Using the JESS Expert system tool to implement an Online Intrusion Detection System based on Snort Rules

GRADUATE PROJECT TECHNICAL REPORT

Submitted to the Faculty of
the Department of Computing Sciences
Texas A&M University-Corpus Christi
Corpus Christi, TX

in Partial Fulfillment of the Requirements for the Degree of
Master of Science in Computer Science

by

Chaitanya Chinthireddy
Spring 2011

Committee Members

Dr. Mario Garcia
Committee Chairperson

Dr. Longzhuang Li
Committee Member

Dr. David Thomas
Committee Member
ABSTRACT

Security is one of the most important and inevitable modules in any modern world systems. Any security system should be able to find all the threats possible to a system and should be able to provide security and safety to the system which can vary from a small indication to quarantining all security threats found. This security can be against the intrusions in a network. To develop a system that provides this kind of security against the intrusions in a network through indication messages and also storing these indications in a database is the main theme of this project. This is done by implementing various signatures developed by Snort also known as Snort Rules in Java based software (JESS). Implementation in JESS gives the advantage of platform independency because of which this can be used on any system making it platform independent. Storage of the results or indications is enabled to do a detailed analysis and observations on the kind of attacks happening in the network offline also.
# TABLE OF CONTENTS

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

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

List of Figures .......................................................................................................................................... v

1. Background and Rationale ............................................................................................................... 1
   1.1 Snort Rules .................................................................................................................................. 1
   1.2 Need for Implementing Snort rules in JESS .............................................................................. 2
      1.2.1 Platform Independency ........................................................................................................ 2
      1.2.2 Rete Rate Matching Algorithm .......................................................................................... 3
   1.3 Related Work of Others .............................................................................................................. 3
      1.3.1 Signature-based Network Intrusion Detection System using JESS ................................. 3
      1.3.2 Two stage decomposition of Snort rules towards efficient hardware
           implementation .......................................................................................................................... 4
      1.3.3 A Framework for Rule Processing in Reconfigurable Network
           systems .................................................................................................................................... 5
      1.3.4 Priority based high speed intelligent rule checking ............................................................. 5

2. Narrative .......................................................................................................................................... 7
   2.1 Expert systems ............................................................................................................................ 7
      2.1.1 Applications of Expert systems .......................................................................................... 7
   2.2 Intrusion Detection Systems ...................................................................................................... 9
   2.3 Snort rules .................................................................................................................................. 10
   2.4 Java a platform Independent language ...................................................................................... 10
   2.5 Implementation of JESS Rules with Java .................................................................................. 11
2.6 Database Implementation.........................................................................................13

3. Developed Research......................................................................................................16

3.1 Code Design..............................................................................................................16

3.1.1 Structure of Snort rule.............................................................................................16

3.1.1.1 The Rule Header ....................................................................................................16

3.1.1.2 Rule Body .............................................................................................................18

3.1.1.3 Implementation using JESS ..................................................................................18

3.1.1.4 Generation of Facts ..............................................................................................20

3.1.2 Embedding JESS program into Java language ......................................................22

3.1.3 Implementation of the database in Java ...............................................................24

4. Testing and Evaluation .................................................................................................27

5. Conclusion and Future Work ......................................................................................41

Bibliography and References............................................................................................42

Appendix A: Types of Snort rules .....................................................................................44

Appendix B: Program Listings .........................................................................................46
LIST OF FIGURES

Figure 3.1. Structure of Snort rule .................................................................17
Figure 3.2. Structure of Snort rule template ....................................................19
Figure 3.3. Output of C program to generate facts ............................................21
Figure 3.4. Facts file and the structure.............................................................22
Figure 3.5. Java code when JESS is embedded in Java .....................................23
Figure 3.6. Run Java program .........................................................................24
Figure 3.7. Showing the output of Java code ....................................................25
Figure 3.8. Showing the database storage of output .........................................26
Figure 4.1. Output obtained from JESS rule engine using rules and facts file ........28
Figure 4.2. Output obtained from Java code with the input taken as output file from C
             program ..................................................................................................29
Figure 4.3. Loading of implemented rules file that contains only one rule ..........31
Figure 4.4. Loading of Facts file into JESS .......................................................32
Figure 4.5. Output display of rule engine when rules are fired .......................33
Figure 4.6. Output of fired rules that contains one rule of each type ...............34
Figure 4.7. Display of rules fired when rules and facts file are given as input ....35
Figure 4.8. C program which takes AllRules.clp as input and gives Facts.dat as
             output ......................................................................................................37
Figure 4.9. Depicts the Java code when embedded with JESS rule engine ..........37
Figure 4.10. Fired rules displayed in console tab .............................................38
Figure 4.11. Output file generated when the system is run continuously ..........39
1. BACKGROUND AND RATIONALE

As the modern world is fast growing, everything and anything can happen at the speed of a mouse click. In the present world where every small task depends on the computers and their networks the need to protect data and other information is always important. With the ever growing technology there is always a threat in the form of intrusions to a system and important data. These types of intrusions can be coming from either internet or in any other form. The intrusions from the internet that occur typically are the data packets and these are then transferred from the internet to systems.

The main purpose to develop an overall system that can give us information about various types of attacks in a system is of much importance in the current day world. Current systems typically concentrate only on a specific type of protection that can be given to a system.

Security system developed in this project can be used on any platform thereby are always of a great advantage in the present day. Such systems are in great demand due to less overhead it generates when deployed on a different platform than the one on which it is developed on.

1.1 Snort Rules

In order to provide security to any system the first pre-requisite is to know the behavior of the security threat. The security threats from the internet in any modern system arise from the data packets that are transferred or exchanged between the host and the server. These data packets which pose a threat to the systems security follow a certain configuration or pattern. All these particular patterns are written and stored in the form of rules for each and every intrusion. These rules are written and tested by the Vulnerability Research Team of Snort organization.

Rules give us the required data about the intrusions that are possible in a system. When these rules are implemented they give us the information about the attack and the actions needed.
1.2 Need for Implementing Snort Rules in JESS

Software which is implemented to describe rules in Snort plays a very important role on the total functionality of the system itself. The kind of support software gives for its system is tremendous and if not properly chosen then even the best system takes form of an average system.

JESS is a Java based rule matching engine. JESS software is used for the implementation of the Snort rules to provide a system that is capable of detecting the intrusions on a network. The selection of this software is mainly done because of various reasons like,

1.2.1 Platform Independency

JESS which is basically a rule based engine developed on the Java platform has all the significant features of Java. System specific libraries that provide the required programming language support for any software to develop a complete system is not available in Java. Due to this reason, Java is also called a platform independent language. It reduces the overhead of either inclusion of required libraries for the developed system to run, or to recompile the software every time there is a change in the system that uses the software. All these benefits that come with Java are also present even in JESS because of which the system becomes a platform independent system.

