**Adapted from:**
**Snort-Setup for Statistics HOWTO**
**Sandro Poppi**
**v1.01, Feb 23, 2002**
**http://www.linuxdoc.org/HOWTO/Snort-Statistics-HOWTO/configuration.html#PRE-SNORT-CONFIG**

- **Download Snort from: http://www.snort.org/dl/**

Download RPM from http://www.lug-burghausen.org/projects/Snort-Statistics/snort-1.8.1-4.i386.rpm

# 1. Configuring Snort

**Snort Configuration**

Since I am using RedHat linux 7.x all the given pathnames and configuration options are eventually RedHat specific while there should be no big problem to transfer it to any other distribution.

This chapter describes the various configuration tasks to get snort and the tools up and running.

Since I am using RedHat linux 7.x all the given pathnames and configuration options are eventually RedHat specific while there should be no big problem to transfer it to any other distribution.

## 1.1. /etc/snort/snort.conf

After installing the RPM we have to edit */etc/snort/snort.conf* to reflect our needs. Martin Roesch created the Snort Users Manual which is shipped with the snort tarball and the RPMS as a PDF version. You should have a look on it to see which options you would like to use as not all but only the ones needed for our configuration here will be covered in this document.

Also the example configuration */etc/snort/snort.conf* shipped with the tarball/RPM is a good place to start because of the detailed remarks.

### 1.1.1. Snort Variables

First we define various variables like HOME_NET, EXTERNAL_NET and DNS_SERVERS to reflect our network topology. Make sure you use the right addresses or you get weird, or worse, no alarms.

When using snort in a complex environment, let's say one sensor with multiple interfaces to watch, the definition of HOME_NET and EXTERNAL_NET may be hard or at least results in a very long list, you can set both variables to *any*. You loose some kind of pre-filtering for the sake of not having to put in dozens of network ranges in a large internal network. And you minimize the performance impact of having snort run through a huge list of addresses for each packet.

To get rid of some nasty messages of (false) portscans define the variable DNS_SERVERS to hold all ip addresses of dns-servers along with other nodes like network management stations triggering snort's portscan module. This is an ongoing process.

You also can define your own variables here which you can refer to in your own rules. This is helpful e.g. if using *pass rules* to suite your environment.

Define all other variables to appropriate values or as in the shipped */etc/snort/snort.conf* to $HOME_NET.

```
      var HOME_NET any
      var EXTERNAL_NET any
      # DNS_SERVERS holds the addresses of "noisy" computers like DNS
or NWM
      # to be ignored from portscans
      var DNS_SERVERS [1.1.1.1/32,2.2.2.2/32]
      var SMTP_SERVERS $HOME_NET
      ...
```

### 1.1.2. Snort Preprocessors

Next we have to set up the preprocessors to be used. While the more preprocessors you use you get more triggers for alarms but for the cost of performance. So be careful in choosing preprocessors.

You should also have a look on Marty's *Snort Users Manual* because some preprocessors are deprecated. For those you should use the new introduced ones.

The preprocessors *minfrag* and *stream* are depricated in favor of *stream4*, and *defrag* is deprecated by *frag2*.

*frag2* is the new IP defragmentation processor introduced in snort v1.8 which should be more memory efficient than *defrag/minfrag*.

From the Snort Users Manual: *The stream4 module provides TCP stream reassembly and stateful analysis capabilities to Snort. Robust stream reassembly capabilities allow Snort*

*to 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. Stream4 should be able to scale to handle 64,000 simultaneous TCP connections.*

The *stream4* module consists of two preprocessors called *stream4* and *stream4_reassemble*, which both have to be used.

There are various options for both preprocessors while we will use only - for *stream4* - *detect_scans* for getting alarms for portscan events and *detect_state_problems* to be informed when stream events like evasive RST packets, data on SYN packets and out of window sequence numbers occur.

With *stream4_reassemble* we use the option *ports all* what makes the reassembly catch all ports instead of only some predefined ones. To be honest, this is some kind of paranoic and impacts the cpu utilization of the snort sensor, but since I didn't get any bad results listening on a Pentium III 800 MHz on three 100 Mbit/s full duplex lines with average to low utilization I think it's the better solution.

