ISSN No. 0976-5697

Volume 3, No. 2, March-April 2012

International Journal of Advanced Research in Computer Science

RESEARCH PAPER

Available Online at www.ijarcs.info

# Snort, BRO, NetSTAT, Emerald and SAX2 : A Comparison

Suchita Patil*
Computer Technology Department
VJTI Mumbai, India
suchitapatil26@gmail.com

Pallavi S.Kulkarni
Computer Technology Department
VJTI Mumbai, India
pskulkarni77@gmail.com

Pradnya B. Rane
Computer Technology Department
VJTI Mumbai, India
pradnyarane@gmail.com

Dr.B.B.Meshram
Computer Technology Department
VJTI Mumbai, India
bbmeshram@vjti.org.in

***Abstract***: Intrusion detection is an important component in network security. Many current Intrusion Detection Systems are designed on rule-based, which have a limitation of identifying the unknown attacks. Some IDS are designed on anomaly based detection technique which have advantage of identifying known and unknown attacks. It has a disadvantage of learning and training the data set to identify the good and bad data. Some IDS are designed on both signature based and anomaly based detection techniques. That are also referred to as hybrid IDS systems. There are many IDS available in which some IDS are open source IDS and some IDS are commercial products used in enterprise network. This paper gives the detailed comparative study of open source software SNORT, BRO, Net STAT also covers the commercial products like NFR, Emerald which is used as research tool and SAX2.

***Keywords:*** signature based system, anomaly detection system, Intrusion Detection system

## I. INTRODUCTION

Intrusion Detection System [1]: An Intrusion Detection System (abbreviated as IDS) is a defense system, which detects hostile activities in a network. The key is then to detect and possibly prevent activities that may compromise system security, or a hacking attempt in progress including reconnaissance/data collection phases that involve for example, port scans. Intrusion detection is a process of identifying and responding to malicious activity targeted at computing and networking resources".

One key feature of intrusion detection systems is their ability to provide a view of unusual activity and issue alerts notifying administrators and/or block a suspected connection.

In addition, IDS tools are capable of distinguishing between insider attacks originating from inside the organization (coming from own employees or customers) and external ones (attacks and the thread posed by hackers).

Intrusion – a series of concatenated activities that pose threat to the safety of IT resources from unauthorized access to a specific computer or address domain;

Incident – violation of the system security policy rules that may be identified as a successful intrusion;

Attack – a failed attempt to enter the system (no violation committed).

Modeling of intrusions – a time-based modeling of activities that compose an intrusion.

The intruder starts his attack with an introductory action followed by auxiliary ones (or evasions) to proceed to successful access; in practice, any attempts undertaken during the attack by any person, for example by the IT resource manager can be identified as a threat.

An intrusion can be defined as "any set of actions that attempt to compromise the integrity, confidentiality or availability of a resource", for example, illegally gaining super user privileges, attacking and rendering a system out of service (i.e., denial-of-service), etc.

Intrusion prevention systems: IPS technologies are differentiated from IDS technologies by one characteristic: IPS technologies can respond to a detected threat by attempting to prevent it from succeeding.

Intrusion prevention techniques, such as user authentication (e.g. using passwords or biometrics), avoiding programming errors, and information protection (e.g., encryption) have been used to protect computer systems as a first line of defense.

This paper is organized as follows. Section I is introduction which gives brief ides about Intrusion Detection system. Section II focused on the Detection Capabilities and Some of the Examples of Intrusion Detection Systems. Section III discusses the problems or drawbacks associated with the Intrusion Detection systems and variations for the same. section IV is Conclusion.

## II. RELATED WORK

To design any IDS/IPS three major techniques are used. As specified in [1],[2],[5],[9].

There are three techniques used for detection

Misuse detection or Signature detection (knowledge based)

Anomaly detection (behavior based)

Misuse detection discovers attacks based on patterns extracted from known intrusions. Anomaly detection identifies attacks based on significant deviations from normal activities.