1.2.2 Rete Rate Matching Algorithm

JESS is a rule based program or software that applies a certain if – else statements on a set of given data. Rule matching algorithms are generally inefficient in terms of the memory usage because of the time taken by them to implement the rules and these disadvantages can be eliminated by the Rete algorithm. Rete algorithm in JESS can be implemented on the basis of
nodes. This can be accomplished by developing a network of nodes, signifying one or more tests that are found on a rule in the Left Hand Side (LHS). The facts or data from the system being tested are eliminated from the working memory and then sent across the network for processing. The nodes present at the end of the network are the rules individually written in JESS. Rules are said to be activated if they pass all the required tests that are available on the LHS of a particular rule. The activated rule may be associated with an RHS which is executed or fired. Rete algorithm makes JESS one of the fastest rule engines available today.

1.3 Related Work of Others

The current systems that are presently in use are either system specific or do not provide security to the entire system. This section details about the various works that are done for making a complete system which can be used on any platform and gives the advantage of performing a detailed analysis of the system attacks.

1.3.1 Signature-based Network Intrusion Detection System using JESS

By using the Signature based Intrusion Detection System, Network based intrusion detection system is developed by using JESS. In this project about fifty different types of rules are implemented. From these fifty types of rules there are about thousand rules that implemented as each and every type of rules contain about thirty to forty different rules that explain the attacks. For these rules, facts are generated and they are used for the simulation purpose for detecting intrusions that are available in the network [Ahmed 2004].

In this current project, there are about forty one types of rules and each type is again divided into 100 rules. There are about many rules that are deleted, modified and added newly to
the snort database. Hence there are about 2595 rules to be implemented. For these rules, facts are implemented which represent the data packets in the network. The fired rules when run are stored in the database for future analysis.

1.3.2 Two-stage decomposition of SNORT rules towards efficient hardware implementation

In the project security to a system is provided by using SNORT rules. The normal and direct implementation of Snort rules in the hardware is an inefficient method. An efficient method of implementation is to break down the Snort rules into simpler signature patterns and then to implement them in the hardware and then the smaller patterns are stored. The advantage of using this method in the project is easier implementation in the hardware along with dynamism. Smaller signature patterns enable high scalability of the matching operation and simplify the online updation of the rules. [Chen 2009]

This project involves more production cost of the hardware components and their design and is very negligible when done by using software tools. By making use of software tools better algorithms can be designed which help to gain the required speed but implementation of all various rules may not be possible in such a system.

The current project because of its implementation in software makes it cost efficient and also it can also be deployed on any system that needs security. Apart from this, the choice of the JESS software which uses a highly efficient algorithm (Rete) enables to achieve and maintain the speed within the software itself.
1.3.3 A Framework for Rule Processing in Reconfigurable Network Systems

This project provides the framework for the implementation of reconfigurable rule processing engine in the hardware. Various components of the Intrusion detection system are enabled in this framework to integrate for a reconfigurable system implemented in hardware. It has also been tested on a system that was developed on Xilinx FPGA hardware. [Attig 2005]

This project gives only the framework that can be used for the development of a security system in hardware. But, it is not a complete system which can be used directly on any machine. It involves an overhead of designing, developing and testing of a system that uses this framework and also makes use of more time and money, hence is not cost efficient. Implementation of these intrusion detection systems in hardware has another disadvantage of being system specific i.e; the overhead needed or required to move intrusion detection system from one platform to another is very high. In some cases entire system might be needed to change for making the system to work with this hardware.

These limitations on the system and its hardware don’t exist if the system is developed in software and also the JESS based intrusion detection system makes it platform independent.

1.3.4 Priority-based high-speed intelligent rule-checking

Another method of implementing the rule checking algorithm for intrusion detection system is the priority based algorithm. In priority based algorithm, priorities are assigned to the packets in order to increase the probability of malicious packets detection during high speed data transfers. It is done by Intelligent String Arrangement (ISA) and using this ISA method gives us a higher probability of the malicious packets being detected even at higher data rates or throughputs. At higher data rates the system tends to drop packets which it is not being able to
process and hence there is a chance for the intrusion packet to be dropped. This kind of an issue with the systems is eliminated by assigning priorities to the packets. [Sachidanada 2006]

Assigning priorities to the packets will enable the processing of high priority packets at the cost of low priority packets. This makes the system vulnerable to risks or intrusions which are given lower priorities. Hence the system is not completely secure but still is prone to some kind of risks always.

Packets are dropped because of slow processing time and the need to display and process the results immediately at high input data speeds. The processing of the results can be made independent. Bringing in such kind of independency enables the system to be able to process the data at higher data rates. Due to this offline analysis, the independency of the results processing is brought in by writing the fired or matched rules onto a file. Storage of the results helps us in processing the results at a later point of time offline also.
2. NARRATIVE

To develop a security or intrusion detection system that can be platform independent i.e. being able to run on any system, can detect any type of intrusions in the system or the network, and can be able to store the results of the intrusions detected in a file or a database so that a detailed analysis can be done for the future comparisons or for any further actions is the main theme of the project.

2.1 Expert systems

Any system that can help a human in accomplishing a task is called an Expert system. These systems are developed mainly to help a human expert in his aid. They are developed using the human expertise for problem solving. Expert systems have the capability of learning new software’s or do not have the capability.

2.1.1 Applications of Expert systems

Expert systems are widely used in order to make the work of the human easier. Major areas of the applications of Expert systems come into picture where huge volumes of data are involved. Some of the fields where huge volumes of data are involved are in the analysis of the climate, space etc.

Example of an Expert system that is already in use is SCREENER. The Expert system screens various projects for the environmental impacts. Here user has the option to select from a menu both the proposed activity and the environmental features where the activity will be taking place. The program takes in the information and refers to its knowledge and rule bases to give a code describing the potential impact of this activity in the designated environment (1 - no impact; 2 - insignificant impacts; 3 - mitigable impacts; 4 - unknown impacts; 5 - significant
impact). Based on the code generated, SCREENER recommends what level of environmental impact assessment is required for the project. SCREENER also provides some suggested mitigation measures that can help prevent or reduce environmental impacts for this project or activity [EIA 1997].

Another example of an Expert system is ORBI. This system was written in prolog with a natural language interface. It helps the user in giving the required inputs in the natural language rather than in the computer coded languages. This kind of feature is present in very few Expert systems that are currently available. ORBI can help in various fields including geology, hydrology, ecology, and microclimate. The system can make judgments about the suitability of an area for uses such as industry, agriculture, and recreation. Various levels of values are obtained via inference of the rules to indicate the suitability of environmental assets in the area of interest relative to requirements of the various development activities that are planned. It can also give the results graphically in the form of maps, via a plotter [EIA 1997].

Another Expert system in the field of environment is IMPACT. This system is currently in use in the U.S. Department of Energy’s Savannah River Site in South Carolina. It assesses several types of affects like air pollution, water pollution and ground water pollution using a simplified GIS method. It does the screening for possible impacts. This program gives out the results in the form of tabular data or report [EIA 1997].
2.2 Intrusion Detection Systems

These systems provide security by detecting the intrusions and are known as Intrusion Detection Systems. These kind of systems are always needed as they make a system secure either by giving a notification or message to the user or by quarantining the security threat.