Two other preprocessors we will use are *portscan* and *portscan-ignorehosts* which are responsible for portscan detection (*portscan*) and for which hosts portscan detection has to be ignored (*portscan-ignorehosts*).

For *portscan* we define to look for every network using the form *0.0.0.0/0*, set the number of port numbers to be accessed in the also to be defined detection period in seconds. Additionally we have to provide the complete path to the portscan logfile.

With *portscan-ignorehosts* we get rid of some weird alarms from hosts which talk too much and trigger portscan detection like name servers and network management stations (see variable *DNS_SERVERS* above).

Some preprocessors which are not (yet) mentioned in Marty's Users Manual but we will use are *unidecode* which is a replacement of *http_decode* and normalizes http and UNICODE attacks, *rpc_decode* to normalize rpc traffic on a given port, *bo* to check for back orifice traffic and *telnet_decode* to normalize telnet negotiation strings.

Other preprocessors like SPADE are not yet covered here but may be in a future version. Contributions are very welcome >;)

After all that theoretical stuff here is the preprocessor part of */etc/snort/snort.conf*:

```
preprocessor frag2
preprocessor stream4: detect_scans detect_state_problems
preprocessor stream4_reassemble: ports all
preprocessor unidecode: 80 8080
preprocessor rpc_decode: 111
```

```
preprocessor bo: -nobrute
preprocessor telnet_decode
preprocessor portscan: 0.0.0.0/0 6 3 /var/log/snort/portscan.log
preprocessor portscan-ignorehosts: $DNS_SERVERS
```

### 1.1.3. Snort Output Modules

The next part is the configuration of the output modules of which we will use the syslog module *alert_syslog* to send alerts to syslog and *database* to additionally log to a MySQL database.

The *alert_syslog* module requires some options for what has to be logged. If like in my case you are using SnortSnarf to analyse the logfile you'll have to add the option *LOG_PID* else SnortSnarf has problems.

As stated before we will use ACID and thus we need to set up snort to log to a database. I chose MySQL for no particular reason (well, I've heard more from MySQL than from postgreSQL but that's all).

The *database* output module requires the following parameters:

log | alert

> Log to the *alert* facility. Also possible would be the *log* facility. If you would like to get portscan alerts into the database you have to use *alert* here.

mysql|postgrsql|odbc|oracle|mssql

> This is the type of database.

user=<username>

> Here you define the username to be used with the database.

password=<password>

> The required password for the given user.

dbname=<databasename>

> The name of the database to be used for logging into.

host=<hostname>

Here you define the host on which the database is running. Use localhost if the database is running on the snort sensor itself.

sensor_name=<sensor name>

Here you put in a unique name which is used to differentiate between various sensors if more than one is logging into a single database.

Now let's take a look on the output module part of */etc/snort/snort.conf*:

```
       output alert_syslog: LOG_AUTH LOG_ALERT LOG_PID
       output database: alert, mysql, user=snort password=mypassword
dbname=snort host=localhost sensor_name=mysensor
```

If you are using more than one physical snort sensor and would log to a database I would recommend using a central database on a separate machine. You then can correlate alert data with a single console getting a better overview when attacks are found.

### 1.1.4. Snort Rule Sets

The rules are the vital part of snort. There are various categories of rules shipped with snort. They can be found in */etc/snort/*, ending with **.rules*. The format in version 1.8+ has changed to reflect the classification types. In addition priority settings of the classtypes can also be defined.

If you're using the original snort tarball I suggest copying all rule files and *classification.config* into it.

The configuration of classification types is done in */etc/snort/classification.config*. Normally you don't have to touch it since it is preconfigured for the shipped snort rules. But if you (again like me) are using Max Vision's *vision.rules* you'll have to add some lines because the classtypes are different. Just copy and paste all *config classification:* lines from *vision.conf* to */etc/snort/classification.config*. And remember to take the *vision.rules* for snort 1.8 (called *vision18.rules* and *vision18.conf* on http://www.whitehats.com/) as the older ones are not prepared for the new format introduced in snort 1.8!

