Building a GenII Honeynet Gateway

Spanish Honeynet Project http://www.honeynet.org.es

Diego González Gómez

<dggomez -at- honeynet.org.es>

11 August, 2004. *Last updated:* 14 November, 2004

Abstract

This is a short guide to build a GenII Honeynet Gateway, also called a Honeywall, under Linux; broaching the most common problems and providing several solutions and tips. This document does not explain the only way to install a Honeywall. It can be installed and configured using other tools, accomplishing the same objectives.

Please note: The author makes no warranties, nor can he be held responsible for damages caused by the instructions held in this paper.

1. Introduction

Honeynet technologies are a great way to improve and to learn about network and system security. However, the implementation of these techniques requires a high level of knowledge in these areas and involves a certain degree of responsibility.

A GenII Honeynet Gateway is the most critical element in a GenII Honeynet. Basically, it is the gateway of the Honeynet, but it is also a firewall, an IPS (Intrusion Prevention System), and a network traffic/system logger.

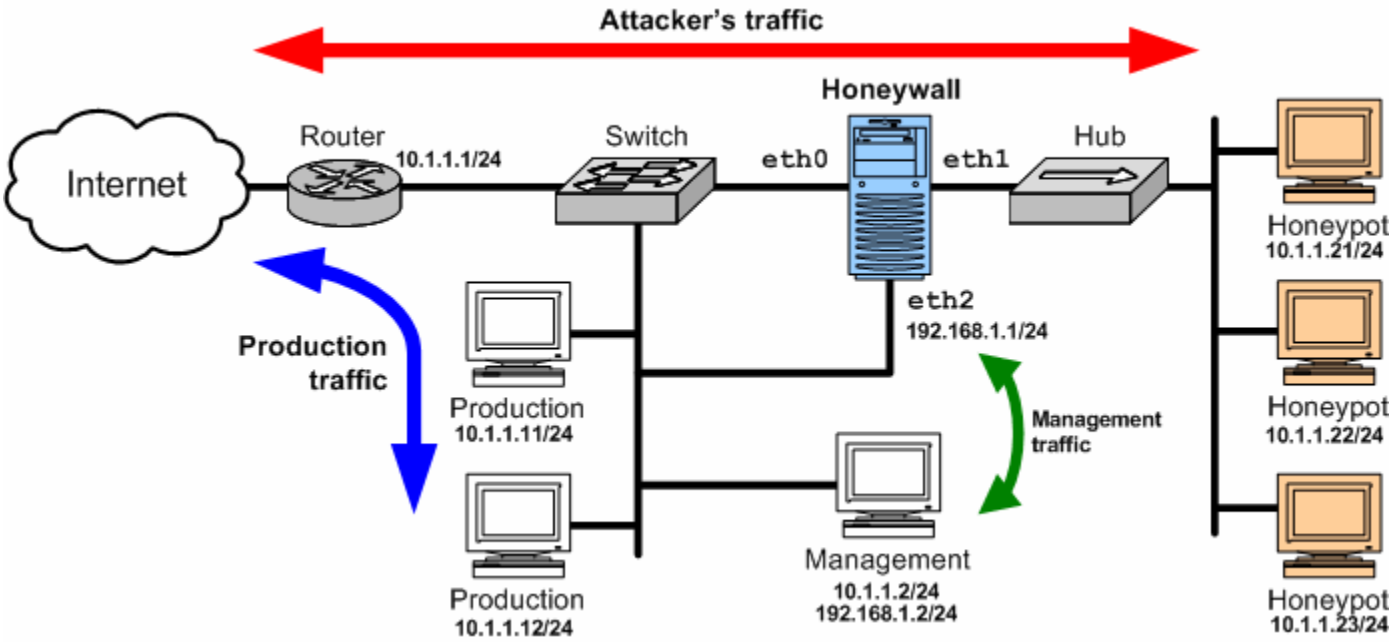
There is a bootable CDROM that makes the implementation of a Honeynet Gateway easer, simply called the Honeywall CDROM. As the authors say: "The intent is to make honeynets easier to deploy and customize. You simply boot off the CDROM, configure it based on your environment, and you should have a Honeywall gateway ready to go". If you do not want to complicate things, you can simply download the CDROM image from http://www.honeynet.org/tools/cdrom/ and stop reading here. On the other hand, if you want to learn how to build a Honeywall from scratch please read on.

This paper explains the overall steps to build a Honewall using Red Hat Linux 9.0, but most of the instructions can be applied to any other Linux distribution. It is assumed that the reader understands the basics of honeypots and the related terminology. In addition, I would also recommend a read of the "Know Your Enemy" series papers from the Honeynet Project at http://www.honeynet.org/papers.

2. Network design

The following diagram is a sample network architecture for implementing the Honeywall.

Figure 1. Proposed GenII Honeynet

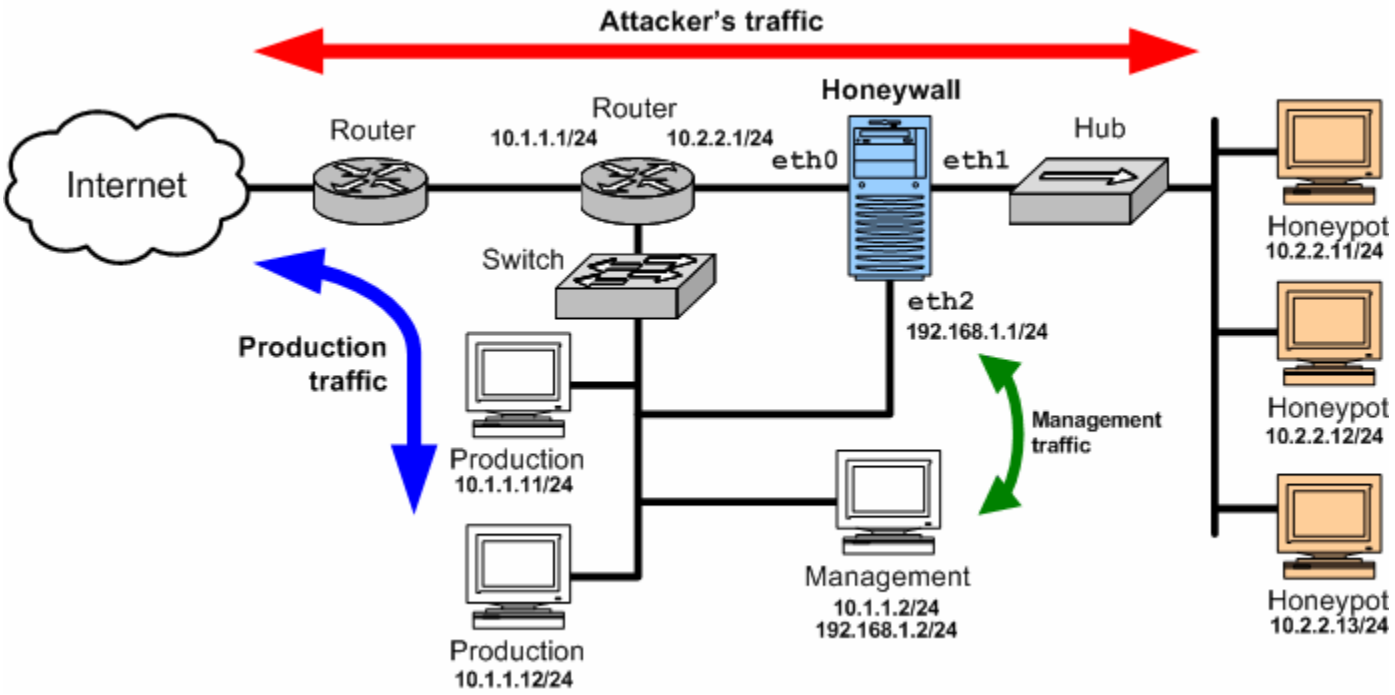


As can be seen in Figure 1, the Honeywall has three network interfaces. Two in bridge mode (`eth0` and `eth1`) and the last one, `eth2`, with an IP stack used for management purposes. The main advantage of the bridge mode is that it is harder to detect by the attackers. For example, since the Honeywall has no IP addresses (except for `eth2`), it does not affect the TTLs (Time to Live) values of the traffic entering/leaving the Honeynet. However, it can still transparently control and capture all the data passing through it.

The management station has one network interface with two IP addresses, one for the main network and another to manage the Honeywall. Another way to communicate directly with the Honeywall could also be achieved using a second dedicated network interface directly connected to the Honeywall using a crossover cable.

Note also in Figure 1, production hosts and honeypots are in the same network. Although this is not the safest method to implement a Honeynet, it can be a feasible scenario. For example, you already know the consequences of this architecture and you wish to study the collateral effects offered by it. But if you do not want the production hosts to be directly accessed by the compromised honeypots, a proper configuration of the bridge firewall rules would be necessary – read crucial. In any case, if a more secure scenario is required, it is recommended to put the Honeynet in a separate network, such as can be seen in Figure 2.

Figure 2. Alternative GenII Honeynet



For security reasons, it is also recommended to implement remote logging, to the management host or to a different host with proper firewall configuration (the configuration of logging is beyond the scope of this paper). In this paper, the first diagram will be adopted.

In the following sections we are going to cover the most important steps to implement the Honeywall functions, namely Data Control, Data Capture and Alerting.

3. Data control

This is perhaps one of the most critical aspects of a Honeywall. Basically, the purpose of a Honeynet is to be compromised to take the opportunity to learn from it. Fine, but keep in mind that if your Honeynet is successfully attacked, it can be used to attack other systems! You must be prepared to circumvent that situation. How? A good solution is to implement some kind of firewall. One of the reasons for choosing Linux is because it has **`IPTables`**, an outstanding firewall with, among other utilities, traffic limiting capabilities that are extremely useful in Honeynet technologies. Another layer of protection can be accomplished by an IPS, such as **`snort_inline`**, used in this scenario to protect the outside world from our (potentially dangerous) Honeynet.

3.1. Firewall

As discussed, we are going to put the Honeywall in bridge mode. For **`IPTables`** to be able to see and filter the bridged IP traffic, the kernel must include the bridge-nf code. In addition, the kernel must also support the IP QUEUE option if we want to install **`snort_inline`** IPS tool.

