89 **SNORT** (NIDS-**N**etwork **I**ntrusion **D**etection **S**ystem) & Packet sniffer/logger

- Much of the following information is taken from the Snort User's Manual from Martin Poesch and Chris Green. In SuSE 8.2, Install `snort` package from CD

`snort` as a packet sniffer:

- `snort -i eth0`	Tells snort to listen on `eth0`(Default)
- `snort -v`	Sniffer mode. Shows only packet headers
- `snort -Xv`	Sniffer mode. Shows packet headers and data.
- `snort -vd`	Same as `snort -Xv`
- `snort -vde`	Same as `snort -Xv` but with more header info.

`snort` as a packet logger:

- `snort -ved -h 192.168.1.0/24 -l ~/snortlog`
  Logs all packets headers and data in hex and ASCII in the `~/snortlog` directory as separate files based on the local network IP address and the service port numbers. eg. (filenames)
  `TCP:33652-22, TCP:59446-110` (for ssh & pop3)

- `snort -l /var/log/snortlog -b`
  Logs the packets in tcpdump-binary format into a single file.eg. `~/snortlog/snort.log.1056473849`

- `snort -dv -r ~/snortlog/snort.log.1056473849`
  Read and decode all the packets from the saved tcpdump-binary format file.

- `snort -dvr ~/snortlog/snort.log.1056473849 icmp`
  Read and decode only the `icmp` packets from the saved tcpdump-binary format file.

`snort` as a Network Intrusion Detector:

```
snort -dev -l ~/log -h 192.168.1.0/24 -c snort.conf
```
- Displays the Packet headers and their data. (`-dev`)
- Logs the same into files in the `~/log` directory per service port relative to the subnet (`-h 192.168.1.0`)
- Uses the rules present in the configutation file (`-c snort.conf`) default is `/root/.snortrc`

```
snort -d -h 192.168.1.0/24 -l /var/log/snort\
     -c /etc/snort/snort.conf
```
- Same as above except without terminal display(`-v`) and no data link headers(`-e`)

In both of the above cases snort is started in IDS mode and will act upon the rules and directives declared in the configuration file `snort.conf`.

`snort` Syntax:
```
snort [-abCdDeGINoOpqsTUvVxXyz?] [-A alert-mode]
[-B address-conversion-mask] [-c rules-file] [-F bpf-file] [-g grpname]
[-h home-net] [-i interface] [-k checksum-mode]  [-l log-dir]
[-L bin-log-file] [-m umask] [-M smb-hosts-file] [-n packet-count]
[-P snap-length]  [-r tcpdump-file ] [-S variable=value ]
[-t chroot_directory]  [-u usrname ] expression
```

Output modes and destinations

There are a number of ways to configure the output of Snort in NIDS mode. The default logging and alerting mechanisms are to log in decoded ASCII format and use "`full`" alerts. There are several other alert output modes available at the command line, as well as two logging facilities.
Packets can be logged to their default decoded ASCII format or to a binary log file via the -b command line switch. If you wish to disable packet logging all together, use the `-N` command line switch.

Alert modes are somewhat more complex. There are six alert modes available at the command line, full, fast, socket, syslog, smb (winpopup), and none. Four of these modes are accessed with the -A command line switch. The four options are:

Output Modes	
`-A full`	(default) writes: alert message and full packet headers.
`-A fast`	fast alert mode. writes: timestamp, alert message, source and destination IPs/ports.
`-A none`	turn off alerting
Output destinations	
`-A unsock`	send alerts to a UNIX socket.
`-A console`	send "fast-style" alerts to the console (screen)
`-s`	Sends the Alert to syslogd as `authpriv.alert`
`-M` *`winhosts`*	Sends the Alert as a WinPopup message to the windows hosts listed in the file (*`winhosts`*) using the `smbclient` for sending the message. Note: To use this alerting mode, you must configure Snort to use it at configure time with the `--enable-smbalerts` switch.

