

3M IPS Multiqueue Multilevel Multiservice

Pierpaolo Palazzoli, Brescia, Italy
Fabio Mostarda, Brescia, Italy
Claudia Ghelfi, Brescia, Italy

The IPS acronym references an Intrusion Prevention System, i.e. a network security device. It features network traffic flow monitoring in order to provide real-time attack blocking capabilities against any threat, both criminal and unintentional.

The IPS technology originates as an extension of Intrusion Detection Systems, and it's still tightly bound to them.

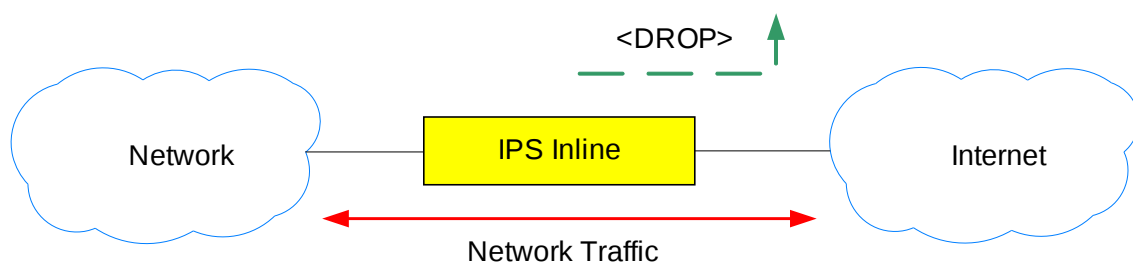
Compared to an Intrusion Detection System, which only listens to the traffic flow, an IPS has the advantage of real time blocking against malicious traffic, following administrator-defined security policies.

An IPS can be implemented with a device where the Snort software does the IPS job, while Iptables endorses firewall duties and, with additional modules, creates software queues for the packets to be analyzed.

SNORT

Snort is an intrusion detection and prevention system (ISPD), and can operate in two different ways:

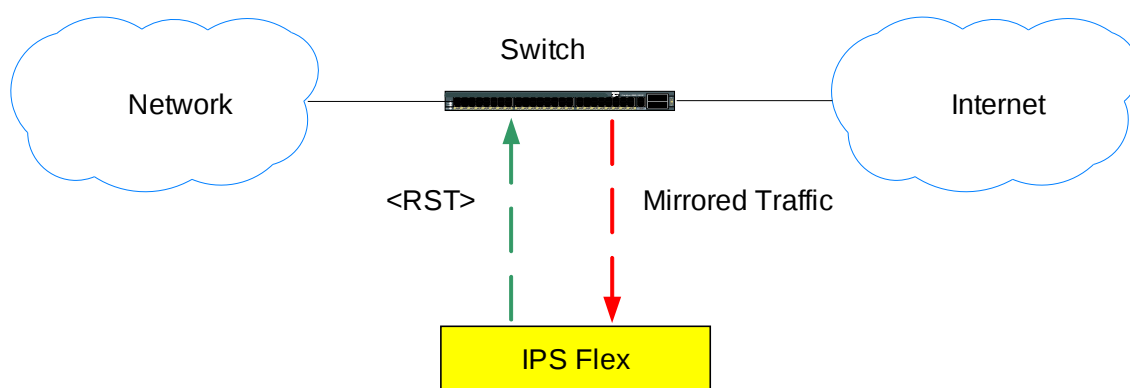
- o *Inline*. In this mode Snort works like an Ethernet bridge, that is, in order to monitor a network segment, it has to be inserted transparently with two bridged NICs.



With this setup, any packet can flow through the bridge from a network card to the other, unless it matches the drop rules; in that case, the switch opens and blocks the packet.

- o *Flex Response*. With this second setup, all the inbound traffic is replicated by a network device (i.e. a switch) to a "mirror port". The IPS is then linked to that

port, analyzing the traffic flow and sending RST packets whenever it detects a match in the drop policies.



In both setups the IPS is transparent and utterly invisible to the network.

IPTABLES

Between the two previous Snort setups, the one featuring the best performance is the Inline mode, because it offers protection against attacks delivered by a small number of packets.

On the contrary, it takes more time to the flex-response mode to identify an intrusion and then to reset the proper port.

The usage of an inline IPS doesn't fix all network security issues, but it contributes to the centralized structuring of a security system that is both dynamic and efficient.

Inline Snort works by analyzing packet queues, and therefore it's mandatory the usage of a queue creation system, additionally to Snort itself.

