ABSTRACT

Intrusion Detection is an area of research in computer security that focuses on developing software systems to detect attempted intrusions in computer systems and networks. These systems are an important component of defensive measures to protect from potential threats and vulnerabilities. This technical report presents a literature survey of different types of intrusion detection systems and the various methods in developing them. It presents the design and implementation of a Signature-based Network Intrusion Detection System using JESS (SNIDJ), an intrusion detection system implemented using Java Expert System Shell (JESS).
# TABLE OF CONTENTS

Abstract .......................................................................................................................... ii

Table of Contents .......................................................................................................... iii

List of Figures .................................................................................................................. vi

1. Introduction and Background ................................................................................... 1
   1.1 Defining Intrusion Detection .............................................................................. 1
   1.2 Need for Intrusion Detection Systems ............................................................... 1
   1.3 Types of Intrusion Detection Systems ............................................................... 2
      1.3.1 Host-based IDS (HIDS) ............................................................................ 3
      1.3.2 Network-based IDS (NIDS) .................................................................... 3
      1.3.3 Signature-based IDS (SIDS) ................................................................... 4
      1.3.4 Anomaly-based IDS (AIDS) ................................................................... 5
   1.4 Intrusion Detection Techniques .......................................................................... 6
      1.4.1 Expert Systems ....................................................................................... 6
      1.4.2 Data Mining ............................................................................................ 7
      1.4.3 Neural Networks ..................................................................................... 7
      1.4.4 Genetic Programming ............................................................................. 8
   1.5 Java Expert System Shell (JESS) ..................................................................... 8

2. Narrative ....................................................................................................................... 10
   2.1 A Simple Expert System ................................................................................... 10
      2.1.1 User Interface ....................................................................................... 11
      2.1.2 Knowledge base .................................................................................... 11
2.1.3 Knowledge Acquisition Facility ........................................... 11
2.1.4 Inference Mechanism ....................................................... 11

2.2 Rule-based System ............................................................. 11
2.2.1 The Inference Engine ....................................................... 12
2.2.2 The Rule Base ............................................................... 13
2.2.3 The Working Memory ....................................................... 13
2.2.4 The Pattern Matcher ....................................................... 13
2.2.5 The Agenda ................................................................. 13
2.2.6 The Execution Engine ..................................................... 14

2.3 Development of SNIDJ as a Rule-based System ......................... 14
2.3.1 Knowledge Engineering ................................................... 14
2.3.2 Structuring the data ....................................................... 15
2.3.3 Testing ..................................................................... 15
2.3.4 Inference Building ......................................................... 16
2.3.5 Writing the rules .......................................................... 16
2.3.6 Iterative Development .................................................... 16

2.4 Snort ............................................................................ 16
2.4.1 Snort Rules ............................................................... 17
2.4.2 Structure of a Snort Rule ................................................ 17

3. System Design ................................................................. 20
3.1 Project Environment ......................................................... 20
3.2 Architecture of SNIDJ ....................................................... 20
3.2.1 Packet Capturing ......................................................... 20
# LIST OF FIGURES

| Figure 1.1 | Host-based Intrusion Detection System | 3 |
| Figure 1.2 | Network-based Intrusion Detection System | 4 |
| Figure 1.3 | A typical Signature-based Intrusion Detection System | 5 |
| Figure 1.4 | A typical Anomaly-based Intrusion Detection System | 6 |
| Figure 1.5 | Intrusion Detection Techniques | 7 |
| Figure 2.1 | A Simple Expert System | 10 |
| Figure 2.2 | A Typical Rule-based System | 12 |
| Figure 2.3 | Development of a Rule-based System | 15 |
| Figure 2.4 | Structure of a Snort Rule | 18 |
| Figure 3.1 | Architecture of SNIDJ | 21 |
1. INTRODUCTION AND BACKGROUND

1.1 Defining Intrusion Detection

An intrusion is an attempt to gain unauthorized access to a system with the purpose of either testing the security of the network, using the facility as a launching pad for further attacks on other systems, modifying or stealing information, etc. Intrusion Detection (ID) is the art of detecting intrusions and taking appropriate action against them. [Sans 2003]

1.2 Need for Intrusion Detection Systems

Intrusion Detection Systems (IDSs) are very important in today’s computing environment because it is difficult to keep pace with the current and potential threats and vulnerabilities in our computing systems. The Internet is in a process of constant evolution and so are the new threats and vulnerabilities associated with the networking environment. IDSs assist in managing threats and vulnerabilities in this rapidly changing environment. [Sans 2003]

Threats include groups of people who have the potential to compromise the computer system, either as a learning experience in the field of computer security (hackers) or as a way of having fun by destroying these systems (crackers). These may be a curious teenager, a disgruntled employee or espionage from a rival company or foreign government. [Sans 2003]

Vulnerabilities are weaknesses in systems. They can be exploited and used to compromise the computer system. New vulnerabilities are discovered all the time. In a worst-case scenario, an intrusion may cause production downtime, sabotage of critical information, theft of confidential information, cash, or other assets, or even negative
public relations that may affect the company's stock price. [Sans 2003] The attacks committed by intruders can be summarized as follows. [George 2002]

1. Sniffer attacks - capturing data as it traverses the net.
2. E-mail attacks - gaining system access through vulnerabilities in network service software.
3. Network File System attacks - gaining data access through vulnerabilities in operating system software.
4. Network infrastructure attacks - denial of service through attack on routers and name servers.
5. IP spoofing attacks - gaining system attacks by tunneling through firewalls.
6. WWW attacks - gaining users or system information through the web or cgi programs.