Here are some output configuration examples:

1) Log to default (decoded ASCII) facility and send alerts to syslog
```
snort -c snort.conf -l ./log -s -h 192.168.1.0/24
```
2) Log to the default facility in `/var/log/snort` and send alerts to a fast alert file:
```
snort -c snort.conf -s -h 192.168.1.0/24
```
3) Log to a binary file and send alerts to Windows workstations:
```
snort -c snort.conf -b -M /etc/WORKSTATIONS
```
4) Log to a binary file and use fast alerting mode, logging to `/var/snort`:
```
snort -c snort.conf -b -A fast -l /var/snort
```

PERFORMANCE

If you want Snort to go *fast* (like keep up with a 100 Mbps net fast) use the "`-b`" and "`-A fast`" or "`-s`" (syslog) options. This will log packets in tcpdump-binary format and produce minimal alerts. For example:

```
snort -b -A fast -c snort-lib
```

In this configuration, Snort has been able to log multiple simultaneuos 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 tcpdump-

binary format to the `snort.log` file.
<u>To read this file back</u> and break out the data in the familiar Snort format, just rerun Snort on the data file with the "`-r`" option and the other options you would normally use. For example:

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

Once this is done running, all of the data will be sitting in the log directory in its normal decoded format. Cool, eh?

CHANGING ALERT ORDER

Some people don't like the default way in which Snort applies it's rules to packets, with the Alert rules applied first, then the Pass rules, and finally the Log rules. This sequence is somewhat counterintuitive, but it's a more foolproof method than allowing the user to write a hundred alert rules and then disable them all with an errant pass rule. For people who know what they're doing, the "`-o`" switch has been provided to change the default rule applicaition behavior to Pass rules, then Alert, then Log:

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

MISCELLANEOUS

If you are willing to run snort in "daemon" mode, you can add `-D` switch to any combination above. PLEASE NOTE that if you want to be able to restart snort by sending SIGHUP signal to the daemon, you will need to use full path to snort binary, when you start it, eg..:

```
/usr/local/bin/snort -a -D -d -h 192.168.1.0/24 \
     -l /var/log/snortlogs -c /usr/local/etc/snort-lib
```

Relative pathes are not supported due to security concerns.

If you're going to be posting packet logs to public mailing lists you might want to try out the `-O` switch. This switch "obfuscates" your the IP addresses in the packet printouts. This is handy if you don't want the people on the mailing list to know the IP addresses involved. You can also combine the `-O` switch with the `-h` switch to only obfuscate the IP addresses of hosts on the home network. This is useful if you don't care who sees the address of the attacking host. For example:

```
snort -d -v -r snort.log -O -h 192.168.1.0/24
```

This will read the packets from a log file and dump the packets to the screen, obfuscating only the addresses from the `192.168.1.0/24` class C network.

If you want to see Snort's packet statistics without stopping the process, send a SIGUSR1 to the Snort process ID and it will dump stats to the screen or syslog if it's running in daemon mode. This will allow you to see which protocols Snort has been seeing, get counts of alerts and logged packets and counts of total packets seen and dropped. It's a very handy capability if you're tweaking Snort for performance.

Configuration file `/etc/snort/snort.conf`

The configuration file **`/etc/snort/snort.conf`** is used for 2 purposes:

Configuration directives that can also be given on the command line when starting snort. They are preceeded by the keyword `config`

Rules for watching packet traffic.

`snort` Configuration Directives in `/etc/snort/snort.conf`

Many of the following these directives can also be given as a command line option.

