The Implementation and Assessment of Snort Capabilities

Aaruni Goel
Department of Computer Science and Engineering
Research Scholar, Mewar University
Chittorgarh, Rajasthan, India

Ashok Vasishtha, PhD
Department of Computer Science and Engineering
Research Supervisor, Mewar University
Chittorgarh, Rajasthan, India

ABSTRACT
The attacks on computer networks are not a new deal. In general except for financial institutions and military or intelligence organizations nobody bothers about it. But in recent times it is being observing that it effects much more than the said calculations. Assuming that somebody (attacker) blocked the access of particular seller's website at peak times then it results that his customer would like to choose another seller's website whose outcome may result to tremendous loss of permanent seller. Likewise, there are many instances where the impact on network attacks has been observed from top notch to common people. Snort has emerged as a powerful solution to those organizations that could not spent much on purchasing licensed intrusion detection and prevention system as snort is free ware. This paper is aiding to popularize the techniques that can help everybody to identify and prevent from these attacks. The discussed medium in this paper is SNORT, an open source and powerful network intrusion detection and prevention tool.

Keywords
Snort, Libcap, swatch, sendmail, packet logging

1. INTRODUCTION
Snort is a most widely used open source Network Intrusion Detection and Prevention System (NIDPS). It is developed by Martin Roesch and his team under the parent organization Sourcefire. Since October 2013 it has been taken over by Cisco. Snort NIDPS is written in Language C and performs real time packet analysis and it also consists of smart logging capabilities. It is used to identify signature based detection, anomaly based detection and stateful protocol analysis. This paper will focus on installation part of snort-2.9.7.2 and another center of attention will to find out the impact of this tool on intrusion detection and prevention data set as provided by MIT Lincoln Institute. In this paper so far the whole work is carrying out on Ubuntu 14.04 operating system (OS) and 2 GB RAM.

2. ARCHITECTURE OF SNORT
The architecture of snort can be categorized into five basic modules namely Libcap, Packet Decoder, Preprocessors, Detection Engine and Output plugins. The traffic comes from Internet are received by routers and passed to switch. The switch then delivers this data traffic to firewalls for first level of evaluation. After that the firewalls passed them to the Ethernet adapter of server. Here the Snort came into focus for any type of evaluation of those data packets.

The detailed pictorial view of integrated architecture of snort is explained in Fig. 1. These modules make the snort as a complete body.

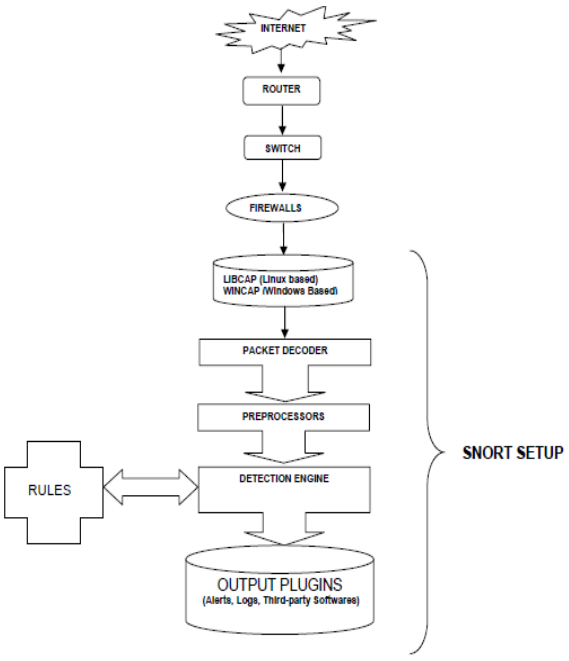

Fig. 1: The integrated modules in Snort architecture

The brief description of each component is explained as under [1],[3]:

- **Libcap**
  With the help of this utility snort is able to capture raw packets directly from Ethernet adapter of server which is needful. Otherwise there are chances that these packets go to any alteration process as instructed by operating system without using this service. Snort always needs a crude or raw packet directly from ethernet. It then analyzes them thoroughly and accurately according to its own norms.
- **Packet Decoder**
  This is the first place where raw data traffic comes and reaches to this module of snort i.e. from ethernet adapter after passing from libcap utility. The packets are decoded here on the manner that each packet is diagnosed in terms of proper and rightful protocol implementation as to follow TCP/IP protocol suite with the help of multiple decoders. It can be thereby said that they are based on signature based detection. The final verdict is calculated and transferred to next module as stated in Fig. 1.
- **Pre-processors**
  They are involved in finding out the abnormalities (i.e., anomaly based detection). It generally involves identifying the anomaly detection by varying the data traffic pattern that was based on signature based detection. With the help of such pre-processors, this module provides the normalized data traffic to detection

engine. By using such normalized traffic, detection engine identifies many evasion techniques and anomaly based attacks. There are different types of preprocessors included in snort 2.9.7.2 with optional options, however in this paper no preprocessor configured but for actual intrusion detection and prevention they are necessary to control. The significant ones are as under:
frag3, stream5, http_inspect, ftp_telnet, smtp, sfportscan, arpspoof, ssh, dns, imap, pop and *reputation*. The description of each of the preprocessor is very wide and beyond the scope of this research paper [5],[7].

- **Detection Engine**
  This portion of snort is principally very dynamic and unified. This module is very vital in terms of multiple rules examination in terms of their priority order. When snort use to inspect the packets multiple rules with different priorities are reported and stored in a queue. However then it only reports out the rule(s) with highest priority. This is specially used to avoid deep evasion techniques if used by attacker. This makes the snort as a highly proficient in terms of attack identification.
- **Output Modules**
  This module design came up after Snort 1.6 version. This is the last segment of snort where packets come from detection engine and disseminated to network in different modes as per the convenience of the network administrator. The convenience of network administrator is in terms to view the real time alerts, logs and other parameters to evaluate the performance of the network of the organization. Third party tools such as *mysql*, a database, can also used for the same purpose. But in this paper the logs are stored into */var/log/snort* directory.

