# Table of Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Pages</th>
</tr>
</thead>
<tbody>
<tr>
<td>Introduction</td>
<td>3-4</td>
</tr>
<tr>
<td>Physical Setup</td>
<td>4-7</td>
</tr>
<tr>
<td>Routing</td>
<td>7-9</td>
</tr>
<tr>
<td>Remote System Logging</td>
<td>10-12</td>
</tr>
<tr>
<td>Firewall Setup</td>
<td>12-14</td>
</tr>
<tr>
<td>Snort Setup (IDS)</td>
<td>14-18</td>
</tr>
<tr>
<td>Swatch Setup</td>
<td>18-19</td>
</tr>
<tr>
<td>Iptables Rule Lists</td>
<td>20-21</td>
</tr>
<tr>
<td>Links</td>
<td>22</td>
</tr>
<tr>
<td>Appendix</td>
<td>23-24</td>
</tr>
</tbody>
</table>
What is a Honeynet?

Network security is critical. Business plans, credit card numbers, and next week's midterm are just a few things that we don't want others to obtain. Unfortunately as the internet continues to grow, the number of services increases and security becomes even more of an issue. In order to defend ourselves and better develop software and hardware, it is important to understand how systems are compromised and what a hacker may or can do to a system. Many times system administrators learn the hard way by being hacked into themselves. There is no better way to understand a hacker's tools and motives than to watch it happen and then be able to piece the steps back together.

A honeynet is a network specifically designed for the purpose of being compromised by the blackhat community. It is a research tool to study the tactics and tools of hackers and research new holes in software. It is different from a single honeypot, which is just a single machine designed to lure in hackers. Often with honeypots, the operating system and rest of the network is emulated, giving a hacker only limited functionality. They are not given the freedom to fully compromise a network. Instead we would like to learn as much as possible by letting a hacker be able to fully exploit many different systems and services. A honeynet sits behind a firewall where all inbound and outbound data is contained, captured, and controlled. We are fully able to monitor everything that goes on. The data is then analyzed to gain information about how the system was compromised and what the hacker did to the system while they were there. The goal is to have the honeynet look like a real, attractive network so that hackers will fully compromise your system and not be suspicious that you are monitoring their every move.

The honeynet we designed here was based on The Honeynet Project (http://project.honeynet.org) described in the book *Know Your Enemy, Revealing the Security Tools, Tactics, and Motives of the Blackhat Community*. Our network was set up with mostly spare parts and software that we already had. In fact, the only cost to us was the $40 hub and of course our own time. You do not need top of the line hardware because your honeynet is simply being compromised. It is important to remember also when designing your own honeynet that the less traffic you create on it yourself, the easier it will be to analyze your data. With that in mind, our
honeypots were not used by anyone for anything except maintenance. Any traffic coming from one of the honeypots is then a successful hack.

Our honeynet consists of a total of six machines. We have two honeypots, with default installations of Windows 2000 Server and Linux Red Hat 6.2. We did nothing to the honeypots to make them more or less secure than a usual installation, including no patching. A firewall, than a router controlled the packets coming in. Outside is the intrusion detection system and our system log server whose purpose is to monitor and log all of our data. The rest of this document describes in detail how to setup and maintain a honeynet.

### Physical Setup

#### Equipment Needed

**6 computers**
- 4 running Red Hat Linux 7.3, 1 running Red Hat 6.2, and 1 running W2k Server
- 2 four port hubs (may also use a one switch and one hub)
- 9 Network Interface Cards (3 computers need 2, other 3 need one)
- 1 crossover cable
- 1 internet connection with port 80 available for incoming requests

#### Setup

Brief description of each machine

**Firewall** - [Red Hat 7.3, 2 Network Cards] The firewall filters all traffic entering and leaving the honeynet. It allows all traffic into the honeypots as we want hackers to be able to find them. The firewall limits outbound traffic to five new connections per day from each honeypot. This limit is in place so the honeypots cannot be used in DoS or other similar attacks. All
incoming and outgoing traffic is also logged by the firewall. More Information (the firewall must be setup for routing packets first)

Router – [Red Hat 7.3, 2 Network Cards] The router may seem redundant in our setup. However, it provides several important functions. The router is used for anti-spoofing, protecting the firewall from a compromised honeypot, and additional logging. More Information

IDS – [Red Hat 7.3, 2 Network Cards] The IDS machine sniffs the traffic entering and leaving the honeypots with Snort. Snort captures and logs all the honeynet traffic and will detect port scans, potential attacks, and other suspicious activity. More Information

Syslog – [Red Hat 7.3] The syslog server is setup to receive all the log information from every machine in the honeynet. By having a remote syslog server, all the data captured in our honeynet is protected and centralized. This allows for only one instance of Swatch to be running on our network. Swatch analyzes log information and sends out email alerts when it finds any specified traffic. More Information

Linux Honeypot – [Red Hat 6.2] The linux honeypot is a default installation of RH 6.2. This version was chosen as it has several known vulnerabilities and is very inviting to the hacker community. The web server (Apache) has only an under construction web page. A modified syslog daemon is used to mask the remote logging.

