Global Information Assurance Certification Paper

Remote Logging and Monitoring of a NIDS/Firewall

By

Brandon Williams

Version Number 1.4b

For

GSEC Practical Assignment

August 8, 2003

Abstract

This document describes one possible configuration for providing log collection and monitoring for a remote NIDS/firewall system that is available through an Internet reachable, static IP address. This might be a system you have installed as a consultant or perhaps this is something you have provided for a friend with a SOHO type of connection to the Internet. In either case, you get to be the watchful eye, constantly monitoring the health of the system and the type of traffic that is hitting it from the Internet. We will not delve into the particulars of the firewall setup but we will cover the setup of a NIDS and a secure remote logging solution. This will be the key to detecting attacks against this remote system.

Introduction

Any system with a connection to the Internet, should have some level of firewall protection locally, or on a separate machine, or on both. The most robust firewalls are typically on separate systems that will take the brunt of attacks from hackers on the Internet, shielding your personal or server systems from being direct targets. This type of firewall will be the base system for this paper. SOHO systems may not have any valuable data for most attackers but they are as vulnerable as corporate systems for those who are just interested in a challenge. Therefore, you must attempt to be as vigilant in protecting small networks as you would large networks.

In one possible scenario as a security specialist, you may be asked, or may wish to pursue on your own, the setup of a multi-layered defense strategy for someone's network. To be as impenetrable as possible, you may want to install a firewall that can automatically respond to attack situations, move critical log files to a safe location, and provide continual system monitoring. Thus, you will have created a solution that can be widely deployed in locations anywhere you wish, and still have up-to-date information on the health of each system in the comfort and security of your own network.

In general terms, the best way to ensure that your system can respond to, and alert you to, network attacks from the Internet, is to have a high quality NIDS, or Network Intrusion Detection System. There are many options here, but a few NIDS software programs stand out as industry standards with proven track records. These programs, often coupled with tools for automated responses to attacks, are usually loaded onto a robust operating system, such as one of the many Linux variants. Once running, they will monitor all network traffic and compare packet contents against a list of known attack signatures. Anything that matches a known attack signature, triggers a series of configured events; some level of logging at the very least. That brings us to the next tool category; logging mechanisms.

It is often preferred to have important log information sent to a dedicated log server rather than residing solely on the Internet connected NIDS/firewall. The log server, commonly referred to as a loghost, is typically in a location that is less accessible for remote attacks. In fact, the configuration used in this document expects the loghost to be residing behind an existing, local firewall, preferably in a DMZ or private LAN that relies on NAT, or Network Address Translation. In addition to collecting log information, it is often helpful to get near real-time updates of the most important log information, and other remote system information, in a centralized, easy to read console. That brings us to the final tool category in our kit; remote monitoring.

Many software packages exist for the specific purpose of remote system monitoring, and some of the better ones provide alerts based on defined resource levels and on key words collected from log information. We will discuss some options I have found that allow for fairly robust functionality in free (GPL licensed) software. This package will run on the loghost without performance issues and therefore saves us from adding another system to the mix.

This document will provide a concise guide, mostly step-by-step, for which packages to install, in which order they should be installed, and how they should be configured and started. First things first, let's take a look at the system layout we need on this network.

System Specifications:

Pentium class systems should suffice for both the remote NIDS/firewall box and the local loghost/monitoring box. In fact, a mere 200MHz Pentium CPU will fit the bill in both cases, so you don't have to break the bank to build these units. Just get the least expensive, but reliable, system you can find. Consider the following as minimum hardware requirements:

- CPU – Pentium class 200Mhz
- RAM – 64Mb
- Hard Drive – 4Gb
- Network Card(s) – 100Mb Ethernet (2-3 in the NIDS/firewall, depending on whether you also have a DMZ network, and 1 in the loghost)

Minimum software requirements:

- OS – Linux/BSD variant with the latest patches.
- Firewall – Recent packet filtering variety; stateful inspection if possible, like iptables or ipfilter.
- Remote Administration – Encrypted traffic through SSH (we will not rely upon web-based interfaces for administration, even encrypted ones; it opens an unnecessary and potentially vulnerable service port).