Furthermore, in this research work it is also not focusing about the details of *snort.conf* file which is very essential to configure for smooth functioning of snort as per the network policy, however important details of this file will be provided timely as per the need. In this research work, snort is postulated as strong NIDPS, an open source. This project work shows how an intrusion detection and prevention capability is open for common man with free of cost charge. In this work it is also focused on how a simple computer educated person can easily manage this tool with the knowledge of basic rules even for small and medium enterprises.

3. INSTALLATION OF SNORT AND RELATED PACKAGES

The installation part encompassed the one to two hours installation with normal internet speed of 80-120 kbps. In this section under sub-section 3.1, 3.2, 3.3, the main focus is on the installation part of snort-2.9.7.2 with the real time intrusion detection and prevention features [11].

3.1 Snort Prerequisite on Terminal

Before fresh installation of snort, it is recommended to run the following commands on the terminal:
 # sudo apt-get update
This command updates the system in terms of providing information of new dependencies, versions etc. but not any type of real upgradation.
 # sudo apt-get upgrade
This command uses the information from previous command and really upgrades the system accordingly to the current level.

3.2 Pre-Installation and Support of different Packages

Before the installation of snort the following command sets are executed to provide proper base for snort. These commands basically install some packages and are prerequisites before the installation of snort. The names of those packages are given itself in the command separated by white line character [11].
The description of installed command parameters are as under describes the necessities of all features are shown in Fig. 2:
 # sudo apt-get install flex bison build-essential checkinstall libpcap-dev libnet1-dev libpcre3-dev libmysqlclient-dev libnetfilter-queue-dev

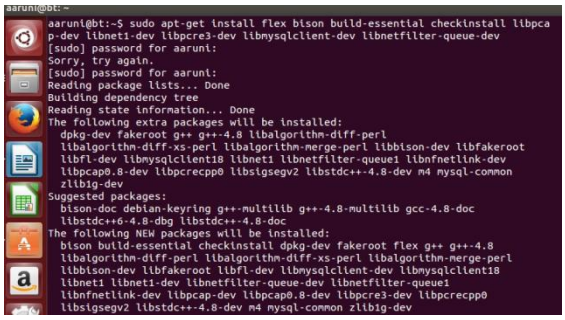

Fig. 2: Installation of prerequisite packages for Snort

flex
It is acronym for fast lexical analyzer which is used to identify the lexical patterns for matching and generating the desired result in a given text.
bison
It is a broadly useful parser generator that changes over an explained setting free linguistic use into a deterministic LR or summed up with LR parser tables. Bison underpins both mnemonic and single-letter alternatives along with choice names. The L implies that the parser peruses input message in one course without moving down; that bearing is normally left to appropriate inside every line, and start to finish over the lines of the full info document. (This is valid for most parsers.) The R implies that the parser delivers a furthest right determination in turn around.
build-essentials
It is generally used to compile the debian packages using gcc and/or g++ compilers.
checkinstall
This is a better substitute to *make install* command. It is used just after the *make* command and generates a debian package with the help of files that came up from make command. It then adds it to the installed packages database, allowing for easy package distribution or removal.
libpcap-dev
As explained in previous section 2 with the help of libcap Snort occupies all the packets directly from interface so that no modification can be performed by operating system.
libnet1
It works generally in the association of *libpcap* for handling and managing packets received from low-level network. Simple to complicated programs can be written at lower level packets can be written with the help *libnet* application program interface (API).
libpcre3-dev
It stands for library Perl-compatible regular expression, a library whose syntax and semantics are nearly very same to the Perl 5 language which uses its API and wrapper functions to support POSIX (Portable Operating System Interface).

libnetfilter_queue
Again an API, which in general, discards the packets that are being in queue and processing old IP address containing packets or reinserted the modified packets.
After the execution of the above commands successfully, the package libdnet is downloaded and manually installed. As said earlier it is used to offer interface which is portable to support many low-level networking functions. The download (wget), unzipping (tar) and installation (./conFig.) part details are mentioned Fig. 3 and Fig. 4 respectively:
 # wget https://libdnet.googlecode.com/files/libdnet-1.12.tgz
 # tar xvfz libdnet-1.12.tgz

Fig. 4: Configuring *libdnet* with *"CFLAGS=-fpic"*

Note that, as mentioned in Fig. 4, "./"conFig. "CFLAGS=-fpic" is necessary for the compatibility with 64 bits operating system for libdnet. Then the next command that is to be executed:
 # make
By use of this command one is permitted to download the most recent variant and update of mainstream tools at host, introducing it close by the majority of the required conditions (dependencies) which will request root get to in the event that you don't have all the required dependencies introduced effectively (see Fig. 5).

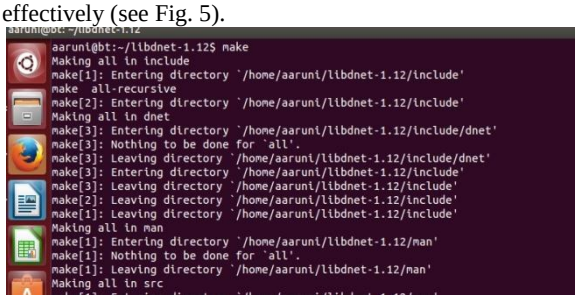

Fig. 5: Running *make* command in *libdnet* package

 # sudo checkinstall
It is a utility that assembles a .deb, .rpm or slackware package from an outsider source code tarball. This permits present such outsider programming utilizing the standard packet administration components for administrator's dispersion as per the linux (Ubuntu, Fedora etc.) is used, (refer Fig. 6) .