W2k Honeypot – [Windows 2000 Server] The W2k honeypot is a default installation of W2k Server. The IIS web server, ftp server, and terminal services are turned on with an under construction page used for the web server. The system should be patched for the Code Red virus (nothing else!). An additional program is used to send log information to the syslog server.

On both the honeypots, fake users with directories and files should be added. We want the machines to appear as though they are in use and a part of an actual network. Productivity software should also be added to the honeypots.

Flow of traffic

The best way to understand how the honeynet works is to follow a little bit of imaginary traffic through the network. Say, for instance, Paul from France sends a request to our web server on the W2k honeypot. This request will go through the first hub to the firewall. The firewall will see it as incoming traffic, allow it through, and make a log entry. While the packets pass through the router, the firewall’s log entry will go through the first hub to the syslog server where Swatch will catch it and send out an email alerting the administrators of network activity (any traffic on a honeynet is regarded as suspicious). Meanwhile, the router passes the web server traffic through to the W2k honeypot where the IIS web server sends our under construction website back to the router. The router checks for ip
spoofing then forwards the traffic to the firewall. The firewall sees the traffic as a response to the incoming request and allows it to go through to Paul’s computer in France. During this entire episode, the IDS is busily capturing every packet passing through the honeynet, logging the contents of each packet, and parsing out worthwhile data for each session.

Let’s look at the simple example of an IIS exploit. Paul sends out his IIS exploit. It is forwarded through the firewall, the firewall logs it, the syslog server emails its alert, and the request is passed onto the router. The router passes it to the W2k honeypot. The IDS sniffs the exploit’s packets and recognizes the attack. The IDS logs an alert to the syslog server and Swatch sends out an email with a more urgent subject line. The exploit is carried out on the W2k honeypot and the system is compromised. The entire exploit has been logged. We know what time it occurred, where it came from, and every command that was executed to compromise the system. At this point, our honeynet is successful.

It is obvious that great care must be taken to ensure the security of the firewall, router, IDS, and syslog server. As such, the flow of traffic through the honeynet is strictly regulated. Below is a diagram of the honeynet with arrows showing the flow of traffic.

Forwarded traffic is allowed both ways through the firewall, but it is filtered according to the firewall’s rules. Likewise with the router. Traffic destined for the firewall and router is limited to local DNS servers and everything else is dropped. The honeypots do no filtering themselves and do not restrict any packets, thus allowing all traffic that gets through the firewall and router. The IDS must be regulated as it is a possible backdoor out of the honeynet without going through the firewall. Therefore, the ethernet interface connected to the second hub (the hub connected to the honeypots) must have a private ip address (192.168.xxx.xxx or 10.0.0.xxx). Also, the only incoming traffic allowed is from Snort. Also, the only outgoing
traffic is the traffic needed for remote logging. With this setup, even if the
IDS system is discovered by a hacker on a compromised honeypot, the
hacker will not be able to get any traffic to pass through the IDS. The
syslog server is also restricted. The only incoming traffic allowed is to
receive remote log entries. This is specified to a specific port and protocol.
The only outgoing traffic allowed is sendmail traffic for the email alerts.
With this flow of traffic, the honeynet’s secure area is kept secure. Example
iptables rules for each of the systems can be found here.

Just the beginning
At this point in the document, we are hoping that the physical setup
of the honeynet is understood. The details for setting up each of the
computers are found in the remaining portions of this document. The next
topic is configuring linux for routing packets, a critical step for the firewall
and router.

Routing Setup
Due to our lack of financial resources, we looked to linux for a way to route
traffic through our honeynet. This setup is needed not only for the router,
but also the firewall. The firewall is just a router with a packet filter.
Therefore, configure both the router and firewall as this section describes,
then move on to the firewall setup.

Linux Installation
Install Red Hat 7.3 as normal, just make sure to select the Router/Firewall
package when selecting software to install.

IP forwarding and Proxy ARP
After RH 7.3 is installed, there are several system options that must be
enabled. First, IP forwarding must be turned on. This is done by setting the
/proc/sys/net/ipv4/ip_forward file to read “1”. This must be set every time
the computer is started, so the easiest way to do this is add the following
line to the /etc/rc.d/rc.local file:

```bash
echo 1 > /proc/sys/net/ipv4/ip_forward
```

Next, the router (and firewall) needs to have the proxy arp enabled. We
need a couple more “1” files, one for each Ethernet interface. These files
are found in the /rpoc/sys/net/ipv4/conf/eth* directories and need to be set
for every boot. The following lines should be added to the /etc/rc.d/rc.local
file:

```bash
echo 1 > /proc/sys/net/ipv4/conf/eth0/proxy_arp
echo 1 > /proc/sys/net/ipv4/conf/eth1/proxy_arp
```
At this point, the system is ready to forward packets, however, it needs a little more direction.

