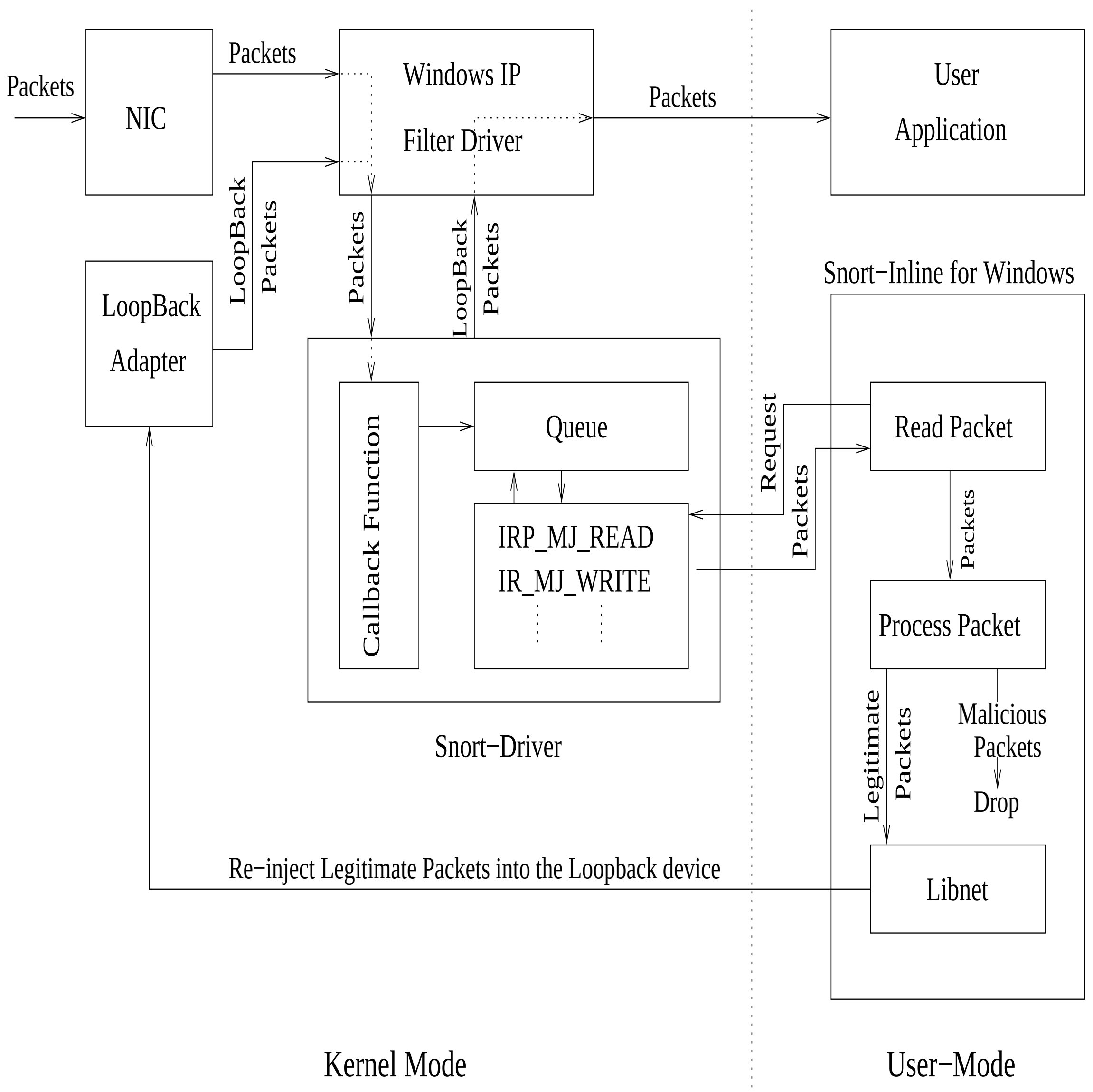

Figure 4.5: Architecture of Intrusion Prevention Module

with the Snort-driver. Snort-application then creates a notification event, known as *PacketArrivedEvent* and sends a handle to this event to the Snort-driver using an IOCTL in IRP_MJ_DEVICE_CONTROL. It then runs in an infinite loop in which it would do the following processing:

1. It reads a packet from the kernel-mode data structure, QUEUE, using IRP_MJ-_READ.

2. It processes the packet to check for attack-patterns.

3. If the packet is found to be malicious, it takes the action corresponding to the rule-type associated with the corresponding Snort signature. The generated alert may be logged, the packet may be dropped, or the connection may be reset.

4. If the packet is found to be legitimate, it re-injects the packet by constructing a raw packet using Libnet. The raw packet would be destined to the ethernet MAC address of the Loopback-adapter.