- NIDS Software – An industry standard, open source variety is almost always a good choice.
- Logging Service – A daemon that can mirror the functionality of Syslog with the reliability of TCP.
- Tunneling Software – The ability to securely tunnel logging information across the Internet is what makes remote logging feasible.
- Remote System Monitoring – Auto updating system monitoring software that shows both system health and alerts the administrator to important log information.

General Recommendations

These systems will both need to be rather scaled-down in their functionality and must be well hardened. The system that is currently just a firewall presumably has a configured packet filter running and not much else. All unnecessary services must be turned off and made to stay that way upon reboots. Any unreasonable level of vulnerability left open for hackers or script kiddies will potentially destroy this security model and may lead to a compromise of the network. If you are building on top of a RedHat Linux platform, and if OS hardening is not already completed, I highly recommend running Bastille Linux on both systems to automate the hardening process http://www.bastille-linux.org/. This product has the additional benefit during installation of providing the administrator some basic instruction on the security issues being addressed by the implementation [1]. In particular, if you will require the NIDS/firewall system to provide DNS services (not covered by this document), Bastille Linux will help configure the service to run in a chroot environment for added security.

SSH is the preferred way to ensure authentication, encryption, and integrity for remote administration commands and file transfers [2]. The open source version, OpenSSH, will be available by default on virtually every Linux/BSD distribution. Ensure that SSH protocol version 2 is the only configured version in your sshd configuration; it's more secure than version 1 and provides some additional features. It's not as pretty as a web-based administration GUI, but it's a lot more bulletproof and a bit more efficient if the majority of your work is done on the command line. The only modification regarding this software is that we will relocate its default service port to another high port number (a little security through obscurity), perhaps port 22222. Many port scanners and vulnerability testers will default to only testing a certain number of port numbers, rather than all 65,535. This movement to a non-standard high port number may only provide a little stealth from the laziest of script kiddies, but every little bit helps.

We will also ensure that TCP Wrapper is configured on the system. This software provides a simple access control list to protect against unwanted connections to inetd-launched services or others services configured to work through its ACL [3]. You get to specify the particular IP's or subnets that you want to allow into your systems through the available service ports. All major Linux distributions will have TCP Wrapper available and OpenSSH is usually setup to work through its ACL. The only requirement

for its activation is that the hosts.deny and hosts.allow must be configured to some degree. You can get very granular in that configuration, specifying different allowances for different services, but if nothing else, you will want to ensure a global level of control over all services. The configurations may contain the following:

sshd_config:

```
Port 22222
Protocol 2
```

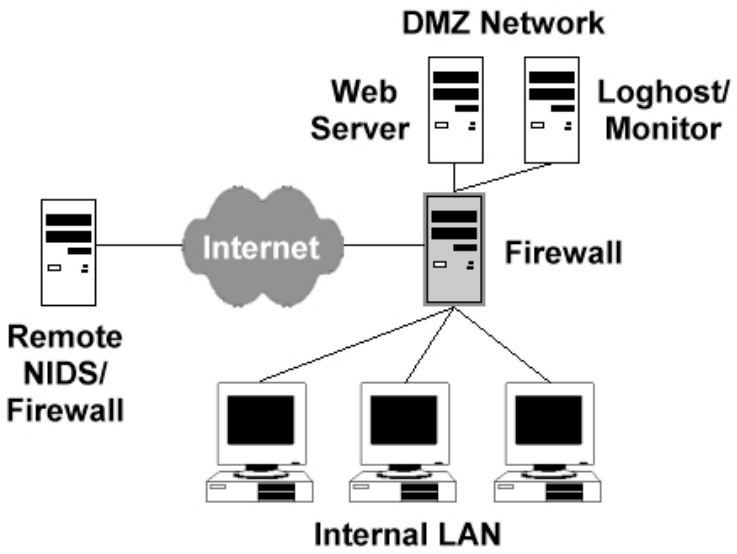
hosts.allow:

```
ALL: 127.0.0.1 [loghost IP] [other administration IP sources]
```

hosts.deny:

```
ALL: ALL
```

The overall structure of the network may look something like the following:



Now comes the specific configuration, starting with our remote logging solution.

