

# Configuring IPTABLES for Snort-Inline

by  
Tim Slighter

*January 23, 2003*

## **Brief background on Snort-Inline**

Snort-Inline is more or less a snort binary that can be configured to receive specific input from IPTABLES. The key syntax used with IPTABLES is "QUEUE". "QUEUE" instructs IPTABLES to pass any matching protocol traffic to the ip\_queue module. Snort-Inline processes the matching traffic that has been passed to ip\_queue and determines response based upon the configuration file (snort.conf) and rules files. This document is for instructions on how to get Snort-Inline working with IPTABLES. First assuming the user has working knowledge of Redhat, IPTABLES and Snort, this procedure has been tested with Redhat 7.3 Kernel 2.4.18-3. Lets break this paper down into logical sections:

- Determining what packages and RPM's are required**
- Obtaining all required packages and RPM's**
- Configuring and compiling required RPM's and source code**
- Configuring your rc.firewall script**
- Configuring and compiling Snort-Inline**
- Test run your new project**
- Understanding and working with the results**
- Security considerations**

Please note that the rc.firewall script has been provided by Rob McMillen <<mailto:rvmcmil@cablespeed.com>> through the generosity of [Honeynet.org](http://www.honeynet.org) (<http://www.honeynet.org/papers/honeynet>). The rc.firewall script provided in this paper has been edited in order to work with Snort-Inline and IPTABLES for a functional Intrusion Response System.

## **Determining what packages and RPM's are required**

First and foremost, a normal build of snort **MUST** currently be on the system. Preferably Snort-1.9.0 should be on the system and it **MUST** be built and running with **"./configure" "make" and "make install"** as well as the testing phase to ensure functionality **"snort -I eth0 -T"** For the scope and purpose of this document, rc.firewall will be run in "Nat" mode. Snort-Inline will also be required for this implementation. Honeynet.org provides a Snort-Inline startup script that is very useful. If anyone is interested in running in "Bridge" mode, please visit the

[Honetnet.org](http://Honetnet.org) website for tools and scripts.

## Obtaining all required packages and RPM's

Unless the source RPM for IPTABLES has been previously built into the kernel, the standard IPTABLES RPM will not work correctly with Snort-Inline. The required RPM can be obtained from Netfilter.org @ <http://www.netfilter.org/downloads.html> - [1.2.7a](http://www.netfilter.org/downloads.html#iptables-1.2.7a). Locate the file iptables-1.2.7a.tar.bz2 and download it. Different iptables builds work for different Redhat releases. Based on best information available, iptables-1.2.7a works for Redhat 7.3 and 8.0, iptables-1.2.6a works with Redhat 7.2. Snort-Inline can be located and downloaded from the Snort.org site @ <http://www.snort.org/dl/contrib/patches/inline/> as file "snort-inline.tgz". Or simply go to <http://www.snort.org/> and click on "downloads" - "contrib" - "patches" - "inline". The HoneyNet provided startup script as well as other tools for Snort-Inline are available at <http://www.honeynet.org/papers/honeynet/tools/>. Make sure to grab the snort-inline startup script!

## Configuring and compiling required RPM's and source code

Unpack the iptables-1.2.7a.tar.bz2 using bunzip2 and change into the newly created directory. Refer to the INSTALL file to correctly install the new IPTABLES source code. In the event that this file is not found, use the brief instructions below to compile the IPTABLES source code:

FOLLOW THESE STEPS:

1) Next, make the package.

```
% make KERNEL_DIR=<<where-you-built-your-kernel>>
```

2) Finally, you need to install the shared libraries, and the binary:

```
# make install KERNEL_DIR=<<where-you-built-your-kernel>>
```

If you are a developer, you can install the headers, development libraries and associated development man pages, with:

```
# make install-devel
```

That's it! Make sure to issue ALL of the above build statements or there will be issues to contend with down the road.

