CHAPTER -3

RESEARCH TOOLS

In this chapter we describe about the working environment in which the research implementation work is done. Here we present the brief description of the research tools which are used.

3.1 Snort

3.1.1 Introduction to Snort

Snort is a free and open source Network Intrusion Detection (NIDS) capable of performing packet logging and real-time traffic analysis on IP networks. Snort performs protocol analysis, and content searching/matching, it is commonly used to actively block or passively detect a variety of attacks and probes, such as buffer overflows, stealth port scans, web application attacks, SMB probes, and OS finger printing attempts amongst other features.

3.1.2 Snort Internal Structure

The module of Snort is illustrated in Figure 3.1[83]. Each module does the following:

- **Packet Capture Module** is based on the popular packet programming library libpcap. It provides a high-level interface to packet capture.

- **Decoder** fits the captured packets into data structures and identifies link level protocols. Then, it takes the next level, decodes IP, and then TCP or UDP (depending on the case) in order to get useful information like ports and addresses. Snort will alert if it finds malformed headers (unusual length TCP options, etc.)

- **Pre-processors** could be seen like some kind of filter, which identifies things that should be checked later in the next modules e.g. the Detection Engine,

[77]
such as suspicious connection attempts to some TCP/UDP ports or too many TCP SYN packets sent in a short period of time (port scan).
Pre-processors function is to take packets potentially dangerous for the detection engine to try to find known patterns.

**Figure-3.1 Snort Modules**

- **Rules Files** are plain text files which contain a list of rules with a known syntax. This syntax includes protocols, addresses, output plug-ins associated and some other things. These rules files can be updated.

- **Detection Plug-ins** are modules referenced from its definition in the rules files. They are used to identify patterns whenever a rule is evaluated.

- **Detection Engine** make use of the detection plug-ins, it matches packets against rules loaded into memory during Snort initialization.

- **Output Plug-ins** are the modules which allow formatting the notifications (alerts, logs) for the user to access them in many ways (console, databases, etc).
3.1.3 Snort Modes

Snort can be configured to run in three modes:[84]:

- **Sniffer mode**, which reads the packets off of the network and displays them in a continuous stream on the console (screen)
- **Packet Logger mode**, which logs the packets to disk.
- **Network Intrusion Detection System (NIDS) mode**, the complex and configurable configuration, which allow Snort to analyze network traffic for matches against a user-defined rule set and performs several actions based upon what it sees.

3.1.4 Prerequisites for Snort Installation

The essential package requirements for the installation of snort are:

- Nmap
- Nbtscan
- Apache2
- Php5
- Php5mysql
- Libpcap0.8dev
- Libpcre3dev
- g++
- Bison
- Flex
- Libpcapruby
- Libdnet
- DAQ (Data Acquisition API)

3.2 Snort Initialization

Fig 3.2 shows the initialization of snort when DAQ is configured in passive mode with various pre-processor objects.
3.3 Installation of snort

Configuration: Snort follows a unixy configuration philosophy. Its configuration consists of global configuration (snort.conf) with optional rules file(s) and additional files. Snort Rules are an important part of the Snort. Main sources of Snort rules are Sourcefire VRT Rules which are known as standard rules and are called, the “gold standard”. Another ones are Community Rules which are contributed by snort users. Rules are the strength of Snort because these are continuously changed at regular intervals of time as per the security need. Basically Bleeding Edge Snort is user framed rules and their focus is on quick releases with minimal testing, but these provide breakthrough for knowing about new threats and for experimental detection. Steps of installation are as follows

- Preparation
- Installation
- Setup the network interface to be used for sniffing traffic in promiscuous mode
- Install Snort
3.3.1 Test Bed Preparation

In our test framework we had made one machine fully functional with Ubuntu 14.04 installed on it and it can run with at least two network interfaces: one network interface dedicated to managing the machine and connected to the actual LAN and the other network interface dedicated for sniffing network traffic. We made network interface that will be sniffing network traffic is eth0 and the interface for managing the machine will be wlan0.

3.4 Snort Alerting and Authenticity

Alerting part of the Alerting and Logging subsystem is activated at run-time. The system administrator can verify that a Snort-generated alert is valid.