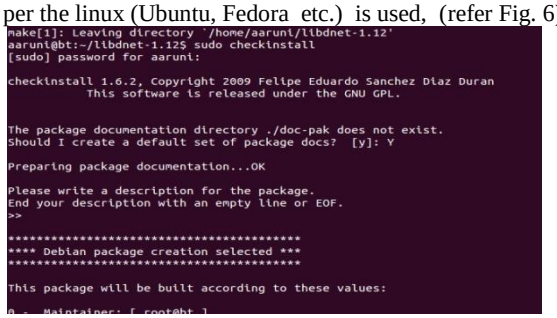

Fig. 6: Usage of *checkinstall* command

 # sudo dpkg -i libdnet_1.12-1_amd64.deb
The *dpkg* or the debian package is a chief package supervisor for debian and other debian based Linux disseminations like Ubuntu. *dpkg* can be utilized for an range of purposes like installation, uninstallation and
knowing the present state of package, refer Fig. 7.

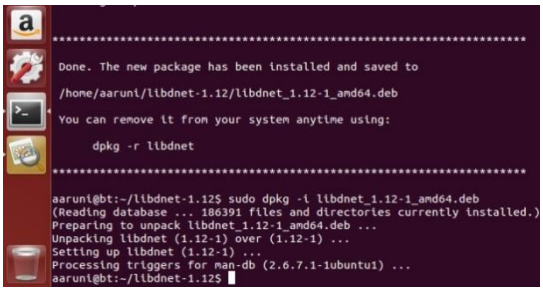

Fig. 7: Conversion of *lidnet* into debian package

After that a symbolic link is made by issuing the command:
 # sudo ln-s /usr/lib/libdnet.1.0.1 /usr/lib/libdnet.1
A *ln –s* is a symbolic view that contains a content string that is naturally translated and taken after by the working operating system (OS). This other document or registry is known as the target and is registered or indexed with OS itself. In the event that a symbolic link connection is erased, its target stays unaffected and acts like a backup. Moreover that a symbolic connection focuses to an target, and at some point later that target is moved, renamed or erased, the typical connection is not consequently redesigned or erased, but rather keeps on existing and still indicates the old target, now a non-existing area or document, refer Fig. 8.

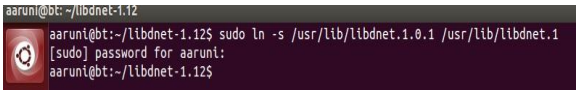

Fig. 8: Creating symbolic link of libdnet.1.0.1 package

3.3 Snort Installation

Now it is time for the entry of our most awaited snort NIDPS. One thing must be kept in notice before going further path names must be carefully learnt throughout the installation process and this tool management. The rest of installation is more or less same like the previous installation of the *libdnet* package [7],[11].
The snort is installed in */etc/snort* directory. Fig. 9 announces the free download of snort from www.snort.org.

Fig. 9: Download of Snort from Snort.org

Then again all the previous commands are going to be repeated from Fig. 10 to Fig. 15 for complete installation of snort. All Figures from Fig. number 10 to 15 are self explanatory.

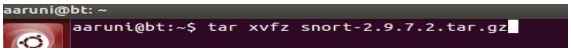

Fig. 10: Unzipping Snort's *tar* package

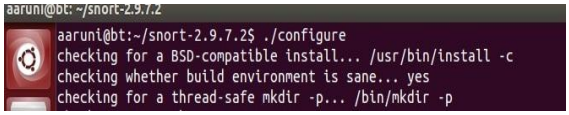

Fig. 11: Configuring Snort

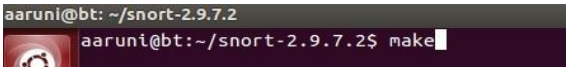

Fig. 12: Updation of Snort through *make* command

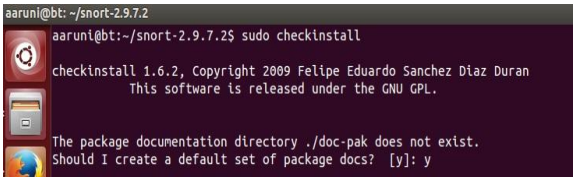

Fig. 13: Running *checkinstall* command for Snort

Fig. 14: Installation of debianpackage for snort

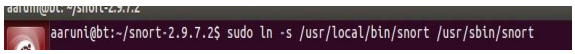

Fig. 15: Making symbolic link of Snort

Now execution of another command:
 # sudo ldcofig -v
This command is again utilized to keep up the common library store. This reserve is regularly put away in the record /etc/so.cache and is utilized by the OS to outline shared library name to the area of the relating shared library document. It makes, upgrades, and removes the essential connection links and cache (for use by the run-time linker, ld.so) to the latest shared libraries found in the registries determined on the order line, in the record /etc/ld.so.cache, and in the trusted libraries (/usr/lib and /lib). It then checks the header and record names of the libraries it experiences while figuring out which variants ought to have their connection links renovated. *ldconfig* neglects symbolic links when examining for libraries [7](refer Fig. 16).

