Intrusion Detection Systems (IDS) - Snort

- Snort is an open source **Network Intrusion Detection System** (NIDS) that is is capable of performing **real-time traffic analysis** and **packet logging on IP networks**.

- Some of the features of Snort are:

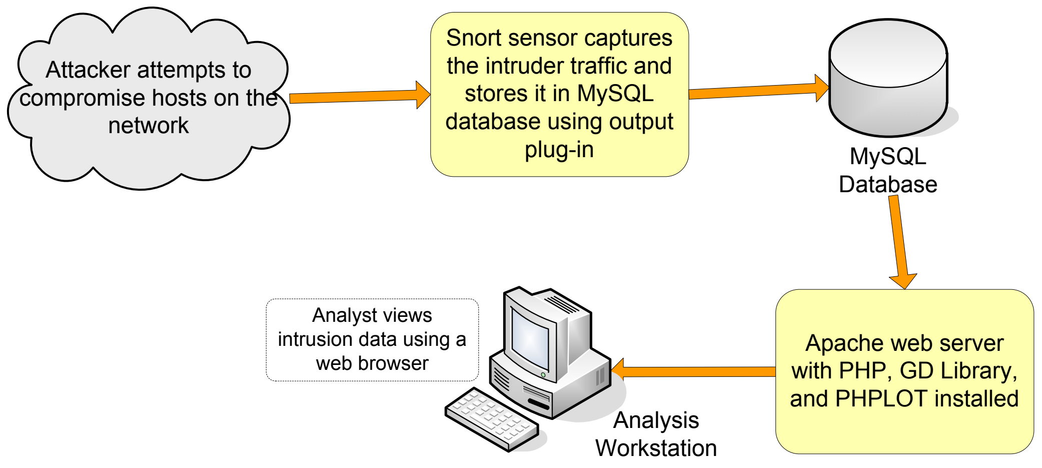
    - Protocol Analysis
    - Content searching / matching
    - Real-time alerting capability
    - Can read in a ***tcpdump*** trace and run against a rule set
    - Flexible rules language to describe traffic that it should collect or pass

- Snort can be combined with other products to design a complete IDS and analysis tool. These products include MySQL database (http://www.mysql.org) and Analysis Control for Intrusion Database (ACID) (http://www.cert.org/kb/acid).

- Snort has the capability to log data collected (such as alerts and other log messages) to a database. MySQL is used as the database engine where all of this data is stored.

- Using Apache web server and ACID, we can analyze this data. A combination of Snort, Apache, MySQL, and ACID makes it possible to log the intrusion detection data into a database and then view and analyze it later, using a web interface.

- The following diagram illustrates this model:


Attacker attempts to compromise hosts on the network
Snort sensor captures the intruder traffic and stores it in MySQL database using output plug-in
MySQL Database
Apache web server with PHP, GD Library, and PHPLOT installed
Analysis Workstation
Analyst views intrusion data using a web browser

A complete network intrusion detection system consisting of Snort, MySQL, Apache, ACID, PHP, GD Library and PHPLOT

- The diagram shown depicts a comprehensive Snort-based IDS that utilizes several tools to provide a web-based user interface with a backend database:

- MySQL is used with Snort to log alert data. Other databases like Oracle can also be used but MySQL is the most popular database with Snort. In fact, any ODBC-compliant database can be used with Snort.

    - Apache acts as a web server.

    - PHP is used as an interface between the web server and MySQL database.

    - ACID is a PHP package that is used to view and analyze Snort data using a web browser.

    - GD library is used by ACID to create graphs.

    - PHPLOT is used to present data in graphic format on the web pages used in ACID. GD library must be working correctly to use PHPLOT.

    - ADODB is used by ACID to connect to MySQL database.

- A system such as the one just described can provide administrators with enough data to make informed decisions on the proper course of action when suspicious activity is observed.

- Snort has also been ported to Win32 and is also being constantly updated with new features.

- An IDS is only effective against attacks it knows about or has signatures for. A clever attacker could slightly modify an attack and alter the signature of the attack and sneak past the watching eyes of the IDS.

- Snort has the ability to receive plug-ins, making it very effective when a new attack emerges and commercial security vendors are slow to release new attack recognition signatures.

- It features rules based logging to perform content pattern matching and detect a variety of attacks and probes, such as buffer overflows, stealth port scans, CGI attacks, SMB probes, and many more.

- Snort has real-time alerting capability, with alerts being sent to syslog, Server Message Block (SMB) "WinPopup" messages, or a separate "alert" file.

- Snort is configured using command line switches and optional Berkeley Packet Filter [BPF93] commands.

- The detection engine is programmed using a simple language that describes per packet tests and actions. Ease of use simplifies and expedites the development of new exploit detection rules.

- Snort is cosmetically similar to tcpdump but is more focused on the security applications of packet sniffing.

- The major feature that Snort has which tcpdump does not is packet payload inspection. Snort decodes the application layer of a packet and can be given rules to collect traffic that has specific data contained within its application layer.

- This allows Snort to detect many types of hostile activity, including buffer overflows, CGI scans, or any other data in the packet payload that can be characterized in a unique detection fingerprint.

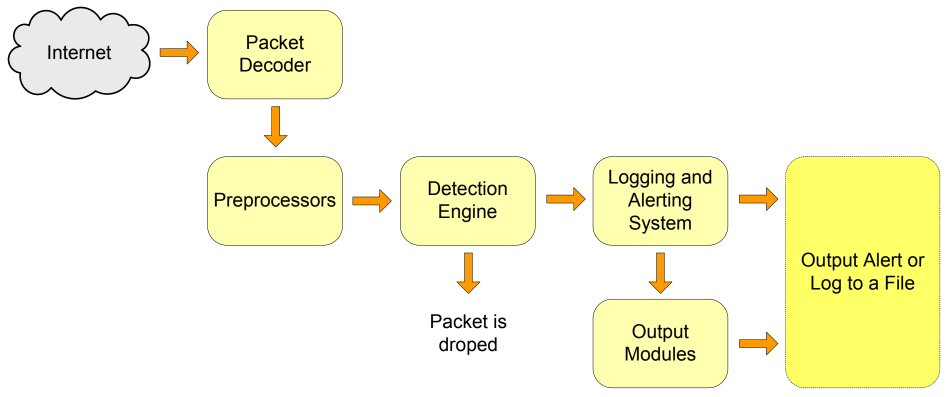
Snort Architecture

- Snort is logically divided into multiple components that work together to detect particular attacks and to generate output in a required format from the detection system.

