

# How to create a Stealth Packet Scrubber using HOGWASH

By Michael Karagiannis

**Purpose:** The following document was written to assist those who would like to add to their network security arsenal by creating a Stealth packet scrubber. It will also walk you through the process of setting up a custom kernel on a UNIX like operating system.

What is a stealth packet scrubber?

A stealth packet scrubber is a machine that will sit in between two network segments and have the ability to disarm known network attacks based on a rules based engine. The stealth part come from the fact that the machine has a modified kernel that does not include any support for any Transport layer protocols (TCP,SPX,Appletalk). This makes the box extremely difficult to take down since there is no IP or other logical address to target. Also the originator of the attack will have a very difficult time trying to figure out why the attacks are failing

## **Requirements:**

\*nix

Redhat 7.1 used in this example but the procedure is the same for any \*nix to include the BSD Family.

## **PC with multiple nic cards**

One could run HOGWASH on a platform as small as a Pentium classic 133 MHz machine with 64 Meg of RAM and 2 AMD PCNET network interface cards. Of course one can use a more powerful hardware suite. Basically if you have more than one nic and can run the \*NIX operating system you should be good to go.

## **HOGWASH**

HOGWASH is the software that runs the packet scrubber. HOGWASH runs on top of SNORT and adds the functionality of being able to drop packets that match a certain SNORT rule. Note that installing HOGWASH automatically installs SNORT.

## **LIBPCAP**

LIBPCAP is a library that is required library to run SNORT therefore do not even think of starting this process without verifying that this is installed. For whatever it is worth Libpcap is included with many of the major \*nix distributions so often you pay attention during the install to make sure that the library is

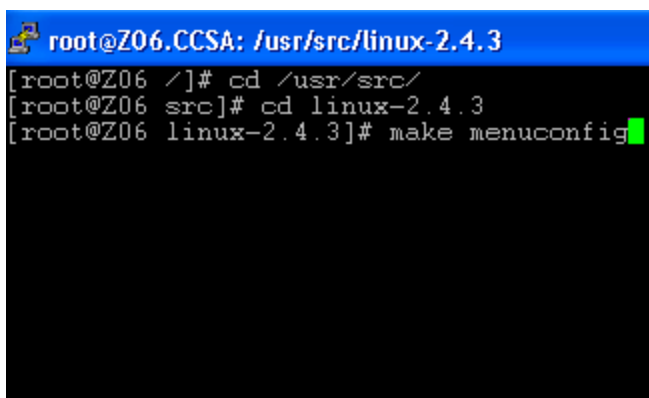
selected during the install I would still recommend having the source tarball available. One can download the sources for this software via the link at the end of this document.

## NET-TOOLS source code

Before you start this procedure make sure that you have the source code for the Net-tools package. Download it from [and](#) have it handy. You will need it in order to recompile the ifconfig utility to allow activation of the network interface without TCP/IP

### Install procedure.

Start by performing a custom installation of the operating system making sure that kernel sources is selected and in all honestly not much else. As far as partitioning, recommend that you crate a separate partition for /var since this machines file system will not be doing anything but writing to log files. Scalability of the packet scrubber is limited by how many processes are running on the machine, and since we will later take out TCP/IP from the kernel, you will find that many of the daemons that are usually installed by default will simply not work anyway. So try to make as small an install footprint as possible. Next verify that both nic cards are working on the install, and download all of the other required software packages to include Libpcap,HOGWASH and the NET-TOOLS source code. Next, recompile the kernel. In our example we will be using a Redhat 7.1 box. Recommend that you download the latest stable kernel sources. Next change into the `/usr/src/linux*` directory that contains your kernel source. From there go into the configuration utility of choice for your kernel. In this example we will use `make menuconfig`. See *example 1-1*

A terminal window screenshot with a blue title bar. The title bar text is "root@Z06.CCSA: /usr/src/linux-2.4.3". The terminal content shows three lines of shell commands and their prompts: "[root@Z06 /]# cd /usr/src/", "[root@Z06 src]# cd linux-2.4.3", and "[root@Z06 linux-2.4.3]# make menuconfig". A green cursor is visible at the end of the third line.

```
root@Z06.CCSA: /usr/src/linux-2.4.3
[root@Z06 /]# cd /usr/src/
[root@Z06 src]# cd linux-2.4.3
[root@Z06 linux-2.4.3]# make menuconfig
```