**Routing**

Now we need to give the router and firewall a roadmap for routing packets. Also the honeypots need a little network configuration to properly use the network. The following is an example honeynet with fake IP addresses.

### Honeynet

Both the firewall and router have two Ethernet cards and therefore two IP addresses (eth0 is the right interface and eth1 is the left interface for this example). The roadmap we are giving to the router and firewall is found in the routing tables stored on each machine. To see a listing of the routing tables, give the `route` command at a prompt. To get our example above to work correctly, the `route` command should give a table similar to the following for the firewall:

```
Destination Gateway      Genmask               Flags Metric   Ref  Use   Iface
10.0.0.6  *  255.255.255.255   UH 0           0      0       eth1
10.0.0.1  *  255.255.255.255   UH 0           0      0       eth1
10.0.0.2  *  255.255.255.255   UH 0           0      0       eth1
127.0.0.0  *  255.0.0.0               U 0           0      0       lo
default  gateway-ip 0.0.0.0               UG 0           0      0       eth0
```

The first entry routes packets to the router, the second entry passes packets for the linux honeypot to the router, and the third entry passes packets for the w2k honeypot to the router. The fourth entry is for the loopback and should be there by default. The last entry is the default gateway for your external network, where gateway-ip symbolizes the IP address of the gateway. When the route command is first issued, your machine will output some assumed routing table entries. We had the best luck by flushing out the routing table and adding only the entries we want. So, the first commands to give are to flush out the table. This is accomplished by the following command:
route del [-host,-net] [ip] netmask [netmask] dev [eth0,eth1]

So, if your initial routing table give an entry such as:

```
10.0.0.0  *  255.255.255.0 UH 0 0 0  eth1
```

This can be removed with the following command:

```
route del -net 10.0.0.0 netmask 255.255.255.0 dev eth1
```

Use this command to remove all of the existing routing table entries except for the loopback entry. Then, we need to add the routing entries for our honeynet so the table resembles the one shown above. This is accomplished with the `route add` command. It has the same syntax as the `route del` command. The following commands will give the routing table shown in our example above.

```
route add -host 10.0.0.6 dev eth1
route add -host 10.0.0.1 dev eth1
route add -host 10.0.0.2 dev eth1
route add default gw gateway-ip dev eth0
```

The routing tables are reset every time the computer starts, so we need to add more to the `/etc/rc.d/rc.local` file. Put all the commands to del the default routing table entries and all the commands to add the entries we want for the honeynet. The routing table for the router is shown below. Add the appropriate commands to the router's `/etc/rc.d/rc.local` file and the routing tables will be set (note the default gateway is the ip address of firewall's internal interface).

```
Destination Gateway Genmask Flags Metric Ref Use Iface
10.0.0.5 * 255.255.255.255 UH 0 0 0 eth0
10.0.0.1 * 255.255.255.255 UH 0 0 0 eth1
10.0.0.2 * 255.255.255.255 UH 0 0 0 eth1
127.0.0.0 * 255.0.0.0 U 0 0 0 lo
default 10.0.0.5 0.0.0.0 UG 0 0 0 eth0
```

**Honeypots**

The honeypots' network configuration is pretty straight forward. The final setup does not require setting routing tables. Simply enter the network interface setup, enter the ip address of the honeypot and set the ip address of the inside interface of the router to be their default gateway (in our example, 10.0.0.7).
Remote System Logging

The capability of remote logging is critical to the functionality of the honeynet. With remote logging, all traffic coming through the firewall, any system changes made to the honeypots, and even shell commands can be reviewed by one machine that is masked from the outside world. The remote system logger can run SWATCH and review all the log files from the entire honeynet and email system administrators when specific events occur.

Syslog Server
This is the machine that receives all of the system logs and runs SWATCH. The configuration for making a Linux machine receive remote logs is fairly simple. There are three files that must be changed.

1. In the file /etc/rc.d/init.d/syslog, there is a line that reads:
   ```bash
   SYSLOGD_OPTIONS="-m 0"
   ```
   Add the 'r' flag to the options being passed to syslog:
   ```bash
   SYSLOGD_OPTIONS="-m 0 -r"
   ```
   The 'r' makes the syslog daemon open port 514 and listen for incoming log information.

2. In the file /etc/sysconfig/syslog, there is a line that is the same as the line found in the above file. The 'r' flag must be added to this line also.