- The IDS consists of the following major components:

    - Packet Decoder
    - Preprocessors
    - Detection Engine
    - Logging and Alerting System
    - Output Modules

- The diagram below shows how these components are positioned. A data packet coming from the Internet enters the packet decoder.



Components of Snort

- It is then preprocessed before being fed to the detection engine where it is either dropped, logged or an alert is generated.

Packet Decoder

- The packet decoder takes packets from different types of network interfaces and prepares the packets to be preprocessed or to be sent to the detection engine. The interfaces may be Ethernet, etc.

Preprocessors

- Preprocessors are components or plug-ins that can be used with Snort to format or modify data packets before the detection engine examines the packet to determine if it matches any of its rules.
- Some preprocessors also perform detection by identifying anomalies in packet headers and generating alerts.
- Preprocessors perform the essential function of preparing the incoming packets so that they can be analyzed against the rules in the detection engine.
- Attackers may use various techniques to fool an IDS. Say we have created a rule to find a signature "**scripts/iisadmin**" in HTTP packets.
- If the rule is matching this string exactly, it can be fooled by making slight modifications to this string:
    - "scripts/./iisadmin"
    - "scripts/examples/../iisadmin"
    - "scripts\iisadmin"
    - "scripts/.\iisadmin"
- To complicate the situation, hackers can also insert in the web Uniform Resource Identifier (URI) hexadecimal characters or Unicode characters that are perfectly legal as far as the web server is concerned.
- Web servers can interpret all of these strings and are able to preprocess them to extract the intended string "scripts/iisadmin".
- However if the IDS is looking for an exact match, it is not able to detect this attack. A preprocessor can rearrange the string so that it is detectable by the IDS.

The Detection Engine

- The detection engine is the most important component in the Snort architecture. Its primary responsibility is to detect the presence of an intrusion signature in a packet by applying the rules to the packet.
- The rules are read into internal data structures or chains where they are matched against all packets.
- If a packet matches any rule, appropriate action is taken; otherwise the packet is dropped.
- Appropriate actions may be logging the packet or generating alerts.
- The detection engine is the time-critical part of Snort. Depending upon how powerful the senor machine is and how many rules have been defined, it may take different amounts of time to respond to different packets.

- If network traffic is too high when Snort is active in NIDS mode, it may drop some packets and may not get a true real-time response.

- The load on the detection engine depends upon the following factors:

    - Number of rules
    - Power of the sensor machine on which Snort is running
    - Internal bus speeds used in the Snort machine
    - Network traffic

- When designing a Network Intrusion Detection System, all of these factors must be considered when selecting the sensor hardware.

- The detection engine can dissect a packet and apply the rules to the different headers and payload within the packet:

- The **IP header** of the packet

- The **Transport layer header**

    - This header includes TCP, UDP or other transport layer headers. It can also work on the ICMP header.

- The **Application layer level header**

    - Application layer headers include, but are not limited to, DNS header, FTP header, SNMP header, and SMTP header.

- **Packet payload**

    - This implies that we have to create a rule that is used by the detection engine to find a string inside the data that is present inside the packet.

- Typically and IDS detection engine stops further processing of a packet when a rule is matched and takes appropriate action by logging the packet or generating an alert.

- This means that if a packet matches criteria defined in multiple rules, only the first rule is applied to the packet; the rest of the rules will be bypassed.

- Snort however matches all the rules against a packet before generating an alert. After matching all rules, the highest priority rule is selected to generate the alert.

Logging and Alerting System

- Depending upon what the detection engine finds inside a packet, the packet may be used to log the activity or generate an alert.

- Logs are kept in simple text files, tcpdump-style files or some other form. All of the log files are stored under /var/log/snort folder by default.

- The ***–l*** command line option can be used to modify the location of where the generating logs and alerts will be stored.

<u>Output Modules</u>

- Output modules or plug-ins can perform different operations depending on how we wish to save the output generated by the logging and alerting system.

- Basically these modules control the type of output generated by the logging and alerting system.

- Depending on the configuration, output modules can perform the following actions:

    - Log to the **/var/log/snort/alerts** file or some other file Sending SNMP traps
    - Send messages to **syslog** facility
    - Logging to a database like MySQL or Oracle.
    - Generating eXtensible Markup Language (XML) output
    - Modifying configuration on routers and firewalls.
    - Sending Server Message Block (SMB) messages to Microsoft Windows-based machines

- Other tools can also be used to send alerts in other formats such as e-mail messages or viewing alerts using a web interface.

- The following table summarizes different components of the IDS:

Name	**Description**
Packet Decoder	Prepares packets for processing.
Preprocessors or Input 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.

Snort Installation Scenarios

- Snort installations will vary depending upon the environment where the IDS is being deployed.
- Some of the typical installation schemes are:

Single Sensor IDS

- An installation of Snort with only one sensor is suitable for small home networks with only one Internet connection.
- The sensor is positioned behind a router or firewall and is used to detect the intruder activity into the network.
- If on the other hand, we are interested in scanning all Internet traffic, the sensor will be positioned outside the firewall.
- The sensor host on which Snort is installed must be dedicate dto the IDS task only and must be configured so that Snort starts automatically at boot time.
- The logging can be done in text or binary files and tools such as SnortSnarf can be used to analyze data.