Here's the */etc/snort/classification.config* I used with *vision.rules*:

```
       #
       # config classification:shortname,short description,priority
       #
       #config classification: not-suspicious,Not Suspicious Traffic,0
       config classification: unknown,Unknown Traffic,1
```

```
       config classification: bad-unknown,Potentially Bad Traffic, 2
       config classification: attempted-recon,Attempted Information
Leak,3
       config classification: successful-recon-limited,Information
Leak,4
       config classification: successful-recon-largescale,Large Scale
Information Leak,5
       config classification: attempted-dos,Attempted Denial of
Service,6
       config classification: successful-dos,Denial of Service,7
       config classification: attempted-user,Attempted User Privilege
Gain,8
       config classification: unsuccessful-user,Unsuccessful User
Privilege Gain,7
       config classification: successful-user,Successful User Privilege
Gain,9
       config classification: attempted-admin,Attempted Administrator
Privilege Gain,10
       config classification: successful-admin,Successful Administrator
Privilege Gain,11

       # added from vision18.conf
       # classification for use with a management interface
       # low risk
       config classification: not-suspicious,policy traffic that is not
suspicious,0
       config classification: suspicious,suspicious miscellaneous
traffic,1
       config classification: info-failed,failed information gathering
attempt,2
       config classification: relay-failed,failed relay attempt,3
       config classification: data-failed,failed data integrity
attempt,4
       config classification: system-failed,failed system integrity
attempt,5
       config classification: client-failed,failed client integrity
attempt,6
       # med risk
       config classification: denialofservice,denial of service,7
       config classification: info-attempt,information gathering
attempt,8
       config classification: relay-attempt,relay attempt,9
       config classification: data-attempt,data integrity attempt,10
       config classification: system-attempt,system integrity
attempt,11
       config classification: client-attempt,client integrity
attempt,12
       config classification: data-or-info-attempt,data integrity or
information gathering attempt,13
       config classification: system-or-info-attempt,system integrity
or information gathering attempt,14
       config classification: relay-or-info-attempt,relay of
information gathering attempt,15
```

```
       # high risk
       config classification: info-success,successful information
gathering attempt,16
       config classification: relay-success,successful relay attempt,17
       config classification: data-success,successful data integrity
attempt,18
       config classification: system-success,successful system
integrity attempt,19
       config classification: client-success,successful client
integrity attempt,20
```

The classification and rule files are included in */etc/snort/snort.conf*. Some rule files used here have been copied from the CVS, e.g. *virus.rules* because they were not shipped with the standard distribution.

As stated before the *vision.rules* file will be fetched via the tool *arachnids_upd* which is discussed later.

Arachnids_upd changes the name from *vision18.rules* to *vision.rules* but the rules are of course the ones prepared for snort 1.8+.

Since the variable definitions for INTERNAL and EXTERNAL in *vision.rules* are not the same as with the snort rules I use a script to change these names. Take a look at the *arachnids_upd* section below.

```
       # Include classification & priority settings
       include /etc/snort/classification.config

       include /etc/snort/exploit.rules
       include /etc/snort/scan.rules
       include /etc/snort/finger.rules
       include /etc/snort/ftp.rules
       include /etc/snort/telnet.rules
       include /etc/snort/smtp.rules
       include /etc/snort/rpc.rules
       include /etc/snort/rservices.rules
       include /etc/snort/backdoor.rules
       include /etc/snort/dos.rules
       include /etc/snort/ddos.rules
       include /etc/snort/dns.rules
       include /etc/snort/netbios.rules
       include /etc/snort/web-cgi.rules
       include /etc/snort/web-coldfusion.rules
       include /etc/snort/web-frontpage.rules
       include /etc/snort/web-iis.rules
       include /etc/snort/web-misc.rules
       include /etc/snort/sql.rules
       include /etc/snort/x11.rules
       include /etc/snort/icmp.rules
```

```
include /etc/snort/shellcode.rules
include /etc/snort/misc.rules
include /etc/snort/policy.rules
include /etc/snort/info.rules
#include /etc/snort/icmp-info.rules
include /etc/snort/virus.rules
include /etc/snort/local.rules

# vision.rules will be catched by arachnids_upd
include /etc/snort/vision.rules
```

When you are done with setting up */etc/snort/snort.conf* you should start snort by calling */etc/rc.d/init.d/snortd start* and correct any errors you get in the log file */var/log/messages* (ignore any database related messages since the database has not been set up at this time, you also may have to document out the output module database). If everything is ok you can go on with configuring the other parts.