The *PacketArrivedEvent* event is used for synchronized access to the QUEUE data structure by the Snort application and the callback-function. Snort-application calls the Snort-driver function that is associated with the IRP major function, IRP_MJ_READ, to read a packet from the QUEUE. The function reads a packet from the QUEUE if it is non-empty, otherwise it waits on the *PacketArrivedEvent* event. When the callback-function receives an incoming packet from the Windows-IP-filter driver, it queues the packet into the QUEUE. If the QUEUE was empty prior to the inclusion of this packet, the callback-function signals the *PacketArrivedEvent* event which would wake the Snort-application if it was waiting for a packet from the Snort-Driver. A spin-lock is used for mutually exclusive access to the QUEUE. The interrupt-request level of the callback-function is DISPATCH_LEVEL and spin-lock is the only windows mechanism for mutual-exclusion that can be used at this level.

When the Snort applicaton is closed, it un-registers the callback-function of Snort-driver from the Windows IP-filter driver, and de-allocates memory allocated to the QUEUE data structure.

Chapter 5

Vulnerability Assessment

Vulnerability Assessment is an internal audit of a network or a system to identify vulnerabilities. A vulnerability is a flaw in the design, implementation, or operation of a computer system or a network that leaves it open to subversion by an unauthorized user. There are broadly two types of vulnerability assessment (VA) tools: Host-based VA tools and Network-based VA tools. A network-based VA tool is used to assess vulnerabilities of the host from the perspective of an intruder who is trying to use the network to break into systems. A host-based VA system looks at the host operating system and applications for vulnerabilities that could be exploited and checks them against the system security policy for non-compliance. Host-based VA tools assess the host from the perspective of the user who has a physical access to the machine.

Signature-based Intrusion Detection Systems detect intrusive actions by searching for attack patterns in network traffic and audit data. Attacks are usually specific to an operating system or a particular version of an application. Therefore, most of the times when an alert is generated, the target machine is not vulnerable to them. Vulnerability Assessment can be used to maintain information about known vunerabilities in the network. This information can be used to associate a severity level with an alert and also to configure the IDS, for example, signatures for which a host is non-vulnerable can be disabled.

We have implemented a vulnerability assessment module for Sachet. In Section

5.1, we give a brief description about the working of Nessus, an open source vulnerability scanner [12]. In Section 5.2, the design and implementation of vulnerability assessment module for Sachet is presented.

5.1 Nessus: Open Source Vulnerability Scanner

Nessus is an open source network vulnerability scanner designed to automate the testing and discovery of known security problems. It is based on client-server technology. Security checking is performed by the server and the client is used to provide user interface. A client can either be installed on the same machine on which the server is installed or on a different machine.

Every security check in Nessus is coded as a 'plugin'. A Nessus plugin is a simple program written in NASL (Nessus Attack Scripting Language) which checks for a given vulnerability. There are currently around 6000 plugins available. Each plugin has a unique nessus-id assigned to it. Information about CVE-id [2] and bugtraq-id [1] of the vulnerabitility is also maintained with the plugin. CVE provides a list of standardized names for vulnerabilities and other information security exposures. It aims to standardize the names for all publicly known vulnerabilities and security exposures. Bugtraq is a security mailing list for discussion and announcement of computer security vulnerabilities.

The user can specify the target(s) to be scanned, enable/disable a plugin or set of plugins and configure other options using the Nessus client. New vulnerabilities are being discovered and disseminated all the time. Nessus community releases plugins for detecting new vulnerabilitis on a daily basis and these can be downloaded from the Nessus website.

5.2 Design and Implementation of Vulnerability Assessment Module in Sachet

We use Nessus to maintain vulnerability profiles of the hosts monitored by Sachet IDS. We considered two design options. In the first option, Nessus is run by the

Server to assess the vulnerabilities of all the monitored hosts and the vulnerability profiles are maintained at the Server. Sachet Server can be used to manage hundreds of Agents and each Agent in turn might be monitoring more than one machine. The assessment needs to be done periodically because vulnerability status of a host may change as a result of starting a new network service, closing an existing service, patching an application vulnerability, etc. Ideally, the periodicity of assessment should be specific to a host machine. But it would be tedious for the Server to manage different time-intervals for vulnerability assessment for different Agents.

Alternatively, an Agent can maintain vulnerability information for hosts monitored by it. Nessus is run periodically by the Control Module to scan the monitored hosts for vulnerabilities. The periodicicity with which vulnerability assessment should be done for the hosts can be configured through the Agent's configuration file. Therefore, this design simplifies Server's configuration and hence has been selected for implementation.

The Control Module of an Agent runs Nessus at periodic intervals to assess the vulnerabilities of the monitored machines. When the Control Module receives an alert from the Misuse detector, it checks the targetted machine's vulnerability profile to determine whether that machine is vulnerable to that attack. It then sends this information along with the alert to the Sachet Server. The Nessus plugins can also be remotely updated from the Console.