Single Sensor IDS with a Database and Web Interface

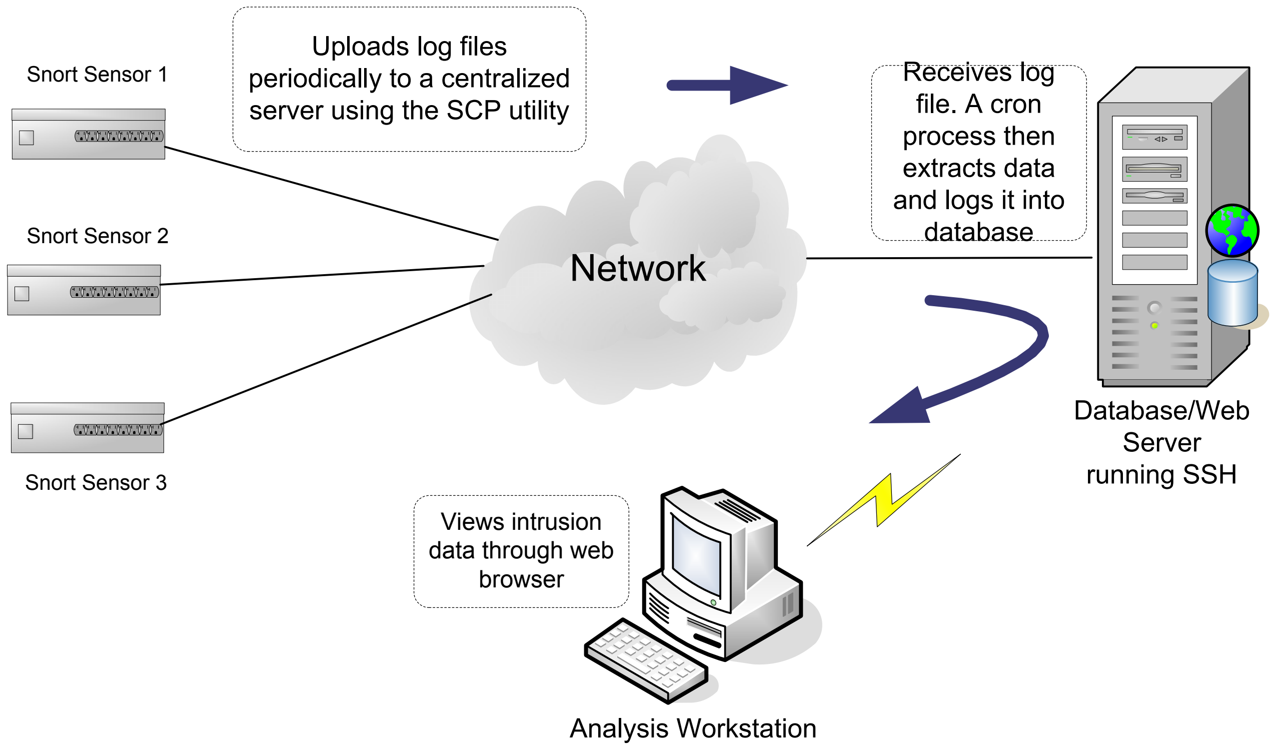
- For small to medium-sized business networks, a Snort-based IDS is best integrated with a database.
- The database is used to log Snort data where it can be viewed and analyzed more conveniently using a web-based interface.
- A typical setup of this type consists of three basic components:

  Snort sensor
  A database server
  A web server

- Different types of database servers like MySQL, PostgresSQL, Oracle, Microsoft SQL server and other ODBC-compliant databases can be used with Snort.
- PHP is used to get data from the database and to generate web pages.
- This configuration provides a very good and comprehensive IDS which is easy to manage and user friendly.

Multiple Snort Sensors with a Centralized Database

- Large corporate networks require multiple IDS sensors deployed at various strategic locations in the network.
- Managing all of these sensors and analyzing the large amount of data captured by these sensors separately is a very resource intensive task.
- The following diagram depicts a configuration that implements a distributed Snort installation.


Snort Sensor 1
Uploads log files periodically to a centralized server using the SCP utility
Receives log file. A cron process then extracts data and logs it into database
Snort Sensor 2
Network
Snort Sensor 3
Database/Web Server running SSH
Views intrusion data through web browser
Analysis Workstation

Distributed Snort installation using SCP and Barnyard

- All of the sensors must have access to the database at the time Snort is started. If Snort is not able to connect to the database at the start time, it will fail to start
- The database must be available all of the time to all sensors. If any of the network links are down, data will be lost.
- If there is a firewall lies between the database server and any of the sensors, additional ports will have to be allowed through the firewall to accommodate the database logging.

- Opening up firewall ports may violate network security policies, not to mention open up opportunities for attackers as well.

- Alternate mechanisms can be used where Snort sensors do not have a direct connection to the database server. The sensors may be configured to log to local files.

- These files can then be uploaded to a centralized server on a periodic basis using utilities like SCP. The SCP utility is a secure file transfer program that uses Secure Shell (SSH) protocol. Firewall administrators usually allow SSH port (port 22) to pass through.

- Utilities such as Barnyard or some other tools can be used to extract data from these log files and put it into the database server.

- Note that this centralized server must be running SSH server so that SCP utility will be able to upload files to this server.