3. The syslog daemon service must be added into the /etc/services files. Place the following line after the isakmp entry (or so it is in numerical order):
   ```plaintext
   syslog 514/udp
   ```

With these three files changed, run the following command:
```
/sbin/service syslog restart
```
A reference to remote logging being enabled should appear in the /var/log/messages file. If the message is there, the remote syslog server is running. It is wise to set up a firewall on the server so the log files cannot be changed or removed. In our honeynet, we have a dedicated syslog server, so the only traffic that should be coming into the machine are udp packets to port 514. By using iptables, a simple firewall can be created:
```
iptables -A INPUT -s honeynet --destination-port 514 ** -j ACCEPT
iptables -A INPUT -j DROP
```

Linux Honeypot
The linux honeypot setup is a little more complicated than the server. One of the first actions taken by a hacker after a system is compromised is to cover his/her tracks. This is done by modifying or removing any type of logging that is done on the system. However, this information is what is critical to the success of a honeynet. So, we want to not only log this
information remotely, but hide that fact that this is being done. Making a
system send its log messages to a remote system is very simple. This is
accomplished by adding the following line to the /etc/syslog.conf file:

```
*.*  @xxx.xxx.xxx.xxx  (ip of syslog server)
```

This tells syslog to send every message to the specified ip address. The
messages will be sent as udp packets to port 514. (A quick note, since the
messages are using the udp protocol, transmission of every message is not
guaranteed.) Restart syslog with the command shown above and logs will
be sent remotely.

The problem with this setup is that a smart hacker will examine the
syslog.conf file for remote syslogging. Therefore, we must hide our remote
logging. First, download the source for syslog. It is packaged as
sysklogd.xx.xx. Unpack the source and open syslogd.c. Search for the line:

```
#define _PATH_LOGCONF    "/etc/syslog.conf"
```

Change the location value to the location where the real syslog.conf file is
located. Something like /etc/.sys/asp.conf will hide the file best. Then,
compile the source with the commands:

```
./configure; make
```

Backup the old version of the syslogd binary then, copy the new syslogd
binary into /sbin/syslogd. Make sure to keep the old syslog.conf file
(without the line for remote logging of course). This will further fool the
hacker. Copy the syslog.conf file to the location that was specified above
and make the remote syslog changes. Restart syslog as shown before and
check for any problems. If everything is working ok, remove the source files
and the backup syslogd binary. Now the system is “secretly” logging to a
remote server.

**Windows Honeypot**

Windows machines do not use a system logger that produces entries in
syslog format. Therefore, a separate program that can convert the Windows
logging reports and send them to the remote syslog server must be
installed. We use a version of the Adison NT EvntSlog. The directions for
installing this application are easy to follow. Unfortunately, this program
does show up as a service that is started automatically. But, it does not
show up in the running process list. This creates a little diversion from
remote logging.

**Firewall, Router, IDS (Snort)**

All three of these machines are configured the same as explained in the first
section for the Linux Honeypot. Logs will be sent remotely by simply adding
the following line to the /etc/syslog.conf file:

```
*.*  @xxx.xxx.xxx.xxx  (address of syslog server)
```
More Information
On the syslog server, the received messages are treated as local syslog messages. They are parsed as specified in the syslog.conf file on the server. In the /var/log/messages file, all remotely logged messages are displayed with the hostname or ip address of the source machine. Further logging can be accomplished by putting a modified bash program on the linux honeypot. This bash logs all inputted commands to the local syslog where they are passed to the remote syslog server. With a little tweaking of the hidden syslog.conf file, it is possible to make these messages not show up in the /var/log/messages file. This is important as logging bash commands is abnormal behavior and will arouse suspicion in a hacker.

Firewall Setup
The honeynet firewall is built with iptables (1.2.5) included in Linux 2.4.18-3 kernel. Iptables is a rules-based filter. When the machine receives a packet, iptables starts at the top of the rule list and compares packets to each rule by looking at characteristics specified by the rules. When a match occurs, the action specified by the rule is applied to the packet.

Iptables Commands Used
The following are brief explanations of commands used in the honeynet firewall:

- `-A policy` "A" appends a rule to the policy specified
- `-s ip` Source ip address of packet
- `-d ip` Destination ip address of packet
- `-p protocol` Protocol of packet (tcp, udp, etc)
- `--destination-port port` Destination port of packet
- `-m` Allows matching capability for rule
- `-j action` Action to be carried out for rule

Firewall Rule List
The following are the rules used for the firewall. Each rule is preceded by an explanation (actual ip addresses are left out for obvious reasons and numbering is only for display purposes) and each rule is meant for one line with no carriage returns. This table without the comments can be found here.