Syslog-ng

Syslog Next Generation - TCP-based syslog replacement for local and remote logging

Product Source: BalaBit IT Ltd. – http://www.balabit.com
Purpose: Allows logging to traverse a TCP-based encrypted tunnel
Setup Requirements: Standard Linux/BSD load, root-level access

Most logging on UNIX-type systems, and even some Windows systems, is handled through the venerable syslog program. This solid package can provide local and remote logging. Since it functions with the UDP protocol and doesn't have security built-in, it is sometimes not the best choice for a highly secure environment. A secure logging setup should ideally have the following [4]:

- Logs sent to a remote logging host
- Logs sent to the log host encrypted
- The log host is a very secure box
- Logs on the log host are sorted by machine
- Logs are archived and stored to save disk space
- Logs are checked by a script and important entries are sent to an administrator

To fill these requirements, a favorite choice is Syslog-ng. This product functions under the TCP protocol, providing much better reliability, and at the same time allows for the use of encrypted tunneling mechanisms that are typically only available for TCP. Syslog-ng will provide this functionality and is just as solid as Syslog. We will keep a minimum of local logging for our attack blocking program to utilize (we will discuss this later). Complete logs amongst all of the enabled logging facilities, will be sent to the loghost.

Installation:

These instructions are assuming that you are presently running Syslog on a RedHat Linux platform and wish to compile the programs yourself [4]. Most steps should be the same for any Linux distribution and possibly many BSD distributions. The version used in this particular installation was syslog-ng-1.5.21. RPM versions are also available and will complete the following steps. They can be found at http://www.rpmseek.com.

1. Configure syslog to not boot at startup to avoid a conflict
   # chkconfig --level 2345 syslog off
2. Install dependencies: libol
   Libol can be acquired from the syslog-ng web page:
   http://www.balabit.com/products/syslog_ng/ or can be installed by rpm. The

version used in this particular installation was libol-0.2.21-1 and libol-devel-0.2.21-1

3. Download and unzip the source of syslog-ng
   - Edit "src/syslog-ng.h".
     Change the line that may look like:
     ```
     #define PATH_SYSLOG_NG_CONF /etc/syslog-ng/syslog-ng.conf
     ```
     to
     ```
     #define PATH_SYSLOG_NG_CONF /etc/hidden/secret.conf
     ```
     or some other hard to guess name. Either way, this file will be replaced with the client or server configuration file detailed on the following pages.
   - The path to the configuration file is changed because many crackers might look for the default log configuration files to find out if you are remote logging.
4. Install syslog-ng
   Do a **configure, make, make install** or a RPM install if applicable
5. Install config scripts and configure system
   # cp contrib/init.d.RedHat /etc/rc.d/init.d/syslog-ng
   Add the following to "/etc/rc.d/init.d/syslog-ng" in the commented out section at the top:
   ```
   # chkconfig: 2345 12 88
   # description: blah
   ```
   This is needed for chkconfig to recognize the script
   # cp contrib/syslog-ng.conf.RedHat /etc/hidden/secret.conf
   # chkconfig --level 2345 syslog-ng on
   # chmod 755 /etc/rc.d/init.d/syslog-ng
6. Test initial configuration
   # /etc/rc.d/init.d/syslog stop
   # /etc/rc.d/init.d/syslog-ng start