The bridge-nf code and ebtables, a powerful filtering tool which acts as a bridge firewall, can be found at http://ebtables.sourceforge.net/. **Ebtables**[1] is a powerful filtering tool which acts as a bridge firewall, it is focused on the Link Layer and although it has many helpful features, we don't really need it for our Honeywall as we have **IPTables**. Both **ebtables** and **bridge-nf** are natively supported by the standard 2.6 kernel. There is a patch available for the stable 2.4 kernel and when the 2.6 kernel is at the stable stage, support for 2.4 will be dropped.

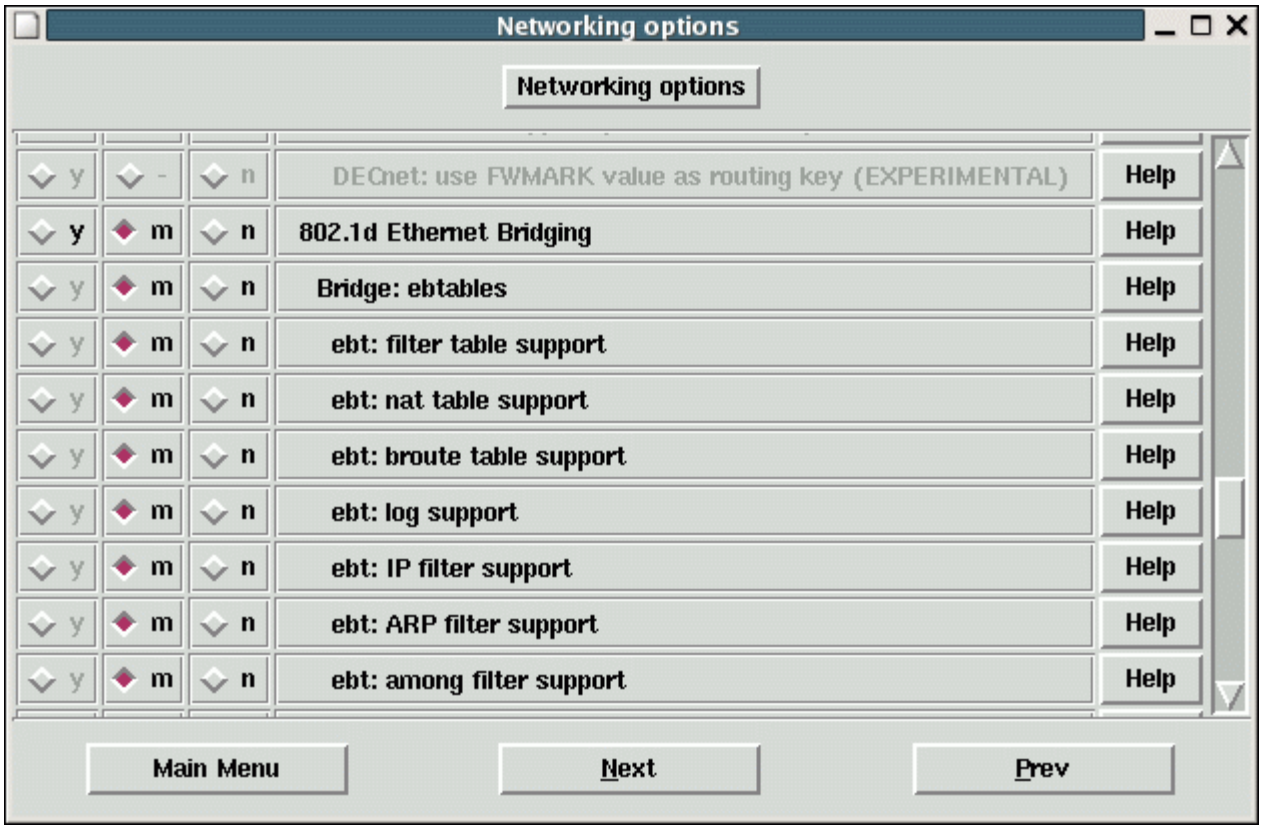
Therefore, we can choose between kernel 2.4 and 2.6. In both cases we will need to recompile the kernel, and probably more than once. At this point I should mention that as touched on before, this document assumes the reader has experience in this area and is comfortable with doing such tasks as recompiling the kernel. If this is not the case, I am afraid that this sort of experience is one of the fundamentals required to continue and I suggest if this is not the level you currently have that you pause here and reconsider installing the pre-built Honeywall CDROM at http://www.honeynet.org/tools/cdrom/.

3.2. Kernel 2.4.X

If we want to build our Honeywall using a standard kernel version 2.4.x, we must download an appropriate **bridge-nf** patch. At the time of this writing, the latest kernel version supported is 2.4.26, and it is probably the last version supported for the 2.4 tree. Download the source kernel version 2.4.26 from http://www.kernel.org and then go to ebtables web site http://ebtables.sourceforge.net/ to download ebtables-brnf-6-vs-2.4.26 file. Extract the contents of the kernel sources and the patch under `/usr/src/` and apply the patch to the kernel.

```
# gzip -d ebtables-brnf-6_vs_2.4.26.diff.gz
# cd /usr/src/linux-2.4.26
# patch -p1 < /home/dggomez/downs/ebtables-brnf-
6_vs_2.4.26.diff
```

When selecting the kernel options, go to **Networking Options**, and select as a module **802.1d Ethernet Bridging**, **Bridge: ebtables**, and all the subsequent ebt options.



The IP QUEUE option does not appear in the kernel options menu. It must be manually compiled as a module. It isn't recommended to manually edit the kernel `.config` file to include IP QUEUE option, if it doesn't exist by default. It is better to manually compile the queue module independently. Following is an extract of the `.config` file.

```
[...]
#
#   IP: Netfilter Configuration
#
CONFIG_IP_NF_CONNTRACK=m
CONFIG_IP_NF_FTP=m
CONFIG_IP_NF_AMANDA=m
CONFIG_IP_NF_TFTP=m
CONFIG_IP_NF_IRC=m
CONFIG_IP_NF_QUEUE=m
CONFIG_IP_NF_IPTABLES=m
CONFIG_IP_NF_MATCH_LIMIT=m
CONFIG_IP_NF_MATCH_MAC=m
```

```
CONFIG_IP_NF_MATCH_PKTTYPE=m
CONFIG_IP_NF_MATCH_MARK=m
CONFIG_IP_NF_MATCH_MULTIPORT=m
CONFIG_IP_NF_MATCH_TOS=m
CONFIG_IP_NF_MATCH_RECENT=m
CONFIG_IP_NF_MATCH_ECN=m
CONFIG_IP_NF_MATCH_DSCP=m
CONFIG_IP_NF_MATCH_AH_ESP=m
CONFIG_IP_NF_MATCH_LENGTH=m
CONFIG_IP_NF_MATCH_TTL=m
[...]
```

Or, better, you can manually compile and install the `ip_queue.o` module.

```
# cd /usr/src/linux-2.4.26/net/ipv4/netfilter
# gcc -D__KERNEL__ -I/usr/src/linux-2.4.26/include -Wall \
-Wstrict-prototypes -Wno-trigraphs -O2 -fno-strict-aliasing \
-fno-common -fomit-frame-pointer -pipe -mpreferred-stack-
boundary=2 \
-march=i686 -DMODULE -DMODVERSIONS \
-include /usr/src/linux-2.4.26/include/linux/modversions.h \
-nostdinc -iwithprefix include -DKBUILD_BASENAME=ip_queue -c \
-o ip_queue.o ip_queue.c
# cp ip_queue.o /lib/modules/2.4.26/kernel/net/ipv4/netfilter
# cd /lib/modules/2.4.26/kernel/net/ipv4/netfilter
# insmod ip_queue.o
```

After saving the kernel options, compile it.

```
# make dep ; make bzImage ; make modules ; make modules_install
```

Tip

You can accelerate the kernel compilation by adding the option '**-j** *n*' to the **make** command. The '*n*' value is the number of simultaneous jobs. I

recommend a value of 5 or less. Note that if you use that option, the overall system performance will decrease during compile time.

After compiling, update your boot image and boot loader configuration and restart. If your kernel boots without problems, you can check if it supports *ip_queue* by typing:

```
# modprobe ip_queue
# lsmod
```

You should see *ip_queue* in the output list.