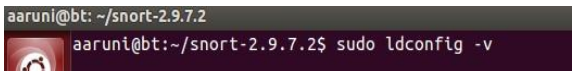

Fig. 16: Execution of *ldconfig* in verbose mode (-v)

Snort 2.9 series presents the DAQ, or Data Acquisition library, for Input/Output[13]. The DAQ straight away guide calls to libpcap capacities with a conceptual layer that encourages operation on varieties of hardware and programming interfaces without agreeable changes to snort. It is feasible to choose the DAQ class and mode when call snort to perform pcap read back or inline operation, and so forth. Such type of changes makes the snort architecture very flexible in nature user friendly for different users using especially different OS platform. Thereby DAQ has been also installed in this work in the similar manner as previous installation took place. The Fig. 17 shows the download of DAQ.
 # wget https://www.snort.org/downloads/snort/daq-2.0.4.tar.gz

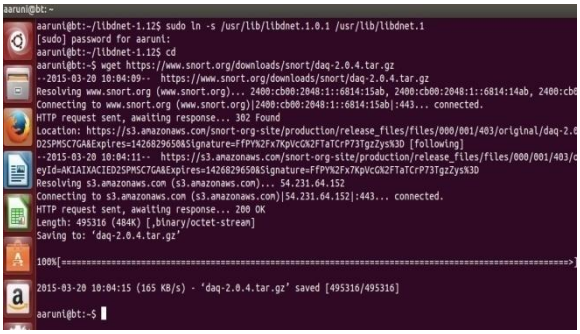

Fig. 17: Download of DAQ from Snort.org

Rest of the process is same as repeated in libdnet and snort installation part. The said commands are once again sequentially listed as:
 # tar xvfz daq-2.0.4.tar.gz (Unzipping the DAQ package)
*# cd daq-2.0.*4 (Moving inside the DAQ directory)
 # make (Usage of make command in DAQ updation)
 # sudo checkinstall (To make DAQ debian package)
 # sudo dpkg -i daq_2.0.4-1_amd64.deb (debian package installation)

Now to check the working that Snort got installed properly it is written on terminal:
 # snort –v (Snort in verbose mode)
The following commands depicts the Snort in verbose. A verbose mode is a choice accessible in numerous systems working OSs, including Microsoft Windows, Mac OS and Linux that gives extra points of interest with respect to what the computer is doing and what drivers and programming it is unpacking during start-up. This level of detail is extremely useful for investigating issues with hardware or software, if mistakes are happening during or after start-up of applications OS. The Fig. 18 depicts that the snort is properly configured and able to detect and prevent attacks related networks [1],[10].

Fig. 18: The Snort in verbose mode

Then a directory is created for snort rules. In this directory the pre-defined snort rules directly from www.snort.org. would be downloaded and stored. In this paper work, along with predefined rules from snort rules, organization policies are stored in */etc/snort/rules* folder.
 # sudo mkdir /etc/snort/rules
In */etc/snort/rules* snort stores its own as downloaded from www.snort.org rules. These are later unzipped in this folder.
 # tar xvfz snortrules-snapshot-2960.tar.gz /etc/snort/rules
At the path */etc/snort/rules* all the unzipped rules would be stored as a result all open accessible rules provided by snort team.
It is to be remembered that for more simplicity to execute snort as daemon (background process). By typing which automatic logging starts to */var/log/snort/* folder related to suspicious events.

 # snort –D –c /etc/snort/snort.conf –l /var/log/snort/

3.4 Snort Logs

As we all know logs are the major source to identify any type of intrusion or for a forensic analysis. The snort with the help of its rule set generates its own logs on the basis of rule set configured. The logs collected by snort can display and log the header and if needed data part of packets. It is demonstrated in the Fig. 19 which contains the header part and data part is clearly marked and snort is working as in packet sniffer mode on the impact of the command:
 # snort –vde

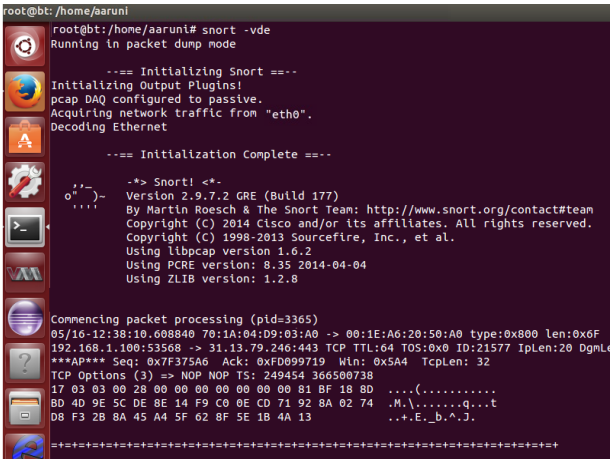

Fig. 19: The display of complete packet (Header + Data) on the basis *–vde*

One can also log these packets of Fig. 19, by issuing the command:
 # snort –vde –l /var/log/snort
Now it is check how the logs are stored in the system itself. For the sake of convenience it is assumed that a rule is crafted (for rules, refer section 4.2) and launch the command as depicted in Fig. 19 along with *–l /var/log/snort* option. According to the rule whenever anybody will ping *gmail.com*, then the corresponding packets will get logged in */var/log/snort* directory. Now it demonstrated with the help of two terminals. The work on first terminal is shown in Fig. 20 which shows that ping command has been executed.

Fig. 20: IP address of system and *ping* command

The Fig. 20 provides the details of logs generated and collected in the /var/log/snort/10.11.5.184 folder. The folder /216.58.199.133 which is also the IP address of gmail.com reflects that it pings using Internet Control Message Protocol (ICMP). On the second terminal the following command executed in parallel:
 # snort -c /etc/snort/snort.conf –l /var/log/snort

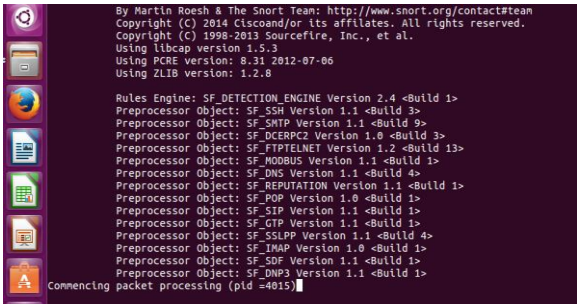

Fig. 21: Snort as a logger to spot rule based packet

Fig. 22 shows the logs generated by ping command along with its anatomy.