Packet Sniffing Mode

- In its simplest form Snort can be used as a packet sniffer. To see how snort can be used to read packets off the subnet (similar to tcpdump) enter the following command:

```
# snort -v
Running in packet dump mode
Log directory = /var/log/snort

Initializing Network Interface eth0

        --== Initializing Snort ==--
Initializing Output Plugins!
Decoding Ethernet on interface eth0

        --== Initialization Complete ==--

-*> Snort! <*-
Version 2.2.0 (Build 30)
By Martin Roesch (roesch@sourcefire.com, www.snort.org)


07/11-20:33:59.675507 192.168.1.10:3436 -> 192.168.1.1:53
UDP TTL:64 TOS:0x0 ID:22707
Len: 36

07/11-20:33:59.916825 192.168.1.1.436 -> 192.168.1.10:3436
UDP TTL:57 TOS:0x0 ID:1177
Len: 91
```

```
07/11-20:33:59.917642 192.168.1.10:3437 -> 192.168.1.153
UDP TTL:64 TOS:0x0 ID:22708
Len: 47

07/11-20:34:00.078208 192.168.1.1:53 -> 192.168.1.10:3437
UDP TTL:57 TOS:0x0 ID:1367
Len: 109

^C
```

```
-*> Snort! <*-
Version 2.2.0 (Build 30)
By Martin Roesch (roesch@sourcefire.com, www.snort.org)
Snort received 343 packets
    Analyzed: 343(100.000%)
    Dropped: 0(0.000%)
========================================================================
========
Breakdown by protocol:
    TCP: 338        (98.542%)
    UDP: 5          (1.458%)
   ICMP: 0          (0.000%)
    ARP: 0          (0.000%)
  EAPOL: 0          (0.000%)
   IPv6: 0          (0.000%)
    IPX: 0          (0.000%)
  OTHER: 0          (0.000%)
DISCARD: 0          (0.000%)
========================================================================
========
Action Stats:
ALERTS: 0
LOGGED: 0
PASSED: 0
========================================================================
========
Snort exiting
```

- We can see from the transfer above that the client made a DNS request to the server.

NIDS Mode

- The real power of snort lies in its ability to read in a rule set, observe the traffic going across the wire, and detect if any of the traffic matches any of the rules.

- Rules can be created that watch for pings, scans, backdoor attempts, cgi-attempts, and many other common methods attackers use to gain control of a target machine.

- Alerts can be logged to a file specified from the command line or even sent through syslog and appended to your system messages. This can all be run as a background (daemon) process.

- Rule sets are written by hand there are freely available rule sets available on the snort homepage at http://www.snort.org/.

- To enable network intrusion detection (NIDS) mode we can use the following command for example:

  snort -Afull –D –X -c snort.conf

- Where snort.conf is the name of the rules file (usually found in **/snort-2.x.x/etc/**) This will apply the rules set in the **snort.conf** file to each packet to decide if an action based upon the rule type in the file should be taken.

- The **–Afull** switch sets it to full alerts.

- The **–D** switch runs it in daemon mode.

- The **–X** switch captures the complete hex dump of the packet (not a good idea on a very busy network).

- If you don't specify an output directory for the alerts, the program will default to **/var/log/snort**.

- The first time you run snort with a rule set it may be a good idea to leave the *-D* switch off so you can ensure snort loads with no errors.

- The directory **/var/log/snort** directory will contain a subdirectory for each host that was detected as violating a rule and the actions they took to trigger the alert.

- The additional log file that is created (under the IP address of the offending machine) contains the actual packet capture of the triggered event.

- One very useful tool that can be used to view snort alert and log files is the "**snortsnarf**" program.

- It is a Perl script that takes alerts and:

    - Produces diagnostic inspection of events.
    - Formats into an html output file.
    - Places the output in directory **snfout.alert**.

- The program is allows you to view Snort alerts in an orderly fashion using a web browser.

- The program creates a directory of html files in the **snfout.alert** subdirectory relative to where you are when you execute the command:

  # snortsnarf.pl /var/log/snort/alert

- Simply point your browser to the **index.html** file.

Automatic Startup and Shutdown

- Snort can be configured to start up at boot time automatically and stop when the system shuts down.
- On UNIX-type machines, this is done through a script that starts and stops Snort. The script is usually created in the **/etc/rc.d/init.d** directory on Linux.
- A link to the startup script may be created in the **/etc/rc.d/rc3.d** directory and shutdown links may be present in **/etc/rc.d/rc2.d**, **/etc/rc.d/rc1.d** and **/etc/rc.d/rc0.d** directories.
- A very useful script file called "**snortd**" is provided with the Snort RPM and can be used to start and stop Snort. This file can be placed in the **/etc/rc.d/init.d** directory.
- The first character in the name of the link file determines if Snort will be started or stopped in a particular run level.
- The startup link file starts with the character **S.** A typical startup file is **/etc/rc.d/rc3.d/S50snort** which is actually linked to **/etc/rc.d/init.d/snortd** file.
- Similarly, a typical shutdown script file starts with the letter **K.** For example, you can create **/etc/rc.d/rc2.d/K50snort** file. The ***init*** daemon will automatically start Snort when the system moves to run level 3 and will stop it when the system goes to run level 2.
- The following example creates a symbolic link to **snortd** from the **rc3.d** directory:

  ln –s ../init.d/snortd S50snort

- The integer "50" in the file specifies when the script will start relative to other scripts in the directory. The higher the number, the later it will be started.
- The symbolic link can be deleted as follows:

  unlink S50snort

- You can start and stop Snort using the script manually as well as follows:

  #/etc/init.d/snortd start
  #/etc/init.d/snortd stop

Running Snort on Multiple Network Interfaces

- When Snort is started it captures traffic on the default interface. This can be changed so that Snort can run on a specified interface using the command line option: ***–i <interface_name>*** .

- If we want to capture traffic on multiple network interfaces, we can to run multiple copies of Snort in parallel.

- The following examples illustrate how to run Snort on network interfaces **eth0** and **eth1** on a Linux machine:

  ```
  #snort -c snort.conf -i eth0 -l var/log/snort0
  #snort -c snort.conf -i eth1 -l var/log/snort1
  ```

- Note that the above commands assume that we have created two log directories, **/var/log/snort0** and **/var/log/snort1**, so that both of the Snort sessions keep their log files separate.

- These directories must exist before Snort is started otherwise the sessions will abort.