Misuse detection has low false positive rate, but cannot detect novel attacks. Anomaly detection can detect unknown attacks, but usually has a high false positive rate. To combine the advantages of both misuse and anomaly detection, many hybrid approaches have been proposed. Data mining is the analysis of large data sets to discover understandable patterns or models. Data mining can efficiently extract patterns of intrusions for misuse detection, identify profiles of normal network activities for anomaly detection, and build classifiers to detect attacks. Data-mining-based systems are more flexible and deployable. The security experts only need to label audit data to indicate intrusions instead of hand coding rules for intrusions. There are many Data mining algorithms that can be used in the Intrusion Detection techniques. There are many papers based on the Probability based algorithm, Information theory based algorithms(based on entropy), Random forest based algorithm which can be used for prediction and probability estimation. The random forests algorithm is an ensemble classification and regression approach, which is one of the most effective data mining techniques.

One of the challenges in IPS/IDS is the feature selection. Feature selection is essential for improving detection rate. The raw data format of network traffic is not suitable for detection. IDSs must construct features from raw network traffic data, and it involves a lot of computation. Thus, feature selection can help reduce the computational cost for feature construction by reducing the number of features. Another challenge of intrusion detection is imbalanced intrusion. Some intrusions such as denial of service (DoS) have much more connections than others (e.g., user to root). Most of the data mining algorithms try to minimize the overall error rate, but this leads to increasing the error rate of minority intrusions. However, in real-world network environments, minority attacks are more dangerous than majority attacks.

As far as the data source is concerned, intrusion detection can be classified into host-based and network-based detections. [5], [7]

a. Host-based approaches detect intrusions utilizing audit data that are collected from the target host machine. As the information provided by the audit data can be extremely comprehensive and elaborate, host-based approaches can obtain high detection rates and low false-alarm rates. However, there are disadvantages for host-based approaches, which include the following.
   Host-based approaches cannot easily prevent attacks: when an intrusion is detected, the attack has partially occurred.
   Audit data may be altered by attackers, influencing the reliability of audit data.
b. Network-based approaches detect intrusions using the IP package information collected by the network hardware such as switches and routers. Such information is not so abundant as the audit data of the target host machine. Nevertheless, there are advantages for network-based approaches, which include the following.

Network-based approaches can detect the so-called "distributed" intrusions over the whole network and thus lighten the burden on each individual host machine for detecting intrusions.

Network-based approaches can defend the machine against attack, as detection occurs before the data arrive at the machine.

## III. DISCUSSION

This section discusses various Intrusion Detection systems available.

Example of IDS are SNORT, BRO, NFR, Emerald, SAX2, and NetSTAT. Architectures of these IDS are explained below.

### *A. Snort [4], [9] [12]:*

Snort runs within numerous parts of networks, gathering data that can be collected at a single point. It also optimizes the hit rate, while keeping the false alarm rate at a minimum. Snort also offers easy-to-use reporting functions, real time detection and alerts and packet capture capabilities in best possible way.

Snort runs in three different modes: sniffer mode, packet logger mode, and Intrusion detection mode.

Running Snort in sniffer mode allows you to dump data in the header and body of each packet to the screen.

Packet logger mode is different from Sniffer mode in that in the former, packet data and/or headers are written to the hard drive of the host on which Snort runs. Sniffer or packet logger modes are appropriate for bulk data capture, but sorting through volumes of packet data to determine whether a security breach has occurred is not practical.

Snort's network intrusion detection mode does not record packets but, rather, allows rules that you select to be applied.

Snort will apply rules defined only in its rule set.

Snort Components

When discussing the internals of Snort, Figure 1 often helps to clarify the components at work and offers a high-level view of the Snort process.