If "Snort inline mode" is used, then you can create just a single queue, while "Snort_inline" allows unlimited queues.

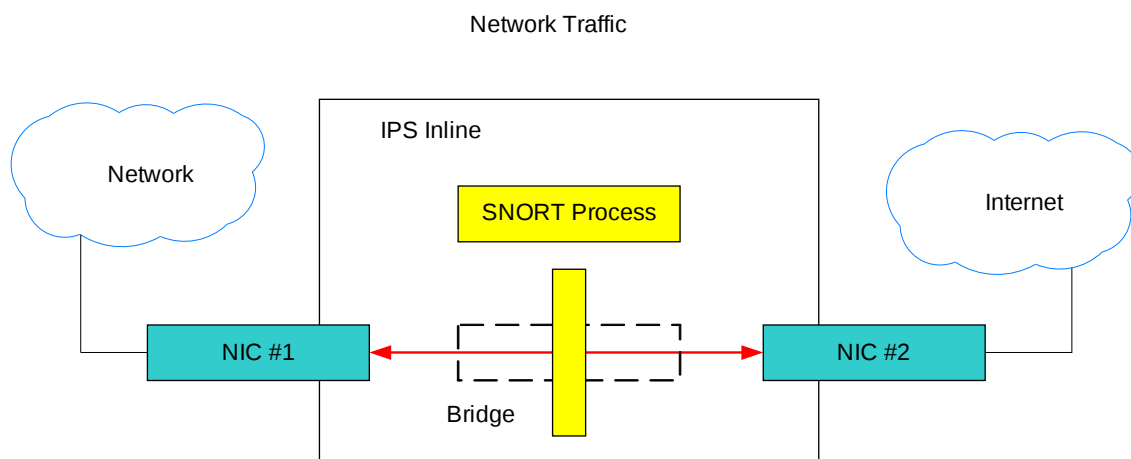
Iptables, additionally to the standard Linux firewall features, performs the queue-creation function, using additional modules like `ip_queue` or `nfnetlink_queue`.

The difference between these two modules marks the mono-queue IPS from the multi-queue IPS.

MONOQUEUE IPS

Whenever `ip_queue` is chosen to create queues, then Snort can use a single queue only, over which every inbound packet is forced to flow.

This queue is then analyzed by Snort according to a single set of rules; it's not feasible to differentiate inbound network flows and to apply custom blocking policies.



MULTIQUEUE IPS

Leveraging the `nfnetlink_queue` module instead of `ip_queue`, it's theoretically possible to create up to 65535 independent queues.

This allows inbound flows differentiation, that is, it's feasible to separate packets flowing through different VLANs by creating virtual bridges.

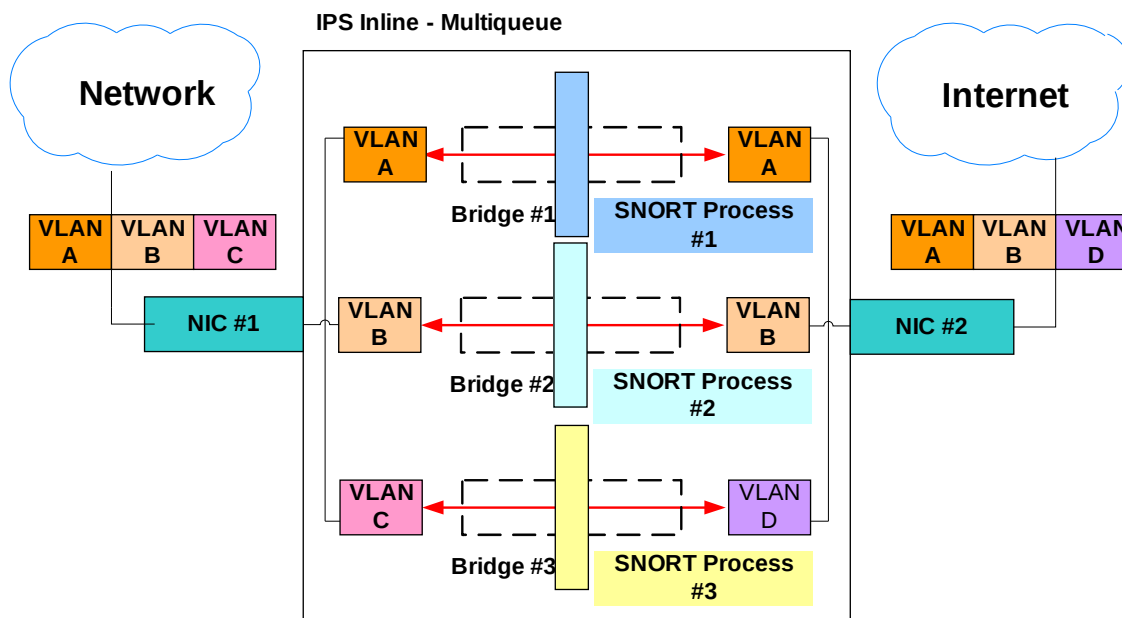
Leveraging the Ethernet VLAN tag, a manageable switch is able to recognize the VLAN of the incoming packet, and therefore to route it to the proper queue.