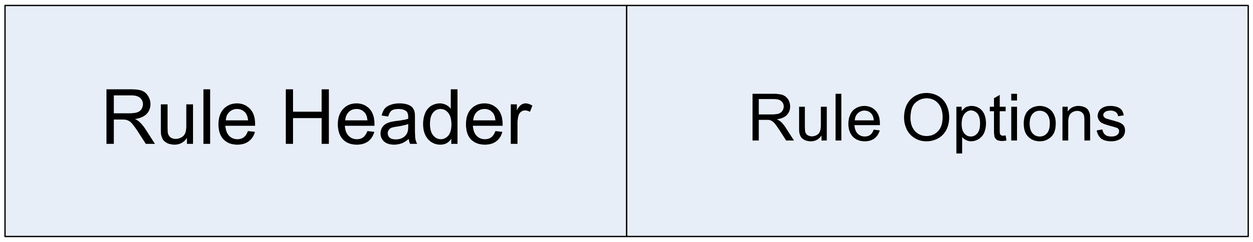
- If both sessions log to a MySQL database, which is configured through **snort.conf** file, the same database can be used.

<u>Writing Snort Rules</u>

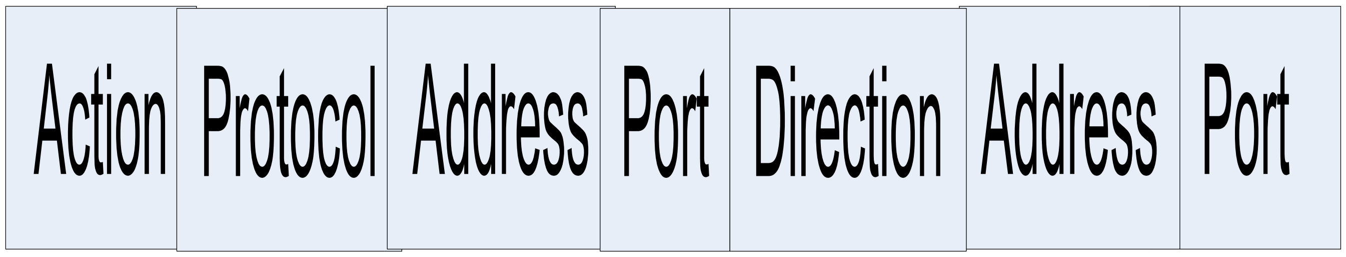
- Snort rules are simple to write, yet powerful enough to detect a wide variety of hostile or merely suspicious network traffic.
- The three base action directives that Snort can use when a packet matches a specified rule pattern are: **pass**, **log**, or **alert**.
- **Pass rules**
  - This action tells Snort to ignore the packet. This action plays an important role in speeding up Snort operation in cases where we don't want to apply checks on certain packets.
- **Log rules**
  - This action will write the full packet to the logging routine that was user selected at run-time.
  - A message can be logged to log files or in a database. Packets can be logged with different levels of detail depending on the command line arguments and configuration file.
- **Alert rules**
  - Generate an event notification using the method specified by the user at the command line, and then log the full packet using the selected logging mechanism to enable later analysis.
  - The functional difference between Log and Alert actions is that Alert actions send an alert message and then log the packet. The Log action only logs the packet.

Rule Structure

- Snort rules have two logical parts: rule ***header*** and rule ***options***:



- 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 also contains some additional criteria for matching a rule against data packets. A rule may detect one type or multiple types of intrusion activity.
- The general structure of a Snort rule header is as follows:



- ***action***
    - 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.

- ***protocol***
    - Used to apply the rule on packets for a particular protocol only. This is the first criterion mentioned in the rule. Examples of protocols used are IP, ICMP, TCP, UDP etc.
- ***address***
    - Defines the source and destination addresses. Addresses may be a single host, multiple hosts or network addresses.
    - These fields can also be used to exclude certain addresses from a complete network.
    - 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.
- ***port***
    - Determines the source and destination ports of a packet on which the rule is applied.
- ***direction***
    - This is the part of the rule that determines which address and port number is used as source and which as destination.

- As an example consider the following simple rule that generates an alert message whenever an ICMP ping packet (ICMP ECHO REQUEST) with TTL equal to 100 is detected:

  alert icmp any any -> any any (msg: "Ping with TTL=100"; ttl: 100;)
- The section of the rule before the **opening parenthesis** is called the rule **header**.
- The section of the rule that is **enclosed by the parentheses** is the **options** part.

- The header contains 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.
    - Depending on the action field, the rule options part may contain additional criteria for the rules.
- **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 type of a packet is not ICMP, the rest of the rule is ignored in order to save CPU cycles.
- **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 address and any source port.
    - Note that 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.
    - The direction can also be reversed using <- symbol.
    - Note that a symbol <> can also be used to apply the rule on packets going in either direction.
- **Destination address and port**.
    - In this example both are set to "**any**", meaning the rule will be applied to all packets irrespective of their destination address.
- In this option ***msg*** is the keyword and **"Ping with TTL=100"** is the argument to this keyword.

- 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.

- The most basic rules contain only protocol, direction, and the port of interest, such as follows:

    log tcp any any -> 192.168.1.0/24 79

- The above rule will record all traffic inbound for port 79 (finger) going to the 192.168.1 network address space.

- Option fields are available for all rule types and may be used to generate complex behaviors from the program:

    alert tcp any any -> 10.1.1.0/24 80 (content: "/cgi-bin/phf"; msg: "PHF probe!";)

- The above rule will detect attempts to access the PHF service on any of the local network's web servers.

- If such a packet is detected on the network, an event notification alert is generated and then the entire packet is logged via the logging mechanism selected at run-time.

- Port ranges can be specified using the colon ":" modifier. For example, to monitor all ports upon which the X Windows service may run (generally 6000 through 6010), the port range could be specified with the colon modifier as shown:

    alert tcp any any -> 192.168.1.0/24 6000:6010 (msg: "X traffic";)

- Both ports and IP addresses can be modified to match by exception with the bang "!" operator, which would be useful in the rule described above to detect X Windows traffic from sources outside of the network as follows:

    alert tcp !192.168.1.0/24 any -> 192.168.1.0/24 6000:6010 (msg: "X traffic";)

- Let us take a closer look at rules and how they are interpreted by snort. Consider the following rule for example:

    alert icmp !$HOME_NET any -> $HOME_NET any (msg:"IDS152 - PING BSD"; content: "|08 09 0a 0b 0c 0d 0e 0f 10 11 12 13 14 15 16 17|"; type: 8; depth: 32;)

- The rule header contains the rule's action (**alert**), protocol (ICMP), and also source and destination IP address and port information.