The KERNEL\_DIR variable should most likely be /usr/src/linux-2.4. In the event there are issues compiling this source code, refer to the Netfilter.org website for support.

## Configuring your rc.firewall script

Configuring the rc.firewall script is fairly straight forward but can be edited to meet any requirements. The main entries that need to be edited are briefly shown below:

COMMON VARS section

MODE="nat"

Use this mode unless your system is operating as a router for other systems with address translation

LAN\_IFACE=""

This value will be the same that Snort-Inline will be running on. Commonly "eth0" is the value

QUEUE="yes"

This must be set to "YES" for snort-inline to work

VARIABLES FOR NAT MODE section

The rc.firewall script also contains entries taking advantage of "-j LOG" for IPTABLES. Log level 6 defaults to "info" and will appear in /var/log/messages according to the -log-prefix that is used. If these LOG entries do not appear in the /var/log/messages file, check /etc/syslog.conf to ensure that \*.info is present and logging to /var/log/messages This script will hopefully be available on the [Snort](#) website. For the time being, the script is available at the end of the document.

## Configuring and compiling Snort-Inline

Tar xzf the "snort-inline.tgz" file and change into the "snort-inline" directory. Issue the command "./configure --enable-inline" in order to build for inline. Follow this with a "make" and "make install". In order to simplify the process run the configure script as "./configure --prefix=/usr/local/snort --enable-inline" or there will be some minor editing waiting for you later on. In the event that configure was run without the --prefix, issue a "cp -R snort-inline /usr/local/snort" and cd to /usr/local/snort/etc and begin edits on the snort.conf file. Much like the original snort, edit this file according to the vars, preprocessors and rules that meet your requirements. This process should be simple enough to not require any further documentation. The next step is to alter all of the rules files. cd to ../rules and either open and edit each rules file that will be used or take advantage of sed and some programming skills to perform a "global" search and replace on the all of the rules files. For the purpose of this document, the proper editing of one rules file will provide a sound starting point. For example, vi the scan.rules files and issue the following command ":1,%s/alert/drop/g" and enter and this should globally replace all occurrences of "alert" with "drop". Do the same to each rules file that will be used as specified in the snort.conf file. Providing that the snort-inline startup script was downloaded from [Honeynet.org](#), "cp" this file to /etc/init.d and edit it according to where the directories and binaries are located. Ensure that ALL instances of eth\* are changed to reflect the interface that snort-inline will be running on. The bottom line of the script should look something close to what is shown below:

```
$$SNORT -D -d -c /usr/local/snort/etc/snort.conf -Q -i eth0 -I $DIR/$DATE
```

However, to avoid any confusion, the simplest form to run the snort-inline binary should be:

```
/usr/local/bin/snort-inline -D -d -c /usr/local/snort/etc/snort.conf -Q -i eth0
```

This will send everything to the standard “alert” file in /var/log/snort but will provide easier reading and troubleshooting.

Important note: Snort-inline generates its own unique binary. This binary will either be located in the snort-inline directory or in the “bin” directory specified with “prefix” when building snort-inline. This specific binary must be used when running snort-inline and not the binary provided by the normal snort build or snort-inline will NOT run. To avoid confusion and simplify the process, locate the snort binary in the snort-inline directory in the “/src” directory and copy it to the same area as the normal snort binary with a different name “cp /snort-inline/src/snort /usr/local/bin/snort-inline”. As always, prior to running this issue a "mkdir /var/log/snort". The first time running snort-inline may generate an error and the program will stop. A quick easy workaround to get rid of this problem is to edit the snort-inline startup script and place the following entry at the top of the file "/sbin/modprobe ip\_queue". Finally, copy the script to a location where snort-inline can be started automatically "cp snort-inline.sh /etc/init.d/snort" and chmod +x to make it executable.