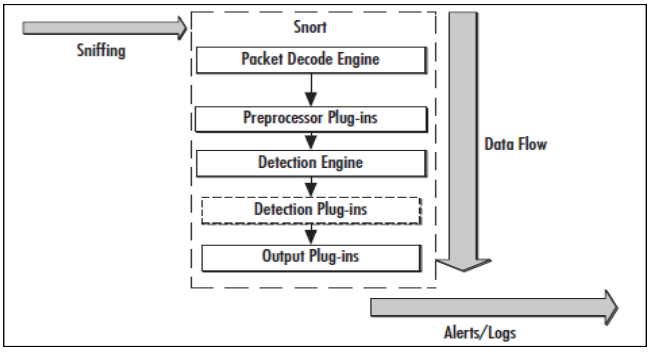


Figure 1 Snort Component Overview

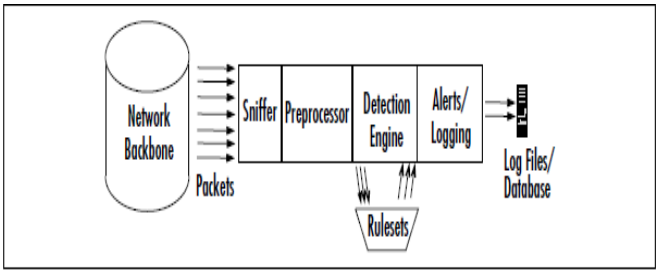


Figure 2 Snort components

The following are the four main components of Snort and the Snort process:

Packet capture/decoder engine First, traffic is acquired from the network link via the libpcap library. Packets are passed through the decode engine that first fills out the packet structure for the link-level protocols, which are then further decoded for higher-level protocols such as TCP and UDP ports.Preprocessor plug-ins Packets are then sent through a set of preprocessors. Packets are examined and manipulated before being handed to the detection engine. Each preprocessor checks to see if this packet is something it should look at, alert on, or modify.

Detection engine Packets are then sent through the detection engine. The detection engine checks each packet against the various options listed in the Snort rules files by performing single, simple tests on an aspect or field of the packet.The detection plug-ins provide additional detection functions on the packets. Each of the keyword options in the rule is linked to a detection plug-in that can perform additional tests.

Output plug-ins Snort then outputs the alerts from the detection engine, preprocessors or the decode engine.

The Table 1 shows Components of NIDs and the functions of every component.

Table 1 functions of components of NIDS (SNORT)

| **Name** | **Description** |
|---|---|
| Packet Decoder | Prepares packets for processing. |
| Preprocessors or Input Plugins | Plugins Used to normalize protocol headers, detect anomalies, packet reassembly and TCP stream re-assembly. |
| Detection Engine | Applies rules to packets. |
| Logging and Alerting System | Generates alert and log messages |
| Output Modules | Process alerts and logs and generate final output. |

All Snort rules have two logical parts: rule header and rule options. This is shown in following figure 3.


Figure 3 The basic structure of snort Rules

The rule header contains information about what action a rule takes. It also contains criteria for matching a rule against data packets. The options part usually contains an alert message and information about which part of the packet should be used to generate the alert message. The options part contains additional criteria for matching a rule against data packets. A rule may detect one type or multiple types of intrusion activity. Intelligent rules should be able to apply to multiple intrusion signatures.

The general structure of a Snort rule header is shown in following Figure 4.

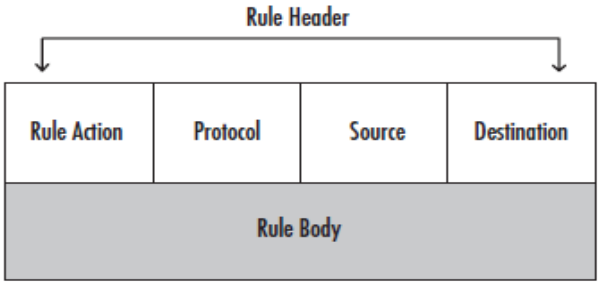


Figure 4 Structure of snort Rule header

The action part of the rule determines the type of action taken when criteria are met and a rule is exactly matched against a data packet. Typical actions are generating an alert or log message or invoking another rule.

The protocol part is used to apply the rule on packets for a particular protocol only. This is the first criterion mentioned in the rule. Some examples of protocols used are IP, ICMP, UDP, TCP etc.

The address parts define source and destination addresses. Addresses may be a single host, multiple hosts or network addresses. there are two address fields in the rule. Source and destination addresses are determined based on direction field. As an example, if the direction field is "->", the Address on the left side is source and the Address on the right side is destination. In case of TCP or UDP protocol, the port parts determine the source and destination ports of a packet on which the rule is applied. In case of network layer protocols like IP and ICMP, port numbers have no significance.