The kind of security that is being talked about is an intrusion detection system that detects any kind of intrusions to the system from a network to the user indicating the detection in the form of messages. Such systems are mainly designed to detect intrusions that might have entered the system breaking through the firewall or other inbuilt securities. These systems can be of two types one that can catch an attempt to break in that is in progress, or those that can detect the intrusions after break-in. In the latter case, one might not be sure of the amount of damage that is already been done to the system but at least information about the kind of problem can be present. Again there are different types of intrusion detection systems one that has the capability to protect networks, and those that can protect individual host machines. For the host based systems the detection can be done by using continuous monitoring on the file systems that detect any changes being done. These files should not be modified expect by the user and any kind of intrusion that gets into the system first tries to make changes to the system files so that a kind of a backdoor activities can be created like attacking another system.[ Hal 2001].

Intrusion detection systems are divided into two basic categories: signature-based intrusion detection systems and anomaly detection systems. Attacks have signatures, like computer viruses, that can be detected using software and by trying to find data packets that contain any known intrusion-related signatures or anomalies related to Internet protocols. Based upon a set of signatures and rules, the detection system is able to find and log
suspicious activity and generate alerts. Anomaly-based intrusion detection usually depends on packet anomalies present in protocol header parts. In some cases these methods produce better results compared to signature-based IDS. Usually an intrusion detection system captures data from the network and applies its rules to that data or detects anomalies in it. [Rehman 2003].

2.3 Snort Rules

Snort is an open source software that detects the network intrusions. Different kinds of intrusions to a network or system can be detected by the information that is sent from the server to the client or from the client to the server. The information is sent in the form of packets from one machine to another, follows certain pattern also called as signatures, basing on which if the packet sent is a type of intrusion or a normal network packet [bovee 2001]. This software uses the characteristic features or signatures of the packets which are meant to harm the system or intrude the network to define the rules. These signatures or attacks may be present in the header part of a packet or in the payload. These rules are classified into different types basing on the kind of intrusion or attack. There are about fifty to hundred varieties of rules defined by Snort to give the user information about kind of intrusions that is being done. 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.

2.4 Java a platform Independent language

In order to implement a platform independent system, Platform Independent Languages must be known and information about these languages should also be known.

Platform Independent Language has 2 main uses:

- As a target language for compilers to abstract from specific Software Platforms.
As an *implementation* language for libraries/applications that need to be run on different platforms. Implementation language works best for applications that are relatively independent on platform-specific libraries and frameworks. The best examples are the web pages which can be loaded onto any system irrespective of its platform.

**Compiler use:** In order to generate System for multiple platforms, multiple compiler back-ends have to be maintained. Platform Independent Language (PIL) allows generating the code in one language to be translated into generated PIL code for any of its supported platform, simplifying the maintenance of multiple compiler back-ends.

### 2.5 Implementation of JESS Rules with Java

When JESS rules are executed in JAVA, JESS library files are also to be loaded into the class path of Java. In JESS two JAR files (JESS.jar,jsr94.jar) are present. These two jar files should be included in order to execute a Java file which has JESS commands embedded in it [Tim 2010].

Instance of JESS Rule engine will be created in Java code. Instance of JESS can then be reused to process each order. (Catalog data is loaded only once; JESS will index it and later accesses will be fast.). JESS Rule Engine is having Special API to execute rules [Corazza 2006].

For Example:

```java
Rete engine = new Rete();
Engine.batch("backdoorrules.clp");
Engine.run();
```
Instance of Rete engine is created by using Ret engine = new Rete(). By using batch function all the rules from backdoorrules.clp will be loaded into local memory and when run function is given to the JESS engine, all the rules will be set to be activated.

Platform Independence can be provided by using the API’s of JESS Rule Engine by executing these rules in Java.

JESS API has special functions which fire rules.

For Example:

```java
Rete engine = new Rete();
engine.executeCommand("(batch "c:files /allrules.clp")");
engine.executeCommand("(load-facts "c:files/facts.txt")");
engine.run();
```

Rete engine = new Rete() is an instance of Rete engine by which JESS is embedded with Java. By using the execute command the path of rules.clp and facts.dat files is given. The files will be loaded into the local memory and when the run command is given then all the rules matching the facts will be fired.

The facts.txt and allrules.clp should reside in the same folder where Java code will be saved or present. If files are saved in different folders then paths need to be mentioned for proper execution. When Java code is made to run it will execute all the fired rules and print them into console tab.

Implementing JESS with JAVA coding will provide a Platform Independent System which can be executed or run on any platform. Each JESS.Rete (Rule engine) object represents an independent engine. A single program can include any number of engines and each individual
Rete objects can have their own working memories, agendas, and rulebases, and all can function in separate threads. Also multiple identical engines can be used in a pool [Proctor 2010].

2.6 Database Implementation

After implementing JESS in Java results have to be saved with date and time so that these can be used for future references. For implementing JESS, the Java functions are to be written and these functions are called from rules file [Manning 2003].

If a rule is fired then that particular rule will invoke Java function and Java function will write the result with time and date into the database which is a file in this case. In order for the fired rules to be loaded into the database, system time and date are needed to ensure to know when the intrusions has been detected. This helps for easily identifying or analyzing the output. For this to happen a database has to be created which is a text file in this project and the fired rules are written into it.

Getting System time:

Java class has been declared to get the system timing when the rule was fired.

This class will import the Date from Java.util, and date format from Java.text packages.

It uses Java API to get the system timing and using the dateFormat, wanted date format can be achieved.

For Example:

```java
String getDateTime() {
    DateFormat dateFormat = new SimpleDateFormat("yyyy/MM/dd HH:mm:ss");
    Date date = new Date();
    return dateFormat.format(date);
}
```
getDateTime function is used to get the system date and time. DateFormat class is taken from Java.text packages for choosing the format of how it should be printed in the output file. Instance of the date is created for capturing the system date. The date is returned in order for printing it in output file which is database in this project. Next it has to be written into a file (Database).

Writing Result into file:

Using Java APIs, results are written to a file.

Print Writer option can be found in Java which is imported from Java.io package.

Using the File Writer class, the path of the file can be mentioned correctly.

When file writer is declared, it should be inside the try-catch block. Try catch block will be used to avoid Errors. If the specified path is wrong then catch block will be executed. And this file writer has Constructor as shown.

FileWriter("file",True);

Which tells the file can be modified.

After doing all these steps results are written into database which is a file in this project.

For Example:

PrintWriter pw=null;

try {
    String filename = path;
    pw = new PrintWriter(new FileWriter(filename, true));
    String s=new ExTime().getDateTime();
    pw.print(s);
pw.print(message);

} catch (IOException ex) {

}

First, PrintWriter has to be assigned with null value. All the messages have to be printed in try catch block. If FileWriter value is true then try block will be executed otherwise catch block has to be executed. PrintWriter class is used writing the output onto the console tab.
3. DEVELOPED RESEARCH

All the required rules are taken from the Snort libraries and are implemented in JESS language. To run these set of rules another set of data is needed called the facts file. These facts stimulate or activate the rules and then the corresponding message will be printed out for the user. When both the files are run together at the same time by using run function, then the accomplished rules are fired.

3.1 Code Design

Coding for the entire system is done in three parts.

1. Implementation of the Snort rules in JESS language.
2. Embedding these JESS implemented rules in Java
3. Implementation of the database part to store the results using Java.