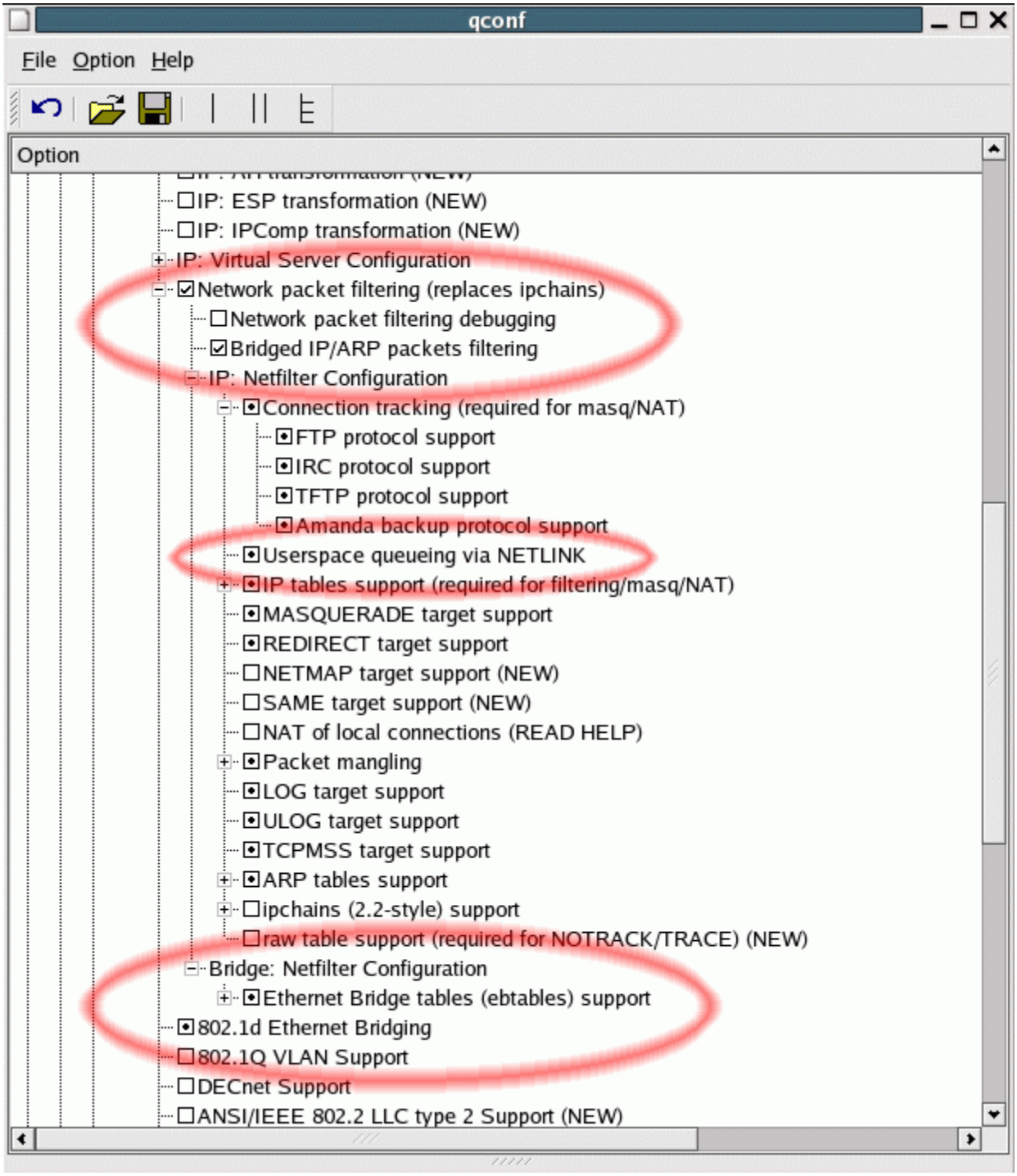
3.3. Kernel 2.6.X

If you have chosen a standard kernel 2.6.x you do not need to apply any **`bridge-nf`** patch because it natively supports it. Recompile the kernel with the following options:

For bridge firewall support, go to '**Device Drivers**', '**Networking Support**', '**Networking Support**', and select '**802.1d Ethernet Bridging**'. Then go to '**Network Packet Filtering**', and select '**Bridged IP/ARP packets filtering**'.

For IP QUEUE support, go to '**Device Drivers**', '**Networking Support**', '**Networking Support**', '**IP: Netfilter configuration**', and select '**Userspace queueing via NETLINK**'.

Lastly, if you want ebtables support you can also select the '**Bridge: Netfilter Configuration**' options under '**Network Packet Filtering**'.



The table below contains the common set of commands needed to recompile the kernel.

Table 1. Kernel compile commands

Kernel 2.4.X	**Kernel 2.6.X**
make mrproper	make mrproper

make menuconfig, or make xconfig	make menuconfig, or make xconfig
make dep	make, or make install (if you have LILO)
make clean	make modules_install
make bzImage	
make modules	
make modules_install	

3.4. Bridge-utils

After recompiling the kernel with bridge firewall support and IP QUEUE (remember, only if you plan to install **`snort_inline`**), it is time to install the **`bridge-utils`** tools.

The **`bridge-utils`** package allows the set up and management of bridges under Linux. First of all though, check if you have it already installed. Otherwise, the latest version can be downloaded from the Linux Ethernet bridging `http://bridge.sourceforge.net` page or use **`apt-get`** tool (from `http://apt.freshrpms.net/`) to install it. Using **`apt-get`** is much simpler, although probably you will not obtain the latest version (it is not necessary neither).

```
# rpm -q bridge-utils
package bridge-utils is not installed
# apt-get install bridge-utils
Reading Package Lists... Done
Building Dependency Tree... Done
[...]
0 packages upgraded, 1 newly installed, 0 removed and
0 not upgraded.
# rpm -q bridge
bridge-utils-0.9.3-8
```

For building a bridge using the interfaces `eth0` and `eth1` we need to issue the following commands:

```
# brctl addbr br0
# ifconfig eth0 0.0.0.0 up -arp
# ifconfig eth1 0.0.0.0 up -arp
# brctl addif br0 eth0
```

```
# brctl addif br0 eth1
# brctl stp br0 off
# ifconfig br0 0.0.0.0 up -arp
```

We have disabled STP (Spanning Tree Protocol) support because we do not need it and to stop unnecessary traffic. ARP (Address Resolution Protocol) is also disabled since the network devices have not an IP stack and because it could reveal their identity and position.

3.5. Firewall rules and bridge mode

After installing **IPTables**, bridge firewall support, and optionally *ip_queue* module, we need to build the bridge and configure the firewall rules. Basically, we should accept all inbound connections, and limit every outbound connections passing through the bridge. However, this can be a difficult task and could become even in a dangerous one if we do not achieve it with care.

Fortunately, the Honeynet Project provides a documented script that automatically builds a bridge and create the appropriate **IPTables** rules for Data Control. The original script can be found at http://www.honeynet.org/tools/. A modified copy of the script can be found in the Appendix section.

3.6. Snort_inline

As you can read at its homepage http://snort-inline.sourceforge.net, **snort_inline** is a modified version of **snort** that accepts packets from **IPTables**, via **libipq**, instead of **libpcap**. Additionally, it can perform some packet mangling using **libnet** library.

Consequently, we will need to install **libipq** and **libnet** libraries before installing **snort_inline**. **Snort_inline** receives packets via QUEUE **IPTables** target, and remember that this option must be supported form the kernel. See how to provide IP QUEUE support in the previous kernel's subsections.

In the following example, we verify that we have `ip_queue.o` installed as a module.

```
# modprobe ip_queue
# lsmod | grep ip_queue
ip_queue 8044 0 (unused)
```

3.6.1. libipq

To install **libipq**, get the source code of the same version of **IPTables** that you have installed on your system. Download it from http://www.netfilter.org/, extract it and issue a '**make install-devel**'.

```
# rpm -q iptables iptables-VERSION
# wget http://www.netfilter.org/files/iptables-VERSION.tar.bz2
[...]
# tar xvjf iptables-VERSION.tar.bz2
[...]
# cd iptables-VERSION
# make install-devel
```

3.6.2. libnet

At the time of this writing, the most current version of **snort_inline** is v2.1.3b, and it needs a **libnet** library version 1.0.x. The 1.0.x tree is deprecated, and the latest version is 1.0.2a. You can download it from http://www.packetfactory.net/libnet/.

```
# wget
http://www.packetfactory.net/libnet/dist/deprecated/libnet-
1.0.2a.tar.gz
[...]
# tar xvzf libnet-1.0.2a.tar.gz
[...]
# cd libnet-1.0.2a
# ./configure ; make ; make install
```

3.6.3. build snort_inline

Once the *ip_queue* module and the required libraries are installed, you can finally configure and install **snort_inline** in the usual way (**configure; make; make install**).

If when executing the **make** command you get a similar sort of ouput as below:

```
[...]
gcc -DHAVE_CONFIG_H -I. -I. -I../.. -I../.. -
I../../src -I../../src/sfutil
```

```
-I/usr/include/pcap -I../../src/output-plugins -
I../../src/detection-plugins

-I../../src/preprocessors -
I../../src/preprocessors/flow

-I../../src/preprocessors/portscan  -
I../../src/preprocessors/flow/int-snort

-I../../src/preprocessors/HttpInspect/include  -
I/usr/include/pcre

-DENABLE_RESPONSE -D_BSD_SOURCE -D__BSD_SOURCE -
D__FAVOR_BSD

-DHAVE_NET_ETHERNET_H -DLIBNET_LIL_ENDIAN -
I/usr/local/include -I/sw/include

-g -O2 -Wall -DGIDS -D_BSD_SOURCE -D__BSD_SOURCE -
D__FAVOR_BSD

-DHAVE_NET_ETHERNET_H -DLIBNET_LIL_ENDIAN -c
spo_alert_fast.c

In file included from
/usr/include/linux/netfilter_ipv4/ip_queue.h:10,

                 from /usr/local/include/libipq.h:37,

                 from ../../src/inline.h:8,

                 from ../../src/snort.h:38,

                 from spo_alert_fast.c:51:

/usr/include/linux/if.h:59: redefinition of `struct
ifmap'

/usr/include/linux/if.h:77: redefinition of `struct
ifreq'

/usr/include/linux/if.h:126: redefinition of `struct
ifconf'

make[3]: *** [spo_alert_fast.o] Error 1

make[3]: Leaving directory

`/home/dggomez/downs/snort_inline-2.1.3b/src/output-
plugins'

make[2]: *** [all-recursive] Error 1

make[2]: Leaving directory
`/home/dggomez/downs/snort_inline-2.1.3b/src'

make[1]: *** [all-recursive] Error 1
```

```
make[1]: Leaving directory
`/home/dggomez/downs/snort_inline-2.1.3b'
make: *** [all-recursive-am] Error 2
```

The kernel headers used by your **glibc** need to be updated. Create a link between `/usr/include` directory and the include directory of your kernel source. More information can be found in the FAQ section of the **snort_inline** page. For example:

```
# cd /usr/include
# mv linux linux.backup
# ln -s /usr/src/linux-2.4.26/include/linux linux
```

After that, go back to **snort_inline** directory and recompile it after doing a '**make clean**'. You can run **snort_inline** issuing:

```
# snort_inline -D -c /etc/snort_inline/snort_inline.conf -u
snort -Q \
-N -l /var/log/snort_inline/YYYYMMDD -t
/var/log/snort_inline/YYYYMMDD
```

Parameters explained:

-D Run in daemon mode
-c Load configuration file
-u Run as UID user
-Q Read packets form QUEUE
-N Turn off logging (does not affect alerts)
-l Log to directory
-t Chroots process to directory after initialization

If your system has not a **snort** user, add one by issuing '**adduser snort -r /bin/nologin/**'. Replace the '*YYYYMMDD*' string with the actual date. Append the '`-T`' option first to check if **snort_inline** works fine. Note that it is not necessary to use '`-i`' (interface) parameter here since **snort_inline** will only receive packets though **IPTables** rules with QUEUE target.

You can find an automatic script for running **snort_inline** in the Appendix section.

3.7. Snort (IDS mode)

The versatile **snort** tool will be used here in NIDS mode to detect known attacks. It can be downloaded from http://www.snort.org. This program is very easy to install and well documented. You can download precompiled binaries or the source files. If you choose the second option, just run the commands '**configure ; make ; make install**' to install it.

To run **snort** in IDS mode see the following command. Note that the network interface used is on the honeynet side (see the Figure 1).

```
# snort -D -c /etc/snort/snort.conf -i eth1 -u snort \
-N -l /var/log/snort/YYYYMMDD -t /var/log/snort/YYYYMMDD
```

Parameters explained:

-D Run in daemon mode
-c Load configuration file
-i Input network interface
-u Run as UID user
-N Turn off logging (does not affect alerts)
-l Log to directory
-t Chroots process to directory after initialization

If your system has not a **snort** user, add one by issuing '**adduser snort -r /bin/nologin/**'. Replace the '*YYYYMMDD*' string above with the actual date. You can run **snort** with '-T' option first to check if it works correctly.