Simply flushes all rules
1. iptables -F

Allows outgoing udp packets destined for port 514 to remote syslog server
2. `iptables -A FORWARD -s w2k-honeypot -d remote-syslogger -p udp --destination-port 514 -j ACCEPT`
3. `iptables -A FORWARD -s linux-honeypot -d remote-syslogger -p udp --destination-port 514 -j ACCEPT`

Allows packets destined for name servers

4. `iptables -A FORWARD -d dns-server1 -j ACCEPT`
5. `iptables -A FORWARD -d dns-server2 -j ACCEPT`

Logs all packets coming into honeynet, logs packet info to syslog with title of "Firewall-In:"


Allows all packets coming into both honeypots

7. `iptables -A FORWARD -d w2k-honeypot -j ACCEPT`
8. `iptables -A FORWARD -d linux-honeypot -j ACCEPT`

Logs all new connection packets and new connection packets from established connections with the w2k-honeypot source ip, logs packet info to syslog with title of "Firewall-Out:"


Allows all packets from an established connection with the w2k-honeypot ip

10. `iptables -A FORWARD -s w2k-honeypot -m state --state ESTABLISHED,RELATED -j ACCEPT`

Allows five new connection packets coming from w2k-honeypot per day (#11), after five new connections, all packets are dropped (#12)

11. `iptables -A FORWARD -s w2k-honeypot -m state --state NEW -m limit --limit 5/day --limit-burst 5 -j RETURN`
12. `iptables -A FORWARD -s w2k-honeypot -j DROP`

Same as rules #9-12, but for linux-honeypot

14. `iptables -A FORWARD -s linux-honeypot -m state --state ESTABLISHED,RELATED -j ACCEPT`
15. `iptables -A FORWARD -s linux-honeypot -m state --state NEW -m limit --limit 5/day --limit-burst 5 -j RETURN`
16. `iptables -A FORWARD -s linux-honeypot -m state --state NEW -j DROP`

If packets do not match any of the above rules, drop the packet

17. `iptables -A FORWARD -j DROP`
What does this mean in non-iptables language?

This firewall allows anyone into the honeynet (7,8). The honeypots can only make five outbound connections a day (11,12,15, 16). This rule is set so that once a honeypot is compromised; it cannot be utilized in DoS attacks, virus propagation, and other similar attacks. Everything else is dropped (17). New connections coming into and going out of the honeynet are logged to the local syslog and then passed to the remote system logger (6,9,13) where SWATCH sends email alerting the admins to honeynet network activity.

Place these rules in the /etc/rc.d/rc.local so they are applied every time the machine starts. Another way is to put the rules in a new file, make the file executable (chmod +x filename), then call the file in the /etc/rc.d/rc.local. This will apply the rules at startup and allow the rules to be reapplied anytime the file is called (ex. want to reset 5/day outbound connections to give a hacker a little more room to work).

Snort

Snort is the intrusion detection system used in our honeynet. Configuring snort requires two different files; a shell script that executes snort and managed the log directories, and a snort.conf file that contains all the rules. We used the most current version of Snort (1.8.6) with just a basic installation of the most current signatures (rule sets) from the www.snort.org website. The machine that Snort runs on requires two ethernet cards, one to sniff the honeynet and a second to send alerts to the syslog server. The second card is not necessary, but saves from having the alerts travel back through the router and firewall and creating extra traffic. With two cards you can lock down the system (really simple with iptables) only allowing UDP on port 514 to the syslog server and make it invisible to the outside world.

The Shell Script (snortsh)

The basic idea of the shell script is to kill off the currently running Snort process, and restart it logging into a different directory named after the current date. We created a new crontab (a new task) that would execute this script once every night at midnight, so that a new date directory would be created. Also we added an entry into the rc.local file that would start snort each time the machine booted.
Snort is executed from the snort.sh file with the following options:

```
$SNORT -b -c $DIR/snort.conf -D -i eth0 -l $DIR/logs/$DATE -s -u $USER
```

- **-b**  Log in binary mode (TCPdump format, much faster)
- **-c <file>**  Use the rules specified in the following file (snort.conf)
- **-D**  Run Snort in the background (daemon mode)
- **-i <inf>**  Listen on the following interface (eth0 or eth1)
- **-l <dir>**  Log in directory (/root/snort-1.8.6/LOGS/$DATE)
- **-s**  Log alert messages to syslog

Below is the actual snort.sh script with comments:

```bash
#!/bin/bash
#
# snort.sh
#
PATH=/bin:/usr/local/bin
PID=`cat /var/run/snort_eth1.pid`
DIR=/root/snort-1.8.6
DATE=`date +%b_%d`
SNORT=/usr/local/bin/snort
USER=snort

### Kill snort
echo "\nKilling snort, PID $PID\n"
kill $PID > /dev/null 2>&1

### Create daily directory to archive log files
if [ -d $DIR/LOGS/$DATE ];then
  :
else
  mkdir $DIR/LOGS/$DATE
fi

### launch snort
$SNORT -b -c $DIR/snort3.conf -D -i eth0 -l $DIR/logs/$DATE
```

### The Rules File (snort.conf)

The rules file is where you specify what types of things snort will be detecting and what data we would like to log. The basic signatures that are all included at the end of the file are all basic rule sets that other snort users have created and that the web site provides for basic intrusion detection.