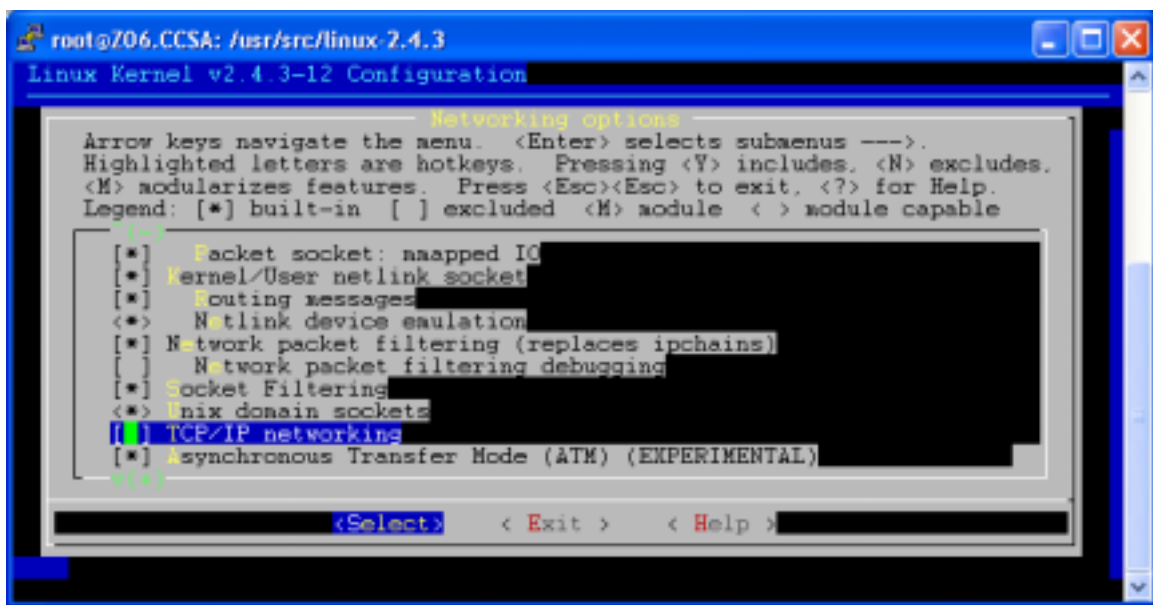
*example 1-1*

For the newbie's out there we are in the scary utility that allows us to define a custom kernel. For with experience, take out IP and remove anything

unnecessary (drivers, protocols, hardware support) to the operation of the packet scrubber.

### Back to the newbie's.

First thing that we need to do is remove TCP/IP support from the kernel. The way that we do that is to go to networking options and deselect TCP/IP support from the menu. You may have to scroll down to find the TCP/IP networking option. An asterisk next to the menu item means that the option will be compiled into the kernel while an M means that the option will be available via a compiled module. TCP/IP will always be selected by default, so you must deselect the box see example 1-2



example 1-2

To make the smallest most efficient kernel possible, recommend that you remove all other transport layer choices including:

- ATM
- Lane emulation
- The IPX protocol
- Decnet
- Bridging
- Wan Router.

So basically everything from TCP/IP down on the screen shown in example 1-2 should be deselected.

In our environment I went ahead and removed everything from the kernel that was not necessary for this machine to work so that I can subsequently port this configuration to a floppy distribution. For those that are unsure of how to further modify the kernel it is ok to stop there save your configuration and proceed to the next step. Otherwise, go ahead and remove every other option in the kernel that is not necessary and then save your configuration.

### Compiling the new kernel.

If you are using the same exact version of the kernel you can get away with just doing *a make dep* and then *make bzimage*. If not, then you must also go through making the modules. Consider the following command to do all of this in one shot:

```
make dep bzImage modules modules_install
```

With any luck and quite a bit of time if you are using as slow a machine as I am, you will eventually have a new kernel ready to go. The new kernel will then be placed in *the /usr/src/linux\*/arch/i386/boot* directory and you can then make it available to the system. The bzImage file needs to be copied into the /boot directory as well as the system.map file.

Examples:

**Copy new kernel to /boot directory**

```
cp /usr/src/linux*/arch/i386/boot/bzimage /boot/
```

**Rename old System.map Just in case you hosed something up**

```
rn /boot System.map oldSystem.
```

**Copy new System.map to /boot directory**

```
cp /usr/src/linux*/System.map /boot
```

## Modifying the boot loader

To save some potential aggravations take a peek at your current lilo configuration by opening up the file `/etc/lilo.conf` in the text editor of your choice.