There is an automatic script for running **snort** in IDS mode in the Appendix section.

4. Data capture

The next step for building a honeynet is to install some kind of activity capture tool, such as **tcpdump** (network traffic) or **sebek** (system activity), for recording details of the conversations between the honeynet and attackers. In this document, **snort** will be used in packet logging mode for capturing network traffic to tcpdump binary log files.

4.1. Snort (Packet logging mode)

If you installed **snort** before, you do not need to install it again. If you did not install **snort** for use as a NIDS, read the instructions above on how to install it.

For running **snort** in logging mode, issue the following. Note that the network interface used is on the honeynet side (see the diagram at the beginning).

```
# snort -D -i eth1 -u snort -l /var/log/snort/YYYYMMDD \
-L tcpdump.YYYYMMDD -t /var/log/snort/YYYYMMDD
```

Parameters explained:

-D Run in daemon mode
-i Input network interface
-u Run as UID user
-l Log to directory
-L Log to a tcpdump file
-t Chroots process to directory after initialization

If your system has not a **snort** user, add one by running '**adduser snort -r /bin/nologin/**'. Replace '*YYYYMMDD*' string with the actual date. You can run **snort** with '-T' option first to check if it works correctly.

You can find an automatic script for running **snort** in logging mode in the Appendix section.

5. Alerting

The alerting functions are extremely useful, and help in managing a Honeynet since they can inform the administrator when events of interest occur. For example, the initiation of an outbound connection from the honeynet (that could reveal a possible compromise), or the incidence of specific attacks (maybe of administrator's interest), etc.

The alerting functions are performed by monitoring programs that looks for changes in certain elements of the system, or in the network traffic itself. In our case, we use **swatch** (The Simple WATCHer) as discussed in the next section.

5.1. Swatch

This Perl based tool looks for events in logfiles. When it matches an event, it can send an email alert to the administrator with the contents of the event found. You can download **swatch** from http://swatch.sourceforge.net/. **Swatch** installs just like a CPAN module. You can obtain more information issuing the **man** command:

```
# man ExtUtils::MakeMaker
```

Alternatively, you can use the **perldoc** command if your man cannot find the document. Instalation of **swatch** is easy, simply run the following commands:

```
# perl Makefile.PL
# make
# make test
# make install
# make realclean
```

If you receive the output below:

```
Warning: prerequisite Date::Calc 0 not found at (eval
1) line 219.
Warning: prerequisite Date::Parse 0 not found at
(eval 1) line 219.
Warning: prerequisite File::Tail 0 not found at (eval
1) line 219.
Warning: prerequisite Time::HiRes 1.12 not found at
(eval 1) line 219.
```

Then you need to install the CPAN modules that have not been found before you can use **`swatch`**. Many operating systems may already provide perl rpm's so you should check with them first. Alternatively, you can find these modules at the below links.

http://search.cpan.org/dist/Date-Calc/
http://search.cpan.org/dist/TimeDate/
http://search.cpan.org/dist/File-Tail/
http://search.cpan.org/dist/Time-HiRes/

To install each Perl module the same set of commands used above to install **`swatch`** need to be executed.

Tip

Another option for installing the Perl modules is to issue the command '**perl -MCPAN -e shell**' and get an interactive installer/shell. The shell is very simple and handles all dependancies. However, as with any installer, manual intervention may be required to install the latest required package. Once in the shell, run the '**install**' command followed by the module name (Date::Calc, Date::Parse, File::Tail, Time::HiRes) like the following.

```
# perl -MCPAN -e shell
```

```
cpan shell -- CPAN exploration and modules
installation (v1.61)
ReadLine support available (try 'install
Bundle::CPAN')


cpan> install Date::Calc
[...]
```

Once **swatch** is installed the next step is to make a configuration file to generate alerts. The most obvious alerts are the ones when the honeynet initiates outbound connections or when connection limits are met.

You can find a script and an example configuration file for running **swatch** in the Appendix section.

6. Testing

Once the required tools are installed for Data Control, Data Capture and Alerting functions, it may be a good idea to test them to make sure that they work correctly. Use the diagram provided at the beginning of the document for any tests, and it is assumed that the scripts used will be the ones found in the Appendices.

6.1. Data Control

Let's see if the **IPTables** logging mechanisms are running correctly. We ran the bridge-firewall script and tried to open one connection from one production host to the honeynet. If you try to open a telnet connection from IP 10.1.1.11 to a honeypot with IP 10.1.1.21, you should see something like this in the `/var/log/message` file:

```
Jul 22 18:47:35 hpot kernel: INBOUND TCP: IN=br0 OUT=br0 PHYSIN=eth0
PHYSOUT=eth1 SRC=10.1.1.11 DST=10.1.1.21 LEN=52 TOS=0x00 PREC=0x00
TTL=96 ID=57225 DF PROTO=TCP SPT=1351 DPT=23 WINDOW=3768 RES=0x00 SYN
URGP=0
```

Then we enabled the 'LAN blocking' option and repeated the telnet connection from the honeypot to a host from the LAN. As expected, we did not find any entry in the logs as the packets were silently dropped.

Now let's verify if **IPTables** is limiting the outbound connections from the honeynet. For example, we are going to make several HTTP connections to the outside world and read the results in `/var/log/messages` [2].

```
Jul 22 18:47:46 hpot kernel: OUTBOUND TCP: IN=br0 OUT=br0 PHYSIN=eth1
PHYSOUT=eth0 SRC=10.1.1.11 DST=192.168.1.20 LEN=52 TOS=0x00 PREC=0x00
TTL=96 ID=57332 DF PROTO=TCP SPT=1032 DPT=80 WINDOW=3768 RES=0x00 SYN
URGP=0

Jul 22 18:48:05 hpot kernel: Drop TCP after 9 attempts IN=br0 OUT=br0
PHYSIN=eth1 PHYSOUT=eth0 SRC=10.1.1.11 DST=192.168.1.20 LEN=52 TOS=0x00
PREC=0x00 TTL=96 ID=57351 DF PROTO=TCP SPT=1032 DPT=80 WINDOW=3768
RES=0x00 SYN URGP=0
```

This is what we would expect. To be completely sure, we can also verify UDP and ICMP protocols.

To test **snort_inline** we can use the rules provided in the `test.rules` file. Include this rules file in the configuration file and start the IPS. We did this and tried to open an external telnet connection. Then we opened an HTTP connection and finally send some pings. The **snort_inline** alert file recorded the following[3].

```
07/22-18:48:13.349121 [**] [1:0:0] Dropping Telnet connection [**]
[Priority: 0] {TCP} 10.1.1.21:1067 -> 192.168.0.1:23

07/22-18:48:16.527301 [**] [1:0:0] Dropping Telnet connection [**]
[Priority: 0] {TCP} 10.1.1.21:1067 -> 192.168.0.1:23

07/22-18:48:23.189037 [**] [1:0:0] Dropping Telnet connection [**]
[Priority: 0] {TCP} 10.1.1.21:1067 -> 192.168.0.1:23

07/22-18:48:49.416280 [**] [1:0:0] Modifying HTTP GET command [**]
[Priority: 0] {TCP} 10.1.1.21:1070 -> 216.239.59.104:80

07/22-18:48:54.476601 [**] [1:0:0] Dropping ICMP packet [**] [Priority:
0] {ICMP} 10.1.1.21 -> 192.168.0.1

07/22-18:48:55.636633 [**] [1:0:0] Dropping ICMP packet [**] [Priority:
0] {ICMP} 10.1.1.21 -> 192.168.0.1

07/22-18:48:57.084787 [**] [1:0:0] Dropping ICMP packet [**] [Priority:
0] {ICMP} 10.1.1.21 -> 192.168.0.1

07/22-18:48:58.603217 [**] [1:0:0] Dropping ICMP packet [**] [Priority:
0] {ICMP} 10.1.1.21 -> 192.168.0.1
```

The IPS worked as expected. Similar tests can be performed to check **snort** in IDS mode. Run them and read the results in the `/var/log/snort/`*`YYYYMMDD`* directory.

6.2. Data Capture

If **snort** was running in logging mode during the above tests, we should be able to find binary log files at `/var/log/snort/`*`YYYYMMDD`* directory with the network traffic generated during the tests. They can be read with `tcpdump` or **snort**.

6.3. Alerting

swatch is configured by default to alert only on outbound connections recorded by iptables, but it can be configured to generate alerts on a variety of events. For example, it can also monitor **snort** and **snort_inline**.

During the previous tests we have opened connections from the Honeynet. If swatch was running, we should receive several email alerts describing the events. Here is an example:

```
Date: Thu, 22 Jul 2004 18:48:14 +0200
From: swatcher <swatcher@origin-domain.com>
To: admin@example.org
Subject: ------ ALERT! OUTBOUND TCP --------
```

```
Jul 22 18:48:13 hpot kernel: OUTBOUND TCP: IN=br0 OUT=br0 PHYSIN=eth1
PHYSOUT=eth0 SRC=10.1.1.21 DST=192.168.0.1 LEN=48 TOS=0x00 PREC=0x00
TTL=128 ID=7327 DF PROTO=TCP SPT=1067 DPT=23 WINDOW=16384 RES=0x00 SYN
URGP=0
```

If all the tests were successful we can configure the scripts to run at boot time. One way to do this is copying the scripts to `/etc/init.d` directory and adding the startup commands to `/etc/rc.d/rc.local` file.

7. Conclusion

The Honeywall is the key element within a Honeynet architecture. In this paper, we have only covered the most common steps needed to install one of these devices and with minimum security requirements. Needless to say that there are a lot of things that could be done to improve security. However, the intent here is to give the reader the opportunity to have a base configuration, and a starting point to experiment with, to learn in this area and to go from there.

Fortunately, Honeynet technologies have lots of possibilities and can be setup in many different ways. If this document has solved some of the most common problems that arose when building a Honeywall, I invite you to share your experiences.

A. Honeywall scripts

This section contains the scripts nedeed to set up and manage the tools comented in this paper. The bridge-firewall script and the main configuration file have been taken from the Honeywall CDROM by the Honeynet Project (with some minor modifications) for simplicity and to preserve compatibility. The other scripts have been developed by the Spanish Honeynet Project for this article.