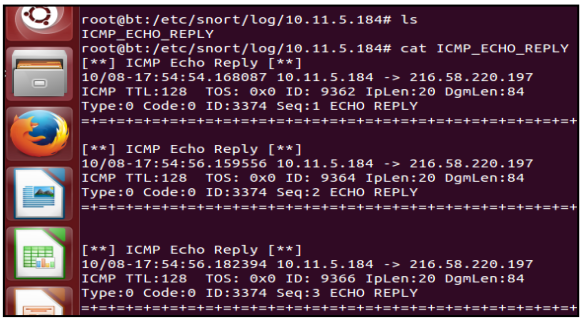

Fig. 22: Verifying the logs generated by Snort

This subsection therefore elaborated the storage and working of snort to accumulate the logs in server. On this basis it can also be said that these logs also provide the valuable forensic details in case of any type of intrusion.

4. RULES

Rules are the laws that give directions to network at the network intrusions. They are composed to prevent network from any attack and also provide guidelines to formulate the network policy of organization. The rules are the whole sole governor in intrusion detection and prevention in NIDPS environment [1].

4.1 Format of Rule

The structure of snort rules are shown in Fig. 23, according to which mandatory parts are action variants, protocols involved, source and destination IP addresses, source and destination port addresses and direction signs as specified. On the other hand optional part contains a message which can be a blend of one or more than one numerous syntaxes as provided by snort team [6],[9].

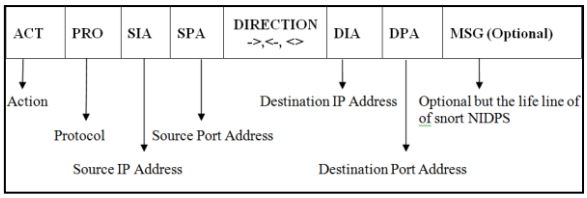

Fig. 23: The Format of a Rule

Action: It defines that what type of action is to be taken before creating a rule related to the policy of the organization. There are eight such actions that can be provided to any packet but only one on a given rule. For the scope of this research work only six actions are discussed. A packet may use:

- Pass: To move directly to network with no hindrance and not being logged.

- Alert: To provide alert notification to administrator and then get logged.
- Log: To directly get logged in pre specified file or database.
- Drop: To directly got dropped before entering the network and then logged.
- Reject: To directly being dropped from ip-tables and then got logged.
- Sdrop: To being directly dropped silently without any logging.

Protocol: The rule can opt only one of the four available protocols -- Transmission Control Protocol (TCP), User datagram Protocol (UDP), Internet Control Message Protocol (ICMP) and Internet Protocol (IP).
Source and Destination IP address: It is 32 bits binary format in network layer to identify any user.
Source and Destination Port address: It is 16 bits binary structure responsible for peer to peer delivery at transport layer.
Direction Sign: It specifies the unidirectional or bidirectional transfer of data from inside network to outside network and vice versa. The sign part is more elaborated in next subsection 4.2.
It is to be noted that snort detects attacks primarily by using signature based detection methodology. Boyer-Moore algorithm is deployed for pattern comparison and matching to identify the attacks in rules. Vulnerability Research Team (VRT) is organized by snort team to test and implement various rules for new attacks. The VRT team analyzes all the real time attacks and creates the rule sets against them. Beside this it also evaluates the rules created by snort user which he submitted to snort forum under its test condition and if found fruitful, this team add such rules in its rule sets. The VRT new rules are freely given to paid subscribers while after 30 days' time period they are also available to free users also. Emerging Threats Pro LLC is another organization which provides the rule sets to Snort. It also provides free rule sets with name ETopen and paid rule sets with title ET pro to users. In Snort both VRT and ET rule sets can be packed simultaneously due to their same format structure.

4.2 Case Study and Configuration of Rules

The first basic rule explains that network administrator of college administration wants to see the activities of students on www.facebook.com. The rule then logs the details of every tcp packet that comes as a part of internet but only alert the traffic activity that has www.facebook.com. The result will display on console in the form of sense message: "The facebook has been accessed".

 # alert tcp $EXTERNAL_NET any -> $HOME_NET 80 (content:"www.facebook.com";msg:"The facebook has been accessed";)

But this will alert every time the network administrator whenever any machine in college will try to attempt facebook.com. This make his time totally at waste. He then uses same command with minor change so that whenever he is free he himself can find out the activities of students.

 # log tcp any any -> $HOME_NET 80
(content:"www.facebook.com";msg: "The facebook has been accessed";)

Further if network administrator wants that student must not waste their time on facebook then he can use another rule with little difference:

 # droptcp $EXTERNAL_NET any -> $HOME_NET any (content:"www.facebook.com";msg:"The facebook has been accessed";)

As per the three case studies, now the basic structure says that (as already mentioned in section 4.1) action, protocol, IP addresses, port addresses and direction are necessary parts to formulate a basic rule.
$EXTERNAL_NET is any IP address that routes through administrtor server $HOME_NET. Next "any" specifies whichever IP address and whatever the port number all are bound to $HOME_ NET as a default gateway for mutual communication. The direction in these examples is unidirectional (->) i.e. from network of network IP systems to administrator's server. Many situation comes when bidirectional angle brackets (<>) sign or unidirectional (<-) is used for local and remote administration.
Now focusing on optional message part also known as body where it is stated that take appropriate action whenever snort finds matched string www.facebook.com with message "The facebook has been accessed". The body part always starts and closed with parenthesis and every optional syntax and its value in the body is separated by semicolon.
Now some important and some advance rule sets that one should understand and are categorized in three parts as under in sub-sections 4.2.1, 4.2.1 and 4.2.3.
4.2.1 Signature Based Detection/Prevention
Snort is able to tackle the signatures based detection with the rules.
The first study to detect the infamous nimda Virus whose signature is already well known. The snort rule to avoid such virus is