With this setup, each flow analysis can be done independently one from another, and it's possible to deploy different Snort versions as well as queue-specific traffic analysis rules, which are customized according to every VLAN's requirements.

According to the IEEE 802.1q standard, the maximum number of VLANs is 1024/2048, and therefore the limit to the bridge number is 512/1024.

Inside every queue it's feasible to define sub-queues, differentiated according to service and marked by the port number.

Thanks to this additional feature, Snort rules can be applied over any single network flow, thus improving the system efficiency.



APPLICATIONS

The creation of an IPS requires the following steps:

1. Operating System and Libraries install

Libnfnetlink-0.0.39 and libnetfilter_queue-0.0.16 libraries are required. If an OS like Debian-4.0rc5 or Fedora-10 is chosen, then you can easily add these libraries later.

The OS install procedure can be completed with the “minimal install” setup; for our walkthrough, we’ll use Debian-4.0r5.

After the install, it’s better to modify the sources.list file in the following way, in order to download the proper libraries later:

```
vi /etc/apt/sources.list
```

Disable all the instructions by putting # before them and add:

```
deb http://ftp.it.debian.org/debian/ unstable main contrib non-free
```

install ssh to allow remote management:

```
apt-get install ssh
```

remove unnecessary packages, i.e. everything but ssh:

```
netstat -taupen
```

(shows the list of applications that are running daemons on specific ports)

```
apt-get remove portmap exim4 exim4-config openbsd-inetd
```

(removes the specified unnecessary packets)

Now the update of the installed packages and their dependencies:

```
apt-get update
```

```
apt-get upgrade
```

and now it’s possible to install the proper libraries for the multique IPS, using the following commands:

```
apt-get install libnfnetlink0
apt-get install libnfnetlink-dev
apt-get install libnetfilter-queue1
apt-get install libnetfilter-queue-dev
```

To check the installed library versions, you can use the command: `dpkg -l [library]`.

2. New kernel installation

To work properly with libnetfilter libraries, it's necessary to use a kernel version higher than 2.6.25; in this case, install the following:

```
apt-get install linux-image-2.6-686
apt-get install linux-image-2.6-686-bigmem
```

(more than 3GB RAM)

```
reboot
```

(reboots the machine)

```
uname -a
```

(checks if the new kernel is currently activated)

3. Additional packet installation

In addition to the previous libraries, it's necessary to install the following packages in order to ensure the correct system operation:

```
apt-get install mlocate
```

(to use functions like updatedb, locate,...)

```
apt-get install ntpdate
```

(PC clock sync protocol)

```
apt-get install vim
```

(to ease the use of vi)

```
apt-get install psmisc
```

(package featuring killall)

```
apt-get install iptraf
```

(statistical traffic generator for IP network)

```
apt-get install bridge-utils
```

(Linux Ethernet bridges configuration utilities)

```
apt-get install gcc
```

(GNU compiler)

```
apt-get install libpcap0.8
```

(packet capture and network monitoring libraries)

```
apt-get install libpcap0.8-dev
```

(source code of the previous libraries that you may have to include)

```
apt-get install libpcre3
```

(PCRE libraries (Perl Compatible Regular Expression) featuring functions that implement commands with Pearl-like syntax and semantics)

```
apt-get install libpcre3-dev
```

(pcre libraries source code)

```
apt-get install libmysqlclient15-dev
```

(libraries needed to access and write mysql)

```
apt-get install iptables-dev
```

(iptables source code, needed to include packet queuing features)

```
apt-get install automake autoconf make
```

(compiler packages)

```
apt-get install libtool
```

(Snort required libraries)

```
apt-get install libclamav-dev clamav-freshclam
```