Syntax: **`config directive [: value]`**

`order`	Change the pass order of rules (`-o`)
`alertfile`	Set the alerts output file. Example: `config alertfile: /var/log/alerts.log`
`classification`	Build rules classifications
`decode_arp`	Turn on arp decoding (`-a`)
`dump_chars_only`	Turn on character dumps (`-C`)
`dump_payload`	Dump application layer (`-d`)
`decode_data_link`	Decode Layer2 headers (`-e`)
`bpf_file`	Specify BPF filters (`-F`). Example: `config bpf_file: filename.bpf`
`set_gid`	Change to this GID (`-g`). Example: `config set_gid: snort_group`
`daemon`	Fork as a daemon (`-D`)
`reference_net`	Set home network (`-h`). Example: `config reference_net: 192.168.1.0/24`
`interface`	Set the network interface (`-i`). Example: `config interface: ppp0`
`alert_with_interface_name`	Append interface name to alert (`-I`)
`logdir`	Set the log directory (`-l`). Example: `config logdir: /var/log/snort`
`umask`	Umask when running (`-m`). Example: `config umask: 022`
`pkt_count`	Exit after N packets (`-n`). Example: `config pkt_count: 13`
`nolog`	Disable Logging. Note: Alerts will still occur. (`-N`)
`obfuscate`	Obfuscate IP Addresses (`-O`)
`no_promisc`	Disable promiscuous mode (`-p`)
`quiet`	Disable banner and status reports (`-q`)
`chroot`	Chroot to specified dir (`-t`) Example: `config chroot: /home/snort`
`checksum_mode`	Types of packets to calculate checksums. Values: `none, noip, notcp, noicmp, noudp, all`
`set_uid`	Set UID (`-u`). Example: `config set_uid: snort_user`
`utc`	Use UTC instead of local time for timestamps (`-U`)
`verbose`	Use Verbose logging to stdout (`-v`)
`dump_payload_verbose`	Dump raw packet starting at link layer (`-X`)
`show_year`	show year in timestamps (`-y`)
`stateful`	set assurance mode for stream4 (`est`).
`min_ttl`	sets a snort-wide minimum `ttl` to ignore all traffic.
`disable`	decode alerts turn off the alerts generated by the decode phase of snort.

`snort` Rules in `/etc/snort/snort.conf`

1. Snort rules are written either on a single line or, like in `bash`, with '\' at the end of the lines, except for the last line.(NO spaces after each '\' !!!!)
2. Snort rules are divided into two logical sections:

rule header (each item is separated by a space)

```
    action protocol source_IP[/mask] port  -> destination_IP[/mask]  port
eg. alert  tcp      any              any   -> 192.168.4.0/24         80
```

rule options (included in Parenteses. items are separated by '`;`')
(What to look for in packet ; alert message)

```
eg.  (content:"CWD incoming"; msg:"cd command detected")
```

Full example:
Command:

```
     snort -b -s -A fast -c /etc/snort/snort.conf \
      -l /var/snort
```

Content of `/etc/snort/snort.conf`:

```
     alert tcp any any -> 192.168.100.0/24 111 \
     (content:"|00 01 86 a5|"; msg:"mountd access";)
```

This rule will `alert` (send the message" `mountd access`") to the prefered output ('`-s`' = syslog daemon) as `authpriv.alert` (default) when any host using `any` port sends a packet to a host from the defined network (`192.168.100.0/24`) on port `111` containing the bytes pathern '`00 01 86 a5`'.

Content of Rule Headers:

action:

`alert`	generate an alert and then log the packet
`log`	log the packet
`pass`	ignore the packet
`activate`	alert and then turn on another dynamic rule
`dynamic`	remain idle until activated by an activate rule , then act as a log rule.

protocol:
`tcp udp icmp` or `ip.`

IP Address: (no hostnames allowed because no DNS resolution is used)

`any`	Any address
`xx.xx.xx.xx/Mask`	Address range in CIDR format. `xx.xx.xx.xx/32` for a single host.
`!xx.xx.xx.xx/32`	Any address except this one

port:

`any`	Any Port
`21`	Port 21
`:600`	Range: Port 1 to 600 (incl)
`1024:`	Range: Port 1024 and higher (<=65535)

`->` and `<>` Traffic direction: `->` Source to Dest , `<>` Bidirectional

Content of Rules options:
The options are enclosed in parenteses.

Char `':'` separates an option keyword and its parameter (if needed).
Char `';'` separates options between each other.
eg. `(content: "Password"; nocase;.......)`