In order to implement the rules in JESS, the structure of the Snort rules must be known.

3.1.1 Structure of Snort rule:

A Snort Rule is made up of two components, the Rule Header and the Rule Body, as shown in Figure 3.1.

3.1.1.1 The Rule Header

The Rule Header is divided in four main categories that are described as follows [Marty 2010]:

Rule Actions: The first section of Snort rule involves rule which depicts the actions needed to be taken if the rule conditions are met. There are three predefined actions are described as follows:
**Pass:** The second section of snort rule involves ignoring the packets. This process plays a prominent role in speeding up snort process in certain cases such as if some packets cannot be applied.

![Figure 3.1 Structure of a Snort Rule](image)

**Figure 3.1** Structure of a Snort Rule [Marty 2010]

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

**Alert:** This type of action is mainly used in sending an alert message in cases where rule conditions for a certain packet are true.

**Protocols:** This is the second part of the Snort rule. This part depicts the type of packet the rule needs to be applied. Snort presently recognizes 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 is originated. It has two parts, the IP address of the source computer and the port number of the source computer. Keyword ‘any’ may be applied to a rule on all addresses and can also be used to be apply on entire packets regardless of the port number.

Destination Information: This gives the information of the computer to which the packet is flowing. It has two parts, the IP address of the destination computer and the port number of the destination computer

3.1.1.2 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 options 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.

3.1.1.3 Implementation using JESS

Implementation of the rules in JESS includes two parts.

1. Defining the rules in JESS corresponding to the rules that are given in the Snort rules data base.

2. Giving the system facts which represent the packets from the network.

For defining the rules in JESS the structure of the packet that is transferred in the network is to be defined in the form of template. This template consists of all the possible information that a normal data packet in the network will have. The information about the various data that a packet can have is defined in JESS as a slot. So the template of the packet consists of the slots that will be used or present. For example a template for fact person could be defined as:
(deftemplate person “A person template”
(slot name)
(slot age)
(slot height)
(slot weight)
)

Slots can be specified as single values or multi values by placing the keyword multi slot instead of slot. The template used for this project is as shown in the figure 3.2.

Once the template is defined, the system needs to be informed with various types of rules that are defined by Snort. A rule in JESS is defined as a set of conditional elements and a set of actions. If there is a matching in all the conditional elements of the Left-Hand Side (LHS), the rule is placed in the local memory. When the inference engine selects a rule for firing from the local memory, the Right-Hand Side (RHS) is executed.

Figure 3.2 Structure of Snort rule template
A rule in JESS has the following syntax:

```
(defrule <rule name> [<comment>]
  <patterns>* ; Left-Hand Side (LHS) of the rule
  =>
  <actions>* ; Right-Hand Side (RHS) of the rule
)
```

For example

```
(defrule rule1 "Health"
  (person (name xyz)
   (age 23)
   (height 5-3)
   (weight 35))
  =>
  (printout t "Person is underweight" crlf)
)
```

Now coming to the second part, JESS software is provided with the facts. Facts are nothing but a “chunk” of data. Facts consist of a relation name (a string) followed by zero or more slots (also strings) and their associated values. So in accordance to this system fact consists of the data that is carried by a packet. When these facts along with the rules are run then the system behaves as if a new data packet is received and generates the output messages in the format and gives an indication to the end user.

**3.1.1.4 Generation of Facts**

JESS implementation of the Snort rules is done using the coding method defined above using the format of the Snort rule. Now to check if there is any attack happening in the system, JESS rule engine needs to be fed with the data packets that are transferred through the network.
The data packets that are fed into the program are called Facts. Currently in this program these facts are generated to simulate the data packets. These facts are generated using a C program written for the same. The output of the C program is as shown in figure 3.3.

The C program takes various rules from Snort as input and generates a data packet from these rules and writes them into a file. This generates the facts by taking all the rules sequentially from a given start number of the rule to an end rule number.

Figure 3.3 show output of C program to generate facts.

Once the data packet is formed in the required format these are written into a file whose name and location are taken at the beginning of the execution of this C file. This facts file is currently given as an input to the system to run the JESS implemented Snort rules. Figure 3.4 shows the facts file that is generated.

Figure 3.4 shows the structure of the facts file. This fact file will be used to fire the rules.
3.1.2 Embedding JESS program into Java Language

This is the second stage of the implementation where the implemented Snort rules in JESS are embedded into a Java code which would invoke the JESS rule engine. The design is chosen as this kind provides many benefits. When the JESS code is embedded in Java all the various advantages of the Java software will be present for making the system robust, platform independent [Netbeans 2011]. Using Java as the master software helps us in implementing any GUI’s for future uses. While the same can be done even in JESS but it is easier to be done in Java than in JESS. Figure 3.5 shows the Java code while Figure 3.6 shows how to run the Java program.

Figure 3.5 shows the Java code which embeds with JESS. ExMyFilter is present in Java class Java class creates instance for Rete.
Figure 3.5 Java code when JESS is embedded in Java

Using the instance of the Rete, rules file(.clp) will be loaded from local memory using `executeCommand` function which is present in API. Path of the rules fired can also be abstracted.

In the same instance of Rete, facts (.dat) will be loaded from local memory. The `run` function has to be called on instance of Rete to fire the rules.

Figure 3.6 Shows the Java program which has to run using the Run file command in the NetBeans IDE. While running the code, it will get executed line by line and the results can be seen automatically in console tab of Netbeans.
3.1.3 Implementation of the data base in Java

Implementation of the database involves first selection of the type of database (file) that is needed for the current system and the method how it is to be implemented (Netbeans).

In the actual system where the data packets would be streaming in continuously storing the results into an online database is going to be a tough task or choice that is to be made. There are many problems that would arise if the database is not chosen properly. In this project the database chosen is a file into which the results are stored along with the time and date at which these rules are fired. Any data which is once written into a file is stored over there and will be present even when the system crashes all of a sudden. And also the size of the database needed for this system is an unknown parameter because the number of attacks or intrusions that can happen to a system cannot be assumed. Furthermore, no additional software is required to retrieve the data that is stored in a file.
According to the current project, rules that are run or fired in the JESS should be stored in a database (file) for future references. This is also implemented in Java and all the rules by default are fired and displayed in the console tab and these displayed results cannot be stored directly into any database. Figure 3.7 Output of Java code that contains the fired rules.

Figure 3.7 Shows build successful message that results after running the Java program. It will give the result as ‘Build Successful’ when it executed, this output will be saved in Database with time and date. And if any error is occured, it will give the line number of the Java code where error occurred and in Console tab it gives Build failed with Error.

Figure 3.7 showing the output of Java code

The figure 3.8 shows the output as stored in the database which is currently a file. Figure 3.8 shows the storing of results into a file. When a rule is fired it will call the Java code, which will get executed and thus saves the result into the file.
Figure 3.8 showing the database storage of output

Using the file Writer class in JAVMA API, file can be opened in append mode and using Print Writer class in Java API results can be written to a file. The result will be saved with time and date when the rule was fired.
4. TESTING AND EVALUATION

Testing is main phase of any project development and it gives an idea with regards to the project aim and its implementation. Testing, depending on the testing method employed, can be implemented at any time in the development process. However, most of the test effort occurs after the requirements have been defined and the coding process has been completed. The success or failure of any project depends on the testing done on that particular project.