(needed to allow Snort to access Clamav (OpenSource Antivirus) signatures)

```
apt-get install libnet1-dev
```

(source code enabling a programmer to build and insert network packages)

```
apt-get install subversion
```

(client server versioning system: a server stores the current project version and its history, and the client connects to it to verify the latest software version)

```
apt-get install ntop
```

(traffic monitoring tool)

```
apt-get install apache2
```

(modular web server platform)

```
apt-get install rrdtool
```

(data storage and presentation graphical system)

```
apt-get install librrdp-perl librrds-perl
```

(perl interfaces to RRD)

4. Snort_inline installation

```
svn co https://snort-inline.svn.sourceforge.net/svnroot/snort-inline/trunk
```

(downloads the updated snort_inline)

```
cd /trunk
```

```
sh autojunk.sh
```

(prepares the package to be compiled)

```
./configure --enable-pthread --enable-memory-cleanup --enable-stream4udp --enable-inline-init-failopen --enable-nfnetlink --enable-clamav --enable-linux-smp-stats --with-mysql
```

(checks the intallation requirements)

```
make
```

```
make install
```

(installs snort_inline)

5. Snort rules download

Whenever a rules backup is not available, those rules can be downloaded online in the following way:

- o Choose the download subscription class regarding the rules:
 - Subscribers: they get the updates as soon as those are available. To access this user class a yearly fee is required, which varies according to the number of sensors:

http://www.snort.org/vrt/why_subscribe.html.

- Registered: to this class belong users who registered on the snort.org portal. They get the new rules 30 days after their development, but they don't pay any fee.
 - Unregistered: unregistered users who get new rules bundled with Snort major version updates only.
 - o Choosing the subscription will allow to receive a code (oinkcode) that will be the key to use oinkmaster.
 - o Now it's possible to install Oinkmaster, an automated rules management software that keeps the rule database updated adding a crontab line like:

```
30 2 * * * oinkmaster.pl -o /etc/snort/rules/ -t oinkmaster
```

 Any change is notified to the user; Oinkmaster can be downloaded at <http://oinkmaster.sourceforge.net/download.shtml>
- Additionally to those rules, it's possible to include the ones available at <http://www.emergingthreats.net/>.

6. Rc.local configuration

Rc.local can be generally found among the startup scripts (in /etc/) and it is run at the end of the startup procedure. It contains user-defined commands that are to be run after all services have been started.

Before compiling this file, it's necessary to clearly understand the architecture that is to be implemented, especially the number of queues and bridges (both physical and logical) that will be required.

In the following pages, four base configuration examples are explained, operating on level 1,2,3,4 of the ISO/OSI architecture. It's then possible, starting from those examples, to create more complex solutions involving multiple levels at the same time, featuring a multilevel service in a multiqueue setup, thus allowing a multiservice protection.

physical
Media, Signal
and Binary Transmission

Case One: two layer 1 queues (physical)

```
/sbin/ifconfig eth4 0.0.0.0 promisc up
/sbin/ifconfig eth5 0.0.0.0 promisc up
/sbin/ifconfig eth0 0.0.0.0 promisc up
/sbin/ifconfig eth1 0.0.0.0 promisc up
brctl addbr br0
brctl addbr br1
```

(two bridges are defined, br0 and br1, both physical)

```
brctl addif br0 eth4
brctl addif br0 eth5
```

(the first bridge is built, between interfaces eth4 and eth5)

```
brctl addif br1 eth0
brctl addif br1 eth1
```

(second bridge between interfaces eth0 and eth1)

```
ifconfig br0 up promisc
ifconfig br1 up promisc
```

Those commands couple the two network interface cards at layer 2, creating two bridges so that the IPS looks transparent to the network traffic.

To setup the bridges, it's necessary to have the bridge-utils package installed.

```
modprobe nfnetlink_queue
modprobe nf_contrack
```

Those modules are needed to create a multiqueue setup

```
echo "VALUE" > /proc/sys/net/nf_contrack_max
```

This command is necessary for a multiqueue system: VALUE is to be replaced with the number of sessions that the kernel must be able to sustain with nf_contrack (i.e. 262140).

```
echo 1 > /proc/sys/net/ipv4/tcp_tw_recycle
echo "3" > /proc/sys/net/ipv4/tcp_fin_timeout
```

Commands oriented to session support at kernel level.

```
sysctl -w net.core.rmem_default='8388608'
sysctl -w net.core.wmem_default='8388608'
sysctl -w net.ipv4.tcp_wmem='1048576 4194304 16777216'
sysctl -w net.ipv4.tcp_rmem='1048576 4194304 16777216'
sysctl -w net.ipv4.route.flush=1
```