The direction part of the rule actually determines which address and port number is used as source and which as destination. For example, consider the following rule that generates an alert message whenever it detects an ICMP1 ping packet (ICMP ECHO REQUEST) with TTL equal to 100.

alert icmp any any -> any any (msg: "Ping with TTL=100"; ttl: 100;)

The part of the rule before the starting parenthesis is called the rule header. The part of the rule that is enclosed by the parentheses is the options part. The header contain the following parts, in order:

A rule action. In this rule the action is "alert", which means that an alert will be generated when conditions are met. Remember that packets are logged by default when an alert is generated. Depending on the action field, the rule options part may contain additional criteria for the rules.

The five rule actions created by default are:

***a.*** ***Pass:*** The pass action simply ignores the packet, and then analysis continue to execute on further captured packets.

***b.*** ***Log:*** The log rule action allows you to log the packet in a manner that you can specify during the configuration of your Snort sensor.

***c.*** ***Alert:*** The alert rule action logs the packet in the same manner as the Log action, and then alerts the user in a manner specified during configuration time. Alerts can be powerful actions and should be used efficiently. An alert log that is too large might prove be a nuisance or an ineffective mechanism for protecting your network.

***d. Dynamic:*** The dynamic action is unique in that it remains dormant until an Activate rule triggers it "on." After it is triggered, it then acts like a Log action rule.

***e. Activate:*** The activate action is the most powerful rule action created by default within Snort, because when triggered, it generates an alert and then starts the specified dynamic rule.These can be an excellent for catching complex attacks, intruders using a variety of tools, or even for categorizing data in a different manner.

In addition to these five rule options, you can create custom rule types. These rule types determine how other applications output the data to other types of output plug-ins. The format is straightforward. First, designate the rule type, and then the actions that you want to occur when the rule action is specified. For example, the following rule provides for the creation of a text file log when a defined hacker anomaly is detected:

```
ruletype hacker_log
{
type log
log_tcpdump: hacker.txt
}
```

This rule is written to send an alert to two different logs when the Gabriel virus is detected:

```
ruletype gabriel_virus
{
type alert output
alert_syslog: LOG_AUTH LOG_ALERT
log_tcpdump: gabriel_virus.log
}
```

Protocol. In this rule the protocol is ICMP, which means that the rule will be applied only on ICMP-type packets. In the Snort detection engine, if the protocol of a packet is not ICMP, the rest of the rule is not considered in order to save CPU time. The protocol part plays an important role when you want to apply Snort rules only to packets of a particular type.

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

a) IP
b) ICMP
c) TCP
d) UDP

If the protocol is IP, Snort checks the link layer header to determine the packet type. If any other type of protocol is used, Snort uses the IP header to determine the protocol type. The protocols only play a role in specifying criteria in the header part of the rule. The options part of the rule can have additional criteria unrelated to the specified protocol. For example, consider the following rule where the protocol is ICMP.

```
alert icmp any any -> any any (msg: "Ping with \
TTL=100"; ttl: 100;)
```

The options part checks the TTL (Time To Live) value, which is not part of the ICMP header. TTL is part of IP header instead. This means that the options part can check parameters in other protocol fields as well. Snort Probably the most-widely deployed NIDS. Snort is the de-facto standard among open-source systems.

Source address and source port. In this example both of them are set to "any", which means that the rule will be applied on all packets coming from any source. Of course port numbers have no relevance to ICMP packets. Port numbers are relevant only when protocol is either TCP or UDP.

Direction. In this case the direction is set from left to right using the -> symbol. This shows that the address and port number on the left hand side of the symbol are source and those on the right hand side are destination. It also means that the rule will be applied on packets traveling from source to destination. You can also use a <- symbol to reverse the meaning of source and destination address of the packet. Note that a symbol <> can also be used to apply the rule on packets going in either direction.

Destination address and port address. In this example both are set to "any", meaning the rule will be applied to all packets irrespective of their destination address. The direction in this rule does not play any role because the rule is applied to all ICMP packets moving in either direction, due to the use of the keyword "any" in both source and destination address parts.