Available Keywords

`msg`	prints a message in alerts and packet logs
`logto`	log the packet to a user specified filename instead of the standard output file
`ttl`	test the IP header s TTL field value
`tos`	test the IP header s TOS field value
`id`	test the IP header s fragment ID field for a specific value
`ipoption`	watch the IP option fields for specific codes
`fragbits`	test the fragmentation bits of the IP header
`dsize`	test the packet s payload size against a value
`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
`icmp_id`	test the ICMP ECHO ID field against a specific value
`icmp_seq`	test the ICMP ECHO sequence number against a specific value
`content`	search for a pattern in the packet s payload
`content-list`	search for a set of patterns 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
`nocase`	match the preceding content string with case insensitivity
`session`	dumps the application layer information for a given session
`rpc`	watch RPC services for specific application/procedure calls
`resp`	active response (knock down connections, etc)
`react`	active response (block web sites)
`reference`	external attack reference ids
`sid`	Snort rule id
`rev`	rule revision number
`classtype`	rule classification identifier
`priority`	rule severity identifier
`uricontent`	search for a pattern in the URI portion of a packet
`tag`	advanced logging actions for rules
`ip_proto`	IP header s protocol value
`sameip`	determines if source ip equals the destination ip
`stateless`	valid regardless of stream state
`regex`	wildcard pattern matching

Using Variables

syntax: `var: variable value`
The use of variables in the rules can prove very efficient and sometimes necessary. They are often used in identifying values that might change with time or are unique to the host environment like:
- Setting the local network addresses
- Defining the messages text
- Grouping addresses etc.

eg.
```
var: MY_NET [192.168.100.0/24,10.35.60.0/24]
alert tcp any any -> $MY_NET any \
     (flags: S; msg: "SYN packet";
```

includes

Include directives are used to add the content of other configuration files to the current one to extend the rules in a modular way.
Syntax: `include /etc/snort/web-attacks.rules`

Preprocessors

Preprocessors are modules that are specialized in a defined type of detection. They are normally used for complex detection involving delays and packet pathern recognition.
They are listed with their possible parameters in the snort configuration file `/etc/snort/snort.conf` and here is a list of some of them:

`Minfrag` Predecessor of Stream4 preprocessor. Specialized in watching for fragmented packets size and alert when the size is smaller than 512 bytes. Stream4 can do the same and much more.
Syntax:`preprocessor minfrag: Threshold_number`
eg. `preprocessor minfrag: 128`

`http_decode`
Specialized in decoding and converting international coded ASCII URIcontent to readable ASCII.
Syntax:
```
preprocessor http_decode portlist \
         [-unicode] [-cginull]
```
eg. `preprocessor http_decode 80 8080 -unicode`

`portscan` Specialized in detecting portscans. It does the following:
- Log the start and end of portscans from a single source IP to the standard logging facility.
- If a log file is specified, logs the destination IPs and ports scanned as well as the type of scan.

Syntax: `preprocessor portscan network_to_watch \`
`number_of_ports` accessed in the detection period \
`detection_period` number of seconds to count \
`/directory/filename` to place alerts in.
(Alerts are also written to the standard alert file)

eg.
```
preprocessor  portscan 0.0.0.0/0 5 7 \     /
var/log/portscan.log
```

portscan-ignorehosts

In combination with portscan it allows to irgnore certain hosts from producing portscan alerts.

Syntax:

```
      preprocessor  portscan-ignorehosts:  hostlist
eg.   preprocessor  portscan-ignorehosts: \
      192.168.100.0/24 192.168.70.0/24
```

Frag2 Replaces the `defrag` preprocessor which uses the memory more efficiently and more options for refined defrag evasion techniques. See Snort User's Manual Page 34.

Syntax:

```
      preprocessor frag2: [memcap <xxx>], \
      [timeout <xx>], [min ttl <xx>], \
      [detect_state_problems], [ttl_limit <xx>]
      noinspect disable stateful inspection
eg.   preprocessor frag2: memcap 8388608, timeout 30
```

Stream4 Powerful module that provides TCP stream reassembly and stateful analysis that ignore stateless attacks such as stick and snot produce. Stream4 also gives large scale users the ability to track more than 256 simultaneous TCP streams. It can be combined with the Stream4_Reassemble module.