Therefore, an Intrusion Detection System is a security system that can assist in protecting a company from intrusion by expanding the options available to manage the risk from threats and vulnerabilities. It monitors computer systems and network traffic and analyzes that traffic for possible attacks originating from outside the organization and also for system misuse and attacks originating from inside the organization. It could be used to detect an intruder, identify and stop the intruder, support investigations to find out how the intruder got in and prevent the exploit from being used by future intruders.

1.3 Types of Intrusion Detection Systems

Intrusion Detection Systems can be broadly classified into four categories. They are discussed in the sections that follow.
1.3.1 Host-based IDS (HIDS)

A Host-based Intrusion Detection System defends and monitors the single host on which it resides. A HIDS gathers information from either the operating system audit trails and/or system logs of the host on which it has been installed (Figure 1.1). It not only monitors the communications traffic in and out of a single computer but also checks the integrity of the system files and processes. Examples of HIDS include BlackIce Defender, Tripwire, Fcheck, Windows NT/2000 Event Viewer, etc. [Beale 2003]

![Host-based Intrusion Detection System Diagram](image)

**Figure 1.1** Host-based Intrusion Detection System [Beale 2003]

1.3.2 Network-based IDS (NIDS)

A Network-based Intrusion Detection System (NIDS) monitors the traffic on its network segment (Figure 1.2). It looks at the packets on the network as they pass some sensor. The sensor can only see the packets that happen to be carried on the network
segment it is attached to. Packets are considered to be of interest if they match a signature [Sans 2003]. Well-known, network-based intrusion detection systems include AXENT (www.axent.com), Cisco (www.cisco.com), CyberSafe (www.cybersafe.com), ISS (www.iss.net), and Shadow (www.nswnavymil/issec/cid). [Beale 2003]

**Figure 1.2** Network-based Intrusion Detection System [Beale 2003]

1.3.3 Signature-based IDS (SIDS)

In Signature-based IDS (SIDS), also known as misuse detection system, the system analyzes system activity, looking for events or sets of events that match a predefined pattern of events that describe a known attack. These patterns are known as signatures. A SIDS looks for a specific signature on a host or a network to detect an intrusion. It works in much the same way as a virus scanner, efficiently sniffing out all
known styles of attack as shown in Figure 1.3. SIDSs are widely available and easy to implement and update. Most IDS in systemic use in the government are signature based. One disadvantage of SIDS is that it is only as good as the database of stored signatures. It requires regular signature updates to keep in touch with variations in hacker technique. Network Intrusion Detector (NID), NetRanger and Network Security Monitor (NSM) are well known signature-based intrusion detection systems. [Sans 2003]

![Figure 1.3 A Typical Signature-based Intrusion detection system](image)

1.3.4 Anomaly-based IDS (AIDS)

Anomaly-based Intrusion Detection Systems (AIDS) construct statistical models of the typical behavior of the system and issue warnings when they observe actions that deviate significantly from those models. AIDS work under the premise that all attacks differ from the normal activity and can be traced if a system that identifies these differences is introduced. These tools create profiles of the normal behavior of users, hosts and networks from the historical data over a period of normal operation of the system / network. Then they collect all available data from the system and check them to find deviations from the normal behavior. When the deviation is observed, an alarm is
generated as shown in Figure 1.4. A well-known Intrusion Detection System developed using this approach is EMERALD, which combines statistical anomaly detection with signature verification. [Sans 2003]

![Diagram of Anomaly-based Intrusion Detection System]

**Figure 1.4** A Typical Anomaly-based Intrusion Detection System.

## 1.4 Intrusion Detection Techniques

There are four different approaches to building Intrusion Detection Systems as shown in Figure 1.5. These are described in the following subsections:

### 1.4.1 Expert Systems

An expert system is intelligent software that contains a set of rules that can be repeatedly applied to a collection of facts about the world. Rules that apply are fired, or executed. These systems can be used to develop knowledge-based intrusion detection systems that identify attacks on the networks or systems as signatures encoded in the system. The expert system continuously monitors audit trails/network packets, and when it finds any suspicious activity in the system, it fires rules. Almost all of the Intrusion
detection systems that are in use today use this technology. Popular among them is NIDES. [George 2002]

1.4.2 Data Mining

Data Mining refers to the process of extracting descriptive models from large stores of data. This approach can be effectively used to develop anomaly based intrusion detection systems by considering intrusion detection as a data analysis process. By identifying bounds for valid network activity, data mining will aid an analyst in his/her ability to distinguish attack activity from common everyday traffic on the network. The recent rapid development in data mining has made available a wide variety of algorithms, drawn from the field of statistics, pattern recognition, machine learning and database that can be used to develop intrusion detection systems. [George 2002]

1.4.3 Neural networks

Artificial neural networks are collections of mathematical models that emulate some of the observed properties of biological nervous systems and draw on the analogies of adaptive biological learning. They can be used in developing anomaly based intrusion
detection systems because of their ability to cope with lots of imprecise and noisy data and their ease of modification for new users entering the system. A neural network is trained to predict a user’s next action or command, given his history of previous actions and commands. All incorrectly predicted events actually measure the deviation of the user from the established profile. [George 2002]

1.4.4 Genetic Programming

Genetic Programming is used in IDS for its learning capability. It allows programs to be evolved, which could then be executed in a production environment. The final results of an evolution run is a set of programs coded in a simple meta-language tailored to solving a specific problem, which are placed in a real system and are run continuously to detect intrusions.