Different types of testing needs to be done to verify various aspects of the code. These are mainly Functionality testing and System Testing.

Functional testing mainly focuses to check if the output obtained is as per the requirement or not. It mainly involves in the identification of functions that the software is expected to perform, the creation of input data based on the function's specifications, the determination of output based on the function's specifications, the execution of the test case, the comparison of actual and expected outputs. This testing can be done at various levels in the project and are Unit testing, Integration testing. Figure 4.1 depicts expected output. Expected output is obtained by running the program in command prompt by using JESS rule engine.
Figure 4.1 Output obtained from JESS rules engine using rules and facts file.

Figure 4.2 depicts the actual output file generated. Actual output is obtained from the Java code in which JESS is embedded into it and the input file for Java program is the output file generated from the C program which takes input as Allrules.clp file which contains all the rules.
By comparing both the outputs from Figure 4.1 and Figure 4.2 both the outputs seem to be same. Hence functional testing is satisfied.

Unit testing is the testing methodology used by which individual units of source code are tested to determine if they can be used. A unit is the smallest testable part of an application. The goal of unit testing is to isolate each part of the program and show that the individual parts are correct. Unit testing provides a strict, written contract that the piece of code must satisfy. As a result, it affords several benefits. Unit tests are done to find the problems early in the development cycle. Unit testing may reduce uncertainty in the units themselves and can be used in a bottom-up testing style approach. By testing the parts of a program first and then testing the sum of its parts, integration testing becomes much easier.

In this project Unit testing is done at each and every individual level of implementation of Snort rules in JESS so that the code is done according to the requirements. The implemented
rules were tested using the JESS command line to check for errors and after firing the rules using JESS command line the result was printed to the console tab according to the fired rule.

First only one rule is considered and tested by using the facts file to determine if the rule is fired or not. The rule is given as the input for the JESS engine by using the batch function.

The rule which is used for unit testing is taken from Attach Response rules in AllRules.clp file.

(defrule Rule1 "Attack Response Rules"

   (packet (protocol tcp)
      (source_ipaddr $HOME_NET)
      (source_port any)
      (direction ->)
      (destination_ipaddr $EXTERNAL_NET )
      (destination_port any)
      (content "Volume Serial Number")
      (ID 1292 )

    )

  =>

    (printout t "Alert, ATTACK-RESPONSES directory listing" crlf)

    )

By using this rule and the whole facts which is named as Facts.DAT are given as the input for JESS rule engine. The format of how the facts file looks file look like can be seen below.

(deffacts startup
   (assert(packet (protocol tcp)(source_ipaddr $HOME_NET)(source_port any)(direction- >)(destination_ipaddr $EXTERNAL_NET)(destination_port any) (content "Volume Serial Number") (ID 1292 )))

)
(assert (packet (protocol tcp) (source_ipaddr $HTTP_SERVERS) (source_port $HTTP_PORTS )(direction ->)(destination_ipaddr $EXTERNAL_NET )(destination_port any) (content "Command completed") (ID 494 )))

The output that is displayed is shown in Figure 4.2.

![Command Prompt - jess.bat](image)

Figure 4.3 loading of implemented rules file that contains only one rule.

Figure 4.3 shows the screen shot when all code containing the rules implementation is loaded onto the JESS CLI (Command Line). The command “batch” indicates that the file that are passing to the CLI is a part of the implementation and JESS loads all the rules present in this batch file will be stored in its local memory.
After loading the rules into the JESS, facts or data packets are needed for simulation to JESS so that it will load these facts also and then the system is able to run. Figure 4.4 shows the output “load-facts” command is executed in the CLI. This command is used to load the facts from a file into the JESS memory.

Run is the command that is used to run the system after the rules and the facts are loaded into the file using the two ‘batch’ and ‘load-facts’ commands. Once run command is given, all the rules that are matched in accordance with the facts or simulated data packets for matching the corresponding messages to be printed on the screen.
Figure 4.5 Output display of rule engine when rules are fired.

Figure 4.5 shows a picture of how the displayed or fired rules look like. The displayed number at the end of the alert messages indicates the total number of rules fired in the current execution of the program. [Emest 2008].

In the similar format, a single rule is taken from each type of rule and then put together in unit.clp file. This file and the facts file is given as the input from JESS rule engine for processing. When the rules are fired they are displayed in the command prompt, total number of rules that are taken as input is equal to the total number of rules that are fired as shown in figure 4.6.
Figure 4.6 Output of fired rules that contains one rule of each type.

When all rules are given as a single file and all facts are given as the input for JESS rule engine then the rules that are fired as output in command prompt is as shown in Figure 4.7.
Figure 4.7 Display of rules fired when rules and facts file are given as input.

JESS Code is embedded in Java to provide Platform independency so when running the main function of the Java program Instance of the Rete Engine will be executed and Rules and facts were supplied to the Rete Engine to execute JESS coding. At this stage it has to print the result to the Java console tab and all the fired rules should be printed.

When executing this command JESS coding has to call the Java functions to write the result to the file with date and time. All the individual Java modules coded were tested to give the 99% accuracy.
Integration testing involves combining all the software modules into a group and testing it as a group. This type of testing occurs before system testing and after unit testing. In integration testing the implemented rules and facts should be embedded in JAVA and it should be executed without errors but the current system includes the implementation of all the rules in JESS Software in which the process would be a very cumbersome method. Hence the rules are selected and passed it to the system that contains C simulation program and the system should be able to execute these rules and give us the exact output. It should be executed and all the fired rules should be printed on console tab. After this it writes the result to the file for future references along with date and time. In this Project two different Java classes were created and Unit Tested. These two files are to be integrated to write the result to the file. The Java code which writes the result to the file will call the date function which is present in the other Java code. This Java code will be invoked by JESS rules.

First, allrules.clp is given as an input for the C simulation program which is written mainly for the testing purpose. Reason for using the C program instead of using JESS software is mainly due to the integration testing part which tells about the functionality between two software’s. Hence by using the C program, Jess file such as Allrules.clp is given as input for C program and facts.dat file is generated as an output as shown in Figure 4.8.
Figure 4.8  C Program which takes Allrules.clp as input and gives facts.dat as output.

The facts file that is generated from the C program is given as an input for the Java code in which JESS I embedded in Java code as shown in Figure 4.9.

Figure 4.9 Depicts the Java code when embedded with JESS rule engine.
To know if the two software’s are compatible or not it has to be executed. If output is displayed then they both are compatible to each other. The output will be printed on the console tab itself. At the bottom of the screen it shows a message as “Build Successful” as shown in Figure 4.10. If there is any error then an error message will be displayed in the console tab itself.

Figure 4.10 Fired rules displayed in console tab.

When the total system is executed, the main system should execute the rules and the result of the fired rules will be stored in a Text File. Stability of the system can be known when it is ruined continuously which is done by System testing. Testing deployed on a complete, integrated system used to investigate the system’s compliance with specified requirements is known as System testing. The inputs to the system testing are integrated software components which passed integration testing. The main goal of integration testing is to identify any inconsistencies among the software units that are integrated together (called assemblages).
System testing is limited testing and identifies defects in the “inter-assemblages” and also within the system.