Below is the actual rules file with comments in hope that it is easily understood.
var INTERNAL your home net IPs
var EXTERNAL_NET any
var PORTS 5
var SECONDS 15
var DNS_SERVERS [your DNS server IPs
var SMTP $HOME_NET
var HTTP_SERVERS $HOME_NET
var SQL_SERVERS $HOME_NET
var RULE_PATH ./rules/
var HTTP_PORTS 80
var ORACLE_PORTS 80
var SHELLCODE_PORTS 80

#### Preprocessors

## Normalizes http requests from remote machines by
## converting any %XX character substitutions to their ASCII equivalent. This is useful for doing things like defeating hostile attackers trying to stealth themselves from IDSs by mixing these substitutions in with the request. It is checking on ports 80, 443, 8080
preprocessor http_decode: 80 443 8080

## Provides TCP stream reassembly and stateful analysis capabilities to snort.
## the 'detect_scans' option turns on alerts for portscan events.
preprocessor stream4: detect_scans

## Detect portscans with a source in our home network, considering more than 5 ports scanned in 15 seconds as a portscan. Log the portscan into the portscan.log file
preprocessor portscan: $HOME_NET $PORTS $SECONDS /root/snort-1.8.6/LOGS/portscan.log

## Ignore portscans from DNS servers
preprocessor portscan-ignorehosts: $DNS_SERVERS

## Back Orifice traffic detector, the -nobrute force option turns off the decryption of the traffic (can slow down the overall performance of snort)
preprocessor bo: -nobrute

## Normalizes telnet negotiation strings from telnet and FTP traffic. It works in much the same way as the http_decode preprocessor, searching for traffic that breaks up the normal data stream of a protocol and replaces it with a normalized representation of that traffic, so that the "content" pattern matching keyword can work without requiring modifications.
preprocessor telnet_decode

## RPC may be sent in alternate encodings besides the usual 4-byte encoding that is used by default. This preprocessor normalizes the RPC traffic on ports 111 and 32771.
preprocessor rpc_decode: 111 32771
Snort was having problems logging alerts to the syslog. It should have worked fine by adding the -s option when we started snort, but it didn't. After much playing around and researching we had to add the following line to make it log successfully.

```
output alert_syslog: LOG_AUTH LOG_ALERT
```

### LOGGING INFORMATION

#### Log all TCP connection
- Log all ASCII TCP activity to session breakout files
  
```
log tcp any any <> any any (session: printable;)
```

#### Log all TCP activity to binary file
- Log all ASCII TCP activity to session breakout files
  
```
log tcp any any <> any any
```

#### Log all UDP activity to binary file
- Log all ASCII TCP activity to session breakout files
  
```
log udp any any <> any any
```

#### Log all ICMP activity
- Log all ASCII TCP activity to session breakout files
  
```
log icmp any any <> any any
```

#### Below is a handy line to use to test your snort setup and swatch setup
```
alert tcp any any -> any any (msg: "TCP traffic");
```