The options part enclosed in parentheses shows that an alert message will be generated containing the text string "Ping with TTL=100" whenever the condition of TTL=100 is met. Note that TTL or Time To Live is a field in the IP packet header.

BRO an open source IDS[11,2]

Bro is a very flexible open-source research system. It provides the starting point for much of our work. Bro is one of the most flexible open-source NIDSs. It is designed and primarily developed by Vern Paxson. In contrast to most NIDSs, it is fundamentally neither an anomaly-based system nor a misuse-based system. Rather, its core is policy-neutral. Bro is written in C++ and covered by a BSD-style license. Bro's primary design goals were (i) separation of mechanism and policy, (ii) efficient operation suitable for high-volume network monitoring, and (iii) resistance to attacks directed at itself.

Architecture of BRO

To this end, Bro's architecture consists of three layers: packet filtering, event generation, and policy script execution.

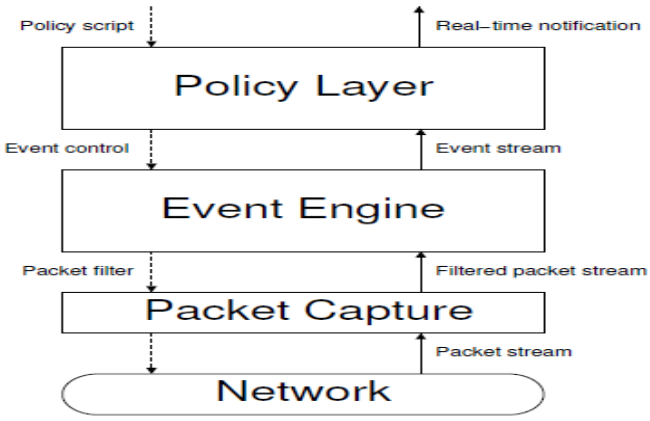


Figure 5 Architecture of BRO

There are three main layers in Bro:

a. ***Packet Capture Unit-*** This unit can be thought as detection unit. It uses libpcap to capture packets from the network. The use of libpcap isolates Bro from the underlying network technology and makes it portable.
b. ***Event Engine-*** It analyzes packet streams captured by the Package Capture Unit, verifies their integration and sends them to the appropriate handler. Handlers are provided by the policy script interpreter.
c. ***Policy Script Interpreter-*** Policy Script Interpreter runs scripts written in Bro language and associated with a handler. Whenever an event arrives, it executes related handler script. This script may execute other arbitrary commands to log events, modify state or record a data.

As Bro is designed to deal with attacks against itself, it defends itself against 3 kinds of different attacks. The first one, overload attack tries to overload the IDS monitor by sending so many packets that exceeds the processing capacity of the IDS.

With increasing layer, the volume of processed data decreases, thereby enabling more expensive processing. Packet filtering is done using a static BPF expression ,leveraging libpcap like Snort. The event engine generates events which in the sense NIDSs often do not make their decisions based on individual network packets but on events which are (policy-neutral) abstractions of network activity. Typical events include the establishment of a connection or the download of a file. If an event (or a sequence of events) violates a site policy, the NIDS should trigger an alert. represent policy-neutral abstractions of network activity at different semantic levels. For example, there are events for attempted/established/terminated/rejected connections, the requests and replies for a number of applications, and successful and unsuccessful user authentication. The user writes policy scripts using a specialized, richly-typed high-level language. These scripts contain event handlers which are executed when the corresponding event is raised. Event handlers codify the actions the NIDS should take:
Updating data structures describing the activity seen on the network, sending out real-time alerts,
Recording activity transcripts to files, and
Executing programs as a means of reactive response.

Bro's main unit of analysis is a connection. While for TCP the definition of a connection is straight-forward, the system also fits UDP and ICMP into its connection model by using a flow-like definition. Connection semantics are interpreted by analyzers which follow the endpoint's communication, extracting basic semantic protocol elements and generating corresponding events. Usually, an analyzer consists of two components:

One inside the event engine which performs policy-neutral analysis and generates events; and
a second component in the form of a policy script containing (predefined yet customizable) policy-specific actions.