The output file that is generated when this system is run continuously is as shown in Figure 4.11.

![Figure 4.11 Output file generated when the system is run continuously.](image)

The system has a fixed set of rules, representing different types of attacks defined by Snort, while the facts, representing the network traffic, that keep changing frequently. These rules and facts were written in JESS and JESS code is embedded in Java program in order to provide platform independency. In Java program, instance of the JESS rule Engine is created and executed then Rules and facts are supplied to the JESS rule engine that performs pattern matching using a very efficient algorithm called the Rete algorithm [Emest 2008]. Rete algorithm is usually implemented by combining network of nodes that represent one or various tests on a rule LHS. The RHS of the associated rule is then executed. Then it will print the result
to the console tab and it will call Java function to write the result into a file. Then the testing is undergone for different sets of rules.

The rules determine the efficiency and accuracy of the system. Hence needs to be tested thoroughly. Testing of each and every rule defined is one of the best methods to ensure the system accuracy. But as the current system includes the implementation of all the rules in JESS software this process would be a very cumbersome method. Hence all rules are selected and passed it to the system and the system should be able to execute these rules and gives the exact output. This can be repeated for multiple times to make sure that the accuracy upto 99%. The results obtained are then stored in File for future comparisons to know if such types of attacks already exist using JAVA program.
5. CONCLUSION AND FUTURE WORK

At the end of development, this project is able to detect intrusions on any network in which it is deployed and it should display the required information to help the user know the type of risk the network is posing to the system. And also this system should be able to give all the information offline so that the intrusions that are more common to that network can be analyzed at any time. Testing is also done on the system by using the tools and the results obtained are stored in the database for future analysis. This output would be in the form of messages displayed on the screen for various rules that were fired. Rules are fired basing on the facts that are given to the system during testing phase. All these results are stored into the database. The storage of the results would be done along with information in the form of various columns like the attack occurred on the system, time at which it occurred, kind of intrusion that has happened, and date of intrusion. This helps the end user to analyze the attacks that happened to the system at any point of time. Apart from this any system is expected to be stable and also to be able to detect all the different kind of intrusions that happen in the network. The testing methodology makes sure that the system will be able to detect the threats present and also results are stored properly in the database.

Further advancements to this system are to implement a GUI mode where the user will be able to interact with the system directly. The system can be improved to give the results basing on the user requirement to ease the end user. Another development that can be possible is to enhance this system to quarantine or stop the intrusions along with the indication messages to the user. This will make the system even more robust and complete.
BIBLIOGRAPHY AND REFERENCES


Appendix A: Types of Snort rules

Each file that represents the type of rule is termed as a class. Various types of rules that are implemented in the current project are as mentioned as:

1. Attack Response Rules
2. Backdoor Rules
3. Bad Traffic Rules
4. Ddos Rules
5. Dns Rules
6. Dos Rules
7. Finger Rules
8. Ftp Rules
9. Icmp Info Rules
10. Icmp Rules
11. Imap Rules
12. Info Rules
13. Misc Rules
14. Multimedia Rules
15. MySql Rules
16. NetBios Rules
17. Nntp Rules
18. Oracle Rules
19. Other Ids Rules
20. P2P rules
21. Pop2 Rules
22. Pop3 Rules
23. Porn Rules
24. Rservices Rules
25. Scan Rules
26. Shell Code Rules
27. Ssmtp Rules
28. Snpmp Rules
29. Sql Rules
30. Telnet Rules
32. WebCgi Rules
33. Web Client Rules
34. Web Cold Fusion Rules
35. Web FrontPage Rules
36. Web IIS Rules
37. Web Misc Rules
38. Web Php Rules
39. Web Attacks
40. X11 Rules

Total number of rules that are implemented in this project are 2597 which includes rules of all the different types.
Appendix B: Program Listings

(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
)

(defrule Rule1 "Attack Response Rules"
   (packet (protocol tcp)
      (source_ipaddr $HOME_NET)
      (source_port any)
      (direction ->)
      (destination_ipaddr $EXTERNAL_NET )
      (destination_port any)
      (content "Volume Serial Number")
      (ID 1292 )
   )
   =>
   (printout t "Alert, ATTACK-RESPONSES directory listing" crlf)
)

(defrule Rule18 "BACKDOOR Rules"
   (packet (protocol tcp)
      (source_ipaddr $EXTERNAL_NET )
      (source_port 27374 )
   )
(defrule Rule94 "Bad Traffic Rules"
  (packet (protocol tcp)
    (source_ipaddr $EXTERNAL_NET)
    (source_port any)
    (direction <>)
    (destination_ipaddr $HOME_NET)
    (destination_port 0)
    (ID 524))
  =>
  (printout t "Alert, BAD-TRAFFIC tcp port 0 traffic" crlf))

(defrule Rule106 "DDos Rules"
  (packet (protocol icmp)
    (source_ipaddr $EXTERNAL_NET)
    (source_port any)
    (destination_ipaddr $HOME_NET)
    (destination_port any)
    (content "1234")
    (ID 221))
  =>
  (printout t "Alert, DDOS TFN Probe" crlf))

(defrule Rule138 "DNS Rules"
  (packet (protocol tcp)
    (source_ipaddr $EXTERNAL_NET)
    (source_port any)
    (destination_ipaddr $HOME_NET)
    (destination_port 53)
    (content "[00 00 FC]"
      (ID 255)))
  =>
  (printout t "Alert, BACKDOOR subseven 22" crlf))
(defrule Rule139 "DNS Rules"
  (packet (protocol udp)
    (source_ipaddr $EXTERNAL_NET)
    (source_port any)
    (direction ->)
    (destination_ipaddr $HOME_NET)
    (destination_port 53)
    (content "|00 00 FC|")
    (ID 1948)
  )
  =>
  (printout t "Alert, DNS zone transfer TCP" crlf)
)

(defrule Rule140 "DNS Rules"
  (packet (protocol tcp)
    (source_ipaddr $EXTERNAL_NET)
    (source_port any)
    (direction ->)
    (destination_ipaddr $HOME_NET)
    (destination_port 53)
    (content "[07|authors")
    (ID 1435)
  )
  =>
  (printout t "Alert, DNS named authors attempt" crlf)
)

(defrule Rule141 "DNS Rules"
  (packet (protocol udp)
    (source_ipaddr $EXTERNAL_NET)
    (source_port any)
    (direction ->)
    (destination_ipaddr $HOME_NET)
    (destination_port 53)
    (content "[07|authors")
    (ID 256)
  )
  =>
  (printout t "Alert, DNS named authors attempt" crlf)
)
(defrule Rule159 "Dos Rules"

  (packet (protocol ip))
  (source_ipaddr $EXTERNAL_NET)
  (source_port any)
  (direction ->)
  (destination_ipaddr $HOME_NET)
  (destination_port any)
  (ID 268))

  =>
  (printout t "Alert, DOS Jolt attack" crlf)
)