1. honeywall.conf - Honeywall Configuration File

```
##########################################################
################
#
# Spanish Honeynet Project <project@honeynet.org.es>
# August, 2004
#
# This file is an improved version of honeywall.conf
config file
# included in the Honeywall CDROM from
# http://www.honeynet.org/tools/cdrom by The Honeynet
Project.
# It has two new options: LAN_BLOCK and
LAN_ALLOWDED_IP.
#
##########################################################
################


# Specify whether or not the Honeywall will operate
as either a bridge or NAT
# [Valid modes: bridge | nat]
MODE=bridge


# This Honeywall's public IP address(es)
# [Valid argument: IP address | space delimited IP
addresses]
PUBLIC_IP=192.168.1.10


# DNS servers honeypots are allowed to communicate
with
# [Valid argument: IP address | space delimited IP
addresses]
DNS_SVRS=
```

```
# To restrict DNS access to a specific honeypot or
group of honeypots, list
# them here, otherwise leave this variable blank
# [Valid argument: IP address | space delimited IP
addresses | blank]
DNS_HOST=

# The name of the externally facing network interface
# [Valid argument: eth* | br* | ppp*]
INET_IFACE=eth0

# The name of the internally facing network interface
# [Valid argument: eth* | br* | ppp*]
LAN_IFACE=eth1

# The IP internal connected to the internally facing
interface
# [Valid argument: IP network in CIDR notation]
LAN_IP_RANGE=192.168.1.0/24

# The IP broadcast address for internal network
# [Valid argument: IP broadcast address]
LAN_BCAST_ADDRESS=192.168.1.255

# Enable traffic blocking from the honeypots to the
LAN, to protect
# the LAN hosts against any attack from the honeypots
# [Valid argument: yes | no]
LAN_BLOCK=no

# The list of the LAN IP addresses that can be
accesed from the honeypots,
# such as the gateway, internal DNS servers, ... This
variable is used
```

```
# only if LAN_BLOCK is enabled
# [Valid argument: space delimited IP addresses]
LAN_ALLOWED_IP=192.168.1.1

# Enable QUEUE support to integrate with Snort-Inline
filtering
# [Valid argument: yes | no]
QUEUE=yes

# The unit of measure for setting oubtbound
connection limits
# [Valid argument: second, minute, hour, day, week,
month, year]
SCALE=hour

# The number of TCP connections per unit of measure
(Scale)
# [Valid argument: integer]
TCPRATE=9

# The number of UDP connections per unit of measure
(SCALE)
# [Valid argument: integer]
UDPRATE=20

# The number of ICMP connections per unit of measure
(SCALE)
# [Valid argument: integer]
ICMPRATE=50

# The number of other IP connections per unit of
measure (SCALE)
# [Valid argument: integer]
OTHERRATE=10
```

```
# Enable the SEBEK collector which delivers keystroke
and files
# to a remote system even if an attacker replaces
daemons such as sshd
# [Valid argument: yes | no]
SEBEK=no


# Specify whether whether to drop SEBEK packets or
allow them to be sent
# outside of the Honeynet.
# [Valid argument: ACCEPT | DROP]
SEBEK_FATE=DROP


# Specify the SEBEK destination host IP address
# [Valid argument: IP address]
SEBEK_DST_IP=10.0.0.1


# Specify the SEBEK destination port
# [Valid argument: port]
SEBEK_DST_PORT=1101


# Enable SEBEK logging in the Honeywall firewall logs
# [Valid argument: yes | no]
SEBEK_LOG=no


# Specify the IP netmask for interface alises.  One
aliases will be created
# on the external interface for each Honeypot
# [Valid argument: IP netmask]
ALIAS_MASK=255.255.255.0


# Space delimited list of Honeypot ips
# NOTE: MUST HAVE SAME NUMBER OF IPS AS PUBLIC_IP
VARIABLE.
```

```
# [Valid argument: IP address]
HPOT_IP=10.10.10.3


# Specify the IP address of the honeywall's internal
ip address.  This is
# used in nat mode.
# [Valid argument: IP address]
PRIV_IP=10.0.0.1


# Specy the network interface for remote management.
If set to br0, it will
# assign MANAGE_IP to the logical bridge interface
and allow its use as a
# management interface.  Set to none to disable the
management interface.
# [Valid argument: eth* | br* | ppp* | none]
MANAGE_IFACE=eth2


# IP of management Interface
# [Valid argument: IP address]
MANAGE_IP=192.168.1.13


# Netmask of management Interface
# [Valid argument: IP netmask]
MANAGE_NETMASK=255.255.255.0


# Default Gateway of management Interface
# [Valid argument: IP address]
MANAGE_GATEWAY=192.168.1.1


# DNS Servers of management Interface
# [Valid argument: space delimited IP addresses]
MANAGE_DNS=
```

```
# TCP ports allowed into the management interface.
If SSH is used this list
# must include the port SSHD is listening on.
# [Valid argument: space delimited list of TCP ports]
ALLOWED_TCP_IN=22


# Specify the IP address(es) and/or networks that are
allowed to connect
# to the management interface.  Specify any to allow
unrestricted access.
# [Valid argument: IP address(es) | IP network(s) in
CIDR notation | any]
MANAGER=any


# Specify whether or not the Honeywall will restrict
outbound network
# connections to specific destination ports.  When
bridge mode is utilized,
# a management interface is required to restrict
outbound network connections.
# [Valid argument: yes | no]
RESTRICT=yes


# Specity the TCP destination ports Honeypots can
send network traffic to.
# [Valid argument: space delimited list of UDP ports]
ALLOWED_TCP_OUT="22 25 43 80 443"


# Specity the UDP destination ports Honeypots can
send network traffic to.
# [Valid argument: space delimited list of UDP ports]
ALLOWED_UDP_OUT="53 123"


# List of files that Swatch should monitor
```

```
# [Valid argument: space delimited list of files with
full path name]
WATCH_FILES="/var/log/messages"


# Specify email address to use for email alerting.
# [Valid argument: any email address]
ALERT_EMAIL=
```