Optimized parameters for Gigabit Ethernet speeds.

```
iptables -A FORWARD -i br0 -j NFQUEUE --queue-num 42
iptables -A FORWARD -i br1 -j NFQUEUE --queue-num 43
```

The following commands generate queues nr. 42 and 43, linked to the physical bridges br0 and br1 respectively.

```
snort_inline -A fast -b -d -D -Q42 -c /etc/snort/snort.conf -l
/var/log/snort/42/ --nolock-pidfile
snort_inline -A fast -b -d -D -Q43 -c /etc/snort/snort.conf -l
/var/log/snort/43/ --nolock-pidfile
```

Snort is started on both queues.

data link
Physical Addressing (MAC & LLC)

physical
Media, Signal
and Binary Transmission

Case Two: two layer 2 queues

```
/sbin/ifconfig eth2 0.0.0.0 promisc up
/sbin/ifconfig eth3 0.0.0.0 promisc up
brctl addbr br0
brctl addif br0 eth2
brctl addif br0 eth3
ifconfig br0 up promisc
```

Physical bridge between interfaces eth2 and eth3.

```
vconfig add eth2 5
vconfig add eth2 7
vconfig set_flag eth2.5 0
```

```
vconfig set_flag eth2.7 0
/sbin/ifconfig eth2.5 0.0.0.0 promisc up
/sbin/ifconfig eth2.7 0.0.0.0 promisc up
```

On the interface eth2, VLAN 5 and 7 are defined on the corresponding virtual interfaces eth2.5 and eth2.7.

```
vconfig add eth3 6
vconfig add eth3 8
vconfig set_flag eth3.6 0
vconfig set_flag eth3.8 0
/sbin/ifconfig eth3.6 0.0.0.0 promisc up
/sbin/ifconfig eth3.8 0.0.0.0 promisc up
```

VLAN 6 and 8 are defined on the interface eth3, with the virtual interfaces eth3.6 e eth3.8.

```
brctl addbr br1
brctl addif br1 eth2.5
brctl addif br1 eth3.6
ifconfig br1 up promisc
brctl addbr br2
brctl addif br2 eth2.7
brctl addif br2 eth3.8
ifconfig br2 up promisc
```

Setup of logical bridges br1 (between interfaces eth2.5 and eth3.6) and br2 (between eth2.7 and eth3.8).

```
modprobe nfnetlink_queue
modprobe nf_conntrack
```

Modules required for multiqueue setup.

```
echo "VALUE" > /proc/sys/net/nf_conntrack_max
```

This command is necessary for a multiqueue system: VALUE is to be replaced with the number of sessions that the kernel must be able to sustain with nf_conntrack (e.g. 262140).

```
echo "3" > /proc/sys/net/ipv4/tcp_fin_timeout
echo 1 > /proc/sys/net/ipv4/tcp_tw_recycle
```

Commands oriented to session support at kernel level.

```
sysctl -w net.core.rmem_default='8388608'
sysctl -w net.core.wmem_default='8388608'
sysctl -w net.ipv4.tcp_wmem='1048576 4194304 16777216'
sysctl -w net.ipv4.tcp_rmem='1048576 4194304 16777216'
sysctl -w net.ipv4.route.flush=1
```

Optimized parameters for Gigabit ethernet speeds.

```
iptables -A FORWARD -i br1 -j NFQUEUE --queue-num 42
iptables -A FORWARD -i br2 -j NFQUEUE --queue-num 43
```

The following commands generate queues nr. 42 and 43, linked to the physical bridges br0 and br1 respectively.

```
snort_inline -A fast -b -d -D -Q42 -c /etc/snort/snort.conf -
l /var/log/snort/42 --nolock-pidfile
snort_inline -A fast -b -d -D -Q43 -c /etc/snort/snort.conf -
l /var/log/snort/43 --nolock-pidfile
```

Snort is started on both queues.