This technique can be used to develop anomaly based Intrusion Detection Systems. It consists of a set of agents that continuously monitor and detect very simple intrusions. Their code is composed of a set of operators and a set of primitives. This set of agents is assigned a fitness score during the test scenario phase. If an agent fails to report a high suspicion of a known intrusion, the agent will be given a low fitness score. In the end of the training, a penalty is assigned to the sensors, based on how this scenario is ranked. These agents are then placed in a production system. [George 2002]

1.5 Java Expert System Shell (JESS)

JESS is a rule engine and scripting environment written entirely in Sun's Java™ language by Ernest Friedman-Hill at Sandia National Laboratories in Livermore, CA. It is a tool for building a type of intelligent software called Expert Systems. An Expert System
is a set of rules that can be repeatedly applied to a collection of facts about the world.
Rules that apply are fired, or executed. [JESS 2003]

JESS supports the development of rule-based expert systems that can be tightly
coupled to code written in powerful, portable Java language. It uses a special algorithm
called Rete to match the rules to the facts. Rete makes JESS much faster than a simple set
of cascading if..then statements in a loop. JESS is small, light, and one of the fastest rule
engines available.

JESS is also a powerful Java scripting environment, from which one can create
Java objects and call Java methods without compiling any Java code. [JESS 2003]
2. NARRATIVE

2.1 A Simple Expert System

An expert system is a computer-based system that uses knowledge, facts and reasoning techniques to solve problems that normally require the abilities of human experts. Most expert systems have certain basic components: a user interface, a knowledge base, and an inference mechanism.

![Diagram of a simple expert system]

**Figure 2.1** A Simple Expert System [Martin 1988]

2.1.1 User Interface

The user interface is software that provides for the communication exchange between the system user and the system. Through the user interface, the user can enter facts about a specific situation that are relevant to the system’s subject domain and can
ask the expert system questions within the system’s subject area. Many expert systems also accept new knowledge through the user interface. [Martin 1988]

2.1.2 Knowledge base

The expert system base contains expert-level knowledge on a particular subject. The knowledge obtained from one or more human experts and or other sources such as books, magazines or web sites is stored in a knowledge representational form that is inherent to the expert system design. Expert systems are also referred to as knowledge-based system because they always include a knowledge base. [Martin 1988]

2.1.3 Knowledge Acquisition Facility

The knowledge acquisition facility (KAF) is software that provides a dialogue between the expert system and the human experts for the purpose of acquiring knowledge from the human experts. The KAF places this acquired knowledge in the system’s database. [Martin 1988]

2.1.4 Inference Mechanism

The inference mechanism is software that performs the inference reasoning tasks for the expert system. The inference mechanism is also called the inference engine. The inference engine uses the knowledge in the expert system knowledge base and the information provided by the user to infer new knowledge. [Martin 1988]

2.2 Rule-based System

The expert system component in SNIDJ is a rule-based system whose primary function is to apply rules to the data contained in the network packets. The architecture of a typical rule based system is shown in Figure 2.2. [Friedman 2003]
Figure 2.2 A typical rule based system [Friedman 2003]

2.2.1 The Inference Engine

The inference engine is the central part of a rule engine and controls the whole process of applying rules to the working memory to obtain the outputs of the system. It works in discrete cycles as follows: [Friedman 2003]

1. All the rules are compared to working memory (using the pattern matcher) to decide which ones should be activated during this cycle. This unordered list of activated rules, together with any other rules activated in previous cycles, is called the conflict set.

2. The conflict set is ordered to form the agenda - the list of rules whose right-hand sides will be executed, or fired. The process of ordering the agenda is called conflict
resolution. The conflict resolution strategy for a given rule engine will depend on many factors, only some of which will be under the programmer’s control.

3. To complete the cycle, the first rule on the agenda is fired (possibly changing the working memory) and the entire process is repeated.

2.2.2 The Rule Base

The rule base contains all the rules the system knows. They may simply be stored as strings of text, but most often a rule compiler processes them into some form that the inference engine can work with more efficiently. JESS’s rule compiler builds a complex, indexed data structure called Rete network, which makes rule processing fast. The rule base can also be stored in the form of relational database that allows the user to select rules based on certain criteria like date, time, and user access rights. [Friedman 2003]

2.2.3 The Working Memory

The working memory or fact-base contains all types of information the rule-based system is working with. It can hold both the premises and conclusion of the rules. [Friedman 2003]

2.2.4 The Pattern Matcher

The pattern matcher decides which rules to apply, given the current contents of the working memory. [Friedman 2003]

2.2.5 The Agenda

The list of rules that should be fired, as determined by the inference engine, is stored on the agenda. The agenda is responsible for using the conflict strategy to decide which of the rules out of all those that apply, have the highest priority and should be fired first. [Friedman 2003]
2.2.6 The Execution Engine

The execution engine is the component of a rule engine that fires the rules. Jess offers a complete programming language that can be used to define what happens when a given rule fires. The execution engine then represents the environment in which this programming executes. [Friedman 2003]

2.3 Development of SNIDJ as a Rule-based system.

The development of SNIDJ is based on the general principles that are important for the development of any rule-based system. These steps are summarized as shown in Figure 2.3. [Friedman 2003]

2.3.1 Knowledge Engineering

This is the first step in the development of SNIDJ where the data used to implement SNIDJ rules is collected. In SNIDJ, the signatures or rules that represent the different types of attack scenarios should be gathered in the knowledge engineering process. This is achieved from the knowledge base of Snort 2.0 Intrusion Detection System. [Beale 2003] Snort is an open source Intrusion Detection System that is based on rules. These rules are, in turn, based on intruder signatures. These signatures may be present in the header part of a network packet or in the payload. A rule may be used to generate an alert message, log a message, or pass the data packet.