Analyzers are available for a wide range of transport- and application-layer protocols, including TCP, HTTP, FTP, SMTP, SSL, and DNS. Moreover, protocol-independent analyzers detect scanners, stepping stones, backdoors etc. A generic connection analyzer generates one-line ASCII summaries of any connection the system sees. To reduce the processing load, Bro examines only packets requested by at least one of the analyzers; others are filtered at the lowest layer by installing a suitable filter. For instance, the connection analyzer requests only TCP control packets (i.e., SYNs/FINs/RESETs) rather than all TCP packets; these are sufficient to deduce basic connection characteristics such as duration and size of the transferred payload. As noted above, Bro was designed to resist attacks against itself. One consequence of this design guideline is the need to avoid predictability; e.g., while the systems needs to expire old state, an adversary should not be able to predict when it does it. To achieve this, Bro's design assumes that an attacker is not aware of the system's concrete parameterization. Therefore, the user needs to adapt the default policy scripts shipped with Bro before using them. This is a variant of Kirchhoff's principle: while the detection mechanisms are public (Bro is open-source), their parameterizations are not. In general, during Bro's development, attack resilience always had priority. In the past, Bro has been a non-distributed NIDS.
In summary, Bro is a policy-neutral research system which provides a large degree of flexibility to experiment with different approaches to network intrusion detection.

### *B. Emerald [11]:*

Emerald is an intrusion detection framework developed by SRI International. Its primary target environment is a large-scale heterogeneous enterprise network, consisting of several independent sub-units with di_ering trust relationships. Emerald is not freely available. Emerald is a highly-distributed system that uses a 3-layer architecture:
The service analysis layer
The domain-wide analysis layer
The enterprise-wide analysis layer

The service analysis layer monitors individual hosts and network components,

The domain-wide analysis layer correlates results across a sub-unit's service layer, and

The enterprise-wide analysis layer coordinates several sub-units.

On all three layers, monitors make up the basic building blocks. Each monitor contains modules for data acquisition, detection, and correlation. All monitors are built from the same code-base; resource objects encapsulate input specifics. On the service layer, a monitor acquires its input from the monitored components. On higher layers, it communicates with other monitors. The communication between monitors uses a subscription-based scheme to exchange asynchronous messages. The communication model provides both \push" and \pull" semantics. Inside a monitor, several detection components can be used which include an anomaly detector, and a misuse detector (based on the expert system).

In general, an Emerald set-up may include both host-based and network-based service-level monitors. So Emerald is Hybrid IDS (HIDS and NIDS).

### *C. STAT Open Source IDS (Net STAT) [2]:*

STAT is an open source IDS. An attack scenario consists of states and transitions. The states represent snapshots of a

system's security relevant characteristics. Transitions from one state to another are triggered by actions which represent steps of an attack. If there is a series of transitions leading from an initial starting state to a "compromised" ending state, a successful attack has been detected. Initially, this state-transition approach was used independently for host- and network-based Detection. The NetSTAT is a real time,distributed network based intrusion detection system. Unlike other network based intrusion detection systems that monitor a single subnetwork for patterns representing malicious activity, NetSTAT is oriented towards the detection of attacks in complex networks composed of several subnetworks. It detects intrusions in real time and monitors events where they are observable. network-based module of STAT, NetSTAT, consists of four types of components:

The network fact base stores all security relevant network information;

The scenario database contains state transition diagrams;

Probes are installed at points of interest across the network to detect attacks; and an analyzer pre-calculates the probes configurations. The network fact base component stores and manages the security relevant information about a network. The fact base is a stand-alone application that is used by the Network Security Officer to construct, insert, and browse the data about the network being protected. It contains information about the network topology and the network services provided.

The state transition scenario database is the component that manages the set of state transition representations of the intrusion scenarios to be detected. The state transition scenario database can be executed as a stand-alone application that allows the Network Security Officer to browse and edit state transition diagrams using a user friendly graphic interface.

The probes are the active intrusion detection components. They monitor the network traffic in specific parts of the network, following the configuration they receive at startup from the analyzer. Probes are general purpose intrusion detection systems that can be configured remotely and dynamically following any changes in the modelled attacks or in the implemented security policy.