## 1.2. /etc/rc.d/init.d/snortd

In */etc/rc.d/init.d/snortd* you should edit at least the line with the interface to be "snort'ed". Replace the definition of *INTERFACE="eth0"* with the interface you use. This can be another ethernet (*ethx*) but also a *pppx* or *ipppx* interface, e.g. if you are using ISDN your definition should be like

```
INTERFACE="ippp0"
```

If your snort sensor is only listening on one interface it's sufficient to use the shipped snortd initscript. But if you have more than one interface you may be interested in having a look onto the script I extended for exactly that case. Even when you only have one interface but wish to use swatch the way I do you could copy the swatch parts to the shipped snortd script (see the contrib section of the RPM's documentation).

Next you find the mentioned snortd initscript I extended for snort to listen on more than one interface. One could now say that you can also use *any* as an interface name since the underlying *libpcap* makes this possible, but that's not what I intended to use because I'm not interested in "snorting" the local network where the snort sensor is set up. This should - in a secure environment - be a separate network segment with additional security set up, e.g. a firewall for that segment, so sniffing does not make much sense except if you want to sniff attacks targeted to the snort network itself. Even then, if you use more than one sensor concentrated in that segment you only need to set up one but not all of the sensors for protecting the segment.

I added a new function *daemonMult* derived from RedHat's *daemon* function found in */etc/rc.d/init.d/functions* which is capable of starting a program more than once. I sent

RedHat a patch for their *daemon* function to introduce a new option *--mult* which eventually will be added. If that happens the *daemonMult* function will be obsolete and the call to snort would change from *daemonMult ...* to *daemon --mult ....* Let's wait and see.

I also changed the subsystem name from snort to snortd to get rid of error messages when rebooting (the killall script on a redhat box depends on the correct name), just a little typo.

With my script you can now define multiple interfaces to be watched on, just use a space separated list with the *INTERFACE* variable, like in the listing shown below.

Some sanity checks are also included to see if the interface to listen on is already up and if there is an IP address defined. If there is an IP address defined the correspondig config which on a RedHat linux box is found in */etc/sysconfig/network-scripts/ifcfg-<interface name>* will be used, else the interface is set up as IP-less in promiscuous mode.

THIS HAS NOT YET BEEN TESTED WITH ANYTHING ELSE THAN ETHERNET INTERFACES! I WILL HOPEFULLY SOON REVIEW IT WITH ISDN INTERFACES AND REPORT HOW THE DIFFERENCES ARE!

A single snort process is then started on each interface, and also *swatch* will be started to check for errors when restarting snort for rule updates (see the *swatch* section below).

When shutting down snort all IP-less interfaces will be shut down but not any interfaces with existing IP configurations because that could last to inaccessability if the "snort'ed" interface is vital for the snort sensor (learned that the hard way >;)

Maybe a better solution would be to check the interface's config file for an entry like

```
     ONBOOT=yes
```

and only if there is not *yes* then the interface will be shut down. But that's not yet implemented.