## Test run your new project

With the snort-inline script configured according to requirements and the rc.firewall in place with a chmod +x in /etc/init.d. Start the rc.firewall script issuing "/etc/init.d/rc.firewall". Uncomment the "set -x" line in the rc.firewall script for verbose debugging if there are problems. Follow this by starting the snort-inline script "/etc/init.d/snort". Issue a ps -ef to ensure that snort-inline is running as anticipated. Check for a new \$DATE directory created in /var/log/snort that will be created when snort-inline initializes.

## Understanding and working with the results

Find a hub or switch and connect the system running snort-inline on the same network with a system that can run nmap and other pen-test oriented tools. Initially, running nmap -sF -sX -sN -sS and -sT scans will most likely return results that are picked up by the stream4 preprocessor and the scans may NOT be blocked. However, scans such as -sU will be picked up by snort rules and will be blocked. Run your own tests according to events that are not picked up by stream4 or other preprocessors that do match snort rules and successful “drops” should take place. Continue running the scans and timeouts will start taking place. For our testing

purposes, the snort.conf file was edited so that the preprocessor stream4 and preprocessor stream4\_reassemble was commented out. Under those conditions, stream4 would not be present to detect scans and the rules would be forced to pick up all scans. Conduct other tests that are known to fire alerts on snort signatures and observe the results. In our lab, the success rate for snort-inline responding with a "drop" was around 80% when scanning a single host. When enough scans were conducted to generate timeouts, subsequent success rates were over 90%. Conduct tests in accordance with what would normally take place from an external scan on your environment.