To fully trust an intrusion detection system alert, examine three complementary data points: a rule representing behaviour that is known or suspect of being anomalous, an alerting message warning of a rule violation or of particular behaviour, and a network packet (or series of packets) causing the rule violation. In lieu of having all three components relate the network packet to at least one of the following:

1. The alerting message
2. The rule violated

If neither of these relationships occurs, it is precluded from characterizing the security event as a positive or negative. The network packet(s) is a critical point-of-examination and without the actual tangible packet(s) we cannot investigate whether a rule is violated or an alerting message displays a positive occurrence of violation to our security policy. Therefore, because Snort allows configuring various levels of alerting and logging, Snort is set up to log the offending packets causing the rule violation and to record the alerts in separate files. When Snort inspects a network packet and detects a match between a rule (describing a violation) and the network packet, Snort sends an alerting message to the user-defined facility and/or logs the packets causing the rule violation. The alerts "may
either be sent to syslog logged to an alert text file in two different formats, or sent as WinPopup messages using the Samba smbclient program. The syslog alerts are sent as security/authorization messages that are easily monitored with tools such as swatch. WinPopup alerts allow event notifications to be sent to a user-specified list of Microsoft Windows consoles running the WinPopup software. There are two options for sending the alerts to a plain text file; full and fast alerting. Full alerting writes the alert message and the packet header information through the transport layer protocol. The fast alert option writes a condensed subset of the header information to the alert file, allowing greater performance under load than full mode. There is a fifth option to completely disable alerting, which is useful when alerting is unnecessary or inappropriate, such as when network penetrations tests are being performed.

Similarly, logging can be set up to log packets in their decoded human-readable format to an IP-based directory structure, or in tcpdump binary format to a single log file. The decoded format logging allows fast analysis of data collected by the system. The tcpdump format is much faster to record to the disk and should be used in instances where high performance is required. Logging can also be turned off completely, leaving alerts enabled for even greater performance improvements.

To put this into perspective, examine the logging and alerting areas of a system. The system sends alerts to the syslog facility and the offending network packet(s) to an IP-based directory structure. All alerts are logged via syslog to a file called "alerts" in the file /var/log/snort/alerts. Any alerting message found in this file will have corresponding offending network packets logged in the same directory as the alert file but under the IP address of the source packet.

When Snort inspects and matches the above rule to an offending network packet(s), an alerting message is sent to syslog stating that a "mountd access" violation has occurred. This message is recorded in the file /var/log/snort/alerts and the actual network packet(s) causing the alert is recorded in a file based on the
source IP address of the offending packet(s), (i.e. /var/log/snort/a.b.c.d).

Some problems may occur when filtering log entries into an IP-named file. For one, multiple alerts may involve one IP address. Under this condition, the offending packets violating each unique rule are sent to the same IP-named file; and mapping the specific alert to the offending packet(s) then demands a search and locates approach that could be time consuming. Snort's output can be used in concert with the output of other security tools to do the following:

1. corroborate a security event that happened by returning the process and user information of the event
2. Identify discrepancies when processes that are reported by programs such as ntop and tcpdump are analyzed.

3.5 Writing Snort Rules

Snort rules are divided into two logical sections, the rule header and the rule options. The rule header contains the rule's action, protocol, source and destination IP addresses and netmasks, and the source and destination ports information. The rule option section contains alert messages and information on which parts of the packet should be inspected to determine if the rule action should be taken.

Rule Headers

**Rule Actions**: The rule header contains the information that defines the "who, where, and what" of a packet, as well as what to do in the event that a packet with all the attributes indicated in the rule should show up. The first item in a rule is the *rule action*. The rule action tells Snort what to do when it finds a packet that matches the rule criteria. There are three available actions in Snort alert, log and pass.

- alert - generate an alert using the selected alert method, and then
log the packet
  • log - log the packet
  • pass - drop (ignore) the packet

Protocols: The next field in a rule is the protocol. There are three IP protocols that Snort currently analyzes for suspicious behaviour: tcp, udp, icmp and ip.

IP Addresses: The next portion of the rule header deals with the IP addresses and port information for a given rule. The keyword "any" may be used to define any address. Snort does not have a mechanism to provide host name lookup for the IP address fields in the rules file. The addresses are formed by a straight numeric IP address and a CIDR block. The CIDR block indicates the netmask that should be applied to the rule's address and any incoming packets that are tested against the rule. A CIDR block mask of /24 indicates a Class C network, /16 a Class B network, and /32 indicates a specific machine address. For example, the address/CIDR combination 192.168.1.0/24 would signify the block of addresses from 192.168.1.1 to 192.168.1.255. Any rule that used this designation for, say, the destination address would match on any address in that range. The CIDR designations give us a nice short-hand way to designate large address spaces with just a few characters.