#### Here are all of the standard snort signatures (rule files)
```
include classification.config
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/smtp.rules
include $RULE_PATH/telnet.rules
include $RULE_PATH/dos.rules
include $RULE_PATH/ddos.rules
include $RULE_PATH/dns.rules
include $RULE_PATH/rfc.rules
include $RULE_PATH/rfc.rules
include $RULE_PATH/rfc.rules
include $RULE_PATH/rfc.rules
include $RULE_PATH/rfc.rules
include $RULE_PATH/rfc.rules
include $RULE_PATH/rfc.rules
include $RULE_PATH/rfc.rules
include $RULE_PATH/rfc.rules
include $RULE_PATH/rfc.rules
include $RULE_PATH/rfc.rules
include $RULE_PATH/rfc.rules
include $RULE_PATH/rfc.rules
include $RULE_PATH/rfc.rules
include $RULE_PATH/rfc.rules
include $RULE_PATH/rfc.rules
include $RULE_PATH/rfc.rules
include $RULE_PATH/rfc.rules
include $RULE_PATH/rfc.rules
include $RULE_PATH/rfc.rules
include $RULE_PATH/rfc.rules
include $RULE_PATH/rfc.rules
include $RULE_PATH/rfc.rules
include $RULE_PATH/rfc.rules
include $RULE_PATH/rfc.rules
include $RULE_PATH/rfc.rules
include $RULE_PATH/rfc.rules
include $RULE_PATH/rfc.rules
include $RULE_PATH/rfc.rules
include $RULE_PATH/rfc.rules
include $RULE_PATH/rfc.rules
include $RULE_PATH/rfc.rules
include $RULE_PATH/rfc.rules
include $RULE_PATH/rfc.rules
include $RULE_PATH/rfc rules
The Data Files

The data files that Snort created are stored in whatever directory you specified into the snort.sh file. There will be two types of files. One type will be a binary file containing all the packets from the network, or also called a 'TCPdump' file. These files will be named <month><day>@<time>.log. You can view the TCPdump file in any sniffer that supports the tcpdump binary format such as tcpdump or Ethereal. Snort can also process the binary file and read back the packets by using the '-r' option. For example to convert the binary log file "0715@0910.log" into a file called 0715.dump you can use the following command:

./snort -dvr 0715@0910.log > 0715.dump

The second type of a file is a TCP session file. These files contain all the printable ASCII characters that a user sees or types during a TCP session. It is useful for seeing what users are typing in telnet, ftp, and web sessions. These files are put into directories by the source IP of the user starting the session. Inside these directories are then one file for each session. They are named SESSION:<source IP port>-<destination IP port> and they require no special commands to view. If there are people making connections outbound from one of your honeypots, there will be a directory with the machines IP address and all of the sessions started from that machine will be stored there. It is useful to go there to see what TCP sessions a hacker was making after he/she broke into your system.

Swatch

Swatch is a utility that monitors system log files, filters out unwanted data and takes specified actions (i.e., sending email, executing a script, etc.) based upon what it finds in the log files. Swatch was run on the syslog server. We configured the program to look for four different types of entries in the syslog (/var/log/messages): alerts from Snort, specific NT/2000 entries, and traffic in and out of our firewall. If found, the specific entry is first emailed to our alert email account. This allows us to have real time notification of suspicious activity. The entry is also archived to the log file /var/log/IDS-scans, which later can be used for data analysis and research.

To Setup Swatch...

Swatch is rather easy to set up. The rules file is easy to understand and write. Below is a copy of our rules file and comments describing each part.
# Swatch configuration file
#
# watch -c /etc/swatchrc -t /var/log/messages
#
###   Snort Alerts
##  -Watch for entries containing the word
##   'snort'
##  -Display it in green on the screen
##  -Mail alert to alerts@yourdomain.com with
##   subject of the email being "----Snort IDS
##  -Log in file /var/log/IDS-scans

watchfor /snort/
  echo green_h
  mail addresses=alerts@yourdomain.com,subject=--- Snort IDS Alert ---
  exec echo $0 >> /var/log/IDS-scans

###    Specific NT/2000 Signatures
##   watch for entry containing 'msadcs.dll' 'ism.dll' or
##   'showcode.asp'
##   and take same actions as above

watchfor /(msadcs.dll|ism.dll|showcode.asp)/
  echo green_h
  mail addresses=alerts@yourdomain.com,subject=--- NT IIS Alert ---
  exec echo $0 >> /var/log/IDS-scans

###  Traffic in through the firewall
##     watch for entry containing 'Firewall-In'

watchfor /Firewall-In/
  echo green_h
  mail addresses=alerts@yourdomain.com,subject=---Firewall In Traffic Alert---
  exec echo $0 >> /var/log/IDS-scans

###  Traffic out through the firewall
##     watch for entry containing 'Firewall-Out'

watchfor /Firewall-Out/
  echo green_h
  mail addresses=alerts@yourdomain.com,subject=---Firewall Out Traffic Alert---
  exec echo $0 >> /var/log/IDS-scans

The following line was then added to the /etc/rc.d/rc.local file to
start up each time the machine reboots.
/usr/bin/swatch -c /etc/swatchrc.txt -t /var/log/messages &
The -c <file> option specifies which rules file to look at and the -t <file>
options tells which file to monitor.
Here are the iptables rules used for each machine in the honeynet. These rules can be added to the `/etc/rc.d/rc.local` file so they are applied every time the machine boots. The names after the `-s` and `-d` flags should be replaced with IP addresses of the machines specified.

**Firewall**

```
iptables -F
iptables -A INPUT -s dns-server1 -j ACCEPT
iptables -A INPUT -s dns-server2 -j ACCEPT
iptables -A INPUT -j DROP
iptables -A FORWARD -s w2k-honeypot -d remote-syslogger -p udp --destination-port 514 -j ACCEPT
iptables -A FORWARD -s linux-honeypot -d remote-syslogger -p udp --destination-port 514 -j ACCEPT
iptables -A FORWARD -d w2k-honeypot -j ACCEPT
iptables -A FORWARD -d linux-honeypot -j ACCEPT
iptables -A FORWARD -d dns-server1 -j ACCEPT
iptables -A FORWARD -d dns-server2 -j ACCEPT
```