Syntax:

```
      preprocessor stream4: [noinspect],\
      [keepstats], [timeout <seconds>], \
      [memcap <bytes>], [detect_scans], \
      [detect_state_problems],   \
      [disable_evasion_alerts], [ttl_limit <count>]
```

`noinspect`
: disable stateful inspection

`keepstats`
: record session summary information in <logdir>/session.log

`timeout <seconds>`
: amount of time to keep an inactive stream in the state table, sessions that are flushed will automatically be picked up again if more activity is seen, default is 30 seconds

`memcap <bytes>`
: number of bytes to set the memory cap at, if this limit is exceeded stream4 will aggressively prune inactive sessions, default is 8MB

`detect_scans`
: turns on alerts for portscan events

`detect_state_problems`
: turns on alerts for stream events of note, such as evasive RST packets, data on the SYN packet, and out of window sequence numbers

`disable_evasion_alerts`
turns off alerts for events such as TCP overlap
`ttl_limit`
sets the delta value that will set off

Stream4 Reassemble Format

Syntax:
```
preprocessor stream4 reassemble: [clientonly],
[serveronly], [noalerts], [ports <portlist>]
```
`clientonly`
provide reassembly for the client side of a connection only
`serveronly`
provide reassembly for the server side of a connection only
`noalerts`
don't alert on events that may be insertion or evasion attacks
`ports <portlist>`
a whitespace separated lit of ports to perform reassembly for, all provides reassembly for all ports, default provides reassembly for ports 21 23 25 53 80 110 111 143 and 513
See Page 37 for more info.

Conversation

The Conversation preprocessor allows Snort to get basic conversation status on protocols rather than just with TCP as done in spp stream4. It can also generate an alert message if it recieves packets with ip protocols that are not allowed on your network.

Syntax:
```
preprocessor conversation: \
     [allowed_ip_protocols protonumbers|all],
     [timeout <sec>], [alert_odd_protocols]
```

portscan2 This module allows portscans to be detected. This module is requires the above Conversation preprocessor 2.4.7 in order to know when a conversation is new. This is intended to pick up quick scans such as a rapid nmap scan.

Syntax:
```
preprocessor portscan2: [scanners_max <num>],\
     [targets_max <num>], [timeout <sec>]
```
`scaners_max`
number of hosts scanning a network to support at once.
`targets_max`
number of nodes to allocate to represent hosts.
`target_limit`
number of hosts a scanner must talk to before a scan is triggered.
`port_limit`
number of ports a scanner must talk to before a scan is triggered.
`timeout`
number of second before a scanner s activity is forgotten.

Portscan2 Defaults
```
scanners_max 1000
targets_max 1000
```

```
target_limit 5
port_limit 20
timeout 60
```

telnet_decode

The telnet decode preprocessor allows snort to normalize telnet control protocol characters from the session data. In Snort 1.9.0 and above, it accepts a list of ports to run on as arguments. Also in 1.9.0, it normalizes into a separate data buffer from the packet itself so that the raw data may be logged or examined with the rawbytes content modifier2.3.38. It defaults to running on ports 21, 23, 25, and 119.

Syntax:

```
preprocessor telnet_decode: <ports> \
     [alert_fragments]\
[no_alert_multiple_requests]\
[no_alert_large_fragments]\
[no_alert_incomplete]
```

rpc_decode

The rpc decode preprocessor normalizes RPC multiple fragmented records into a single unfragmented record.

Syntax:

```
preprocessor rpc_decode: [alert_fragments] \
     [no_alert_multiple_requests] \
     [no_alert_large_fragments] \
     [no_alert_incomplete]
```

Options:

`alert_fragments`
Alert on any fragmented RPC record

`no_alert_multiple_requests`
Don t Alert when there are multiple records in one packet

`no_alert_large_fragments`
Don t Alert when the sum of fragmented records exceeds one packet

`no_alert_incomplete`
Don t Alert when a single fragment record exceeds the size of one packet

Output Modules
The output modules allow to send to more than one place the various alerts.

The default is to send them to `/var/log/snort/` directory or to the directory set by the option `-l` on the command line. In addition the command line `-s` option would also send them to the syslog system as `authpriv.alert`.