Now here is the extended snort initscript:

```
#!/bin/sh
#
# snortd         Start/Stop the snort IDS daemon.
#
# chkconfig: 2345 40 60
# description:  snort is a lightweight network intrusion detection tool
that
#               currently detects more than 1100 host and network
```

```
#               vulnerabilities, portscans, backdoors, and more.
#
# June 10, 2000 -- Dave Wreski Dave Wreski <dave at linuxsecurity.com>
#   - initial version
# July 08, 2000 Dave Wreski <<dave at guardiandigital.com>
#   - added snort user/group
#   - support for 1.6.2
# April 11, 2001 Sandro Poppi <spoppi at gmx.de>
#   - added multiple interfaces option for use with dial up lines
#     or more than one sniffer interface
#     I don't think the libpcap option to use "-i any" is a good
choice,
#     because snort would be set up to monitor one or more ip-less
interfaces
#     while leaving the monitor interface "unprotected"
#   - changed the subsystem name from snort to snortd to get rid of
error messages
#     when rebooting (the killall script on a redhat box depends on the
correct name)
#   - added a function daemonMult derived from the function daemon in
/etc/rc.d/init.d/functions
#     to allow starting multiple instances of snort with the
convenience of the daemon function
#     (eventually this could be integrated into the normal daemon
function of redhat, have to get
#     in touch with the author)
# January 01, 2002 Sandro Poppi <spoppi at gmx.de>
#   - added check if swatch is installed
#   - added check for interfaces other than ethernet since only those
are expected to work with ifconfig
#
# Source function library.
. /etc/rc.d/init.d/functions

# A function to start a program even more than once
# rewritten version of the daemon function in
/etc/rc.d/init.d/functions
daemonMult() {
        # Test syntax.
        gotbase=
        user=
        nicelevel=0
        while [ "$1" != "${1##-}" -o "$1" != "${1##+}" ]; do
          case $1 in
            '')    echo '$0: Usage: daemon [+/-nicelevel] {program}'
                   return 1;;
            --check)
                   shift
                   base=$1
                   gotbase="yes"
                   shift
                   ;;
            --user)
```

```
                    shift
                    daemon_user=$1
                    shift
                    ;;
              -*|+*) nicelevel=$1
                    shift
                    ;;
               *)   nicelevel=0
                    ;;
          esac
        done

        # Save basename.
        [ -z $gotbase ] && base=`basename $1`

        # make sure it doesn't core dump anywhere; while this could
mask
        # problems with the daemon, it also closes some security
problems
        ulimit -S -c 0 >/dev/null 2>&1

        # Echo daemon
        [ "$BOOTUP" = "verbose" ] && echo -n " $base"

        # And start it up.
        if [ -z "$daemon_user" ]; then
           nice -n $nicelevel initlog $INITLOG_ARGS -c "$*" && success
"$base startup" || failure "$base startup"
        else
           nice -n $nicelevel initlog $INITLOG_ARGS -c "su $daemon_user
-c \"$*\"" && success "$base startup" || failure "$base startup"
        fi
}

# Specify your network interface(s) here
INTERFACE="eth1 eth2"

# See how we were called.
case "$1" in
start)
        if [ -x /usr/bin/swatch ] ; then
          echo -n "Starting swatch: "
          # inserted poppi to make use of swatch
          # starting it before snort to get hints on startup errors of
snort
          # if using the snort option -s use /var/log/secure,
          # if using output alert_syslog: in snort.conf use
/var/log/messages
          /usr/bin/swatch --daemon --tail /var/log/messages --config-
file /etc/swatch/swatchrc &
          touch /var/lock/subsys/swatch
          echo "done."
          echo
```

```
        fi

        # added multiple interfaces option
        for i in `echo "$INTERFACE"` ; do
          echo -n "Starting snort on interface $i: "
          # inserted to implement ip-less sniffer interface for snort
at startup
          # if the interface is not yet loaded or if the interface
isn't up yet
          if [ `/sbin/ifconfig $i 2>&1 | /bin/grep -c "Device not
found"` = "0" \
               -o `/sbin/ifconfig $i 2>&1 | /bin/grep -c "UP"` =
"0" ] ; then

            # check for interfaces other than ethernet!
            if [ `echo $i | /bin/grep -c "^eth"` = "1" ] ; then
              # check if there is a config for the given interface
              # normally this should be omitted for security reasons
for a sniffer interface
              if [ -s "/etc/sysconfig/network-scripts/ifcfg-$i" ]; then
                # use the config
                /sbin/ifup $i
              else
                # ip less sniffer interface
                /sbin/ifconfig $i up promisc
              fi
            fi
          fi
          # call the rewritten daemon function from above
          daemonMult /usr/sbin/snort -u snort -g snort -d -D \
                 -i $i -I -l /var/log/snort -c /etc/snort/snort.conf
          echo
        done

        touch /var/lock/subsys/snortd

        ;;
  stop)
        echo -n "Stopping snort: "
        killproc snort
        rm -f /var/lock/subsys/snortd

        # inserted Poppi
        if [ -x /usr/bin/swatch ] ; then
          echo
          echo -n "Stopping swatch: "
          kill `ps x|grep "/usr/bin/swatch"|grep -v grep|awk '{ print
$1 }'`
          rm -f /var/lock/subsys/swatch
        fi

        # shutdown interface if and only if it has NO ip address
        # and if it is a ethernet interface
```

```
        # this is done because we don't want to shutdown interfaces
still needed
        for i in `echo "$INTERFACES"`; do
          if [`echo $i | /bin/grep -c "^eth"` = "1" -a \
              `/sbin/ifconfig $i 2>&1 | /bin/grep -c "inet addr:"` =
"0" ] ; then
            /sbin/ifconfig $i down
          fi
        done
        echo
        ;;
  restart)
        $0 stop
        $0 start
        ;;
  status)
        status snort
        #status swatch
        ;;
  *)
        echo "Usage: $0 {start|stop|restart|status}"
        exit 1
esac
exit 0
```