2. rc.firewall - Bridge-Firewall Script File

```
#!/bin/sh
#
# Copyright 2003 Honeynet Project
<project@honeynet.org>
# License BSD http://www.opensource.org/licenses/bsd-
license.php
#
# This is an improved version of the rc.firewall
script v0.8
# found in Honeywall CDROM from
http://www.honeynet.org/tools/cdrom
# that supports the new LAN_BLOCK option. On the
other hand, the
# handlers' section has been simplified.
#
# Spanish Honeynet Project <project@honeynet.org.es>
# August, 2004
#


PATH="/sbin:/usr/sbin:/usr/local/sbin:/bin:/usr/bin"
. /etc/default/honeywall.conf


start ()
{
```

```
   lsmod | 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 recommened that
you permanently"
       echo "disable IPChains in the /etc/rc.d
startup scripts and enable"
       echo "IPTables instead."
       ipchains -F
       rmmod ipchains
   fi

   #########
   # Flush rules
   #
   iptables -F
   iptables -F -t nat
   iptables -F -t mangle
   iptables -X

   echo ""

   ##########
   # Let's setup the firewall according to the Mode
selected: bridge or nat
   #
   if [ ${MODE} = "nat" ]; then
```

```
       echo "Starting up Routing mode and enabling
Network Address Translation."
       #Let's bring up our internal interface
       ifconfig ${LAN_IFACE} ${PRIV_IP} netmask
${LAN_BCAST_ADDRESS} up


       i=0
       z=1
       tempPub=( ${PUBLIC_IP} )


       for host in ${HPOT_IP}; do
           if [ ${i} = "0" ]; then


              #This is the first honeypot. Let's
attach it to our nic
              ifconfig ${INET_IFACE} ${tempPub[$i]}
netmask ${ALIAS_MASK} up
           else


              # Bring up eth aliases
              ifconfig ${INET_IFACE}:${z}
${tempPub[$i]} netmask ${ALIAS_MASK} up
              let "z += 1"
           fi


           # Ensure proper NATing is performed for
all honeypots
           iptables -t nat -A POSTROUTING -o
${INET_IFACE} -s ${host} \
                    -j SNAT --to-source ${tempPub[$i]}
           iptables -t nat -A PREROUTING -i
${INET_IFACE} -d ${tempPub[$i]} \
                    -j DNAT --to-destination ${host}
           let "i += 1"
       done
```

```
    fi

    # Let's figure out dns
    if [ -z "${DNS_HOST}" ]; then
        if [ "${MODE}" = "bridge" ]; then
            DNS_HOST="${PUBLIC_IP}"
        else
            DNS_HOST="${HPOT_IP}"
        fi
    fi



    #########
    # 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 [ ${QUEUE} = "yes" ]; then
        # Insert kernel mod
        modprobe ip_queue

        # check to see if it worked, if not exit with
error
        lsmod | grep ip_queue &>/dev/null
        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


    ### Support for connection tracking of FTP and
IRC.
    modprobe ip_conntrack_ftp
    modprobe ip_conntrack_irc


    ### Enable ip_forward
    echo "1" > /proc/sys/net/ipv4/ip_forward


    ### Create protocol handling chains
    iptables -N tcpHandler
    iptables -N udpHandler
    iptables -N icmpHandler
    iptables -N otherHandler
```

```
   # Forward Chain:
   #     Some of these rules may look redundant, but
they allow us to catch
   #    'other' protocols.



   # Internet -> honeypot  -
   #     This logs all inbound new connections and we
must
   #     specifically allow all inbound traffic
because
   #     the default policy for forwarding traffic
   #     will be drop.  This will ensure if something
   #     goes wrong with outbound connections, we
   #     default to drop.
   #
   # Also, in case we have something listening to the
QUEUE, we
   #     will send all packets via the QUEUE.



   # Since this is a bridge, we want to allow
broadcast.  By default, we allow all
   # inbound traffic (including broadcast).  We also
want to allow outbound
   # broadcast # (such as NetBIOS) but we do not want
to count it as an outbound
   # session.  So we allow it here *before* we begin
counting outbound connections


   #iptables -A FORWARD -i ${LAN_IFACE} -d
${LAN_BCAST_ADDRESS} -j LOG \
           #--log-prefix "Legal Broadcast: "
```

```
   iptables -A FORWARD -d ${LAN_BCAST_ADDRESS} -j
ACCEPT


   #iptables -A FORWARD -i ${LAN_IFACE} -d
255.255.255.255 -j LOG \
            #--log-prefix "Legal Broadcast: "


   iptables -A FORWARD -d 255.255.255.255 -j ACCEPT


   ### Inbound TCP
   iptables -A FORWARD -i ${INET_IFACE} -p tcp -m
state --state NEW -j LOG \
            --log-prefix "INBOUND TCP: "
   iptables -A FORWARD -i ${INET_IFACE} -p tcp -m
state --state NEW -j ACCEPT


   ### Inbound UDP
   iptables -A FORWARD -i ${INET_IFACE} -p udp -m
state --state NEW -j LOG \
            --log-prefix "INBOUND UDP: "
   iptables -A FORWARD -i ${INET_IFACE} -p udp -m
state --state NEW -j ACCEPT


   ### Inbound ICMP
   iptables -A FORWARD -i ${INET_IFACE} -p icmp -m
state --state NEW -j LOG \
            --log-prefix "INBOUND ICMP: "
   iptables -A FORWARD -i ${INET_IFACE} -p icmp -m
state --state NEW -j ACCEPT


   ### Inbound anything else
   iptables -A FORWARD -i ${INET_IFACE} -m state --
state NEW -j LOG \
            --log-prefix "INBOUND OTHER: "
```

```
   iptables -A FORWARD -i ${INET_IFACE} -m state --
state NEW -j ACCEPT


   # The remainder of established connections will be
ACCEPTED.  The rules above
   # are required in order to log new inbound
connections.
   iptables -A FORWARD -i ${INET_IFACE} -j ACCEPT



   # Okay, this is where the magic all happens.  All
outbound traffic is counted,
   # logged, and limited here.   Targets (called
Handlers) are what actually limit
   # the connections.  All 'Handlers' are defined at
the bottom of the script.


   # Egress filtering, don't want to let our
compromised honeypot send spoofed
   # packets.  Stops most outbound DoS attacks.
However, we might want to allow
   # our honeypots to use dhcp to get an ip while in
bridge mode.
   if [ ${MODE} = "bridge" ]; then
       iptables -A FORWARD -i ${LAN_IFACE} -p udp --
sport 68 \
               -d 255.255.255.255 --dport 67 -j LOG \
               --log-prefix "DHCP OUT REQUEST: "


       iptables -A FORWARD -i ${LAN_IFACE} -p udp --
sport 68 \
               -d 255.255.255.255 --dport 67 -j
ACCEPT
   fi
```

```
   # This rule is for use with sebek.  If sebek is
used, and we don't want
   # the logs filled by SPOOFED SOURCE entries
because sebek uses spoofed
   # IPs, we should drop all traffic in the sebek ip
range.
   if [ ${SEBEK} = "yes" ]; then
       if [ ${SEBEK_LOG} = "yes" ]; then
           iptables -A FORWARD -i ${LAN_IFACE} -p udp
-d ${SEBEK_DST_IP} \
               --dport ${SEBEK_DST_PORT} -j LOG --
log-prefix "SEBEK"
       fi
       iptables -A FORWARD -i ${LAN_IFACE} -p udp -d
${SEBEK_DST_IP} \
               --dport ${SEBEK_DST_PORT} -j
${SEBEK_FATE}
   fi



   ### DNS / NTP Perhaps one of your honeypots needs
consistent
   ### outbound access to provide internal service.


   # If we did not identify a specific destination
dns server, let's go ahead
   # and allow any.
   if [ -z "${DNS_SVRS}" ]; then
      DNS_SVRS="0.0.0.0/0"
   fi


   for srvr in ${DNS_SVRS}; do
       for host in ${DNS_HOST}; do
           iptables -A FORWARD -p udp -i ${LAN_IFACE}
-s ${host} -d ${srvr} \
```

```
                    --dport 53 -j LOG --log-prefix
"Legal DNS: "
            iptables -A FORWARD -p tcp -i ${LAN_IFACE}
-s ${host} -d ${srvr} \
                    --dport 53 -j LOG --log-prefix
"Legal DNS: "


            iptables -A FORWARD -p udp -i ${LAN_IFACE}
-s ${host} -d ${srvr} \
                    --dport 53 -j ACCEPT
            iptables -A FORWARD -p tcp -i ${LAN_IFACE}
-s ${host} -d ${srvr} \
                    --dport 53 -j ACCEPT
        done
    done


    ### Count and limit all other outbound connections


    # This will ensure we don't restrict Honeypots
talking to eachother, and
    # we don't log them as outbound connections (in
bridge mode, the
    # firewall sees all packets; therefore, we have to
make sure it doesn't
    # log packets incorrectly and give false
positives).
    # If you do not want to see this log, comment out
the logging rule.
    # You will still need the ACCEPT rule to ensure
they honeypots can talk
    # to eachother freely.
        iptables -A FORWARD -i ${LAN_IFACE} -o
${LAN_IFACE} -j LOG \
                --log-prefix "Honeypot -> Honeypot: "
```

```
       iptables -A FORWARD -i ${LAN_IFACE} -o
${LAN_IFACE} -j ACCEPT


   # LAN Protect/Blocking denies access to the LAN IP
addresses not included
   # in the LAN_ALLOWED_IP variable.
   # If we activated this feature, allow access to
the sepecified IP addresses
   # and finally block the access to the rest of the
LAN IP address space
   if [ ${LAN_BLOCK} = "yes" ]; then
       for host in ${LAN_ALLOWED_IP}; do
           iptables -A FORWARD -i ${LAN_IFACE} -d
${host} -j ACCEPT
       done
       iptables -A FORWARD -i ${LAN_IFACE} -d
${LAN_IP_RANGE} -j DROP
   fi


   if [ ${LAN_BLOCK} = "yes" ]; then
       for host in ${LAN_ALLOWED_IP}; do
           iptables -A FORWARD -i ${INET_IFACE} -s
${host} -j ACCEPT
       done
       iptables -A FORWARD -i ${INET_IFACE} -s
${LAN_IP_RANGE} -j DROP
   fi


   if [ ${MODE} = "nat" ]; then
       LIMIT_IP="${HPOT_IP}"
   elif [ ${MODE} = "bridge" ]; then
       LIMIT_IP="${PUBLIC_IP}"
   fi


   for host in ${LIMIT_IP}; do
```

```
   # TCP:
   #   This next rule is the connection limiter.  If
it has not exceeded
   #   the limit, the packet will be sent to the
tcpHandler.  The
   #   tcpHandler will log and either QUEUE or ACCEPT
depending on
   #   the Architecture selected.
   #
   #   NOTE: The purpose of the drop rule is to
ensure we can catch 'other'
   #   protocols that enter our network.  If this
statement is not here
   #   we will get false log entries stating Drop
other after xxx
   #   connections.
       iptables -A FORWARD -p tcp -i ${LAN_IFACE} -m
state --state NEW \
               -m limit --limit ${TCPRATE}/${SCALE} \
               --limit-burst ${TCPRATE} -s ${host} -j
tcpHandler


       iptables -A FORWARD -p tcp -i ${LAN_IFACE} -m
state --state NEW \
               -m limit --limit 1/${SCALE} --limit-
burst 1  -s ${host} \
               -j LOG --log-prefix "Drop TCP after
${TCPRATE} attempts"


       iptables -A FORWARD -p tcp -i ${LAN_IFACE} -m
state --state NEW \
               -s ${host} -j DROP


       # This rule is for Mike Clark in order to give
him RELATED information.  For
```

```
       # example, this will tell him the data channel
related to an ftp command
       # channel of a connection.
       iptables -A FORWARD -p tcp -i ${LAN_IFACE} -m
state --state RELATED \
               -s ${host} -j tcpHandler


   #
   # UDP - see TCP comments above.
   #
       iptables -A FORWARD -p udp -i ${LAN_IFACE} -m
state --state NEW \
               -m limit --limit ${UDPRATE}/${SCALE} \
               --limit-burst ${UDPRATE} -s ${host} -j
udpHandler


       iptables -A FORWARD -p udp -i ${LAN_IFACE} -m
state --state NEW \
               -m limit --limit 1/${SCALE} --limit-
burst 1 -s ${host} -j LOG \
               --log-prefix "Drop udp after
${UDPRATE} attempts"


       iptables -A FORWARD -p udp -i ${LAN_IFACE} -m
state --state NEW \
               -s ${host} -j DROP



   #
   # ICMP - see TCP comments above.
   #
       iptables -A FORWARD -p icmp -i ${LAN_IFACE} -m
state --state NEW \
               -m limit --limit ${ICMPRATE}/${SCALE}
\
```

```
               --limit-burst ${ICMPRATE} -s ${host} -
j icmpHandler


       iptables -A FORWARD -p icmp -i ${LAN_IFACE} -m
state --state NEW \
               -m limit --limit 1/${SCALE} --limit-
burst 1 -s ${host} -j LOG \
               --log-prefix "Drop icmp after
${ICMPRATE} attempts"


       iptables -A FORWARD -p icmp -i ${LAN_IFACE} -m
state --state NEW \
               -s ${host} -j DROP



   #
   # EVERYTHING ELSE - see TCP comments above.
   #
       iptables -A FORWARD -i ${LAN_IFACE} -m state -
-state NEW -m limit \
               --limit ${OTHERRATE}/${SCALE} --limit-
burst ${OTHERRATE} \
               -s ${host} -j otherHandler


       iptables -A FORWARD -i ${LAN_IFACE} -m state -
-state NEW -m limit \
               --limit 1/${SCALE} --limit-burst 1 -s
${host} -j LOG \
               --log-prefix "Drop other after
${OTHERRATE} attempts"


   done



   # This portion of the script will ensure that
established or related
```

```
   # connections that were allowed, continue to work.
If these lines
   # are not here, only the first packet of each
connection that hasn't
   # reached the limit will be allowed in because we
are dropping
   # all outbound connections by default.

   if [ "${QUEUE}" = "yes" ]; then
       TARGET=QUEUE
   else
       TARGET=ACCEPT
   fi

   iptables -A FORWARD -i ${LAN_IFACE} -m state --
state RELATED,ESTABLISHED \
           -j ${TARGET}

   ###  These define the handlers that actually limit
outbound connection.
   #
   # tcpHandler - The only packets that should make
it into these chains are new
   #              connections, as long as the host
has not exceeded their limit.
   #
   iptables -A tcpHandler -j LOG --log-prefix
"OUTBOUND TCP: "
   iptables -A tcpHandler -j ${TARGET}

   #
   # udpHandler - see tcpHandler comments above.
   #
   iptables -A udpHandler -j LOG --log-prefix
"OUTBOUND UDP: "
```

```
   iptables -A udpHandler -j ${TARGET}


   #
   # icmpHandler - see tcpHandler comments above.
   #
   iptables -A icmpHandler -j LOG --log-prefix
"OUTBOUND ICMP: "
   iptables -A icmpHandler -j ${TARGET}


   #
   # otherHandler - see tcpHandler comments above.
   #
   iptables -A otherHandler -j LOG --log-prefix
"OUTBOUND OTHER: "
   iptables -A otherHandler -j ${TARGET}


   iptables -A INPUT -m state --state
RELATED,ESTABLISHED -j ACCEPT


   ### Lets make sure our firewall can talk to itself
   iptables -A INPUT -i lo -j ACCEPT
   iptables -A OUTPUT -o lo -j ACCEPT



   ##############################
   # MANAGEMENT INTERFACE RULES #
   ##############################
   if [ ${MANAGE_IFACE} != "none" ]; then
       for ports in ${ALLOWED_TCP_IN}; do
           if [ "${MANAGER}" = "any" ]; then
               #iptables -A INPUT -i ${MANAGE_IFACE}
-p tcp --dport $ports \
                       #-m state --state NEW -j LOG \
```

```
                        #--log-prefix "MANAGE
port:$ports=>"


                iptables -A INPUT -i ${MANAGE_IFACE} -
p tcp --dport $ports \
                        -m state --state NEW -j ACCEPT
            else
                for ips in ${MANAGER}; do
                    #iptables -A INPUT -i
${MANAGE_IFACE} -p tcp -s $ips \
                                #--dport $ports -m state -
-state NEW -j LOG \
                                #--log-prefix "MANAGE
port:$ports=>"


                    iptables -A INPUT -i
${MANAGE_IFACE} -p tcp -s $ips \
                                --dport $ports -m state --
state NEW -j ACCEPT
                done
            fi
        done


        iptables -A OUTPUT -o ${MANAGE_IFACE} -p tcp -
m state \
                --state RELATED,ESTABLISHED -j ACCEPT
    fi


    ### Set default policies for the INPUT, FORWARD
and OUTPUT chains
    # By default, drop all connections sent to
firewall
    iptables -P INPUT DROP


    # If we selected to restrict the firewall, lets
implement it here.
```

```
   if [ ${RESTRICT} = "yes" ]; then
       for port in ${ALLOWED_TCP_OUT}; do
           iptables -A OUTPUT -p tcp --dport $port -m
state \
                   --state NEW,ESTABLISHED,RELATED -j
ACCEPT
       done

       for port in ${ALLOWED_UDP_OUT}; do
           iptables -A OUTPUT -p udp --dport $port -m
state \
                   --state NEW,ESTABLISHED,RELATED -j
ACCEPT
       done

      # By default, drop firewall outbound connection
      iptables -P OUTPUT DROP
   else
      # By default, accept firewall outbound
connection
      iptables -P OUTPUT ACCEPT
   fi



   # By default, if FORWARDED connections are not
within limit, DROP.
   # This is a fail close policy, and more secure.
   iptables -P FORWARD DROP
}

stop ()
{
   echo "Stopping Firewall."
   #########
```

```
   # Flush rules
   #
   iptables -F
   iptables -F -t nat
   iptables -F -t mangle
   iptables -X

   # Set default forward to drop
   iptables -P FORWARD DROP
   iptables -P INPUT DROP
   iptables -P OUTPUT DROP

   # Allow the firewall to talk to itself
   iptables -A INPUT -i lo -j ACCEPT
   iptables -A OUTPUT -o lo -j ACCEPT

   if [ -n ${MANAGE_IFACE} ]; then
      iptables -A INPUT -i ${MANAGE_IFACE} -j ACCEPT
      iptables -A OUTPUT -o ${MANAGE_IFACE} -j ACCEPT
   fi

}

initial ()
{
   #########
   # Flush rules
   #
   iptables -F
   iptables -F -t nat
   iptables -F -t mangle
   iptables -X
```

```
   # Set default forward to drop
   iptables -P FORWARD DROP
   iptables -P INPUT DROP
   iptables -P OUTPUT DROP

   # Allow the firewall to talk to itself
   iptables -A INPUT -i lo -j ACCEPT
   iptables -A OUTPUT -o lo -j ACCEPT
}

restart ()
{
   stop
   start &>/dev/null
}

case "$1" in
    start)
        start
        ;;
    stop)
        stop
        ;;
    restart)
        restart
        ;;
    initial)
        initial
        ;;
    *)
        echo $"Usage: $0
{start|stop|restart|initial)"
```

```
        exit 1
esac
```