In theory, this should work on most platforms. I found this process to not work as flawlessly as the RPM installation however. The actual compilation of the syslog-ng program was fine but the placement of files and the structure of the rc script caused problems. Since I had also done a RPM installation on a nearby RedHat Linux test platform, I was able to utilize the customized script generated from its installation on the real server where I performed the above compilation. This script, with any necessary changes to reflect actual file paths, should work immediately as a replacement; if you find the one mentioned above to be faulty. It should look like the following:

```
#!/bin/bash
#
# syslog-ng        Starts syslog-ng.
#
#
# chkconfig: 2345 12 88
# description: Syslog is the facility by which many daemons use to log \
# messages to various system log files.  It is a good idea to always \
# run syslog.

# Source function library.
. /etc/init.d/functions
```

```
[ -f /usr/local/sbin/syslog-ng ] || exit 0

# Source config
if [ -f /etc/sysconfig/syslog-ng ] ; then
        . /etc/sysconfig/syslog-ng
else
        SYSLOGD_OPTIONS=""
fi

RETVAL=0

umask 077

start() {
        echo -n $"Starting system logger: "
        daemon /usr/local/sbin/syslog-ng $SYSLOGD_OPTIONS -f /etc/hidden/secret.conf
        RETVAL=$?
        echo
        [ $RETVAL -eq 0 ] && touch /var/lock/subsys/syslog-ng
        return $RETVAL
}

stop() {
        echo -n $"Shutting down system logger: "
        killproc syslog-ng
        RETVAL=$?
        echo
        [ $RETVAL -eq 0 ] && rm -f /var/lock/subsys/syslog-ng
        return $RETVAL
}

reload() {
        echo -n "Reloading syslog service: "
        killproc /usr/local/sbin/syslog-ng -HUP
        RETVAL=$?
        echo
        return $RETVAL
}

rhstatus() {
        status syslog-ng
}

restart() {
        stop
        start
}

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

```
        ;;
  condrestart)
        [ -f /var/lock/subsys/syslog-ng ] && restart || :
        ;;
  *)
        echo $"Usage: $0 {start|stop|status|restart|reload|condrestart}"
        exit 1
esac

exit $?
```

Configuration:

Example client configuration file to replace secret.conf

```
options { use_fqdn(yes); keep_hostname(yes); use_dns(yes); long_hostnames(off);
sync(3); log_fifo_size(300); };

#
# This is the default behavior of sysklogd package
# Logs may come from unix stream, but not from another machine.
#
# In Linux you will need the following
#
source src { unix-stream("/dev/log"); internal(); };
#
# In BSD you will need the following
#
# source src {
#         unix-dgram("/var/run/log");
#         file("/dev/klog");
#         internal();
#         udp();
#   };
#
# After that set destinations.

# First some standard logfile
#
destination lpr { file("/var/log/lpr.log"); };
destination mail { file("/var/log/mail.log"); };

# Some `catch-all' logfiles.
#
destination messages { file("/var/log/messages"); };

# The root's console.
#
destination console { usertty("root"); };

# Virtual console.
#
# In Linux you will need the following
#
destination console_all { file("/dev/tty8"); };
#
# In BSD you will need the following
#
# destination console_all { file("/dev/console"); };
```

```
#
# Here's come the filter options. With this rules, we can set which
# message go where.

filter f_lpr { facility(lpr); };
filter f_mail { facility(mail); };
filter f_messages { level(info..emerg) and not facility(mail,lpr); };
filter f_emergency { level(emerg); };

#############################################################
# Comment out for no local logging at all
#
# log { source(src); filter(f_lpr); destination(lpr); };
# log { source(src); filter(f_mail); destination(mail); };
# Left on to ensure local /var/log/messages for Guardian.pl actions
log { source(src); filter(f_messages); destination(messages); };
# log { source(src); filter(f_emergency); destination(console); };