### 1.3. /etc/snort/snort-check

This shell script is used to generate winpopups via *smbclient* or sending emails to given persons. It was inspired by Bill Richardson's script published on the snort homepage.

The winpopup part may be obsoleted by the *smb* output module introduced in snort 1.8 but I haven't tested it yet.

```
#!/bin/sh

# Script to be run from within swatch to send alerts in multiple
formats
# inspired from script on www.snort.org by Bill Richardson
# extended to read a file called "hosts" with names of
# workstation to send a winpopup, syntax is the same as with snortd
option -M
# Poppi, 02.05.2001

# Prerequisites:
# Samba set up correctly
# Change the following variables according to your system (for RedHat
7.x user it should be ok)

# hostfile holds the name of the file containing the workstation for
winpopups
```

```
hostfile="/etc/snort/hosts"

# recipientfile holds the addresses of all recipients in a single file,
# seperated by newline
recipientfile="/etc/snort/recipients"

# if a recipient file exists
if [ -s "$recipientfile" ] ; then
  # generate the recipientlist with email adresses.
  for i in `cat $recipientfile` ; do
    recipients="$recipients "$i
  done

  echo "$*" | mail -s "Snort-Alert!!!" "$recipients"
fi

# if a hostfile exists, send winpopups
if [ -s "$hostfile" ] ; then
  for i in `cat $hostfile` ; do
    echo "Snort-Alert! $*" | smbclient -M $i > /dev/null 2>&1
  done
fi
```

#### 1.3.1. /etc/snort/hosts

In this file you put in all the workstation names of the hosts which should get the snort message, one per line:

```
ws001
ws002
ws003
```

#### 1.3.2. /etc/snort/recipients

In */etc/snort/recipients* you put in email addresses of recipients who wish (or are urged to ;) receive your snort alarms, one address per line:

```
jane@internal.local.com
henk@snort.info
sandro@snort.info
```

If any of these two files is omitted then the corresponding feature is disabled.

### 1.4. Snort internal Statistics

Snort has the ability built in to print out some internal statistics. This can be achieved using the following command:

**/bin/kill -SIGUSR1 <pid of snort>**

or if you have more than one snort process running on the same machine and want to get info about all at once:

