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.

- It 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.

- For example, when the IIS Showcode [IISBT99] web exploits were revealed on the Bugtraq mailing list [BTQ99], Snort rules to detect the probes were available within a few hours.

- 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.

Packet Sniffing Mode

- 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 2.0.1 (Build 88)
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 analyzed 70 out of 72 packets, .
Breakdown by protocol:                Action Stats:
    TCP: 0          (0.000%)          ALERTS: 0
    UDP: 4          (5.556%)          LOGGED: 0
   ICMP: 0          (0.000%)          PASSED: 0
    ARP: 66          (91.667%)
  EAPOL: 0          (0.000%)
   IPv6: 0          (0.000%)
    IPX: 0          (0.000%)
  OTHER: 0          (0.000%)
DISCARD: 0          (0.000%)
===============================================================================
Wireless Stats:
Breakdown by type:
    Management Packets: 0          (0.000%)
    Control Packets:    0          (0.000%)
    Data Packets:       0          (0.000%)
===============================================================================
Fragmentation Stats:
Fragmented IP Packets: 0          (0.000%)
    Fragment Trackers: 0
   Rebuilt IP Packets: 0
   Frag elements used: 0
Discarded(incomplete): 0
   Discarded(timeout): 0
  Frag2 memory faults: 0
===============================================================================
TCP Stream Reassembly Stats:
        TCP Packets Used: 0          (0.000%)
         Stream Trackers: 0
          Stream flushes: 0
           Segments used: 0
   Stream4 Memory Faults: 0
===============================================================================
Snort exiting
```

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

- 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 first part of the rule set contains the rule's action (alert), protocol (ICMP), and also source and destination IP address and port information. This is known as the "rule header".

- The rest of the rule set, known as the "rule option", contains alert messages and information on what should be inspected in a packet to see if the rule matches.

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 godo 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 contains the actual packet capture of the offending 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.

Writing Snort Rules

- 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 simply drop the packet.
- Log rules write the full packet to the logging routine that was user selected at run-time.
- 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 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.
- Snort interprets keywords enclosed in parentheses as "option fields". 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";)

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.
- 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 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:"New 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 new IMAP buffer overflow is detected.

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 first part of the rule set contains the rule's action (alert), protocol (ICMP), and also source and destination IP address and port information. This is known as the "rule header".
-
- The rest of the rule set, known as the "rule option", contains alert messages and information on what should be inspected in a packet to see if the rule matches.

- To get a better idea of what information is being examined in the packet we will take a look at a trace of a ping:

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

- Notice how the content section for the rule above is actually contained in the trace sample above, starting at the 9th byte, and continuing through the 24th byte.

- Our rule is set 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 above rule is set to 32. This means snort will search 32 bytes into each packet looking for the specified "content".

- If matching content is discovered, snort generates an "alert" logs the packet. The message contained in the rule will be logged to the appropriate log file

  "IDS152 - PING BSD".

- The "itype" is the ICMP type. In this case, it is type 8, which is an echo request.

- The following is another rule:

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

- This rule is also using the "alert" option, but this time 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.

- 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 generate 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.

High Performance Configuration

- 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 configuration, 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 completes, all of the data will be in the log directory in its normal decoded format.