(defrule Rule175 "Finger Rules"

  (packet (protocol tcp))
  (source_ipaddr $EXTERNAL_NET)
  (source_port any)
  (direction ->)
  (destination_ipaddr $HOME_NET)
  (destination_port 79)
  (ID 320)
  (content "cmd_rootsh")

  =>
  (printout t "Alert, FINGER cmd_rootsh backdoor attempt" crlf)
)

(defrule Rule189 "INFO Rules"

  (packet (protocol tcp))
  (source_ipaddr $EXTERNAL_NET)
  (source_port 80)
  (direction ->)
  (destination_ipaddr $HOME_NET)
  (destination_port any)
  (content "Connection closed by foreign host")
  (ID 488)

  =>
  (printout t "Alert, INFO Connection Closed MSG from Port 80" crlf)
)

(defrule Rule198 "FTP Rules"

  (packet (protocol tcp))
  (source_ipaddr $EXTERNAL_NET)
  (source_port any)
  (direction ->)
(destination_ipaddr $HOME_NET )
(destination_port 21)
(ID 2546)
(content "MDTM")

=>
  (printout t "Alert, FTP MDTM overflow attempt" crlf)
)

(defrule Rule267 "ICMP-INFO Rules"
  (packet (protocol icmp )
    (source_ipaddr $EXTERNAL_NET )
    (source_port any )
    (direction -> )
    (destination_ipaddr $HOME_NET )
    (destination_port any )
    (ID 363 )
  )
  =>
  (printout t "Alert, ICMP IRDP router advertisement" crlf)
)

(defrule Rule360 "ICMP Rules"
  (packet (protocol icmp )
    (source_ipaddr $EXTERNAL_NET )
    (source_port any )
    (direction -> )
    (destination_ipaddr $HOME_NET )
    (destination_port any )
    (content "ISSPNGRQ")
    (ID 465 )
  )
  =>
  (printout t "Alert, ICMP ISS Pinger" crlf)
)

(defrule Rule382 "IMAP Rules"
  (packet (protocol tcp )
    (source_ipaddr $EXTERNAL_NET )
    (source_port any )
    (direction -> )
    (destination_ipaddr $HOME_NET )
    (destination_port 143 )
    (content "LOGIN")
    (ID 1993 )
  )
)
(printout t "Alert, IMAP login literal buffer overflow attempt" crlf)

(defrule Rule420 "NET-BIOS Rules"
(packet (protocol tcp )
(source_ipaddr $EXTERNAL_NET )
(source_port any )
(direction -> )
(destination_ipaddr $HOME_NET )
(destination_port 139 )
(content ":00\.|00|E|00|M|00|L"
(ID 1293 )
)
=>
(printout t "Alert, NETBIOS nimda .eml" crlf)
)

(defrule Rule850 "NNTP Rules"
(packet (protocol tcp )
(source_ipaddr $EXTERNAL_NET )
(source_port 119 )
(direction -> )
(destination_ipaddr $HOME_NET )
(destination_port any )
(content "200"
(ID 1792 )
)
=>
(printout t "Alert, NNTP return code buffer overflow attempt" crlf)
)

(defrule Rule863 "MISC Rules"
(packet (protocol ip )
(source_ipaddr $EXTERNAL_NET )
(source_port any )
(direction -> )
(destination_ipaddr $HOME_NET )
(destination_port any )
(ID 500 )
)
=>
(printout t "Alert, MISC source route lsrr" crlf)
)

(defrule Rule923 "MULTIMEDIA Rules"

)
(packet (protocol tcp )
  (source_ipaddr $EXTERNAL_NET )
  (source_port 80 )
  (direction -> )
  (destination_ipaddr $HOME_NET )
  (destination_port any )
  (content "Content-Type|3A")
  (ID 1437 )
)
=>
  (printout t "Alert, MULTIMEDIA Windows Media download" crlf)
)

(defrule Rule933 "MYSQL Rules"
  (packet (protocol tcp )
    (source_ipaddr $EXTERNAL_NET )
    (source_port any )
    (direction -> )
    (destination_ipaddr $SQL_SERVERS )
    (destination_port 3306 )
    (content "|0A 00 00 01 85 04 00 00 80|root|00|")
    (ID 1775 )
  )
  =>
  (printout t "Alert, MYSQL root login attempt" crlf)
)

(defrule Rule937 "RSERVICES Rules"
  (packet (protocol tcp )
    (source_ipaddr $EXTERNAL_NET )
    (source_port any )
    (direction -> )
    (destination_ipaddr $HOME_NET )
    (destination_port 513 )
    (content "bin|00|bin|00|"
    (ID 602 )
  )
  =>
  (printout t "Alert, RSERVICES rlogin bin" crlf)
)

(defrule Rule949 "ORACLE Rules"
  (packet (protocol tcp )
    (source_ipaddr $EXTERNAL_NET )
    (source_port any )
    (direction -> )
    (destination_ipaddr $SQL_SERVERS )
    (destination_port 3306 )
    (content "|0A 00 00 01 85 04 00 00 80|root|00|")
    (ID 1775 )
  )
  =>
  (printout t "Alert, ORACLE root login attempt" crlf)
)

(source_port any)
(direction ->)
(destination_ipaddr $SQL_SERVERS)
(destination_port $ORACLE_PORTS)
(content "EXECUTE_SYSTEM")
>ID 1673
)
=>
(printout t "Alert, ORACLE EXECUTE_SYSTEM attempt" crlf)

(defrule Rule1226 "OTHER-IDS Rules"
(packet (protocol tcp)
  (source_ipaddr $HOME_NET)
  (source_port 902)
  (direction ->)
  (destination_ipaddr $EXTERNAL_NET)
  (destination_port any)
  (content "6ISS ECNRA Built-In Provider, Strong Encryption")
>ID 1760
)
=>
(printout t "Alert, OTHER-IDS ISS RealSecure 6 event collector connection attempt" crlf)

(defrule Rule1229 "P2P Rules"
(packet (protocol tcp)
  (source_ipaddr $HOME_NET)
  (source_port any)
  (direction ->)
  (destination_ipaddr $EXTERNAL_NET)
  (destination_port 8888)
  (content "|00 02 00|")
>ID 549
)
=>
(printout t "Alert, P2P napster login" crlf)

(defrule Rule1247 "POP2 Rules"
(packet (protocol tcp)
  (source_ipaddr $EXTERNAL_NET)
  (source_port any)
(direction -> )
(destination_ipaddr $HOME_NET )
(destination_port 109 )
(content "FOLD")
(ID 1934 )

=>

(printout t "Alert, POP2 FOLD overflow attempt" crlf)

(defrule Rule1251 "POP3 Rules"
  (packet (protocol tcp )
    (source_ipaddr $EXTERNAL_NET )
    (source_port any )
    (direction -> )
    (destination_ipaddr $HOME_NET )
    (destination_port 110 )
    (content "DELE")
    (ID 2121 )
  )

=>

(printout t "Alert, POP3 DELE negative argument attempt" crlf)