3. snort_inline.conf - Snort_inline Configuration File

```
var HOME_NET any
var HONEYNET any
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 SNMP_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_inline/rules/drop-rules

config checksum_mode: none

preprocessor flow: stats_interval 0 hash 2
preprocessor frag2
preprocessor stream4: disable_evasion_alerts
detect_scans
preprocessor stream4_reassemble: both
```

```
preprocessor http_inspect: global \
    iis_unicode_map /etc/snort_inline/unicode.map
1252
preprocessor http_inspect_server: server default \
    profile all ports { 80 8080 8180 }
oversize_dir_length 500
preprocessor rpc_decode: 111 32771
preprocessor bo
preprocessor telnet_decode

output alert_full: snort_inline-full.log
output alert_fast: snort_inline-fast.log

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

#include $RULE_PATH/test.rules
include $RULE_PATH/local.rules
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/pop2.rules
include $RULE_PATH/pop3.rules
include $RULE_PATH/virus.rules
include $RULE_PATH/nntp.rules
# include $RULE_PATH/other-ids.rules
# include $RULE_PATH/web-attacks.rules
# include $RULE_PATH/backdoor.rules
# include $RULE_PATH/shellcode.rules
# include $RULE_PATH/policy.rules
# include $RULE_PATH/porn.rules
# include $RULE_PATH/info.rules
# include $RULE_PATH/icmp-info.rules
# include $RULE_PATH/chat.rules
# include $RULE_PATH/multimedia.rules
# include $RULE_PATH/p2p.rules
include $RULE_PATH/experimental.rules
```

4. snort_inline.sh - Snort_inline Script File

```
#!/bin/sh
#
# Copyright 2004 Spanish Honeynet Project
<project@honeynet.org.es>
# License BSD http://www.opensource.org/licenses/bsd-
license.php
#
# NAME:         snort_inline.sh
# DATE:         August, 2004
# VERSION:      0.1
# DESCRIPTION: Setup script for running snort_inline
#

# Load variables
. /etc/default/honeywall.conf

# Script variables
RETVAL=0
BINARY=/usr/local/bin/snort_inline
PATH=/bin:/usr/local/bin
PID=/var/run/snort_inline.pid
DIR="/var/log/snort_inline"
DATE=`/bin/date +%Y%m%d`
CONF_FILE=/etc/snort_inline/snort_inline.conf
PROG=snort_inline
USER=snort

if [ ! -x "$BINARY" ]; then
    echo "ERROR: $BINARY not found."
    exit 1
fi
```

```
if [ ! -r "$CONF_FILE" ]; then
    echo "ERROR: $CONF_FILE not found."
    exit 1
fi

start()
{
    # Check if log directory is present. Otherwise,
create it.
    if [ ! -d $DIR/$DATE ]; then
        mkdir $DIR/$DATE
        chown -R $USER $DIR/$DATE
    fi
    /bin/echo "Starting $PROG: "
    # Snort_inline parameters
    # -D Run snort_inline in background (daemon) mode
    # -Q Use ip_queue for input vice libpcap
(iptables only)
    # -u <uname> Run snort_inline uid as <uname> user
(or uid)
    # -c Load configuration file
    # -N Turn off logging (alerts still work)
    # -l Log to directory
    # -t Chroots process to directory after
initialization

    $BINARY -D -Q -u $USER -c $CONF_FILE -N -l
$DIR/$DATE -t $DIR/$DATE
    /bin/echo "$PROG startup complete."
    return $RETVAL
}

stop()
{
```

```
    if [ -s $PID ]; then
        /bin/echo "Stopping $PROG, with PID `cat
$PID`: "
        kill -TERM `cat $PID`
        /bin/echo "$PROG shutdown complete."
        rm -f $PID
    else
        /bin/echo "ERROR: PID in $PID file not
found."
        RETVAL=1
    fi
    return $RETVAL
}

restart()
{
    stop
    start
    RETVAL=$?
    return $RETVAL
}

case "$1" in
 start)
       start
    ;;
 stop)
       stop
    ;;
 restart|reload)
       restart
    ;;
 *)
```

```
    /bin/echo "Usage: $0 {start|stop|restart|reload}"
    RETVAL=1
esac

exit $RETVAL
```

5. snort.conf - Snort Configuration File

```
var HOME_NET any
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 SNMP_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/rules

preprocessor flow: stats_interval 0 hash 2
preprocessor frag2
preprocessor stream4: disable_evasion_alerts
detect_scans
preprocessor stream4_reassemble
preprocessor http_inspect: global \
```

```
    iis_unicode_map /etc/snort/unicode.map 1252
preprocessor http_inspect_server: server default \
    profile all ports { 80 8080 8180 }
oversize_dir_length 500
preprocessor rpc_decode: 111 32771
preprocessor bo
preprocessor telnet_decode
preprocessor portscan: $HOME_NET 4 4 portscan-
snort.log
preprocessor perfmonitor: time 600 file
/var/log/snort/snort.stats pktcnt 10000


output alert_fast: snort-fast.log


include /etc/snort/classification.config
include /etc/snort/reference.config


include $RULE_PATH/local.rules
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/pop2.rules
include $RULE_PATH/pop3.rules
include $RULE_PATH/nntp.rules
include $RULE_PATH/other-ids.rules
# include $RULE_PATH/web-attacks.rules
include $RULE_PATH/backdoor.rules
include $RULE_PATH/shellcode.rules
# include $RULE_PATH/policy.rules
# include $RULE_PATH/porn.rules
# include $RULE_PATH/info.rules
# include $RULE_PATH/icmp-info.rules
# include $RULE_PATH/virus.rules
# include $RULE_PATH/chat.rules
# include $RULE_PATH/multimedia.rules
# include $RULE_PATH/p2p.rules
include $RULE_PATH/experimental.rules
```