Case three: multiple layer 3 queues (IP)

```
/sbin/ifconfig eth0 0.0.0.0 promisc up
/sbin/ifconfig eth1 0.0.0.0 promisc up
/usr/sbin/brctl addbr br0
/usr/sbin/brctl addif br0 eth0
/usr/sbin/brctl addif br0 eth1
/sbin/ifconfig br0 up
```

Physical bridge between interfaces eth0 and eth1.

```
modprobe nfnetlink_queue
modprobe nf_conntrack
```

Modules required for multiqueue setup.

```
echo "VALUE" > /proc/sys/net/nf_conntrack_max
```

This command is necessary for a multiqueue system: VALUE is to be replaced with the number of sessions that the kernel must be able to sustain with nf_conntrack (e.g. 262140).

```
echo "3" > /proc/sys/net/ipv4/tcp_fin_timeout
echo 1 > /proc/sys/net/ipv4/tcp_tw_recycle
```

Commands oriented to session support at kernel level.

```
sysctl -w net.core.rmem_default=8388608
sysctl -w net.core.rmem_max=8388608
sysctl -w net.core.wmem_max=8388608
sysctl -w net.ipv4.tcp_rmem='4096 87380 8388608'
sysctl -w net.ipv4.tcp_wmem='4096 65536 8388608'
sysctl -w net.ipv4.tcp_mem='8388608 8388608 8388608'
sysctl -w net.ipv4.route.flush=1
```

Optimized parameters for Gigabit ethernet speeds.

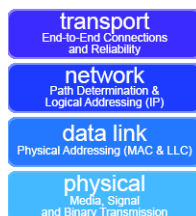
```
/sbin/iptables -A FORWARD -i br0 -d "ip1" -j NFQUEUE --queue-num
43
/sbin/iptables -A FORWARD -i br0 -s "ip1" -j NFQUEUE --queue-num 43
/sbin/iptables -A FORWARD -i br0 -d "ip2" -j NFQUEUE --queue-num 44
/sbin/iptables -A FORWARD -i br0 -s "ip2" -j NFQUEUE --queue-num 44
/sbin/iptables -A FORWARD -i br0 -j NFQUEUE --queue-num 42
```

Traffic from and to ip addresses “ip1” and “ip2” is selected and routed to queue 43 and 44 respectively.

Queue 42 elaborates everything that is not selected in the queues 43 and 44.

```
/usr/local/bin/snort_inline -A fast -b -d -D -Q42 -c /etc/snort/snort.conf
-l /var/log/snort/42/ --nolock-pidfile
/usr/local/bin/snort_inline -A fast -b -d -D -Q43 -c /etc/snort/snort.conf
-l /var/log/snort/43/ --nolock-pidfile
/usr/local/bin/snort_inline -A fast -b -d -D -Q44 -c /etc/snort/snort.conf
-l /var/log/snort/44/ --nolock-pidfile
```

Snort is started on every queue.



Case four: two layer 4 queues (TCP/UDP)

```
/sbin/ifconfig eth0 0.0.0.0 promisc up
/sbin/ifconfig eth1 0.0.0.0 promisc up
/usr/sbin/brctl addbr br0
/usr/sbin/brctl addif br0 eth0
/usr/sbin/brctl addif br0 eth1
ifconfig br0 up promisc
```

Physical bridge between interfaces eth0 and eth1.

```
modprobe nfnetlink_queue
modprobe nf_conntrack
```

Modules required for multiqueue setup..

```
echo "VALUE" > /proc/sys/net/nf_conntrack_max
```

This command is necessary for a multiqueue system: VALUE is to be replaced with the number of sessions that the kernel must be able to sustain with nf_conntrack (e.g. 262140).

```
echo "3" > /proc/sys/net/ipv4/tcp_fin_timeout
echo 1 > /proc/sys/net/ipv4/tcp_tw_recycle
```

Commands oriented to session support at kernel level..

```
sysctl -w net.core.rmem_default='8388608'
sysctl -w net.core.wmem_default='8388608'
sysctl -w net.ipv4.tcp_rmem='4096 87380 8388608'
sysctl -w net.ipv4.tcp_wmem='4096 65536 8388608'
sysctl -w net.ipv4.tcp_mem='8388608 8388608 8388608'
sysctl -w net.ipv4.route.flush=1
```

Optimized parameters for Gigabit ethernet speeds.

```
/sbin/iptables -A FORWARD -i br0 -p tcp --dport "port" -j NFQUEUE --
queue-num 43
```

```
/sbin/iptables -A FORWARD -i br0 -p tcp --sport "port" -j NFQUEUE --queue-num 43
```

```
iptables -A FORWARD -i br0 -j NFQUEUE --queue-num 42
```

Creation of queue 43 containing all TCP traffic to and from port “port”, while queue 42 handles all the remaining.

```
/usr/local/bin/snort_inline -A fast -b -d -D -Q43 -c /etc/snort/snort.conf -l /var/log/snort/43/ --nolock-pidfile
```

```
/usr/local/bin/snort_inline -A fast -b -d -D -Q42 -c /etc/snort/snort.conf -l /var/log/snort/42/ --nolock-pidfile
```

Snort is started on both queues.