(defrule Rule1278 "PORN Rules"
  (packet (protocol tcp )
    (source_ipaddr $EXTERNAL_NET )
    (source_port $HTTP_PORTS )
    (direction -> )
    (destination_ipaddr $HOME_NET )
    (destination_port any )
    (content "alt.binaries.pictures.erotica")
    (ID 1836 )
  )

=>

(printout t "Alert, PORN alt.binaries.pictures.erotica" crlf)

(defrule Rule1299 "WEB-MISC Rules"
  (packet (protocol tcp )
    (source_ipaddr $EXTERNAL_NET )
    (source_port any )
    (direction -> )
    (destination_ipaddr $HTTP_SERVERS )
    (destination_port $HTTP_PORTS )
    (content "<SCRIPT>")

54
(ID 1497)

) =>
(printout t "Alert, WEB-MISC cross site scripting attempt" crlf)
)

(defrule Rule1627 "SCAN Rules"
(packet (protocol tcp)
 (source_ipaddr $EXTERNAL_NET)
 (source_port any)
 (direction ->)
 (destination_ipaddr $HOME_NET)
 (destination_port 113)
 (content "VERSION\0A")
 (ID 616)
)

=>
(printout t "Alert, SCAN ident version request" crlf)
)

(defrule Rule1644 "SHELLCODE Rules"
(packet (protocol ip)
 (source_ipaddr $EXTERNAL_NET)
 (source_port $SHELLCODE_PORTS)
 (direction ->)
 (destination_ipaddr $HOME_NET)
 (destination_port any)
 (content ";\082\010\017\091 D0\089")
 (ID 647)
)

=>
(printout t "Alert, SHELLCODE sparc setuid 0" crlf)
)

(defrule Rule1665 "SMTP Rules"
(packet (protocol tcp)
 (source_ipaddr $EXTERNAL_NET)
 (source_port any)
 (direction ->)
 (destination_ipaddr $SMTP_SERVERS)
 (destination_port 25)
 (content "rcpt to\03A")
 (ID 654)
)

=>

55
(printout t "Alert, SMTP RCPT TO overflow" crlf)
)

(defrule Rule1724 "SNMP Rules"
  (packet (protocol udp )
    (source_ipaddr $EXTERNAL_NET )
    (source_port any )
    (direction -> )
    (destination_ipaddr $HOME_NET )
    (destination_port 161 )
    (content "|04 00|")
    (ID 1893 )
  )
=>
  (printout t "Alert, SNMP missing community string attempt" crlf)
)

(defrule Rule1741 "SQL Rules"
  (packet (protocol tcp )
    (source_ipaddr $EXTERNAL_NET )
    (source_port any )
    (direction -> )
    (destination_ipaddr $SQL_SERVERS )
    (destination_port 139 )
    (content "|s|00|p|00|_|00|s|00|t|00|a|00|r|00|t|00|_|00|j|00|o|00|b|00|")
    (ID 676 )
  )
=>
  (printout t "Alert, MS-SQL/SMB sp_start_job - program execution" crlf)
)

(defrule Rule1787 "TELNET Rules"
  (packet (protocol tcp )
    (source_ipaddr $EXTERNAL_NET )
    (source_port any )
    (direction -> )
    (destination_ipaddr $TELNET_SERVERS )
    (destination_port 23 )
    (content ":|A0 23 A0 10 AE 23 80 10 EE 23 BF EC 82 05 E0 D6 90|\%|E0|"")
    (ID 1430 )
  )
=>
  (printout t "Alert, TELNET Solaris memory mismanagement exploit attempt" crlf)
)

(defrule Rule1802 "TFTP Rules"
(packet (protocol udp )
  (source_ipaddr any )
  (source_port any )
  (direction -> )
  (destination_ipaddr any )
  (destination_port 69 )
  (content "]00 01|")
  (ID 1941 )
)
=>
  (printout t "Alert, TFTP GET filename overflow attempt" crlf)
)

(defrule Rule1813 "WEB-CGI Rules"
  (packet (protocol tcp )
    (source_ipaddr $EXTERNAL_NET )
    (source_port any )
    (direction -> )
    (destination_ipaddr $HTTP_SERVERS )
    (destination_port $HTTP_PORTS )
    (content "/hsx.cgi")
    (ID 803 )
  )
  =>
    (printout t "Alert, WEB-CGI HyperSeek hsx.cgi directory traversal attempt" crlf)
  )

(defrule Rule2162 "WEBCLIENT Rules"
  (packet (protocol tcp )
    (source_ipaddr $HOME_NET )
    (source_port any )
    (direction -> )
    (destination_ipaddr $EXTERNAL_NET )
    (destination_port $HTTP_PORTS )
    (content ".eml")
    (ID 1233 )
  )
  =>
    (printout t "Alert, WEB-CLIENT Outlook EML access" crlf)
  )

(defrule Rule2187 "WEBCOLDFUSION Rules"
  (packet (protocol tcp )
    (source_ipaddr $EXTERNAL_NET )
    (source_port any )
    (direction -> )
  )
(destination_ipaddr $HTTP_SERVERS )
(destination_port $HTTP_PORTS )
(content "/cfcache.map")
(ID 903 )
)
=>
(printout t "Alert, WEB-COLDFUSION cfcache.map access" crlf)
)

(defrule Rule2222 "WEB-FRONTPAGE Rules"
(packet (protocol tcp )
(source_ipaddr $EXTERNAL_NET )
(source_port any )
(direction -> )
(destination_ipaddr $HTTP_SERVERS )
(destination_port $HTTP_PORTS )
(content "/fp30reg.dll")
(ID 1248 )
)
=>
(printout t "Alert, WEB-FRONTPAGE rad fp30reg.dll access" crlf)
)

(defrule Rule2257 "WEB-IIS Rules"
(packet (protocol tcp )
(source_ipaddr $EXTERNAL_NET )
(source_port any )
(direction -> )
(destination_ipaddr $HOME_NET )
(destination_port $HTTP_PORTS )
(content "/msadcs.dll")
(ID 1970 )
)
=>
(printout t "Alert, WEB-IIS MDAC Content-Type overflow attempt" crlf)
)

(defrule Rule2378 "WEB-PHP Rules"
(packet (protocol tcp )
(source_ipaddr $EXTERNAL_NET )
(source_port any )
(direction -> )
(destination_ipaddr $HTTP_SERVERS )
(destination_port $HTTP_PORTS )
(content "/bb_smilies.php")
(ID 1774 )
)
(defrule Rule2504 "WEB-ATTACKS wget command attempt"
(packet
(protocol tcp)
(source_ipaddr $EXTERNAL_NET)
(source_port any)
(direction ->)
(destination_ipaddr $HTTP_SERVERS)
(destination_port $HTTP_PORTS)
(content "wget")
)
=>
(printout t "Alert, WEB-ATTACKS wget command attempt" crlf)
)

(defrule Rule2596 "WEB-ATTACKS apache DOS attempt"
(packet
(protocol tcp)
(source_ipaddr $EXTERNAL_NET)
(source_port any)
(direction ->)
(destination_ipaddr $HTTP_SERVERS)
(destination_port $HTTP_PORTS)
(content "|2f2f2f2f|")
)
=>
(printout t "Alert, WEB-ATTACKS apache DOS attempt" crlf)
)