Rules for SNIDJ are developed by understanding those available in Snort and scripting them in the JESS environment.
2.3.2 Structuring the data

In this step, the knowledge collected in the previous step is examined and the data structures are designed that will make it easy to implement the rules clearly and directly. This process resembles object-oriented analysis. First, the major concepts of SNIDJ are identified and then all the variable characteristics of each concept are listed. [Friedman 2003]
2.3.3 Testing

Testing is a very important step in the development of rule-based system. If rigorous tests are applied to the system at every stage of its development, it will be more robust, more modular and better understood than a system that wasn’t tested until the end.

In the development of SNIDJ, tests were conducted on rules. [Friedman 2003]

2.3.4 Interface building

For the rule based expert system component of SNIDJ to do some useful work, it needs to be connected in some way to its environment. This involves how the expert system monitors the network packets, the network from which the packets should be detected etc. [Friedman 2003]

2.3.5 Writing the rules

Once the data structures are defined, the interfaces are specified, and the tests are in place, it’s time to begin writing the rules. The rules in SNIDJ are developed on the basis of those in Snort. The rule base in Snort is interpreted in plain English and then encoded in JESS scripting language. [Friedman 2003]

2.3.6 Iterative development

Once the rules have been developed in SNIDJ, it is possible that new rules might have to be developed because computer systems face new attacks every day from hackers. When this happens, more knowledge engineering needs to be done. The development of rule-based system lends itself well to this sort of iterative procedure. [Friedman 2003]
2.4 Snort

Snort is an open source packet sniffer / packet logger / network intrusion detection system, available for free under GNU Public license. As a packet sniffer, it is used to intercept packets on the screen and display them on the screen. As a packet logger, it is used to log specified network traffic either as ASCII text or binary format. As a network Intrusion Detection System, it is used to screen the network traffic for predetermined or content using a set of rules that trigger an alert whenever errant packets are detected in the network. [Beale 2003]

2.4.1 Snort Rules

Like viruses, most intruder activity has some sort of signature. Information about these signatures is used to create Snort rules. These signatures or attacks may be present in the header part of a packet or in the payload. Snort's detection system is based on rules. These rules, in turn, are based on intruder signatures. Snort rules can be used to check various parts of a data packet. [Beale 2003]

2.4.2 Structure of a Snort Rule

A Snort Rule is made up of two components, the Rule Header and the Rule Body, as shown in Figure 2.4.

The Rule Header

The Rule Header is divided in four main categories that are described as follows:

Rule Actions: A rule action is the first part of a Snort rule. It shows what action will be taken when rule conditions are met. There are three predefined actions as described below:
**Pass:** This action tells Snort to ignore the packet. This action plays an important role in speeding up Snort operation in cases where we don't want to apply checks on certain packets.

![Figure 2.4 Structure of a Snort Rule](image)

**Log:** This action tells Snort to log the packet in a manner as specified during the configuration of the Snort sensor.

**Alert:** The alert action is used to send an alert message when rule conditions are true for a particular packet.

**Protocols:** Protocol is the second part of a Snort rule. The protocol part of a Snort rule shows on which type of packet the rule will be applied. Currently Snort understands the following protocols:

- Internet Protocol (IP)
- Internet Control Message Protocol (ICMP)
- Transmission Control Protocol (TCP):
- User Datagram Protocol (UDP)
**Source Information:** This gives the information of the source computer from where the packet originated. It has two parts, the IP address of the source computer and the port number of the source computer. The keyword *any* can be used to apply a rule on all addresses. Similarly, the keyword *any* can be used to apply the rule on all packets irrespective of the port number.

**Destination Information:** This gives the information of the computer to where the packet is flowing to. It has two parts, the IP address of the destination computer and the port number of the destination computer. The keyword *any* can be used to apply a rule on all addresses. Similarly, the keyword *any* can be used to apply the rule on all packets irrespective of the port number.

**Rule Body**

The Rule Body contains various sections enclosed inside a pair of parentheses. Each section defines an option trailed by the desired option value. There may be one option or many and the options are separated with a semicolon. When multiple options are used, they form a logical AND. The action in the rule header is invoked only when all criteria in the options are true. There are several rule options; the discussion of each of them is beyond the scope of this report.
3. SYSTEM DESIGN

3.1 Project Environment

The Signature based Intrusion Detection System using JESS (SNIDJ) is a rule based expert system developed using Java Expert System Shell (JESS). The knowledge base for SNIDJ is obtained from the rule files of the source code of Snort. These rules that represent different types of attack signatures are encoded using the constructs provided by JESS. The input to SNIDJ is the network traffic obtained by using Snort as a packet-sniffing tool. This traffic is encoded into facts using the constructs provided by JESS. Network traffic capture is done by installing Snort on a host computer.