 # drop tcp $EXTERNAL_NET->$HOME_NET 80 (msg: "Web-IIS cmd.exe access";flags:A+;content:"cmd.exe", nocase; classtype:web-application-attack;sid:1002;rev:2;)

This is a case where a known attack of nimda virus is identified; the body is read as under:

- msg: WEB-IIS cmd.exe access

flags: A+ : As shown in Fig. 24, a TCP header contains the six flags in total are implemented. Here Acknowledgement flag (ACK) is set. In general other options may also incorporate like SA+, !R, etc.

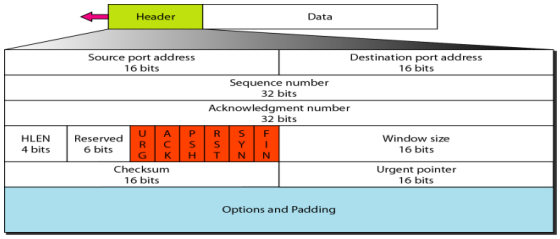

Fig. 24: TCP protocol structure Header

- Content: If data traffic contains cmd.exe type of pattern.
- Nocase: The content part should be treated as case insensitive.
- Classtype: It is used to classify the type of rule. The value web-application-attack is a class name; network administrator can create other classifications for the attacks related to organization policy or any new that he wishes to add. By default snort has its own classification with four priorities and exist in *classification.config* file. To grouping the rules in *classtype* is a better way to shape the events related to attacks. It has a format that looks like as --config classification: <class name>,<class description>,<default priority>

The top priority is 1 which means High alert and 4 implies merely a simple suspicious activity.

- Sid: Snort Identifier; here it is 1002. It uniquely uses to find out Snort rules i.e. every rule is unique. In general less than 100 is kept for future distribution; The range100-

999,999 is for Snort's own use and over 999,999 is for local use of organizations.

- Rev: It shows the number of times a revision has been done to improve this rule either to detect new or modified attacks or to reduce the false positive. The value in this rule illustrates the two times revision of the said rule.

It is to be noticed carefully that "!" is used as NOT, "+" is for AND, and "*"is for OR operations. Finally after detection as an Snort IPS can dropped it too using either drop or sdrop syntax [6].

4.2.2 Anomaly Based Detection/Prevention

Sometimes buffer overflow exploits can be identified when host receive large packets size. In such cases the size of packet is measured and dropped by utilizing the *dsize* keyword in the rule. This is used to permit that what maximum or minimum or equal size is needed by network administrator.

 # drop ip any any ->$HOME_NET ANY (dsize:>6000; msg: "IP packet has more than 6000 bytes";)

As looking the IP header in Fig. 25 the maximum size of data 65,536 –(20 or 60) is 65,516 to 65,476 bytes. By this analogy if Snort finds the packet of size greater than 6000 bytes it can be dropped. This is in keeping view to avoid buffer over flow problem [12].

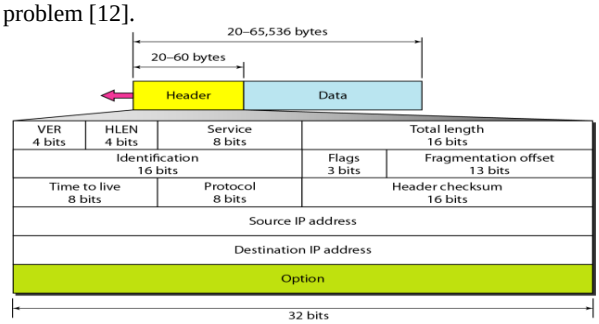

Fig. 25: IP protocol structure Header

4.2.3 Deep Packet Inspection

This rule gives the details of vulnerable protocol implementation. According to TCP header SYN and FIN cannot be set at the same time. The blend of SYN and FIN being set in TCP header is unlawful and it has a place with the class of illicit combination since it calls for both foundation of connection(via SYN) and end of connection(via FIN). So for this special case a rule is created as:

 # alert tcp any any -> $HOME_NET any (msg:"SYN-FIN packet detected";flags:SF;)

This is a major miserable situation toward security group since aggressors might abuse this by get to know that which type of OS or its version victim belongs to, so that they craft packets for that particular OS.

As a NA this rule is very useful rule which depicts if any user or unauthorized porn sites may try to communicate then:

 # alert tcp any any <> $HOME_NET any (msg:Censored!!! Porn Content"; content-list= "porn", react:block;)

Here two new parameters are declared:

- Content-list: This is list configured in snort.conf and many such lists can be created as per the need of the organization. A list is created with name 'porn' that includes hundreds of websites those are involved in pornography.
- React: This is the way by which Snort as an IPS blocks those websites that are added in one of the list named as porn.

Finally the most common but very important part needs to be discussed as per the rule statistics, that uses to show the actual *content* syntax working. In general the next defined rule is to explain the basic working of the said keyword.

 # alert tcp any any -> !HOME_NET any (content: "|47|45|54|"; msg: "GET matched";)

Actually the content part reads the hexadecimal-binary-ASCII trilogy. The "|" symbol represents byte-code for binary data (or the representation of binary data in hexadecimal). In this rule hexadecimal 47 is converted to binary and then mapped to ASCII values. The result is 47=G, 45 = E, 54=T. So every TCP packet that is *not* the part of HOME_NET will be alerted and logged.