- The type field in the ICMP header of a data packet is used to determine the type of the ICMP packet. Type 8 refers to an ICMP Echo Request packet.

- The rule option in this case is a bit more complex and has some new keywords that are described next.

<u>The ***content*** keyword</u>

- One important feature of Snort is its ability to find a data pattern inside a packet's payload.

- The pattern may be represented as an ASCII string or as binary data in the form of hexadecimal characters.

- Malicious packets have signatures and the content keyword is used to find these signatures in the packet.

- In the above example, the pattern being matched is the hex string ***"|08 09 0a 0b 0c 0d 0e 0f 10 11 12 13 14 15 16 17|"***.

- For example, the following rule detects a pattern "**GET**" in the data portion of all TCP packets that are leaving 192.168.1.0 network and going to an address that is not part of that network:

  alert tcp 192.168.1.0/24 any -> ![192.168.1.0/24] any (content: "GET"; msg: "GET \ matched";)

- The following rule does the same thing but the pattern is listed in hexadecimal:

  alert tcp 192.168.1.0/24 any -> ![192.168.1.0/24] any \ (content: "|47 45 54|"; msg: "GET matched";)
- We can also match both ASCII strings and binary patterns in hexadecimal form inside one rule by enclosing the hexadecimal characters inside a pair of bar symbols: ||.

- The following **caveats** must be kept in mind when using the **content** keyword:

  - Content matching is a computationally intensive process and should be used judiciously by avoiding the use too content matching rules.

  - If you provide content as an ASCII string, you should escape the double quote, colon and bar symbols.

  - Multiple content keywords cane be used in one rule to find multiple signatures in the data packet.

  - Content matching is case sensitive.

- There are three other keywords that are used with the content keyword. These keywords provide additional filtering criteria when searching for a pattern inside a packet:
    - The **offset** keyword
    - The **depth** keyword
    - The **nocase** keyword
- The first two keywords are used to confine the search within a certain range of the data packet.
- The **nocase** keyword is used to make the search case-insensitive.
- This keyword, in conjunction with the ***offset*** keyword, can also be used to search into the application layer header.

The ***offset*** keyword

- The offset keyword is used in combination with the ***content*** keyword. Using this keyword, we can start your search at a certain offset from the start of the data part of the packet.
- The argument to this keyword is an integer. The following rule starts searching for the word "HTTP" after **4 bytes** from the start of the data portion of a TCP packet:

  alert tcp 192.168.1.0/24 any -> any any \
  (content: "HTTP"; offset: 4; msg: "HTTP matched";)

- The ***depth*** keyword can also be used to specify the point at which Snort should stop searching for the pattern in the data packets.

The ***depth*** keyword

- The ***depth*** keyword is also used in conjunction with the ***content*** keyword to specify an upper limit to the pattern matching.
- Using the ***depth*** keyword, we can specify an ***offset*** from the start of the data portion of a packet.
- Data after that offset is not searched for pattern matching. Using the ***offset***, ***depth*** and ***content*** keywords, we can specify the range of data within which pattern matching should be done.
- The following rule searches for the word "**HTTP**" between characters 4 and 40 of the data portion of the TCP packet:

  alert tcp 192.168.1.0/24 any -> any any (content: \
  "HTTP"; offset: 4; depth: 40; msg: "HTTP matched";)

- This keyword is important since it is used to limit searching inside the packet.
- For example, information about HTTP GET requests is found in the start of the packet. There is no need to search the entire packet for such strings.
- Many packets captured using this rule will be very large in size resulting in wasted time while searching for these strings in the entire packet.

- Consider yet another example rule:

  **alert tcp !192.168.1.0/24 any -> 192.168.1.0/24 111 **
  (content: "|00 01 86 a5|"; msg: "external mountd access";)

- This rule generates an "alert" if all the following criteria are met:

  1. Traffic is generated from any source IP address that us not within the internal network, any port
  2. The destination source IP is the internal network
  3. Client connects to TCP port 111 (portmapper)
  4. Data sent from client contains **"|00 01 86 a5|"**

- If all these criteria are met the msg " **external mountd access**" will be logged to the log file.

<u>IP Address Matching</u>

- The address fields are used to check the source from which the packet originated and the destination of the packet.
- The address may be specified as a single IP address or a network address. The *any* keyword can also be used to apply a rule on all addresses.
- The address specification is an IP address is in CIDR notation. The following are some examples of address specification in Snort rules:
- 192.168.1.5 specifies a single host.
- 192.168.1.0/24 defines a class C network with addresses ranging from 192.168.1.0 to 192.168.1.255.
- 142.232.0.0/16 defines a class B network with addresses ranging from 142.232.0.0 to 142.232.255.255.

- An address 192.168.1.16/28 defines an address range of 192.168.1.16 to 192.168.1.31.

IP Address Exclusion

- Snort provides a mechanism to exclude addresses by the use of the negation symbol "!".

- This symbol is used with an address to instruct Snort not to match packets coming from or going to that address.

- For example, the following rule is applied to all packets except those that originate from class C network 192.168.2.0.

  alert icmp ![192.168.2.0/24] any -> any any (msg: "Ping with TTL=100"; ttl: 100;)

- This is useful if we wish to exclude our home network from the rule matching.

IP Address Lists

- We can also specify list of addresses in a Snort rule. For example, if a home network consists of two C class IP networks 192.168.1.0 and 192.168.2.0 and we want to apply the above rule to all addresses but hosts in these two, we can use the following modified rule where the two network addresses are separated by a comma:

  ***alert icmp ![192.168.1.0/24,192.168.2.0/24] any -> any ***
  any (msg: "Ping with TTL=100"; ttl: 100;)

- Note the use of square brackets with the negation symbol. These are omitted if the negation symbol is not used.