5.2.1 Preprocessing

Snort signatures refer to well-known lists of vulnerabilities such as CVE, Bugtraq, Nessus, etc. We parse the Snort signature database and extract a list, known as ***Signature-Reference*** list, containing the following set of values: Snort signature-id, CVE-id, bugtraq-id, and nessus-id. A Nessus plugin is uniquely identified by nessus-id and references CVE-id and bugtraq-id if the corresponding vulnerability has been documented by CVE and Bugtraq. A nessus-id may reference more than one CVE-id and bugtraq-id. The nessus-plugins are parsed to extract lists of nessus-ids, CVE-ids, and bugtraq-ids into a data structure, known as ***Nessus-Reference*** structure. Nessus generates a report which gives a list of the vulnerabilities found.

Each vulnerability is associated with a nessus-id and references CVE and Bugtraq. For each monitored host, we parse the generated report to extract lists of nessus-ids, CVE-ids, and bugtraq-ids into a data structure, known as ***Host-Vulnerability*** structure.

5.2.2 Generation of Vulnerability Profile

We generate a vulnerability profile for each monitored host using the above extracted information. The vulnerability profile is a set of 2 tuples: Snort signature-id, 'vulnerability-status'. The field 'vulnerability-status' is used to specify whether the host is vulnerable to the attack corresponding to the signature-id. It can take 3 values: a value of 0 implies that it is not vulnerable, a value of 1 implies that vulnerability information for this signature could not be determined, and a value of 2 implies that it is vulnerable. Initially, the 'vulnerability status' of all signatures is set to unknown, that is, 1. The vulnerability profile is generated as follows: The lists in ***Host-Vulnerability*** structure are scanned and if a match with a tuple in ***Signature-Reference*** list is found, then vulnerabilty status of the corresponding signature-id is set to 2, that is, the host machine is vulnerable to this attack. Then the lists in 'Nessus-Vulnerability' structure are scanned and if a match with a tuple in ***Signature-Reference*** is found, the status of the corresponding signature-id is set to non-vulnerable if it had not been earlier set to vulnerable.

When the Misuse Detector generates an alert, the Control Module uses the vulnerability profile of the target machine to find its vulnerability status with respect to the corresponding signature-id and sends the status along with the alert to the Server. This status information is also displayed on Console. The administrator may configure IDS by disabling the signatures to which the host is not vulnerable, or apply patches for a vulnerability. This information also allows him to concentrate on alerts which pose a real threat to the system and require his immediate attention.

An Agent might be monitoring more than one host. The host(s) to be scanned for vulnerability assessment can be specified in the Agent's configuration file. By default, all monitored machines are scanned. The periodicity of vulnerability assessment can also be configured through this file. The administrator can also specify other configuration information such as, scanning be done only when an Agent starts or periodic scanning be disabled.

5.2.3 Updation of Nessus Plugins

Nessus plugins are released on a daily basis. It would be tedious to manually update the plugins-database at each Agent. A capability has been provided to the administrator to remotely update the plugins-database for all Agents from the Console. The administrator may download the latest plugins-database and request the Console to update the database for all Agents by specifying the path to the downloaded (compressed) file.

A version-number for the plugins database is maintained at the Server. It is initialized with 1 and incremented by one each time a request for updation comes from the administrator. The version-number for plugins database is also maintained for each Agent at the Server. On receiving a request for updation, the Server sends the compressed file to all alive Agents. On successful updation of nessus database at an Agent, the Server updates the version number for that Agent. Whenever an Agent authenticates itself to the Server, the Server compares the current version-number of Nessus plugins-database with the Agent's version-number, and if there is a mismatch then the plugins-database of that Agent is updated. This ensures that at any particular instance of time, all alive Agents have the latest nessus plugins-database.

Chapter 6

Conclusions and Future Work

We have developed intrusion prevention and vulnerability assessment modules for the Sachet Intrusion Detection System. We have built a kernel-mode Filter-Hook driver for Windows which communicates with Snort to provide capability to actively respond to an attack packet by either dropping the packet or resetting the connection. We also maintain vulnerability profile of each monitored host and when an alert is generated by Misuse Detector, we use this profile to determine whether the host is vulnerable to the attempted attack. This allows the administrator to concentrate on alerts which pose a real threat to the system and require his immediate attention, and also provides him with vulnerability information which he can use to configure the IDS, for example, he can disable a signature to which a monitored-host is not vulnerable.