**/bin/killall -USR1 snort**

With either of these commands you get internal statistics in the following way in your syslog (*/var/log/messages* with RedHat):

```
Sep 29 07:51:48 ids01 snort[8000]:
===============================================================================
Sep 29 07:51:48 ids01 snort[8000]: Snort analyzed 27316 out of 27316 packets,
Sep 29 07:51:48 ids01 snort[8000]: dropping 0(0.000%) packets
Sep 29 07:51:48 ids01 snort[8000]: Breakdown by protocol:                Action
Stats:
Sep 29 07:51:48 ids01 snort[8000]:     TCP: 27152      (99.400%)
ALERTS: 0
Sep 29 07:51:48 ids01 snort[8000]:     UDP: 0          (0.000%)
LOGGED: 0
Sep 29 07:51:48 ids01 snort[8000]:    ICMP: 164        (0.600%)
PASSED: 0
Sep 29 07:51:48 ids01 snort[8000]:     ARP: 0          (0.000%)
Sep 29 07:51:48 ids01 snort[8000]:    IPv6: 0          (0.000%)
Sep 29 07:51:48 ids01 snort[8000]:     IPX: 0          (0.000%)
Sep 29 07:51:48 ids01 snort[8000]:   OTHER: 0          (0.000%)
Sep 29 07:51:48 ids01 snort[8000]: DISCARD: 0          (0.000%)
Sep 29 07:51:48 ids01 snort[8000]:
===============================================================================
Sep 29 07:51:48 ids01 snort[8000]: Fragmentation Stats:
Sep 29 07:51:48 ids01 snort[8000]: Fragmented IP Packets: 0          (0.000%)
Sep 29 07:51:48 ids01 snort[8000]:     Fragment Trackers: 0
Sep 29 07:51:48 ids01 snort[8000]:    Rebuilt IP Packets: 0
Sep 29 07:51:48 ids01 snort[8000]:    Frag elements used: 0
Sep 29 07:51:48 ids01 snort[8000]: Discarded(incomplete): 0
Sep 29 07:51:48 ids01 snort[8000]:    Discarded(timeout): 0
Sep 29 07:51:48 ids01 snort[8000]:   Frag2 memory faults: 0
Sep 29 07:51:48 ids01 snort[8000]:
===============================================================================
Sep 29 07:51:48 ids01 snort[8000]: TCP Stream Reassembly Stats:
Sep 29 07:51:48 ids01 snort[8000]:         TCP Packets Used: 27152
(99.400%)
Sep 29 07:51:48 ids01 snort[8000]:          Stream Trackers: 1
Sep 29 07:51:48 ids01 snort[8000]:           Stream flushes: 0
Sep 29 07:51:48 ids01 snort[8000]:            Segments used: 0
Sep 29 07:51:48 ids01 snort[8000]:    Stream4 Memory Faults: 0
Sep 29 07:51:48 ids01 snort[8000]:
===============================================================================
```

But remember: With versions prior to 1.8.3 you have to restart snort to get new statistics, so always combine the **kill -SIGUSR1** with a snort restart if not using the actual version!

You first should have a look on the first 2 lines. If snort tells you that there are dropped packets you have to take a very close look on your configuration of the snort box itself not only (but including) the snort configuration.

E.g. stop all unnecessary services which are not vital for the box. And take a look on the output of the **top** command. If the idle counter is very low you should figure out which processes eat up all of your cpu time and eventually outsource the corresponding program packets. This is e.g. true when using ACID and the underlying database and snort on the same machine with less memory and/or cpu.

The other statistical data lines give you an overview of some of the preprocessors and their work. You should also have a look on the memory faults sections. If the number is not 0 you should have a look on your memory usage and eventually configure the preprocessors to use more memory (take a look to the appropriate section in */etc/snort/snort.conf*).

Now a short script which I was inspired by Greg Sarsons to get snort's internal statistics, save them to a file and restart snort.