Example 1-3a

**Before**

```
root@Z06.CCSA: /etc
[root@Z06 /etc]# more lilo.conf
boot=/dev/ida/c0d0
map=/boot/map
install=/boot/boot.b
prompt
timeout=50
message=/boot/message
linear
default=linux

image=/boot/vmlinuz-2.4.3-12
    label=linux
    root=/dev/ida/c0d0p1
    read-only
    initrd=/boot/initrd-2.4.3-12.img
[root@Z06 /etc]#
```

Example 1-3b

**After**

```
root@Z06.CCSA: /etc
UW PICO(tm) 4.0 File: /etc
boot=/dev/ida/c0d0
map=/boot/map
install=/boot/boot.b
prompt
timeout=50
message=/boot/message
linear
default=NO-IP

image=/boot/vmlinuz-2.4.3-12
    label=linux
    root=/dev/ida/c0d0p1
    read-only
    initrd=/boot/initrd-2.4.3-12.img

image=/boot/bzImage
    label=NO-IP
    root=/dev/ida/c0d0p1
    read-only
    initrd=/boot/initrd-2.4.3-12.img
```

In example 1-3 we have your current lilo configuration, for those of you that are new to this. All the information for the `image=` line down has to do with your current configuration. We will now add another `image=` block to the file with a label of `NO-IP`. We will then change the `default =` line to point to `NO-IP`. This will allow us to get into the machine using the original configuration if anything went wrong with your new kernel. To go back to the original configuration you would just enter `Linux` at the lilo prompt, or select it from the menu depending on how your box is setup. Now to invoke the configuration you would and run the lilo

command. Literally just type *lilo*. The asterisk next to the entry means that the configuration is the default. See example 1-4

Example 1-4 just type lilo

```
root@Z06.CCSA: /usr/src/linux-2.4/arch/i386/boot
[root@Z06 boot]# lilo
Added linux
Added NO-IP *
[root@Z06 boot]# █
```

### Reboot and Kiss you IP goodbye.

Now we will go into the configuration of the host itself. When you reboot your machine, you will probably notice that you are unable to activate the network interfaces. This is because you will need to recompile *ifconfig* via the *net-tools* package. So untar the *net-tools* package and run the configure script.

```
cd nettools-1.54
./configure.sh config.in
```

You will then be answering a series of questions that you will answer no to with the exception of answering yes to Unix protocol Family and Ethernet support. After that you can simply run

```
make all
make install
```

Viola you now should be able to bring up you network interfaces with

```
ifconfig eth0 up
ifconfig eth1 up .
```

Finally, we are now ready to install HOGWASH. First go ahead and create the directories for SNORT and HOGWASH to log to, this will prevent a startup error when you are done installing HOGWASH

```
mkdir /var/log/SNORT
mkdir /var/log/HOGWASH
```

It is amazing how many users I have seen ask why SNORT gives an error about this simple step. So make sure to make these directories.

Next go ahead and run the setup utility. If you are truly lucky (and have the PCAP sources in the right place), you will have a working SNORT and HOGWASH in a few moments. For the rest of us, you will have to modify the Makefile.in file to point to the source directory for PCAP. This is why I recommended having the source tarball for Libpcap available. You can usually just point the Makefile to the directory that you extracted pcap (usually found in */usr/include/pcap*).

Try a make followed by a make install and you should be on your way.

Example syntax for starting HOGWASH is

```
HOGWASH -i eth0 -e eth1 -c stock.rules.
```

I usually move the HOGWASH binary to */usr/sbin* and create a folder in */etc* to hold the HOGWASH rules.

```
cp ./hogwash /usr/sbin
mkdir /etc/hogwash
cp ./stock.rules /etc/hogwash
```

At this point you are ready to start modifying the rule set to tailor it to your environment. Please use this on a test LAN before you move to a production environment. You do not want to take down your production network to a bad rule set configuration or false positives. But that is another paper.

### References

SNORT <http://www.SNORT.org>

HOGWASH <http://HOGWASH.sourceforge.net/>

LibPcap <http://www.tcpdump.org/>

Net-tools <http://www.tazenda.demon.co.uk/phil/net-tools/>

redhat Linux <http://www.redhat.com>

Michael Karagiannis <mailto:michaelkaragiannis@melly.net>

*Michael Karagiannis spends his days at the Command and Control Support Agency of the US Army as a government contractor.*