Port Numbers: Port numbers may be specified in a number of ways, including "any" ports, static port definitions, ranges, and by negation. "Any" ports are a wildcard value, meaning literally any port. Static ports are indicated by a single port number, such as 111 for portmapper, 23 for telnet, 80 for http, etc. Port ranges are indicated with the range operator "-:". The range operator may be applied in a number of ways to take on different meanings.
**The Direction Operator:** The *direction operator* "->" indicates the orientation, or "direction", of the traffic that the rule applies to. The IP address and port numbers on the left side of the direction operator is considered to be the traffic coming from the source host, and the address and port information on the right side of the operator is the destination host. There is also a *bidirectional operator*, which is indicated with a "<>", symbol. This tells Snort to consider the address/port pairs in either the source or destination orientation. This is handy for recording/analyzing both sides of a conversation, such as telnet or POP3 sessions.

**Rule Options:** Rule options form the heart of Snort's intrusion detection engine, combining ease of use with power and flexibility. All Snort rule options are separated from each other using the semicolon ";" character. Rule option keywords are separated from their arguments with a colon ":" character. There are fifteen rule option keywords available for Snort:

- **msg** - prints a message in alerts and packet logs
- **logto** - log the packet to a user specified filename instead of the standard output file
- **minfrag** - set a threshold value for the smallest acceptable IP fragment size
- **ttl** - test the IP header's TTL field value
- **id** - test the IP header's fragment ID field for a specific value
- **dsize** - test the packet's payload size against a value
- **content** - search for a pattern in the packet's payload
- **offset** - modifier for the content option, sets the offset to begin attempting a pattern match
- **depth** - modifier for the content option, sets the maximum search depth for a pattern match attempt
- **flags** - test the TCP flags for certain values
- seq - test the TCP sequence number field for a specific value
- ack - test the TCP acknowledgement field for a specific value
- itype - test the ICMP type field against a specific value
- icode - test the ICMP code field against a specific value
- session - dumps the application layer information for a given session

# msg: The msg rule option tells the logging and alerting engine the message to print along with a packet dump or to an alert. It is a simple text string that utilizes the "\" as an escape character to indicate a discrete character that might otherwise confuse Snort's rules parser (such as the semi-colon ";" character).

Format:

```plaintext
msg: "<message text>";
```

# logto: The logto option tells Snort to log all packets that trigger this rule to a special output log file. This is especially handy for combining data from things like NMAP activity, HTTP CGI scans, etc. It should be noted that this option does not work when Snort is in binary logging mode.

Format:

```plaintext
logto: "<filename>";
```

# minfrag: Minfrag sets a minimum size threshold for a fragmented packet. It is generally used in conjunction with a single alert rule to set a boundary for the minimum fragment size that is acceptable on a network segment. It makes a handy detector for attackers that like to break their fragments into tiny pieces before transmitting them to try to avoid detection mechanisms. Generally
speaking, there is virtually no commercial network equipment available that generates fragments smaller than 256-bytes, so people can take advantage of this fact by setting their minfrag value somewhere below that threshold.

Format:

    minfrag: "<number>";

# ttl: This rule option is used to set a specific time-to-live value to test against. The test it performs is only successful on an exact match. This option keyword was intended for use in the detection of traceroute attempts.

Format:

    ttl: "<number>";

# id: This option keyword is used to test for an exact match in the IP header fragment ID field. Some hacking tools (and other programs) set this field specifically for various purposes, for example the value 31337 is very popular with some hackers. This can be turned against them by putting a simple rule in place to test for this and some other "hacker numbers".

Format:

    id: "<number>";

# dsize: The dsize option is used to test the packet payload size. It may be set to any value, plus use the greater than/less than signs to indicate ranges and limits. For example, if we know that a certain service has a buffer of a certain size, you can set this option to watch for attempted buffer overflows. It has the added advantage of being a much faster way to test for a buffer overflow than a payload content check.
# content: The content keyword is one of the more important features of Snort. It allows the user to set rules that search for specific content in the packet payload and trigger response based on that data. Whenever a content option pattern match is performed, the Boyer-Moore pattern match function is called and the (rather computationally expensive) test is performed against the packet contents. If data exactly matching the argument data string is contained anywhere within the packet's payload, the test is successful and the remainder of the rule option tests are performed. This test is case sensitive.