## Security considerations

In order to further secure a snort-inline system, edit the rc.firewall script or create your IPTABLES that enhances restrictions on the INPUT table, such that the INPUT table becomes IPTABLES -P INPUT DROP. Originally setting the INPUT table to ACCEPT allows all traffic to pass through primarily for testing purposes. Feel free to edit the rc.firewall script to meet the needs of your organization.

```
.....  
rc.firewall script  
.....
```

```
#!/bin/bash  
#  
# rc.firewall, ver 0.6 - 5 Jan 2003  
# http://www.honeynet.org/papers/honeynet/tools/  
# Rob McMillen <rvmmil@cablespeed.com>  
# This scripts has been borrowed from honeynet.org and customized for  
strict control of IPTABLES in conjunction with Snort-Inline  
#  
# PURPOSE  
#  
# This scripts has been redesigned for use with IPTABLES and Snort_Inline  
specifically. Use IPTABLES to specify  
# what traffic is analysed, logged, or QUEUED. The primary method to  
deploy a working and successful IPTABLES  
# is by placing allowed hosts, services, traffic etc at the beginning of  
the chain, followed by -j LOG and finally -j QUEUE  
# This will vary according to environment and requirements for each  
organization. Make note that anything used with -j QUEUE  
# will pass everything in that IPTABLES rule to Snort-Inline. Providing  
that all snort rules have "drop" in place of "alert"  
# the specific traffic will ONLY be blocked providing it meets any of the  
criteria in any of the snort rules. In many situations  
# it would be recommendable to place -j QUEUE at the top of the IPTABLES  
chain but run the risk of the traffic being dropped  
# since it will not match any criteria in the IPTABLES chain or Snort-  
Inline. Use your own judgement according to the requirements  
# of your organization.  
#  
# REQUIREMENTS  
# This utility will not work with a standard IPTABLES RPM, one must compile  
the source RPM or the best solution would be to download  
# the most recent source from Netfilter.org. The Tarball from Netfilter  
contains INSTALL and README files on how to recompile your
```

```
# new IPTABLES into the kernel.  Providing that works well for you, then
you should be able to proceed with this script with no issues.
#
#### If you want to see all the commands or which command is giving your
#      problems, remove the comment below.
#set -x

#*****
# USER VARIABLE SECTION
#*****

#####
# COMMON VARS #
#####

# The MODE variable tells the script to #setup a bridge HoneyWall
# or a NATing HoneyWall.
MODE="nat"

### IPTables script can be used with the Snort-Inline filter
### You can find the current release at
### http://www.snort.org/dl/contrib/patches/inline/
QUEUE="yes"          # This must be set to "yes" or Snort-Inline will not
work as anticipated

#####
# SPECIAL CONSIDERATION VARIABLE #
#####
LAN_IFACE="eth0"
#####
# VARIABLES THAT RESTRICT WHAT THE FIREWALL CAN SEND OUT #
#####

RESTRICT="no"

#####
# END RESTRICT VARIABLES #
#####

#####
# LOCATION OF PROGRAMS USED BY THIS SCRIPT #
#####
IPTABLES="/sbin/iptables"
BRIDGE="/usr/sbin/brctl"
IFCONFIG="/sbin/ifconfig"
ROUTE="/sbin/route"
MODPROBE="/sbin/modprobe"
LSMOD="/sbin/lsmmod"

#####
# END OF PROG VARS #
#####

#*****
# END OF USER VARIABLE SECTION (DO NOT EDIT BEYOND THIS POINT)
#*****

#####
```

```

# First, confirm that IPChains is NOT running.  If
# it is running, clear the IPChains rules, remove the kernel
# module, and warn the end user.

/sbin/lsmmod | grep ipchain
IPCHAINS=$?

if [ "$IPCHAINS" = 0 ]; then
    echo ""
    echo "Dooh, IPChains is currently running! IPTables is required by"
    echo "the rc.firewall script. IPChains will be unloaded to allow"
    echo "IPTables to run.  It is recommended that you permanently"
    echo "disable IPChains in the /etc/rc.d startup scripts and enable"
    echo "IPTables instead."
    ipchains -F
    rmmmod ipchains
fi

#####
# Flush rules
#
$IPTABLES -F

#####
# Load all required IPTables modules
#

### Needed to initially load modules
/sbin/depmod -a

### Add iptables target LOG.
$MODPROBE ipt_LOG

### Add iptables QUEUE support (Experimental)
if test $QUEUE = "yes"
then
    # Insert kernel mod
    $MODPROBE ip_queue

    # check to see if it worked, if not exit with error
    $LSMOD | grep ip_queue
    IPQUEUE=$?

    if [ "$IPQUEUE" = 1 ]; then
        echo ""
        echo "It appears you do not have the ip_queue kernel module compiled"
        echo "for your kernel.  This module is required for Snort-Inline and"
        echo "QUEUE capabilities.  You either have to disable QUEUE, or"
        echo "compile"
        echo "the ip_queue kernel module for your kernel.  This module is part"
        echo "of the kernel source."
        exit
    fi

    echo "Enabling Snort-Inline capabilities, make sure Snort-Inline is"
    echo "running in -Q mode, or all outbound traffic will be blocked"
fi