`alert_syslog` Output Module:
Does the same as the `-s` command line option but allows to change the syslog facility and priority. (`-s` send messages as `authpriv.alert`)
NOTE: Use either the `-s` comand line option or this Output Module but not both.
If Both are used than the `-s` will be the only one active.
Syntax: `output alert syslog: ` *`Facility Priority options`*
eg. `output alert syslog: LOG_AUTH LOG_ALERT LOG_PERROR`

Facilities:	Priorities:
LOG_AUTH	LOG_EMERG
LOG_AUTHPRIV	LOG_ALERT
LOG_DAEMON	LOG_CRIT
LOG_LOCAL0	LOG_ERR
LOG_LOCAL1	LOG_WARNING
LOG_LOCAL2	LOG_NOTICE
LOG_LOCAL3	LOG_INFO
LOG_LOCAL4	LOG_DEBUG
LOG_LOCAL5	Options :
LOG_LOCAL6	LOG CONS (Log to the console)
LOG_LOCAL7	LOG NDELAY
LOG_USER	LOG PERROR
	LOG PID (Log snort PID)

`alert_fast` Output Module
This module allows to save fast one-line alerts into a file.
Syntax: `output alert_fast: ` *`Filename`*
eg. `output alert_fast: /var/log/snort/fast_alerts.log`

`alert_full` Output Module
Print Snort alert messages with full packet headers. The alerts will be written in the default logging directory (`/var/log/snort`) or in the logging directory specified at the command line. Inside the logging directory, a directory per IP will be created. These files will be decoded packet dumps of the packets that triggered the alerts. The creation of these files slows snort down considerably. This output method is discouraged for all but the lightest traffic situations.
Syntax: `output alert full: ` *`output_filename`*

`alert_smb` Output Module
This module allows to send a WinPopup message to multiple Windows workstations using the `smbclient` program. Similar to the `-M` *`winstation`* command line option but for multiple Windows targets.
Syntax: `output alert_smb: ` *`Worstations_List_Filename`*
eg. `output alert_smb: /etc/winhostslist.txt`

`alert_unixsock` Output Module
Sets up a UNIX domain socket and sends alert reports to it. External programs or processes can listen in on this socket and receive Snort alert and packet data in real time. This is currently an experimental interface.
Syntax: `output alert_unixsock`

`log_tcpdump` Output Module
The log_tcpdump module logs packets to a tcpdump-formatted file. This is useful for performing post process analysis on collected traffic with the vast number of tools that are avialable for examining tcpdump formatted files. This module only takes a single argument, the name of the output file. Note that the file name will have the *`<Month><Date>@<Time>-`* prepended to the file name. This is so data from separate snort runs can be kept distinct.
Syntax: `output log_tcpdump:` *`output_filename`*
eg. `output log_tcpdump: /var/log/snort/raw.log`
This above example will create new files at every new snort start. Their names will be always different since the time of start is different.

`log_null` Output Module
Sometimes it is useful to be able to create rules that will alert to certain types of traffic but will not cause packet log entries. In Snort 1.8.2, the log null plugin was introduced. This is equivalent to using the -N command line option but it is able to work within a ruletype.
Syntax: `output log_null` Does the same as using `-N` command line option

Creating an information only rule type:
```
ruletype info {
     type alert
     output alert_fast: /var/log/snort/info.alert
     output log_null
     }
```

Then use the ruletype:
```
info tcp any any -> $HOME_NET 110 \
     (Content:"Password"; nocase; msg: "Password gone by";)
```

Practical things with SuSE 8.2:

- Installation of `snort` package is from SuSE CD
  SuSE 8.2 has the Version 1.9.1
  Debian 'Woody' has the version 1.8.4
- Edit the `/etc/snort/snort.conf` and:
  1) Set the network variables for your network

```
var HOME_NET $ppp0_ADDRESS
var RULE_PATH /etc/snort
```

  2) Configure preprocessors (change to do is in underlined here)

```
preprocessor frag2
preprocessor stream4: detect_scans, disable_evasion_alerts
preprocessor stream4_reassemble
preprocessor http_decode: 80 unicode iis_alt_unicode \
                  double_encode iis_flip_slash full_whitespace
preprocessor rpc_decode: 111  2049 <---- For NFS
preprocessor bo: -nobrute
preprocessor telnet_decode
preprocessor conversation: allowed_ip_protocols all, \
                  timeout 60, max_conversations 32000
```

  3) Configure output plugins

```
output alert_syslog: LOG_LOCAL0 LOG_ALERT
```

  4)Edit the file `/etc/syslog.conf` and set the proper destination for `local0.alert` messages.
  4) Customize your rule set as needed (may have none for now)
  5) Edit the file `/etc/sysconfig/snort` and set the paramters of the variables as needed.

```
SNORT_INTERFACE="ppp0"
SNORT_ACTIVATE="no"
SNORT_AUTO="yes"      Auto IP Number setting of HOME_NET
SNORT_PROMISC="no"
SNORT_USER="snort"
SNORT_GROUP="snort"
SNORT_EXTRA_OPTIONS=""  <----Extra command line options come here
```