It is to be remembered without pipes only simply string is compared. The content part string comparison is done through the Boyer Moore pattern matching algorithm.

An additional versatile rule is based on ICMP protocol. For e.g. a rule type like:

 # alert icmp any any -> any any (msg:"Ping withTTl=100" ttl:100;)

This is a wonderful rule in sense that the TTL is a part of IP protocol but here ICMP protocol is used for alert. This simply shows the richness of Snort's rule in terms of keeping knowledge of other protocols too [8].

Snort uses hundreds of syntaxes. In this paper so far discussed the important and highly used syntaxes. One more parameter that one may ask is *reference* syntax. The reference syntax permits standards to incorporate references to outside third parties that have created. At the end it is to be said that the Snort rules are very much flexible and easy to learn. But as complexity arises the situation goes worst. It is due to the fact that multiple plugins are joined to filter the traffic [4].

5. TESTING OF SNORT ON DARPA DATA SET 1999

The Cyber Systems and Technology Group (earlier the DARPA Intrusion Detection Evaluation Group) of MIT Lincoln Laboratory, under Defense Advanced Research Projects Agency (DARPA ITO) and Air Force Research Laboratory (AFRL/SNHS) sponsorship, has gathered and circulated the primary standard for assessment of network system to identify intrusions.

These assessments measured likelihood of discovery and probability of false alarm for every network under test. These assessments contributed essentially to the intrusion finding of recorded network attack to look into field by giving guidance. They are important to all scientists chipping away at the general issue of workstation and system intrusion recognition [2].

To begin first of all download the file outside.tcpdump from MIT Lincoln Laboratory website to snort folder [14]. For the sake of this work this file is renamed as IDPS_darpa_dataset.tcpdump. The IDPS_darpa_dataset.tcpdump file is nothing but the dump of data traffic. The data traffic itself contains many attacking scenarios. This is a standard file and provided to identify the number of intrusions to check the working of NIDPS. Its values can be changed as per the rules policy of the organization [2].

Then the following is a command to run snort in an IDS mode and to read and analyze this file also as shown in Fig. 26 [11].

 # snort eth0 -c /etc/snort/snort.conf -r IDPS_darpa_dataset.tcpdump

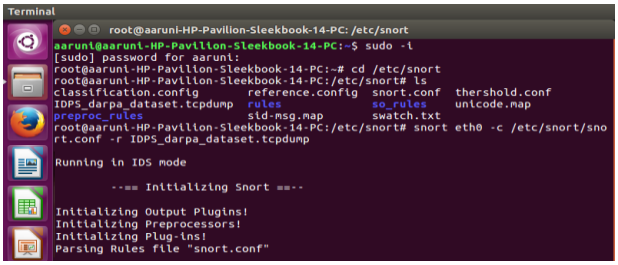

Fig. 26: Running Snort as IDS to analyze DARPA Dataset

The Fig. 27 describes the analysis of Snort in an IDS mode where the whole dump is reviewed and checked out as per the rules configured in Snort.

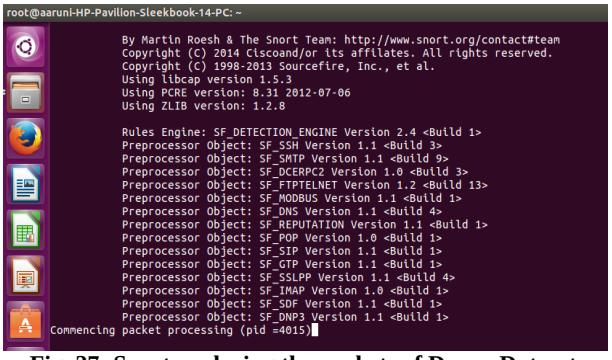

Fig. 27: Snort analyzing the packets of Darpa Dataset

As per statistics the Snort runs for merely 55 seconds to complete IDPS_darpa_dataset.tcpdump analysis of 57 MB. After the snort quits itself (see Fig. 28).

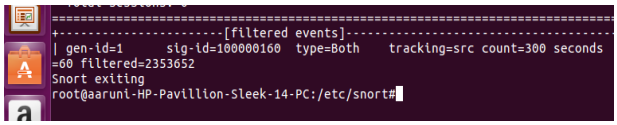

Fig. 28: Snort quits after analyzing the IDPS_darpa_dataset.tcpdump

As said earlier after quitting of snort when mouse is scrolled up it finds the important details of this session as shown in Fig. 29. This Fig. shows that in 55 seconds snort captured 2291319 packets in total.

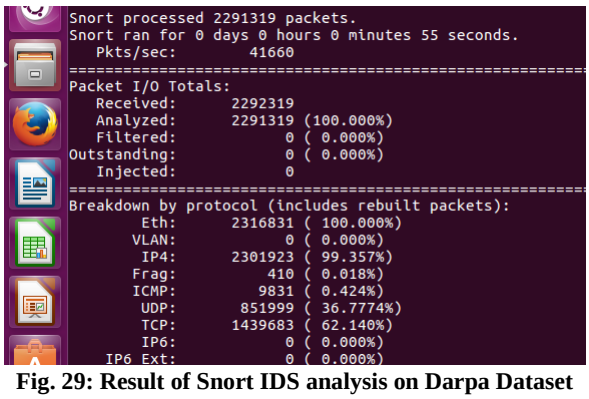

Fig. 29: Result of Snort IDS analysis on Darpa Dataset

The second row in packet I/O reflects how many packets are analyzed by snort after receiving 2291319 packets. The third row Filtered suggests that no packets are given to snort as they are not prepared and simply the raw packets. In precise all the packets are given to snort for analysis.