3.2 Architecture of SNIDJ

The working of SNIDJ is represented in the Figure 3.1. As seen in the system diagram, the packets from the network are captured using Snort. Once these packets are captured, they are sent to the expert system component of SNIDJ, where they are converted into facts using the constructs provided in JESS. These facts are then matched against the rules contained in the knowledge base of SNIDJ. If any of these rules match against the facts then the rule is fired and the corresponding action is taken.

3.2.1 Packet Capturing

Packet capturing is done using Snort. Snort was used as our packet-capturing tool because the network traffic should be compatible with the format of the rules in the knowledge base.
Each network packet captured by Snort has the following format:

{date} - {time} {source-ip-address:port} -> {destination-ip-address:port} {protocol}
{TTL} {TOS} {ID} {IP-length} {datagram-length} {payload-length} {hex-dump}
{ASCII-dump}

{date} - {time}: This gives the date and time when the network packet was captured.

{source-ip-address:port}: This gives the information of the source computer in the form of ip-address and the port number from where the packet originated.

{destination-ip-address:port}: This gives the information of the source computer in the form of ip-address and the port number from where the packet originated.
{protocol}: This gives the information of the protocol that is being used during transmission of the packet.

{TTL}: This gives the time to live value of the packet

{TOS}: This value determines the type of service value of the packet

{ID}: This gives the fragment identification number of the packet.

{IP-length}: This gives the length of the packet

{datagram-length}: This gives the datagram length of the packet

{payload-length}: This gives the length of the payload data carried by the packet.

{hex-dump} & {ASCII-dump}: This gives the information about the data that is contained in the packet.

A sample output of three packets captured using Snort running on a host computer in the campus labs is as follows:

04/09-12:00:05.994275 165.95.7.5:13039 -> 165.95.6.183:22
TCP TTL:125 TOS:0x0 ID:11105 IpLen:20 DgmLen:40 DF

***A**** Seq: 0x7BA91B0C Ack: 0x1B2A8E67 Win: 0xF8DC TcpLen: 20

04/09-12:00:05.994348 165.95.6.183:22 -> 165.95.7.5:13039
TCP TTL:64 TOS:0x10 ID:24469 IpLen:20 DgmLen:648 DF

***A**** Seq: 0x1B2A8E67 Ack: 0x7BA91B0C Win: 0x3B80 TcpLen: 20

04/09-12:00:06.177808 165.95.7.5:13039 -> 165.95.6.183:22
TCP TTL:125 TOS:0x0 ID:11106 IpLen:20 DgmLen:40 DF

***A**** Seq: 0x7BA91B0C Ack: 0x1B2A90C7 Win: 0xFD5C TcpLen: 20
The network packets captured using Snort can be implemented as a template using JESS. This can be done using the deftemplate construct. It is implemented as shown below:

(deftemplate packet

   "A network packet"

   (slot source_ipaddr) ; IP Address of the source
   (slot source_port) ; Port Number of the source
   (slot direction) ; Direction of the flow of the packet
   (slot destination_ipaddr) ; IP Address of the destination
   (slot destination_port) ; Port Number of the destination
   (slot protocol) ; Protocol followed by the packet
   (slot TTL) ; Time To Live value of the packet
   (slot TOS) ; Type Of Service value of the packet
   (slot ID) ; Fragment Identification Number of the packet
   (slot IpLen) ; Length of the IP field
   (slot DgmLen) ; Length of the datagram
   (slot Seq) ; Value of Sequence Number of the packet
   (slot Ack) ; Value of Acknowledgement Number of the packet
   (slot content) ; ASCII or Binary data in the packet
)

The keyword `deftemplate` is used to define the template with the name `packet` and the description "A network packet". The different portions of the packet header are represented in the form of slots and their descriptions are provided in comments.

### 3.2.2 Writing the rules

The rules in SNIDJ are written using the `defrule` construct provided in JESS. This rule is used to define format. The content for these rules is obtained from the source code of Snort.

For example, the following Snort rule sends an alert message with message “Porn word matched”, on all IP packets starting from any source address on any port number and flowing towards a destination with address 192.168.1.0/24 and any port number, whose payload contains the word “porn”

```
alert ip any any -> 192.168.1.0/24 any (content: "porn"); msg: "Porn word matched").
```

The above rule can be redefined in JESS using the `defrule` construct as shown below:

```
(defrule rule1 "porn rule"

  (packet (protocol ip)
    (source_ipaddr any)
    (source_port any)
    (direction ->)
    (destination_ipaddr 192.168.1.0/24)
    (destination_port any)
    (content "porn"))
```
(printout t "Alert, "Porn word matched")
)

This rule makes use of the packet template as defined earlier.

3.2.3 Writing the facts

The facts in SNIDJ represent all the conditions that determine which rule is to be fired. The facts that can fire the rule as described in the previous section are coded as:
(packet (protocol ip)(source_ipaddr any)(source_port any)(direction ->)
(destination_ipaddr 192.168.1.0/24)(destination_port any)(content "porn")

The set of all the facts captured from the network are stored in a separate file and given as input to the SNIDJ expert system.