The option data for the content keyword is somewhat complex; it can contain mixed text and binary data. The binary data is generally enclosed within the pipe ("|"") character and represented as bytecode. Bytecode represents binary data as hexadecimal numbers and is a good shorthand method for describing complex binary data.

Format:

```
content: "<content string>";
```

# offset: The offset rule option is used as a modifier to rules using the content option keyword. This keyword modifies the starting search position for the pattern match function from the beginning of the packet payload. It is very useful for things like CGI scan detection rules where the content search string is never found in the first four bytes of the payload. Care should be taken against setting the offset value too "tightly" and potentially missing an attack. This rule option...
keyword cannot be used without also specifying a content rule option.

Format:

    offset: <number>;

# depth: depth is another content rule option modifier. This sets the maximum search depth for the content pattern match function to search from the beginning of its search region. It is useful for limiting the pattern match function from performing inefficient searches once the possible search region for a given set of content has been exceeded. e.g. if we are searching for "cgi-bin/phf" in a web-bound packet, we probably don't need to waste time searching the payload beyond the first 20 bytes.

Format:

    depth: <number>;

# flags: This rule tests the TCP flags for an exact match. There are actually 8 flags variables available in Snort:

- F - FIN (LSB in TCP Flags byte)
- S - SYN
- R - RST
- P - PSH
- A - ACK
- U - URG
- 2 - Reserved bit 2
- 1 - Reserved bit 1 (MSB in TCP Flags byte)

The reserved bits can be used to detect unusual behaviour, such as IP stack fingerprinting attempts or other suspicious activity. All of the flags are
considered as a whole for this test, they must all be "up" for this rule option to be successful.

Format:

    flags: <flag values>;

# seq: This rule option refers to the TCP sequence number. Essentially, it detects if the packet has a static sequence number set, and is therefore pretty much unused. It was included for the sake of completeness.

Format:

    seq: <number>;

# ack: The ack rule option keyword refers to the TCP header's acknowledge field. This rule has one practical purpose so far: detecting NMAP TCP pings. A NMAP TCP ping sets this field to zero and sends a packet with the TCP ACK flag set to determine if a network host is active.

Format:

    ack: <number>;

# itype: This rule tests the value of the ICMP type field. It is set using the numeric value of this field. It should be noted that the values can be set out of range to detect invalid ICMP type values that are sometimes used in denial of service and flooding attacks.

Format:

    itype: <number>;

# icode: The icode rule option keyword is pretty much identical to the itype rule, just set a numeric value and Snort will detect any traffic using that ICMP code value. Out of range values can also be set to detect suspicious traffic.
Format:

    icode: <number>;

# session: The session keyword is used to extract the user data from TCP sessions. It is extremely useful for seeing what users are typing in telnet, rlogin, ftp, or even web sessions. There are two available argument keywords for the session rule option, printable or all. The printable keyword only prints out data that the user would normally see or be able to type. The all keyword substitute non-printable characters with their hexadecimal equivalents. This function can slow Snort down considerably, so it shouldn't be used in heavy load situations, and is probably best suited for post-processing binary (tcpdump format) log files.

Format:

    session: [printable|all];

Though Snort has a rich command language but still there are gaps and there is a scope for bridging the gaps of snort by building good rules. From the study made above in this chapter pitfalls of Snort are as under

    When Snort works in its active detection mode it occupies the processor fully thereby slowing down the performance of the system. With increased number of rules, memory consumption also increases and hence will take longer to initialize all the rules. Each and every field mentioned in the rule is checked by Snort normally. It will decrease the processing throughput by performing several unnecessary comparisons with all the fields in the rule.

    Automata of the Snort are built by considering every attribute present in the rule. This results in a much deeper tree structure requiring more processing power and memory in order to compare the rules and for storage of the automata in the memory respectively, thereby resulting in an increased search time, processor and
memory use.

By default snort will not provide any anomaly detection and is purely a misuse based system. Extra plug-in is required as Snort is purely an intrusion detection system. Snort will start to drop the packets at a massive rate when the incoming packet rate is more. Therefore possibilities of detecting possible attack patterns are less since it fails to analyze those dropped packets. So work will be done in this direction that these gaps are bridged so that Snort works in an efficiently in all scenarios of security requirements.