The analyzer is a stand-alone application used by the Network Security Officer to analyze and instrument a network for the detection of a number of selected attacks. It takes as input the network fact base and the state transition scenario database and determines, which events have to be monitored, where the events need to be monitored, what information must be maintained about the state of the network in order to be able to verify state assertions, etc.

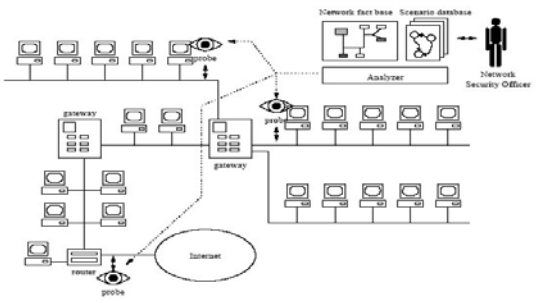


Figure 6 NetSTAT architecture

### *D. NFR Security [11]:*

The NFR software differs from most other products, as it provides analysis of data starting at the packet level as a sniffer would, and then it provides stateful packet inspection, misuse detection and protocol anomaly detection using a scriptable open source language called N-Code. NFR can operate as a true IDS hybrid solution, inspecting everything in the OSI reference model from layer 2 protocols up to layer 7 applications.

NFR Methodology does not believe that examining TCP header information will provide sufficient information to successfully detect an attack. In fact, it is necessary to buffer the entire connection, including the headers and bodies of the messages transmitted, to truly identify what is happening in a connection, and most IDS products fail in that respect. Solutions that do not guarantee that data is reassembled correctly have an almost impossible time ensuring the correctness of the originating data.

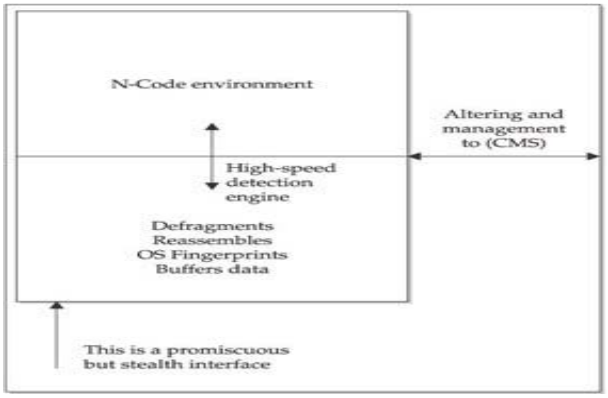


Figure 7 NFR Detection mechanism

NFR's methodology is focused on achieving data correctness during the capture and

decoding of packets. NFR provides a two-layer detection mechanism, shown in Figure 2. The lower layer is a high-speed engine with advanced buffering techniques that ensures proper state is maintained in packet transactions, data is correctly defragmented, and entire message bodies are recorded and reassembled in sequence for accurate detection and analysis. The upper layer provides a scriptable detection language. N-Code is a unique detection language that provides a rapid signature-development platform for intrusion detection.

## IV. CONCLUSION

When the computer is connected to network, there has to be more security provided to it. There are many tools available now a day's using which many attacks can happen easily. So to detect the attacks it requires and counter measures of these attacks. There are Intrusion detection system designed to detect the attack and also to store the new attack signatures in to the log file. Snort is the De-facto standard for IDS. And It is open source software. So it is used for research work. Snort is rule-based Intrusion detection system. It is single layer system. The Snort is de-facto standard for IDS and It is open source software. Snort is light weighted Single layer architecture. It is rule based system. All rules specifies what action to be

performed and when to be performed. NFR is distributed Inspector IDS. It is Hybrid because it operates in both signature based and anomaly based methods. It is best suited for small network. Emerald is also distributed anomaly based IDS system. It is also operating in both signature and anomaly detection methods, so it is referred to as hybrid system. This IDS is best suited for complex and large network. NetSTAT is realtime multilayer network Intrusion detection system. This NIDS operates on event State transitions. The events state transition are stored. It uses DFA to represent state transitions. This NIDS is best suitable for Large and complex network. BRO is multilayer Intrusion detection system. It is core policy-neutral. It supports customize policy script. It is best suitable for small and medium network. SAX-2[1] is single layer real-time network intrusion detection and prevention system. It is also hybrid IDS. It is best for both IT professionals and novice users.