3.2.4 Matching rules and facts

SNIDJ has a fixed set of rules, representing different types of attacks, while the facts, representing the network traffic, keep changing frequently. These rules and facts are supplied to the JESS rule engine that performs pattern matching using a very efficient algorithm called the Rete algorithm. The Rete algorithm is implemented by building a network of nodes, each of which represents one or more tests found on a rule LHS. The network traffic as represented by facts is processed by this network of nodes. At the bottom of the network are nodes representing individual SNIDJ rules. When a set of facts filters all the way down to the bottom of the network, it has passed all the tests on the LHS of a particular rule. The RHS of the associated rule is then executed.
4. EVALUATION AND RESULTS

4.1 Running SNIDJ

Signature-based Network Intrusion Detection System using JESS (SNIDJ) is essentially implemented with two files, \textit{snidj.clp} and \textit{attacks.dat}. The file \textit{snidj.clp} contains the data-structures in the form of templates and rules written using the constructs provided by JESS. The file \textit{attacks.dat} is a data file containing the facts. The rules from \textit{snidj.clp} and facts from \textit{attacks.dat} are loaded into the JESS Rule Engine. When the system is run, all the rules for which the matching facts are found will be executed and the corresponding alerts are sent to the Administrator.

Execution of SNIDJ takes place in three steps. The first step is to load the rules file \textit{snidj.clp} in JESS rule engine as shown below:

\begin{verbatim}
Jess> (batch snidj/snidj.clp)
TRUE
\end{verbatim}

The second step involves loading the data file \textit{attacks.dat} into the JESS rule engine as shown below:

\begin{verbatim}
Jess> (load-facts snidj/attack.dat)
TRUE
\end{verbatim}

The third step involves execution of rules and facts as shown below:

\begin{verbatim}
Jess> (run)
\end{verbatim}

4.2 Testing the rules

The most important part in the development of SNIDJ is the process of writing the rules. The rules determine the efficiency and accuracy of the system. The better the rules, the better is the performance of SNIDJ. It is therefore extremely important to test each and every rule that is included in the rule definition file of SNIDJ. Improperly defined
rules can reduce the system performance drastically. In general, there are two methods for testing the rules. The first and the most difficult is conducting stress tests against the rules and system hardware. The second is testing the rule syntax. These are described in the following sub-sections. [Beale 2003]

4.2.1 Stress Tests

The goal of any stress test is to identify thresholds. In the case of SNIDJ, the stress test can be carried out under conditions when there is excess traffic over the network. In these conditions, some of the packets might go undetected. As a result the system might fail to detect potential intrusions. [Beale 2003]

4.2.2 Individual Rule Tests

Testing and interpreting each rule individually for proper syntax is one of the best and most accurate methods for testing rules. It is a cumbersome task while developing a full-fledged network intrusion detection system, because patterns corresponding to each individual attack should be entered. If a pattern matches against the rules in the rule base the appropriate alerts will be generated. [Beale 2003]

4.3 Output of SNIDJ

The output of SNIDJ is a set of messages that is generated upon the execution of rules. These messages give information about the intrusions taking place on the network and the user is expected to take appropriate action to against them. It also gives number of rules that are executed. The sample output format of SNIDJ is as shown below:

Jess> (run)
Alert, ATTACK RESPONSES id check returned root
Alert, DDOS TFN Probe
Alert, SNMP missing community string attempt
Alert, CHAT ICQ Access
Alert, BAD TRAFFIC tcp port 0 traffic
Alert, Virus-Snowwhite Trojan Incoming
Alert, IP Packet detected
5. FUTURE WORK

5.1 A Better Knowledge-base

In the development of SNIDJ, the Snort rules were used as the knowledge base for the signatures of attacks. These rules are based on the network traffic generated by Snort and contain attributes related to it. Effort should be made to find a better knowledge base that represents signatures of viruses that are accepted by a larger group of people.

5.2 An Inbuilt Packet-sniffing Tool

Snort was used as a packet-sniffing tool for capturing network traffic. In the future, effort should be made to develop a packet-sniffing tool that is part of SNIDJ. This tool should capture the network traffic and output it in the form that conforms to the data contained in the knowledge base.

5.3 An Effective Parsing Engine

While developing SNIDJ, the network traffic generated by Snort was manually converted into facts as represented in JESS. Effort should be made to develop an effective parsing engine that automatically interprets the network traffic and converts it into facts in a real-time environment.

5.4 A Hybrid System

SNIDJ is a signature-based intrusion detection tool that detects intrusions by matching signatures in the knowledge base with the facts in the working memory. As such this system is as good as the knowledge base of the stored signatures. In the future, effort should be made to develop a more sensitive anomaly based intrusion detection component that can be coupled with SNIDJ to produce a hybrid system. This system
would be more effective as it will have the advantages of both signature-based and anomaly-based intrusion detection systems.
6. CONCLUSION

Intrusion Detection is still a fledgling field of research. However, it is beginning to assume enormous importance in today's computing environment. The combination of facts such as the rapid growth of the Internet, the vast financial possibilities opening up in electronic trade, and the lack of truly secure systems make it an important and pertinent area of study.

The Signature based Intrusion Detection System using JESS (SNIDJ) is the first attempt towards developing a Network-based Intrusion Detection System using the Java Expert System Shell (JESS). About fifty different types of simulated attacks were tested against SNIDJ rules and the results were found to be correct.