6. snort.sh - Snort Script File (NIDS)

```
#!/bin/sh
#
# Copyright 2004 Spanish Honeynet Project
<project@honeynet.org.es>
# License BSD http://www.opensource.org/licenses/bsd-
license.php
#
# NAME:         snort.sh
# DATE:         August, 2004
# VERSION:      0.1
# DESCRIPTION: Setup script for running snort
#

# Load variables
. /etc/default/honeywall.conf

# Script variables
RETVAL=0
BINARY=/usr/local/bin/snort
PATH=/bin:/usr/local/bin
PID=/var/run/snort_${LAN_IFACE}_ids.pid
DIR="/var/log/snort"
DATE=`/bin/date +%Y%m%d`
CONF_FILE=/etc/snort/snort.conf
PROG=snort
USER=snort

if [ ! -x "$BINARY" ]; then
    echo "ERROR: $BINARY not found."
    exit 1
fi
```

```
if [ ! -r "$CONF_FILE" ]; then
    echo "ERROR: $CONF_FILE not found."
    exit 1
fi

start()
{
    # Check if log diratory is present. Otherwise,
create it.
    if [ ! -d $DIR/$DATE ]; then
        mkdir $DIR/$DATE
        chown -R $USER $DIR/$DATE
    fi
    /bin/echo "Starting $PROG: "
    # Snort parameters
    # -D Run Snort in background (daemon) mode
    # -i <if> Listen on interface <if>
    # -u <uname> Run snort uid as <uname> user (or
uid)
    # -c Load configuration file
    # -N Turn off logging (alerts still work)
    # -l Log to directory
    # -R <id> Include 'id' in snort_intf<id>.pid file
name

    $BINARY -D -i $LAN_IFACE -u $USER -c $CONF_FILE -
N -l $DIR/$DATE -R _ids
    /bin/echo "$PROG startup complete."
    return $RETVAL
}

stop()
{
    if [ -s $PID ]; then
```

```
        /bin/echo "Stopping $PROG with PID `cat
$PID`: "
        kill -TERM `cat $PID` 2>/dev/null
        RETVAL=$?
        /bin/echo "$PROG shutdown complete."
        rm -f $PID
    else
        /bin/echo "ERROR: PID in $PID file not
found."
        RETVAL=1
    fi
    return $RETVAL
}

restart()
{
    stop
    start
    RETVAL=$?
    return $RETVAL
}

case "$1" in
 start)
       start
    ;;
 stop)
       stop
    ;;
 restart|reload)
       restart
    ;;
 *)
```

```
    /bin/echo "Usage: $0 {start|stop|restart|reload}"
    RETVAL=1
esac

exit $RETVAL
```

7. snort_pcap.sh - Snort Script File (Packet logging)

```
#!/bin/sh
#
# Copyright 2004 Spanish Honeynet Project
<project@honeynet.org.es>
# License BSD http://www.opensource.org/licenses/bsd-
license.php
#
# NAME:         snort_pcap.sh
# DATE:         August, 2004
# VERSION:      0.1
# DESCRIPTION: Setup script for running snort
#

# Load variables
. /etc/default/honeywall.conf

# Script variables
RETVAL=0
BINARY=/usr/local/bin/snort
PATH=/bin:/usr/local/bin
PID=/var/run/snort_${LAN_IFACE}_pcap.pid
DIR="/var/log/snort"
DATE=`/bin/date +%Y%m%d`
CONF_FILE=/etc/snort/snort.conf
PROG=snort
```

```
USER=snort

if [ ! -x "$BINARY" ]; then
    echo "ERROR: $BINARY not found."
    exit 1
fi

if [ ! -r "$CONF_FILE" ]; then
    echo "ERROR: $CONF_FILE not found."
    exit 1
fi

start()
{
    # Check if log diratory is present. Otherwise,
create it.
    if [ ! -d $DIR/$DATE ]; then
        mkdir $DIR/$DATE
        chown -R $USER $DIR/$DATE
    fi
    /bin/echo "Starting $PROG: "
    # Snort parameters
    # -D Run Snort in background (daemon) mode
    # -i <if> Listen on interface <if>
    # -u <uname> Run snort uid as <uname> user (or
uid)
    # -l Log to directory
    # -L Log to a tcpdump file
    # -t Chroots process to directory after
initialization
    # -R <id> Include 'id' in snort_intf<id>.pid file
name
```

```
    $BINARY -D -i $LAN_IFACE -u $USER -l $DIR/$DATE -
L tcpdump.$DATE -t $DIR/$DATE -R _pcap
    /bin/echo "$PROG startup complete."
    return $RETVAL
}

stop()
{
    if [ -s $PID ]; then
        /bin/echo "Stopping $PROG with PID `cat
$PID`: "
        kill -TERM `cat $PID` 2>/dev/null
        RETVAL=$?
        /bin/echo "$PROG shutdown complete."
        rm -f $PID
    else
        /bin/echo "ERROR: PID in $PID file not
found."
        RETVAL=1
    fi
    return $RETVAL
}

restart()
{
    stop
    start
    RETVAL=$?
    return $RETVAL
}

case "$1" in
 start)
```

```
        start
    ;;
 stop)
        stop
    ;;
 restart|reload)
        restart
    ;;
 *)
    /bin/echo "Usage: $0 {start|stop|restart|reload}"
    RETVAL=1
esac

exit $RETVAL
```

8. tcpdump.sh - tcpdump Script File

```
#!/bin/sh
#
# Copyright 2004 Spanish Honeynet Project
<project@honeynet.org.es>
# License BSD http://www.opensource.org/licenses/bsd-
license.php
#
# NAME:         tcpdump.sh
# DATE:         August, 2004
# VERSION:      0.1
# DESCRIPTION: Setup script for running tcpdump.
#
# Comments: The default log directory is
/var/log/tcpdump
#           The filter file is optional
#
```

```
# Load global variables
. /etc/default/honeywall.conf

# Script variables
RETVAL=0
BINARY=/usr/sbin/tcpdump
PATH=/bin:/usr/local/bin
FILTER_FILE=/etc/tcpdump/tcpdump.filter
DATE=`/bin/date +%Y%m%d`
LOG_DIR=/var/log/tcpdump
LOG_FILE=tcpdump.log.`/bin/date +%s`
PROG=tcpdump

if [ ! -x "$BINARY" ]; then
    echo "ERROR: $BINARY not found."
    exit 1
fi

start()
{
    # Check if log directory is present. Otherwise,
create it.
    if [ ! -d $LOG_DIR/$DATE ]; then
        mkdir $LOG_DIR/$DATE
        chown -R $USER $LOG_DIR/$DATE
    fi
    /bin/echo "Starting $PROG: "
    if [ -s "$FILTER_FILE" ]; then
        $BINARY -i $LAN_IFACE -F $FILTER_FILE -w
$LOG_DIR/$DATE/$LOG_FILE &
    else
        $BINARY -i $LAN_IFACE -w
$LOG_DIR/$DATE/$LOG_FILE &
```

```
    fi
    /bin/echo "$PROG startup complete."
    return $RETVAL
}

stop()
{
    /bin/echo "Stopping $PROG: "
    for pid in `/sbin/pidof $PROG`; do
        /bin/kill -TERM $pid 2>/dev/null
        RETVAL=$?
    done
    /bin/echo "$PROG shutdown complete."
    return $RETVAL
}

restart()
{
    stop
    start
    RETVAL=$?
    return $RETVAL
}

case "$1" in
 start)
       start
    ;;
 stop)
       stop
    ;;
 restart|reload)
```

```
       restart
    ;;
 *)
    /bin/echo "Usage: $0 {start|stop|restart|reload}"
    RETVAL=1
esac

exit $RETVAL
```

9. swatch.conf - Swatch Configuration File

```
watchfor /OUTBOUND TCP/
mail=admin@example_domain.com,subject=------ ALERT!
OUTBOUND TCP --------
throttle 10:0:0


watchfor /OUTBOUND UDP/
mail=admin@example_domain.com,subject=------ ALERT!
OUTBOUND UDP --------
throttle 10:0:0


watchfor /OUTBOUND ICMP/
mail=admin@example_domain.com,subject=------ ALERT!
OUTBOUND ICMP --------
throttle 10:0:0


watchfor /OUTBOUND OTHER/
mail=admin@example_domain.com,subject=------ ALERT!
OUTBOUND OTHER --------
throttle 10:0:0


watchfor /Drop/
mail=admin@example_domain.com,subject=------ ALERT!
Connection Limit Reached --------
```

```
throttle 10:0:0
```

10. swatch.sh - Swatch Script File

```
#!/bin/sh
#
# Copyright 2004 Spanish Honeynet Project
<project@honeynet.org.es>
# License BSD http://www.opensource.org/licenses/bsd-
license.php
#
# NAME:         swatch.sh
# DATE:         August, 2004
# VERSION:      0.1
# DESCRIPTION: Setup script for running Swatch.
#

# Load variables
. /etc/default/honeywall.conf

# Script variables
RETVAL=0
BINARY=/usr/bin/swatch
PATH=/bin:/usr/local/bin:/usr/bin
CONF_FILE=/etc/swatch/swatch.conf
PROG=swatch

if [ ! -x "$BINARY" ]; then
    echo "ERROR: $BINARY not found."
    exit 1
fi
if [ ! -r "$CONF_FILE" ]; then
    echo "ERROR: $CONF_FILE not found."
```

```
    exit 1
fi


start()
{
   /bin/echo "Starting $PROG: "
   # Launch one Swatch process for each file included
in $WATCH_FILES var
   for FILE in $WATCH_FILES; do
       $BINARY --config-file=$CONF_FILE --tail-
file=$FILE --daemon &
   done
   /bin/echo "$PROG startup complete."
   return $RETVAL
}


stop()
{
   /bin/echo "Stopping $PROG: "
   for PID in `/sbin/pidof $PROG`; do
       /bin/kill -TERM -$PID 2>/dev/null
       RETVAL=$?
   done
   /bin/echo "$PROG shutdown complete."
   return $RETVAL
}


restart()
{
   stop
   start
   RETVAL=$?
   return $RETVAL
```

```
}

case "$1" in
 start)
       start
    ;;
 stop)
       stop
    ;;
 restart|reload)
       restart
    ;;
 *)
    /bin/echo "Usage: $0 {start|stop|restart|reload}"
    RETVAL=1
esac

exit $RETVAL
```


---

[1] Apart from filtering, ebtables also gives the ability to alter the Ethernet MAC addresses and implement a brouter.

[2] The external IP have been removed for privacy.

[3] The external IP have been removed for privacy.