The following table shows the comparison of open source IDS as well as the Emerald and SAX2 which is not open source IDS systems.

Table I Comparison Table for Architectural differences between different IDS

| | SNORT | NFR | Emerald | NetSTAT[2] | BRO | SAX2[2] |
|---|---|---|---|---|---|---|
| **Architecture** | Single layer | Distributed | Distributed | Multi layer | Multi layer | Single layer |
| **Framework** | Light weighted NIDS | Inspector based IDS | Distributed anomaly-based intrusion detection system | real time,distributed network based intrusion detection system | BRO is fundamentally neither an anomaly-based system nor a misuse-based system. Rather, its core is policy-neutral. | Real-time Network Intrusion detection and prevention system |
| **Detection capabilities** | Rule based/ Signature based | Hybrid (Signature and Anomaly based) | Hybrid (Signature and Anomaly based) | Event state-transition approach | events which are (policy-neutral) abstractions of network activity | Rule based/Signature based |
| **Scalability and flexibility** | Not much | yes | yes | yes | Yes | Not much |
| **Database required** | Signature database is large and regular updating | Large and regular updating | Depends on third party products | The set of state transition is stored in state transition scenario database | recording activity transcripts to files, and writing policy script. | Signature database is large and constantly updating |
| **Regular updates** | Required | Required | Not required | Not required | required | Required |
| **Best for** | Small and medium network | Small and medium network | Large network | complex networks | Small and medium network | both IT professionals and novice users |
| **Support** | Open source product | Commercial product | Research product | Open source product | Research product | Commercial tool |

## V. REFERENCES


[1] Intrusion Deection System using Sax 2.0 and wireshark 1.2.2

[2] Nong Ye, Senior Member, IEEE, Syed Masum Emran, Qiang Chen, and Sean Vilbert(2002),"Multivariate Statistical Analysis of Audit Trails for Host-Based Intrusion Detection", IEEE Transactions on Computers, Vol. 51, No. 7, July 2002.

[3] George Lawton, "Open Source Security: Opportunity or Oxymoron?" March 2002.

[4] K. Salah A. Kahtani (2009), "Improving Snort performance under Linux", IET Commun., 2009, Vol. 3, Iss. 12.

[5] Fang Yu, T. V. Lakshman, Randy H. Katz (2006), "Efficient Multimatch Packet Classification for Network Security Applications", IEEE Journal on Selected Areas in Communications, Vol. 24, NO. 10, October 2006.

[6] Jianchao Han, Mohsen Beheshti, Kazimierz Kowalski, Joel Ortiz, Johnly TomeldenComponent-based Software Architecture Design for Network Intrusion Detection and Prevention System, 2009 IEEE Computer society Sixth International Conference on Information Technology: New Generations 2009.

[7] David J., Chaboya, Richard A. Raines, Rusty O. Aldwin, and Barry E. Mullins,"Network ntrusion etection Automated and Manual Methods Prone to Attack and Evasion", PUBLISHED by the IEEE Computer Society, 2006.

[8] Hui Li, Dihua Liu, "Research on Intelligent Intrusion Prevention System Based on Snort", International Conference on Computer, Mechatronics, Control and Electronic Engineering (CMCE) 2010.

[9] Snort Manual and Whitepapers on Rule Optimization, Detection, High-performance multi rule detection engine, Protocol Flow analyzer. All available at the Snort homepage: http://www.sourcefire.com/products/library.html

[10] Jiong Zhang, Mohammad Zulkernine, and Anwar Haque(2008), "Random-Forests-Based Network Intrusion Detection Systems", IEEE Transactions on Systems, man, and Cybernetics—Part C: Applications and Reviews, Vol. 38, NO. 5, September 2008

[11] Raghuram Ponnaganti "Comparative study of three IDS system (NFR, Emerald, Snort)"

[12] SNORT R Users Manual 2.9.1