The statistics file will be archived to */var/log/snort/archive* so you have to create that directory first ;)

```
#!/bin/bash
# Script to generate and extract snort statistics from syslog or given
file
# generated after kill -USR1 <snort-pid>
#
# This script assumes that the pid is logged into the logfile!
# This can be obtained using  the following line in snort.conf:
# output alert_syslog: LOG_AUTH LOG_ALERT LOG_PID
#
# (c) Sandro Poppi 2001
# Released under GPL

echo "Starting gathering snort internal statistics. Please be
patient..."

if [ "$1." == "." -o ! -e "$1" ] ; then
  # no or unexistent file given, using default
  log_file="/var/log/messages"

else
```

```
  # when using non-standard logfile location make sure snort uses this
logfile
  # when sending signal USR1 else this script won't work!
  log_file="$1"
fi

# find out snort pids
snort_pid=`/sbin/pidof snort`

# get internal statistics for all snort processes
# not using killall to get already sorted output
for i in `echo $snort_pid` ; do
  kill -USR1 $i

  # sleep for 2 secs to let snort time to send statistics to syslog ;)
  sleep 2
done

# immediately restart snort after sending signal USR1
# this may be ommitted when using CVS version of snort after about
01.11.2001
# or any version from 1.8.2 or higher
/etc/rc.d/init.d/snortd restart

for i in `echo $snort_pid` ; do
  # process logfile

  filename=/var/log/snort/archive/snort.`date "+%Y-%m-%d"`.$i.log

  # check for existing file and rename it if existing
  if [ -e "$filename" ] ; then
    mv "$filename" "$filename.bak"
  fi

  egrep "snort\[$i\]:" $log_file > "$filename"

  # check if there are dropped packets using lines like
  # Oct 22 18:02:06 xbgh17183 snort[573]: dropping 0(0.000%) packets
  if [ "`egrep "dropping" $filename | awk -F "[ (]" '{ print $7 }'`" !=
"0" -a \
       "`egrep -c "dropping" $filename`" != "0" ] ; then
    echo "Snort's dropping packets!!! Take a look on the configuration
and/or the system's performance!!!"
  fi

done

echo "Gathering snort internal statistics finished..."
```

## 1.5. Testing Snort

To test snort you should edit */etc/rc.d/init.d/snortd* and make the interface listen on the loopback device *lo*. For people with a network card installed you can use *eth0* instead but you have to use a second pc to run snot because no packet is sent over the interface if snot and snort are run on the same machine!

Probably the simplest way to test snort is to use *snot* which can be found on http://www.sec33.com/sniph/.

You have to have libnet installed for snot. Since on RedHat 7.x there is no RPM available you could use *libnet-1.0.2-6mdk.i586.rpm* from Mandrake Soft, which can be found on http://rpmfind.net/ and of course on Mandrake's site http://www.mandrake.com/. Most Mandrake RPMs could be used with no problem on a RedHat system. But be warned: Mandrake does not provide *i386* RPMs so you can't use them with a processor less than an old Pentium P5. In such a case you have to get the sources from http://www.packetfactory.net/projects/libnet and compile it from scratch yourself.

To compile snot you only have to untar the tarball, cd into the snot directory and call *make*. If compilation exits without an error snot is ready to use, if not you are almost always missing some development packages.

To prepare snot you should first copy */etc/snort/snort.conf* into the snot directory and *cat* one or more rule files to the end of the copied *snort.conf* using e.g.:

**cat /etc/snort/backdoor.rules >> snort.conf**

Then on one console you should call **tail -f /var/log/messages**, while on another you should try to run the tests.

Snot can then be called the following way assuming you used *lo* as the interface name in the snortd initscript:

**./snot -r snort.conf -d localhost -n 5**

With that command you tell snot to use the copied *snort.conf*, the destination is *localhost* and for not triggering too many alerts restrict it to a maximum of 5.

You'll probably get some messages saying ignoring additional parameters because snot can not handle yet the new parameters introduced in snort 1.8. Don't panic, just ignore the messages, snot works fine though.

In */var/log/messages* you should now see some snort alerts, e.g.:

```
Sep 10 18:22:33 ids01 snort[1536]: <lo> GateCrasher access:
192.168.213.151:6969 -> 127.0.0.1:3170
```

```
Sep 10 18:22:33 ids01 snort[1536]: <lo> GateCrasher access:
192.168.213.151:6969 -> 127.0.0.1:3170
Sep 10 18:22:33 ids01 snort[1536]: <lo> GateCrasher access:
192.168.155.231:6969 -> 127.0.0.1:57580
Sep 10 18:22:33 ids01 snort[1536]: <lo> GateCrasher access:
192.168.155.231:6969 -> 127.0.0.1:57580
Sep 10 18:22:33 ids01 snort[1536]: <lo> Deep Throat access:
192.168.170.42:2140 -> 127.0.0.1:60521
```

If you get similiar alerts it's ok, if not please take again a look on your configuration until you get this far.

Now it's time to edit */etc/snort/snort.conf* again and put in the correct value to the *INTERFACE* variable, restart snort and get a cup of coffee. You have deserved it!