Remarkably, if a different snort.conf were used in the command line starting snort, then it would be possible to customize the applied rule set.

After compiling rc.local, the directories to store the logs have to be created, according to the OS guidelines; in our example:

```
mkdir /var/log/snort
```

```
mkdir /var/log/snort/42
```

```
mkdir /var/log/snort/43 ..etc..
```

It's very important to reboot the system at this stage, in order to automatically build the bridges at startup and to have Snort up and running.

7. Snort configuration

The last element to customize is the Snort configuration file: snort.conf.

The following sections have to be edited for this specific configuration:

➤ Network variables definition:

```
var HOME_NET [10.0.0.0/8]
```

Through this line it is specified the network that has to be defended; in our example, we'll use 10.0.0.0/8

```
var EXTERNAL_NET !$HOME_NET
```

```
var HONEYNET $HOME_NET
```

```
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 SNMP_SERVERS $HOME_NET
```

```
var SSH_PORTS 22
```

```
var HTTP_PORTS [80,443]
```

```
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]
```

The previous instructions define the network variables.

```
var RULE_PATH /etc/snort/rules
```

Declares where the rules are.

➤ Preprocessor configuration

Preprocessors, introduced since version 1.5, perform a first traffic check to prepare it for the following rule-based analysis.

Every preprocessor handles a different protocol.

First of all, we declare the paths where the preprocessors are to be found:

```
config checksum_drop : all
dynamicpreprocessor directory /usr/local/lib/snort_dynamicpreprocessor/
dynamicengine /usr/local/lib/snort_dynamicengine/libsfe_engine.so
```

The command that allows the insertion of a preprocessor is the following:

```
preprocessor <name>: <options>
```

Usually, the most employed preprocessors are:

- The frag3 preprocessor reassembles fragmented traffic:

```
preprocessor frag3_global: max_frag 65536
```

the option (max_frag) deals about the maximum number of alignable fragments (the default value is 8192)

```
preprocessor frag3_engine: policy first detect_anomalies ttl_limit 255
```

those options handle the defragmenting modes (policy first), the anomalous packet detection (detect_anomalies) and the maximum TTL delta allowed on the first packet of the fragment (ttl_limit 255)

- The following preprocessors are complementary: they handle TCP and UDP protocols, and their setup is the following:

```
preprocessor flow: memcap 83886080, rows 8198, stats_interval 0 hash 2
```

Memcap defines the number of bytes to be allocated, rows the number of rows in the addressing table and hash the addressing mode.

```
preprocessor stream4: disable_evasion_alerts, enable_udp_sessions,
norm_window, max_sessions 32768, memcap 1000000000
```

Disable_evasion_alerts disables the warnings for events like TCP overlap, enable_udp_sessions enables UDP session tracking, and finally max_sessions and memcap define the maximum number of sessions and the maximum number of bytes respectively.

```
preprocessor clamav: ports 25, toserveronly, action-drop,
dbdir /var/lib/clamav, dbreload-time 3600
```

ClamAV preprocessor performs an antivirus action, actually on port 25 for the mail traffic only. Virus contaminated packets are dropped; the rules regarding this section are located in the directory /var/lib/clamav. Clamav works with stream4 only, and therefore with flow.

- Preprocessor Stream5 is used as an alternative to the previous group of preprocessors, because it handles TCP/UDP protocols too. We chose to use stream5. A stream5 configuration for our example could be:

```
preprocessor stream5_global: max_tcp 131070, track_tcp yes, track_udp yes, memcap 256000000
```

max_tcp identifies the maximum number of tcp session, and the two following options allow the analysis of TCP and UDP protocols, memcap defines the number of bytes to be allocated.

```
preprocessor stream5_tcp: policy first, use_static_footprint_sizes
```

this line configures stream5 in the TCP protocol analysis.

```
preprocessor stream5_udp: ignore_any_rules
```

this line configures stream5 in the UDP protocol analysis.

- The perfmonitor preprocessor delivers real time Snort benchmarking.

```
preprocessor perfmonitor: time 60 file /var/log/snort/perfmon.txt pktcnt 500
```

defines the check interval between measures and the file where the statistics are written.