To Start/Stop the snort Daemon:

```
rcsnort {start|stop|restart|reload|status|activate|deactivate}
```

`activate|deactivate` is for automatic snort startup on interface startup

In SuSE 8.2 It will start with the following parameters:

```
-Dd  -i $SNORT_INTERFACE $PROMISC \
     -l /var/log/snort \
     -u $SNORT_USER \
     -g $SNORT_GROUP \
     -c /etc/snort/snort.conf \
     $SNORT_EXTRA_OPTIONS
```

(The settings for above variables are found in `/etc/sysconfig/snort`)

In Debian 'Woody' it will start with the following parameters:

```
-Dbd -S "HOME_NET=[$DEBIAN_SNORT_HOME_NET]" \
     -h "$DEBIAN_SNORT_HOME_NET" -c /etc/snort/snort.conf
     -l /var/log/snort -u snort -g snort \
      $DEBIAN_SNORT_OPTIONS >/dev/null
```

(The settings of above `Variables` are set in `/etc/snort/snort.debian.conf`)

Below is the content of the SuSE 8.2 `/etc/snort/snort.conf`(without comments)

```
var HOME_NET $ppp0_ADDRESS
var EXTERNAL_NET any
var DNS_SERVERS $HOME_NET
var SMTP_SERVERS $HOME_NET
var HTTP_SERVERS $HOME_NET
var SQL_SERVERS $HOME_NET
var TELNET_SERVERS $HOME_NET
var HTTP_PORTS 80
var SHELLCODE_PORTS !80
var ORACLE_PORTS 1521
var AIM_SERVERS \
      [64.12.24.0/24,64.12.25.0/24,64.12.26.14/24,64.12.28.0/24,64.12.29.0/
     24,64.12.161.0/24,64.12.163.0/24,205.188.5.0/24,205.188.9.0/24]
var RULE_PATH /etc/snort
preprocessor frag2
preprocessor stream4: detect_scans, disable_evasion_alerts
preprocessor stream4_reassemble
preprocessor http_decode: 80 unicode iis_alt_unicode double_encode
iis_flip_slash full_whitespace
preprocessor rpc_decode: 111 2049
preprocessor bo: -nobrute
preprocessor telnet_decode
preprocessor conversation: allowed_ip_protocols all, \
                        timeout 60, max_conversations 32000
include classification.config
include reference.config
include $RULE_PATH/bad-traffic.rules
include $RULE_PATH/exploit.rules
include $RULE_PATH/scan.rules
include $RULE_PATH/finger.rules
include $RULE_PATH/ftp.rules
include $RULE_PATH/telnet.rules
include $RULE_PATH/rpc.rules
include $RULE_PATH/rservices.rules
include $RULE_PATH/dos.rules
include $RULE_PATH/ddos.rules
include $RULE_PATH/dns.rules
include $RULE_PATH/tftp.rules
include $RULE_PATH/web-cgi.rules
include $RULE_PATH/web-coldfusion.rules
include $RULE_PATH/web-iis.rules
include $RULE_PATH/web-frontpage.rules
include $RULE_PATH/web-misc.rules
include $RULE_PATH/web-client.rules
include $RULE_PATH/web-php.rules
include $RULE_PATH/sql.rules
include $RULE_PATH/x11.rules
include $RULE_PATH/icmp.rules
include $RULE_PATH/netbios.rules
include $RULE_PATH/misc.rules
include $RULE_PATH/attack-responses.rules
include $RULE_PATH/oracle.rules
include $RULE_PATH/mysql.rules
include $RULE_PATH/snmp.rules
include $RULE_PATH/smtp.rules
include $RULE_PATH/imap.rules
include $RULE_PATH/pop3.rules
include $RULE_PATH/pop2.rules
include $RULE_PATH/nntp.rules
include $RULE_PATH/other-ids.rules
include $RULE_PATH/experimental.rules
include $RULE_PATH/local.rules
```