It shows the actual operation and corresponding result. The fourth row states that the number of packets waiting in a buffer, for further processing by snort, is zero. Finally the fifth row indicates the Injected packet which says the number of packets created and sent by snort are also zero. During TCP reset conditions such activity in general takes place.

After complete execution the output that is related to NIDPS is depicted in Fig. 30 (a). This Fig. provides us the total action statistics on the basis of rule sets configured. It shows that in general after analyzing IDPS_darpa_dataset.tcpdump file by snort, 1679 alerts are generated and logged. No packet is bypassed with respect to rule contradiction and priority.

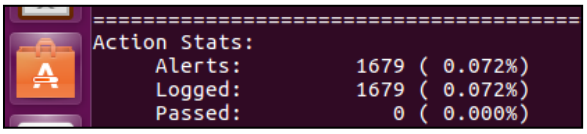

Fig. 30(a): Statistics of Alerts generated by Snort IDPS analysis

One more important thing that is analyzed during this data analysis is verdicts parameter (see Fig. 30 (b)).

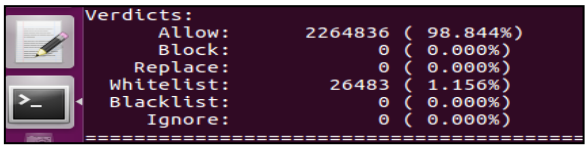

Fig. 30(b): Snort IDPS micro analysis

The Table 1 provides the brief details of verdicts and other shown keywords as displayed on the basis of file IDPS_darpa_dataset.

Table 1

Verdicts: These values may updated after snort continues to read data stream		
Allow	2264836	No action is taken by snort on theses packets after examining them.
Block	0	No packets are dropped by rule-set configuration.
Replace	0	No packet is altered during normalization process
Whitelist	26483	These are the packets which snort allows without any examination by itself.
Blacklist	0	These are the packets which snort seeks to examine them.

6. CONCLUSION

We have executed this project on Ubuntu 14.04 TLS (Trusty Tahr) and if all is well the complete package takes around two to three hours for complete installation including sendmail, swatch and snort and all its supporting packages. We are satisfied with the complete functioning of snort and its related IDPS technology but silently it can be said that there are some things that little bit hamper the task of functioning of NAs somehow.

Starting from the very basic it is necessary for user to be have good understanding of Linux and associated packages configuration files. In general it has found that many user are not find them familiar and self-comfortable in using Linux based operating system. The said statement is not on the basis of NAs but on the basis of foundations of large networks where supporting staff is not proficient in Linux OS. The said statement is not on the basis of NAs but on the basis

of foundations of large networks where supporting staff is not proficient in Linux OS.

Another thing there are in general more than 23000 signatures which we found in snort rules files provided by snort.org. Moreover NAs also have to create many rules for the sake of the organization policy time to time, what so ever are required by the management. Due to the said purpose it is quite difficult to work on the different other unknown but genuine applications on the computers which are under snort's umbrella. At last it can be said that in near future and with more versatile tools the associated problems in snort will be more easily checked.

7. REFERENCES

[1] Goel, Aaruni and Vasishtha, A.K., A Review on Foundation of Network Intrusion Detection and Prevention Systems (NIDPS)", csjournals, 2017, Volume 9, Issue 1, pp.125-137.

[2] L. Emilie and E. Jonsson, "Survey of Intrusion Detection Research", Chalmers University of Technology, (2002)

[3] E. D. Dorothy, "An intrusion-detection model", Software Engineering, IEEE Transactions, vol. 2, (1987), pp. 222-232.

[4] R. Suman and V. Singh, "SNORT: An Open Source Network Security Tool for Intrusion Detection in Campus Network Environment", International Journal of Computer Technology and Electronics Engineering, vol. 2, no. 1, (2012), pp. 137-142.

[5] R. R. Ur, "Intrusion detection systems with Snort: advanced IDS techniques using Snort, Apache, MySQL, PHP, and ACID", Prentice Hall Professional, (2003).

[6] K. Vinod and O. P. Sangwan, "Signature based intrusion detection system using snort", International Journal of Computer Applications & Information Technology, vol. 1, no. 3, (2012), pp. 35-41.

[7] Salah, K. and Qahtan, A.; "Boosting throughput of Snort NIDS under Linux", Proceedings of IEEE International Conference on "Innovations in Information Technology", pp: 643 – 647, 2008.

[8] Ahmed, M.; Pal, R.; Hossain, M.; Hasan, K. and Bikas, A.N.; "A Comparative Study on the Currently Existing Intrusion Detection Systems", Proceedings of IEEE International Conference on "Computer Science and Technology", pp: 151 – 154, 2009.

[9] Salah, K. and Kahtani, A.; "Improving snort performance under linux", Proceedings of Communications, IET, vol. 3, Issue: 12, pp: 1883 – 1895, 2009.

[10] Ismail, M.N. and Ismail, M.T.; "Framework of Intrusion Detection System via Snort Application on Campus Network Environment", Proceedings of IEEE International Conference on "Future Computer and Communication", pp: 455 – 459, 2009.

[11] Brian Caswell and Jeremy Hewlett. Snort Users Manual (http://www.snort.org/docs/).

[12] Chang-Su Moon and Sun-Hyung Kim. (2014). Integrated Security System based Real-time Network Packet Deep Inspection. International Journal of Security and Its Applications, pp. 123– 135.

[13] S. Vikrama Teja, S. Kranthi Kumar, T.V. Rao, G.Dayanandam. (2013, August). In-line Prevention System using Snort. International Journal of Application and Innovation in Engineering management.

[14] DARPA Data Set for Intrusion Detection and Prevention (1999) (https://www.ll.mit.edu/ideval/data/)