SNIDJ may not be a full-fledged Intrusion Detection System due to its reliance on Snort and lack of required hardware infrastructure like setting up a large network with many hosts to provide various levels of network traffic; nevertheless, it will be the first step towards developing a powerful Intrusion Detection System that uses the best tools in existence today.
Acknowledgements

I would like to express my deepest gratitude to Dr. Mario Al. Garcia, Assistant Professor of Computer Science, Texas A&M University – Corpus Christi, for his expertise in the area of Expert Systems, excellent guidance and encouragement during the period of this project work.

My sincere thanks to Dr. John D. Fernandez, Assistant Professor of Computer Science, Texas A&M University – Corpus Christi, for his keen interest, unending support and warm wishes that provided me the much-needed motivation in completing my project.

My sincere thanks to Dr. Dulal Chandra Kar, Assistant Professor of Computer Science, Texas A&M University – Corpus Christi, for his vast knowledge and experience in the field of Network Security. His patience and timely help in answering my questions has been very useful.

And finally heartfelt gratitude to all the faculty and staff of the Department of Computing and Mathematical Sciences, my colleagues and friends for their moral support.
BIBLIOGRAPHY AND REFERENCES


Appendix A: Intrusion Detection Terminology

This section includes a brief explanation of some of the terms that are commonly used with Intrusion Detection Systems [Security Focus 2003].

1. Alert

An alert is a warning issued by the IDS to the system operator that an intrusion is taking place or being attempted. On detecting an intrusion, the IDS will alert the analyst using a variety of methods. If the console is local to the IDS the alert would normally appear on the monitor. The use of a warning sound can be used, though on a busy IDS this will soon be turned off. Alerts to a remote console can be sent using the vendor proprietary method (usually securely), SNMP (often insecurely), email, SMS/Pager, or any combination of these methods.

2. Anomaly

An anomaly based IDS will build a profile of the host or network activity over time. When an anomaly occurs, which is an event outside this profile, the IDS will alarm. i.e, when someone does something they have not done before.

3. Attacks

Attacks can be considered attempts to penetrate a system or to circumvent a system's security in order to gain information, modify information or disrupt the intended functioning of the targeted network or system. The following is a list and explanation of the most common types of Internet attacks that an IDS is set up to detect.

DOS - Denial Of Service attack

Rather than penetrating a systems security by hacking, a DOS attack will just take the
system out, denying the service to its user. The means of achieving this are varied from buffer overflows to flooding the systems resources.

**DDOS - Distributed Denial of Service**

A standard DOS attack, the type that use large quantities of data from a single host to a remote host, cannot deliver sufficient packets to achieve the desired result, therefore the attack will be launched from many dispersed hosts, hence the name DDOS.

**Smurfing**

A type of network security breach in which a network connected to the Internet is swamped with replies to ICMP echo (PING) requests. A smurf attacker sends PING requests to an Internet broadcast address. These are special addresses that broadcast all received messages to the hosts connected to the subnet.

**Trojans**

Trojans are referred to software that appears to be legitimate, but that actually contains hidden malicious software. When the legitimate program is run, the malicious software is installed, unknown to the user.

**IP Spoofing**

A technique used to gain unauthorized access to computers, whereby the intruder sends messages to a computer with an IP address indicating that the message is coming from a trusted host.

4. **CERT - Computer Emergency Response Team**

This term was chosen for the first Computer Emergency Response Team, founded at the Carnegie Mellon University, which responded to computer security incidents.
These days many organizations will have a CERT, a computer security incident handling team.

5. **CIDF - Common Intrusion Detection Framework**

The Common Intrusion Detection Framework (CIDF) was an effort to standardize intrusion detection to some degree by developing "protocols and application programming interfaces so that intrusion detection research projects can share information and resources and so that intrusion detection components can be reused in other systems".

6. **CIRT - Computer Incident Response Team**

The Computer Incident Response Team (CIRT) indicates the change in philosophy towards security occurrences. Whereas CERTs were initially targeted at specific computer emergencies, the term incident in CIRT indicates that while not all incidents are necessarily emergencies, all emergencies can be considered incidents.

7. **CISL - Common Intrusion Specification Language**

CISL is the language used for CIDF components to communicate with each other. As CIDF is an attempt to standardize protocols and interfaces, so CISL is an attempt to standardize the language of intrusion detection research.

8. **Exploits**

For every vulnerability there is an exploit, i.e. a mechanism by which to exploit the vulnerability. An exploit can be considered the means of taking advantage of the structural weakness of the vulnerability. In order to attack a system, a hacker 'exploits' vulnerabilities in the code.
9. False Negatives

A false negative occurs when an attack or an event is either not detected by the IDS or is considered harmless by the analyst.

10. False Positives

An event that is picked up by the IDS and declared an attack but is actually harmless.

11. Firewalls

A firewall is also known as a network security door. A firewall is not an IDS but their logs can provide valuable IDS information. A firewall works by blocking unwanted connections based on rules or criteria, such as source address, ports etc.

12. Honeypot

A honeypot is a system that can simulate one or many vulnerable hosts, providing an easy target for the hacker to attack.

13. Signatures

Signatures or rules describe an attack and this is what makes the IDS trigger on an event.


The purpose of the Intrusion Detection Working Group is to define data formats and exchange procedures for sharing information of interest to intrusion detection systems and response systems, and to management systems which may need to interact with them.
15. Application IDS

Application IDSs are aware of the intrusion signatures for specific applications, usually the more vulnerable applications such as Web servers, databases etc.