```

```
#####  
####IPTABLES ENTRIES#####  
#####
```

```
# Set up the new policy on the IPTABLES INPUT chain. Use ACCEPT first to  
make sure that everything works and then lock this down later
```

```
$IPTABLES -P INPUT ACCEPT  
$IPTABLES -P FORWARD ACCEPT  
$IPTABLES -P OUTPUT ACCEPT
```

```
# Allow all Traffic on your loopback interface  
$IPTABLES -A INPUT -i lo -p all -j ACCEPT
```

```
# Start off on the INPUT table and allow anything back in that was  
originated from your system (Use this when IPTABLES -P is DROP)  
$IPTABLES -A INPUT -i $LAN_IFACE -p all -m state --state  
ESTABLISHED,RELATED -j ACCEPT
```

```
# USE these lines below if you need to log every protocol that is coming  
into your system. This can be disk and resource intensive!!
```

```
### Inbound TCP
```

```
$IPTABLES -A INPUT -i $LAN_IFACE -p tcp -m state --state NEW -j LOG --log-  
level 6 --log-prefix "INBOUND TCP: "  
$IPTABLES -A INPUT -i $LAN_IFACE -p tcp -m state --state NEW -j ACCEPT
```

```
### Inbound UDP
```

```
$IPTABLES -A INPUT -i $LAN_IFACE -p udp -m state --state NEW -j LOG --log-  
level 6 --log-prefix "INBOUND UDP: "  
$IPTABLES -A INPUT -i $LAN_IFACE -p udp -m state --state NEW -j ACCEPT
```

```
### Inbound ICMP
```

```
$IPTABLES -A INPUT -i $LAN_IFACE -p icmp -m state --state NEW -j LOG --log-  
level 6 --log-prefix "INBOUND ICMP: "  
$IPTABLES -A INPUT -i $LAN_IFACE -p icmp -m state --state NEW -j ACCEPT
```

```
### Inbound anything else
```

```
$IPTABLES -A INPUT -i $LAN_IFACE -m state --state NEW -j LOG --log-level 6  
--log-prefix "INBOUND OTHER: "
```

```
# This might be a good time to pass all incoming traffic over to Snort
```

```
$IPTABLES -A INPUT -i $LAN_IFACE -p all -j QUEUE
```

```
# After all of this hard work...make sure that all of the IPTABLES entries  
are saved
```

```
if [ $MODE = "nat" ]  
then  
/sbin/iptables-save > /etc/sysconfig/iptables  
fi
```

For a very easy and good starting point, use the following as your first  
firewall script:

```

MODE="nat"
QUEUE="yes"
LAN_IFACE="eth0"
RESTRICT="no"
IPTABLES="/sbin/iptables"
BRIDGE="/usr/sbin/brctl"
IFCONFIG="/sbin/ifconfig"
ROUTE="/sbin/route"
MODPROBE="/sbin/modprobe"
LSMOD="/sbin/lsmmod"
$IPTABLES -F

#####
# Load all required IPTables modules
#

### Needed to initially load modules
/sbin/depmod -a

### Add iptables target LOG.
$MODPROBE ipt_LOG
$MODPROBE ip_queue

### Add iptables QUEUE support (Experimental)
if test $QUEUE = "yes"
then
    # Insert kernel mod
    $MODPROBE ip_queue

    # check to see if it worked, if not exit with error
    $LSMOD | grep ip_queue
    IPQUEUE=$?

    if [ "$IPQUEUE" = 1 ]; then
        echo ""
        echo "It appears you do not have the ip_queue kernel module compiled"
        echo "for your kernel. This module is required for Snort-Inline and"
        echo "QUEUE capabilities. You either have to disable QUEUE, or"
compile"
        echo "the ip_queue kernel module for your kernel. This module is part"
        echo "of the kernel source."
        exit
    fi

    echo "Enabling Snort-Inline capabilities, make sure Snort-Inline is"
    echo "running in -Q mode, or all outbound traffic will be blocked"
fi

#####
#####IPTABLES ENTRIES#####
#####

# Set up the new policy on the IPTABLES INPUT chain. Use ACCEPT first to
make sure that everything works and then lock this down later
$IPTABLES -P INPUT ACCEPT
$IPTABLES -P FORWARD ACCEPT
$IPTABLES -P OUTPUT ACCEPT

```

```
# Allow all Traffic on your loopback interface
$IPTABLES -A INPUT -i lo -p all -j ACCEPT
```

```
# Start off on the INPUT table and allow anything back in that was
originated from your system (Use this when IPTABLES -P is DROP)
$IPTABLES -A INPUT -i $LAN_IFACE -p all -m state --state
ESTABLISHED,RELATED -j ACCEPT
```

```
# Pass all incoming traffic to ip_queue so that snort-inline will pick this up
```

```
$IPTABLES -A INPUT -p ALL -j QUEUE
```

```
*****
```

```
*** Remember to use the snort-inline binary instead of the snort binary ***
```