```
"Firewall-In: " --log-tcp-sequence --log-tcp-options --log-ip-options
iptables -A FORWARD -d honeynet -m state --state NEW,RELATED -j LOG --log-prefix
"Firewall-Out: " --log-tcp-sequence --log-tcp-options --log-ip-options
iptables -A FORWARD -d w2k-honeypot -m state --state ESTABLISHED,RELATED -j ACCEPT
iptables -A FORWARD -d linux-honeypot -m state --state ESTABLISHED,RELATED -j ACCEPT
iptables -A FORWARD -d w2k-honeypot -m state --state NEW -m limit --limit 5/day --limit-burst 5 -j RETURN
iptables -A FORWARD -d w2k-honeypot -j DROP
iptables -A FORWARD -d linux-honeypot -m state --state NEW,RELATED -j LOG --log-prefix
"Firewall-Out: " --log-tcp-sequence --log-tcp-options --log-ip-options
iptables -A FORWARD -d linux-honeypot -m state --state ESTABLISHED,RELATED -j ACCEPT
iptables -A FORWARD -d linux-honeypot -m state --state NEW -m limit --limit 5/day --limit-burst 5 -j RETURN
iptables -A FORWARD -d linux-honeypot -m state --state NEW -j DROP
iptables -A FORWARD -j DROP
```
**Router**

iptables -F
iptables -A INPUT -s dns-server1 -j ACCEPT
iptables -A INPUT -s dns-server2 -j ACCEPT
iptables -A INPUT -j DROP

**IDS (Snort)**

iptables -F
iptables -A INPUT -s dns-server1 -j ACCEPT
iptables -A INPUT -s dns-server2 -j ACCEPT
iptables -A INPUT -j DROP
iptables -A OUTPUT -d remote-syslogger -p udp --destination-port 514 -j ACCEPT
iptables -A OUTPUT -j DROP

**Remote Syslog Server**

iptables -F
iptables -A INPUT -s dns-server1 -j ACCEPT
iptables -A INPUT -s dns-server2 -j ACCEPT
iptables -A INPUT -s firewall -d remote-syslogger -p udp --destination-port 514 -j ACCEPT
iptables -A INPUT -s router -d remote-syslogger -p udp --destination-port 514 -j ACCEPT
iptables -A INPUT -s w2k-honeypot -d remote-syslogger -p udp --destination-port 514 -j ACCEPT
iptables -A INPUT -s linux-honeypot -d remote-syslogger -p udp --destination-port 514 -j ACCEPT
iptables -A INPUT -s ids -d remote-syslogger -p udp --destination-port 514 -j ACCEPT
iptables -A INPUT -j DROP
Links

The Honeynet Project Website
http://www.honeynet.org

Red Hat Website
http://www.redhat.com

Windows 2000 Website

Iptables/Netfilter Website
http://www.iptables.org

Swatch Website
http://www.oit.ucsb.edu/~eta/swatch/

Snort Website
http://www.snort.org

The Linux Documentation Project Website
http://www.tldp.org/
Appendix

Here is a brief description of all of the various exploits included in the EXPLOITS.tar.gz package.

In the Windows Directory:

------------------------------------------------------------------------------
UnicodeIISexploit.tar.gz:
Unicode Exploit, used to run commands on IIS 4.0 or IIS 5.0 web server. Script enclosed creates a backdoor by executing commands to anonymously ftp to your own ftp server, download netcat, and run a batch file containing a command for netcat to listen on a port and upon connection run 'cmd.exe' This is a remote exploit that does not give you administrative privileges.
------------------------------------------------------------------------------

IISexploit2.tar.gz

This exploit is a Windows 2000 ISAPI printer buffter overflow that overflows ISAPI printer buffer and spawns a reverse cmd.exe shell. This is a remote exploit that gives administrator access.

------------------------------------------------------------------------------

In the Linux Directory:

------------------------------------------------------------------------------
ptrace_exploit.tar.gz

This is a local exploit for execvc/ptrace race condition in Linux kernels up to 2.4.9 that sometimes will offer root access.

------------------------------------------------------------------------------
rwhois_exploit.tar.gz

This is a remote exploit against rwhoisd up to version 1.5.5 that works on Linux and FreeBSD that spawns root shell.

wuftpd_exploit.tar.gz

This is a remote exploit against wuftpd 2.6.0(1) on RedHat 6.2. This exploit also gives a root shell.

sxp.tar.gz

Local sendmail exploit against Sendmail versions below 8.11.6 that gives you root privileges.

leshka.tar.gz

A second local sendmail exploit against Sendmail versions 8.7-8.8.2 for Linux and FreeBSD.