The performance of intrusion prevention module is not satisfactory. A overhead of around 10 milli-second is incurred for processing a packet. This means it will effect the network performance. A possible way to improve it is by incorporating the Snort code required for detecting an attack into the driver, that is, by doing the signature-based attack detection in the callback-function . This is likely to improve the performance significantly as there would no longer be a need to switch between kernel-mode and user-mode for processing each packet. This would also not require any synchronization mechanism and re-injection of packet to the Loopback adapter.

Another possible improvement in the intrusion prevention module is that it should provide a capability to configure the policy of the firewall depending upon detected attacks.

Bibliography

[1] Bugtraq mailing list. *http://lists.insecure.org/about/bugtraq.txt*.

[2] Common vulnerabilities and exposures. *www.cve.mitre.org/cve/*.

[3] eeye digital security's retina network security scanner. *http://www.eeye.com/html/products/Retina/*.

[4] Handling irps: What every driver writer needs to know. *http://msdn.microsoft.com/en-us/dndevice/html/IRP_Handle.asp*.

[5] Information assurance vulnerability alerts. *https://infosec.navy.mil/*.

[6] Internet security systems proventia intrusion prevention system. *http://www.iss.net/products_services/enterprise_protection/proventia/g_series.php*.

[7] Juniper networks intrusion detection and prevention system. *http://www.juniper.net/products/intrusion/*.

[8] libipq - iptables userspace packet queuing library. *http://www.cs.princeton.edu/ nakao/libipq.htm*.

[9] Libnet, open source packet construction utility. *http://libnet.sourceforge.net*.

[10] Libpcap - user-level packet capture. *http://sourceforge.net/projects/libpcap/*.

[11] Msdn - microsoft developer network. *http://msdn.microsoft.com/*.

[12] Nessus, open source network vulnerability scanner. *http://www.nessus.org*.

[13] netfilter/iptables - firewall packet for linux. *http://www.netfilter.org/*.

[14] Saint vulnerability scanner. *www.saintcorporation.com/saint/*.

[15] Snort-inline, open source network intrusion prevention system for linux. *http://snort-inline.sourceforge.net*.

[16] Snort, open source network intrusion detection system. *http://www.snort.org*.

[17] Symantec host-based intrusion prevention system. *http://enterprisesecurity-.symantec.com/products/products.cfm?ProductID=48&EID=0*.

[18] Top player's attack mitigator. *http://www.toplayer.com/content/products/intrusion_ detection/-attack_ mitigator.jsp*.

[19] Windows packet filtering mechanism. *http://www.ndis.com/papers/winpktfilter.htm*.

[20] Locks, deadlocks, and synchronization. *Windows Hardware and Driver Central* (2004).

[21] Scheduling, thread context, and irql. *Windows Hardware and Driver Central* (2004).

[22] User-mode interactions: Guidelines for kernel-mode drivers. *Windows Hardware and Driver Central* (2004).

[23] Goel, S. Sachet - a distributed real-time network based intrusion detection system. Master's thesis, Indian Institute of Technology, Kanpur, June 2004.

[24] 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.

Appendix A

New Messages Included in the Sachet Protocol

The new messages added to the Sachet protocol for implementing Vulnerability Assessment 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.

- **UPDATE_PLUGINS:** The Server uses this message to update Nessus-plugins database at an Agent. The data part of this message contains three fields: 'more', 'filename', 'file-data'. The Server sends the plugins as a compressed file to an Agent. A single message cannot hold the entire file as there is an upper limit on the size of a UDP datagram. To overcome this drawback, the file is sent in multiple messages and the 'more' flag is used to inform the Agent when the transfer is complete. A value of '1' for this flag indicates more messages containing the remaining part of the file can be expected and a value of '0' indicates that the file has been completely transferred. 'filename' gives the name of the file being transfered and is used to determine the compression format. 'file-data' contains the contents of the file being transfered.

  UPDATE_PLUGINS | more (2)| filename (512)| file-data (variable)

The reply to this message contains the message code **UPDATE_PLUGINS_-REPLY**. It contains two fileds: ‘more’ and ‘success’. A value of ‘0’ for ‘more’ flag indicates that the compressed file has not yet been completely received, and a value of ‘1’ indicates that the file transfer is complete. ‘success’ field indicates the success of nessus-plugins database updation. A value of ‘1’ indicates that the database has been successfully updated and a value of ‘0’ indicates failure.

UPDATE_PLUGINS_REPLY | more (2)| success (2)

- **I_UPDATE_PLUGINS:** The Console sends this message to the Server requesting it to update the Nessus plugins-database of all the Agents. The data part contains the path to the latest plugins-database.

  I_UPDATE_PLUGINS | Path to latest plugins-database (variable)

  The reply to this message from the Server contains message code **I_UPDATE_-PLUGINS_REPLY**. The data part contains the list of Agent-ids that have been updated successfully.

  I_UDPATE_PLUGINS_REPLY | agentId (2) | agentId (2) ......