16. Consoles

In order to make IDS suitable for the corporate environment, the dispersed IDS agents need to report to a central console.

17. Personal Firewall

Personal firewalls sit on individual systems and prevent unwanted connections, incoming or outgoing. While not infallible, they are very effective in protecting hosts from attack.
Appendix B: Default Snort Rules

Snort comes with a rich set of rules. These rules are divided into different files.

Each file represents one class of rules. The source code distribution of Snort has the following set of rules:

- attack-responses rules
- backdoor rules
- bad-traffic rules
- chat rules
- ddos rules
- deleted rules
- dns rules
- dos rules
- experimental rules
- exploit rules
- finger rules
- ftp rules
- icmp-info rules
- icmp rules
- imap rules
- info rules
- local rules
- misc rules
- multimedia rules
- mysql rules
- netbios rules
- nntp rules
- oracle rules
- other-ids rules
- p2p rules
- policy rules
- pop3 rules
- porn rules
- rpc rules
- rservices rules
- scan rules
- shellcode rules
- smtp rules
- snmp rules
- sql rules
- telnet rules
- tftp rules
- virus rules
- web-attacks rules
- web-cgi rules
- web-client rules
- web-coldfusion rules
- web-frontpage rules
- web-iis rules
- web-misc rules
- web-php rules
- x11 rules
Appendix C: Program Listings

;; Signature-based Network Intrusion Detection System using JESS (SNIDJ)
;; SNIDJ is a network intrusion detection system built with expert system
;; technology using Java Expert System Shell (JESS).
;; Name: Aijaz Ahmed
;; Degree: Master of Science
;; Major: Computer Science
;; Semester: Spring 2004

(defun program-name (snidj.clp)
  ;; Program Name: snidj.clp
  ;; Description: This file contains rules and templates that define
  ;; different types of attacks in a network.

  ;; The packet template contains information about the different fields
  ;; in the packet header as generated by the network traffic.

  (deftemplate packet
    "A network packet"
    (slot protocol) ; Protocol of the packet
    (slot source_ipaddr) ; IP Address of the source
    (slot source_port) ; Port Number of the source
    (slot direction) ; Direction of flow of the network traffic
    (slot destination_ipaddr) ; IP Address of the destination
    (slot destination_port) ; Port Number of the destination
    (slot TTL) ; Time To Live value of the packet
    (slot TOS) ; Type of Service value of the packet
    (slot ID) ; Fragment Identification Number of the packet
    (slot IpLen) ; Length of the IP packet
    (slot DgmLen) ; Length of the datagram
    (slot Seq) ; Value of the Sequence Number
    (slot Ack) ; Acknowledgement Number value
    (slot TcpLen) ; Length of the TCP packet
    (slot content) ; Payload content of the packet
  )
)
The rules form the knowledge base of SNIDJ.
Each rule identifies the signature for a particular intrusion.
Rules are fired when the facts from the working memory match them.

---

; ;; VIRUS RULES
; ;;---------------------------------------------

(defrule rule1 "Snowwhite Trojan"
  (packet (protocol tcp)
    (source_ipaddr any)
    (source_port 110)
    (direction ->)
    (destination_ipaddr any)
    (destination_port any)
    (content "Suddently"))
=>
  (printout t "Alert, Virus-Snowwhite Trojan Incoming" crlf)
)

(defrule rule2 "pif Worm"
  (packet (protocol tcp)
    (source_ipaddr any)
    (source_port 110)
    (direction ->)
    (destination_ipaddr any)
    (destination_port any)
    (content ".pif")

=>
  (printout t "Alert, Virus - Possible pif Worm" crlf)
)
(defrule rule3 "NAVIDAD Worm"
  (packet (protocol tcp)
    (source_ipaddr any)
    (source_port 110)
    (direction ->)
    (destination_ipaddr any)
    (destination_port any)
    (content "NAVIDAD.EXE")

=>
  (printout t "Alert, Virus - Possible NAVIDAD Worm" crlf)
)
;; BAD TRAFFIC RULES

(defvar rule4 "bad traffic rule"
  (packet (protocol tcp)
    (source_ipaddr $HOME_NET)
    (source_port 0)
    (direction <>)
    (destination_ipaddr $EXTERNAL_NET)
    (destination_port any))
=>
  (printout t "Alert, BAD TRAFFIC tcp port 0 traffic" crlf)
)

(defvar rule5 "bad traffic rule"
  (packet (protocol udp)
    (source_ipaddr $EXTERNAL_NET)
    (source_port any)
    (direction <>)
    (destination_ipaddr $HOME_NET)
    (destination_port 0))
=>
  (printout t "Alert, BAD TRAFFIC udp port 0 traffic" crlf)
)

(defvar rule6 "bad traffic rule"
  (packet (protocol tcp)
    (source_ipaddr $EXTERNAL_NET)
    (source_port any)
    (direction ->)
    (destination_ipaddr $HOME_NET)
    (destination_port any))
=>
  (printout t "Alert, BAD TRAFFIC data in TCP SYN packet" crlf)
)

;; CHAT RULES
;; These signatures look for people using various types of chat programs
;; (Eg: AIM, ICQ, IRC) which may be against corporate policy.

(defvar rule7 "chat rule"
  (packet (protocol tcp)