Port Number Matching

- Port number matching is used to apply a rule on packets that originate from or destined for a particular port or a range of ports.

- For example, source port number 23 can be used to apply a rule to match those packets that originate from a Telnet server.

- The keyword ***any*** can be sued to apply the rule on all packets irrespective of the port number.

- The following rule is applied to all packets that originate from any external machine to port 7597 on any machine in the home network.

- The signature is that of the QAZ worm program, which infects Windows machines. The worm disguises itself as Notepad.exe and spreads by scanning for shared drives on the network, which increases network traffic.

- It also sends the IP address of the infected machine back to the creator of the worm, who can then gain access to the computer through the worm.

```
alert tcp $EXTERNAL_NET any -> $HOME_NET 7597 \
msg:"BACKDOOR QAZ Worm Client Login access"; \
flow:to_server,established; content:"qazwsx.hsq";\
reference:MCAFEE,98775; classtype:misc-activity; sid:108; rev:6;)
```

- The keyword ***flow*** is used to apply a rule on TCP sessions to packets flowing in a particular direction. In other words it can be used to determine the direction of packets.

- The following options can be used with this keyword determine direction:

    - **to_client**
    - **to_server**
    - **from_client**
    - **from_server**

- The ***classtype*** keyword is used to assign a class to the rule. The **misc-activity** class is defined with a default priority in **classification.config** file.

    - The rule ID is 108.
    - The **rev** keyword is used to show version of the rule.

- Port numbers are useful in applying a rule to a particular type of data packet only. For example, if a vulnerability is related to a backdoor Trojan, say on port 21111, we can design a rule so it matches traffic to that port only and not to any other TCP packets.

- The following is an example of such a rule:

```
alert tcp !$HOME_NET any -> $HOME_NET 31337\
(msg:"BACKDOOR ATTEMPT-Backorifice";flags:S;)
```

- This rule is looking for "tcp" based traffic originating from outside the home network "!$HOME_NET" and destined to the monitoring machine "->$HOME_NET" on a specified port "31337".

- If this pattern is detected, the message "BACKDOOR ATTEMPT - Back Orifice" will be logged in the system log file.

- The flags option in this rule is looking for a SYN. In a nutshell, this rule is looking for any traffic originating from outside our network and attempting to connect to port 31337.

Port Range Matching

- Rules can be applied to a range of ports instead of only one port in the port field. This is done using a colon to separate starting and ending port numbers.

- For example, the following rule will create an alert for all UDP traffic coming from ports 1024 to 2048 from all hosts.

  alert udp any 1024:2048 -> any any (msg: "UDP ports";)

Upper and Lower Port Boundaries

- While listing port numbers, we can also use only the starting port number or the ending port number in the range.

- For example, a range specified as **:1024** includes all port numbers up to and including port 1024. A

- port range specified as **1000:** will include all ports numbers including and above port 1000.

Port Negation

- Just as was the case with addresses, we can also use the negation symbol with port numbers to exclude a port or a range of ports from the scope of the Snort rule.

- The following rule logs all UDP traffic except for source port number 53.

  log udp any !53 -> any any log udp

Rule Development

- Snort is extremely flexible as far as developing new rules is concerned.
- The general method for development consists of getting the exploit of interest, such as a new buffer overflow and running the exploit on a test network with Snort recording all traffic between the target and attack hosts.
- Typically we would run the experiment on a controlled testbed and capture all the exploit traffic using tcpdump.
- The resulting data is then analyzing the data for a unique signature and the signature is condensed into a rule.
- Consider the "**IMAP buffer overflow**" packet dump shown below:

```
052499-22:27:58.403313 192.168.1.4:1034 -> 192.168.1.3:143
TCP TTL:64 TOS:0x0 DF ***PA* Seq: 0x5295B44E
 Ack: 0x1B4F8970 Win: 0x7D78

90 90 90 90 90 90 90 90 90 90 90 90 90 90 EB 3B     ..............;
5E 89 76 08 31 ED 31 C9 31 C0 88 6E 07 89 6E 0C    ^.v.1.1.1..n..n.
B0 0B 89 F3 8D 6E 08 89 E9 8D 6E 0C 89 EA CD 80   .....n....n.....
31 DB 89 D8 40 CD 80 90 90 90 90 90 90 90 90 90   1...@...........
90 90 90 90 90 90 90 90 90 90 90 E8 C0 FF FF FF     ................
2F 62 69 6E 2F 73 68 90 90 90 90 90 90 90 90 90    /bin/sh.........
```

- The unique signature data in the application layer is the machine code just prior to the ***/bin/sh*** text string, as well as the string itself.
- Using this information, a new rule can be developed quickly, such as:

    alert tcp any any -> 192.168.1.0/24 143
    (content:"|E8C0 FFFF FF|/bin/sh"; msg:"IMAP Buffer Overflow detected!";)

- The content field of the rule contains mixed plain text and hex formatted bytecode, which is enclosed in pipes.
- At run-time, this data is converted into its binary representation, as displayed in the decoded packet dump above and then stored in an internal list of rules by Snort.
- The rule for this signature will log an alert any time a packet containing the "fingerprint" of the IMAP buffer overflow is detected.