- The HttpInspect preprocessor decodes http traffic and identifies application-level attacks exploiting http vulnerabilities. The configuration of this preprocessor is split in two parts, a global one and a server one:

```
preprocessor http_inspect: global iis_unicode_map unicode.map 1252
```

iis_unicode_map unicode.map 1252 has to be always specified because it handles the global IIS Unicode map

```
preprocessor http_inspect_server: server default \
```

```
profile all ports { 80 8080 8180 } oversize_dir_length 500 flow_depth 0
```

- The preprocessor dcerpc analyzes SMB, DCE/RPC protocols:

```
preprocessor dcerpc: \
  ports smb { 139 445 } \
  ports dcerpc { 135 } \
  max_frag_size 3000 \
  memcap 150000 \
  reassemble_increment 0
```

defines the ports where smb and dce/rp monitoring is activated., the maximum fragment size and the number of bytes to be allocated.

- The smtp preprocessor handles the email flow, and it is configured as follows:

```
preprocessor smtp: \
  ports { 25 } \
  inspection_type stateful \
  normalize_cmds \
  normalize_cmds { EXPN VRFY RCPT } \
  alt_max_command_line_len 260 { MAIL } \
  alt_max_command_line_len 300 { RCPT } \
  alt_max_command_line_len 500 { HELP HELO ETRN } \
```

```
alt_max_command_line_len 255 { EXPN VRFY }
```

those commands specify the port which the protocol analysis is to be activated on, and the stateful inspection mode. The last lines set the length of the smtp command lines that have to trigger alerts for some kind of services.

➤ Rule set customization:

In this last section of the configuration file there are the declarations of the rules to be activated. In our example:

```
include $RULE_PATH/classification.config  
include $RULE_PATH/reference.config
```

The command to add a rule is:

```
include $RULE_PATH
```

followed by the rule name. The complete rule list is available in the following pages. Depending on the specific requirements, the proper rules have to be selected and enabled.

- attack-responses.rules*
- backdoor.rules*
- bad-traffic.rules*
- chat.rules*
- community-bot.rules*
- community-deleted.rules*
- community-dos.rules*
- community-exploit.rules*
- community-ftp.rules*
- community-game.rules*
- community-icmp.rules*
- community-imap.rules*
- community-inappropriate.rules*
- community-mail-client.rules*
- community-misc.rules*
- community-nntp.rules*
- community-oracle.rules*
- community-policy.rules*
- community-sip.rules*
- community-smtp.rules*
- community-sql-injection.rules*
- community-virus.rules*
- community-web-attacks.rules*
- community-web-cgi.rules*
- community-web-client.rules*
- community-web-dos.rules*
- community-web-iis.rules*
- community-web-misc.rules*
- community-web-php.rules*

content-replace.rules
ddos.rules
dns.rules
dos.rules
emerging-attack_response.rules
emerging-botcc-BLOCK.rules
emerging-botcc.rules
emerging-compromised-BLOCK.rules
emerging-compromised.rules
emerging-dos.rules
emerging-drop-BLOCK.rules
emerging-drop.rules
emerging-dshield-BLOCK.rules
emerging-dshield.rules
emerging-exploit.rules
emerging-game.rules
emerging-inappropriate.rules
emerging-malware.rules
emerging-p2p.rules
emerging-policy.rules
emerging-rbn-BLOCK.rules
emerging-rbn.rules
emerging.rules
emerging-scan.rules
emerging-virus.rules
emerging-voip.rules
emerging-web.rules
emerging-web_sql_injection.rules
experimental.rules
exploit.rules
finger.rules
ftp.rules
icmp-info.rules
icmp.rules
imap.rules
info.rules
misc.rules
multimedia.rules
mysql.rules
netbios.rules
nntp.rules
oracle.rules
other-ids.rules
p2p.rules

policy.rules
pop2.rules
pop3.rules
porn.rules
rpc.rules
rservices.rules
scada.rules
scan.rules
shellcode.rules
smtp.rules
snmp.rules
specific-threats.rules
spyware-put.rules
sql.rules
telnet.rules
tftp.rules
virus.rules
voip.rules
web-activex.rules
web-attacks.rules
web-cgi.rules
web-client.rules
web-coldfusion.rules
web-frontpage.rules
web-iis.rules
web-misc.rules
web-php.rules
x11.rules

A detailed description of every Snort configuration parameter is available at:
http://www.snort.org/docs/snort_htmanuals/htmanual_283/