- Consider the following trace generated by the ping program:

```
07/23-09:46:41.866911 192.168.1.10 -> 192.168.1.1 ICMP TTL:50 TOS:0x0
ID:2403
ID:8474 Seq:256 ECHO

36 12 7B 39 1B C6 0B 00 08 09 0A 0B 0C 0D 0E 0F     6.{9............
10 11 12 13 14 15 16 17 18 19 1A 1B 1C 1D 1E 1F     ................
20 21 22 23 24 25 26 27 28 29 2A 2B 2C 2D 2E 2F     !"#$%&'()*+,-./
30 31 32 33 34 35 36 37 01234567
```

- Note the content contained in the trace sample above, starting at the 9th byte, and continuing through the 24th byte: “***08 09 0a 0b 0c 0d 0e 0f 10 11 12 13 14 15 16 17”.***
- This is a typical pattern used in Unix-type ping programs and we will use this signature in developing the rule.
- The rule will be designed to look for any ICMP traffic not originating on our monitoring machine "**!HOME_NET**" that is destined for our monitoring machine "**->HOME_NET**".
- The "depth" in the rule is set to 32. This means snort will search 32 bytes into each packet looking for the specified "content".
- The "**itype**" is the ICMP type. In this case, it is **type 8**, which is an **echo request**.
- The following is the rule that can be used to detect ping activity matching the above signature:

```
 alert icmp !$HOME_NET any -> $HOME_NET any \
(msg:"IDS152 - PING BSD"; \
content: "|08 09 0a 0b 0c 0d 0e 0f 10 11 12 13 14 15 16 17|";\
 type: 8; depth: 32;)
```

High Performance Considerations

- For fast packet captures on a busy network we can use the ***-b*** and ***-A fast*** or ***-s*** (syslog) options.
- This will log packets in ***tcpdump*** format and produce minimal alerts. For example:

  snort -b -A fast -c snort.conf

- In this invocation, Snort has been able to log multiple simultaneous probes and attacks on a 100 Mbps LAN running at a saturation level of approximately 80 Mbps.
- In this configuration, the logs are written in **binary format** to the **snort.log** tcpdump-formatted file.
- To read this file back and break out the data in the familiar Snort format, just rerun Snort on the data file with the -r option together with any other options.
- For example:

  snort -d -c snort.conf -l ./log -h 192.168.1.0/24 -r snort.log

- Once this command is completes, all of the data will be in the log directory in its normal decoded format.

Fast Mode

- The fast alert mode logs the following information:
  - Timestamp
  - Alert message (configurable through rules)
  - Source and destination IP addresses
  - Source and destination ports
- As mentioned earlier, this mode is invoked using the ***"-A fast***" command line option.
- This alert mode results in significantly reduced overhead when capturing packets on a large and busy network.
- The following command will start Snort in fast alert mode:

  snort -c snort.conf -q -A fast

- The ***–q*** option used on the command line stops the initial messages and final statistical summary from being displayed on the screen.

- A typical trace using this mode will be displayed as follows:

```
05/28-22:16:25.126150 [**] [1:0:0] Ping with TTL=100 [**]
{ICMP} 192.168.1.100 -> 192.168.1.3
```

- Note that the actual packet (hex dump) is not logged when using this alert mode.

<u>Sending Alerts to *syslog*</u>

- This mode allows Snort to send alerts to the **syslog** daemon.
- **syslog** is a system logger daemon and it is configured to generate log files for system events.
- The configuration file used by the daemon is /etc/syslog.conf, and among other parameters it also specifies the location of the log files.
- The default location of syslog files is **/var/log** directory. On Linux systems, the file **/var/log/messages** is the main logging file.
- The following command enables Snort to log to the syslog daemon:

```
snort -c snort/etc/snort.conf -s
```

- Using the default configuration, the messages are logged to the **/var/log/messages** file.
- When an alert is generated a message will written into **/var/log/messages** file. The following is an example of one such message:

```
May 28 22:21:02 snort snort[1750]: [1:0:0] Ping with TTL=100
{ICMP} 192.168.1.100 -> 192.168.1.3
```

Running Snort in Stealth Mode

- If the Snort sensor is deployed outside a network perimeter it is important to protect the IDS from being compromised.

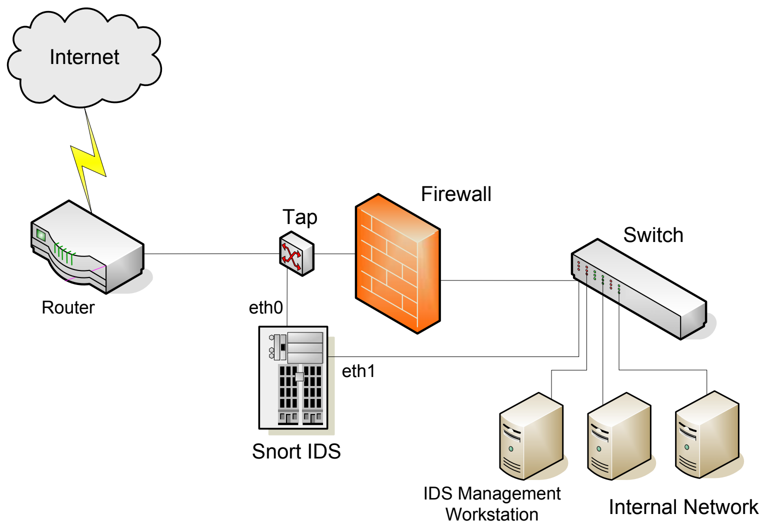
- This is achieved by running Snort in stealth mode. In stealth mode, the sensor host becomes invisible to external hosts.

- There are multiple ways to run Snort in stealth mode. One of these methods is to run Snort on a network interface to which there is no IP address is assigned.

- Running Snort on a network interface without any IP bindings is achieved in one of two ways:

    1. A stand-alone Snort sensor with only one network adapter.
    2. A Snort sensor with two network adapters: one to access the sensor from an isolated network and the other one connected to the public network and running in stealth mode.

- This arrangement is shown in the diagram below where network interface **eth1** is connected to a **private isolated network** and **eth0** is connected to a **public network.**



Snort IDS in Stealth Mode using 2 Network Cards

- Access to the IDS itself is enabled through eth1 which has an IP address configured to it.

- The management workstation shown in the figure may be used to connect to the sensor either to collect data or to log information to a centralized database.

- Network interface eth0 has all the IP bindings removed so it will operate in stealth mode but can still listen to the network traffic from this side of the network.

- Before starting Snort on eth0, it will have to be brought up using the following command:

  ifconfig eth0 up

- Following that Snort is started on this interface by using "-i eth0" command line option:

  snort -c snort.conf -i eth0 -D