#############################################################
## set up logging to a loghost forwarded from localhost via stunnel
#
# In Linux you will need the following
#
destination loghost {tcp("127.0.0.1" port(514));};
#
# In BSD you will need the following
#
# destination loghost {tcp("127.0.0.1" port(514)); sync(0) };
#
# send everything to loghost, too
log { source(src); destination(loghost); };
#############################################################
```

Example server configuration file to replace secret.conf on the loghost

```
options { use_fqdn(yes); keep_hostname(yes); use_dns(yes); long_hostnames(off);
sync(3); log_fifo_size(300); };

# network logs come from the local network and from stunnel on 127.0.0.1
source src { unix-stream("/dev/log"); internal(); };
source remote { tcp(ip("127.0.0.1") port(514) keep-alive(yes)); };

#
# This is the default behavior of sysklogd package
# Logs may come from unix stream, but not from another machine.
#
source src { unix-stream("/dev/log"); internal(); };

# After that set destinations.

# First some standard logfile
#
destination lpr { file("/var/log/lpr.log"); };
destination mail { file("/var/log/mail.log"); };

# Some `catch-all' logfiles.
#
destination messages { file("/var/log/messages"); };
```

```
# The root's console.
#
destination console { usertty("root"); };

# Virtual console.
#
destination console_all { file("/dev/tty8"); };

# Here's come the filter options. With this rules, we can set which
# message go where.

filter f_lpr { facility(lpr); };
filter f_mail { facility(mail); };
filter f_messages { level(info..emerg) and not facility(mail,lpr); };
filter f_emergency { level(emerg); };

#############################################################

log { source(src); filter(f_lpr); destination(lpr); };
log { source(src); filter(f_mail); destination(mail); };
log { source(src); filter(f_messages); destination(messages); };
log { source(src); filter(f_emergency); destination(console); };


#############################################################
  # automatic host sorting

  # set it up
  destination hosts { file("/var/log/HOSTS/$HOST/$FACILITY" owner(root) group(root)
perm(0600) dir_perm(0700) create_dirs(yes)); };

  # log by host
  log { source(remote); destination(hosts); };
#############################################################
```

7. (Optional if not loaded via RPM) An adjusted logrotate entry is needed. Remove "syslog" from "/etc/logrotate.d/" and replace with the script listed at this link, with appropriate path and file settings: "syslog-ng". This provides an archive going back as many versions as you wish to configure, often with weekly rotations, and keeps the log directory from filling up.
8. Add entries into "/etc/services", both port numbers are arbitrary, just make sure they match between the syslog-ng.conf file and the stunnel command as it will be utilized.

    - syslog-ng 514/tcp (will most likely be assigned to “shell” in RedHat)
    - syslog-ngs 5140/tcp

* Ensure that 127.0.0.1 is in the TCPWrapper /etc/hosts.allow file or no connections can be made to stunnel and that this traffic is allowed by the firewall.

Now that syslog-ng is available to control our logging functions, we need to fulfill the requirement to send the information in an encrypted form. This will provide confidentiality and integrity of the data. Enter, Stunnel.

Stunnel

Provides SSL encryption of TCP protocols that do not contain their own encryption mechanisms

Product Source: Michal Trojnara – http://www.stunnel.org
Purpose: Allows syslog-ng logging data to safely traverse public connections
Setup Requirements: Standard Linux/BSD load, root-level access

To enable remote logging over a public connection, some method of data protection must be implemented. It also doesn't hurt to provide this functionality between systems on related networks however, just in case someone manages to compromise a box on your inside network and starts sniffing network traffic. If the log information can provide any useful clues to an attacker as to your network structure, services you are running, or whether the attacker has been detected, it's worth the little bit of extra effort to provide this protection. In the case of logging, Stunnel provides this capability by accepting connections bound for the localhost on a given port number and encrypting the data as it is sent to a predefined remote host and port number. The SSL cryptographic protocol used by Stunnel, is a widely accepted, industry standard protocol that ensures encryption of the entire communication, provides integrity protection for the data, and performs authentication between the systems [3].

Installation:

The version of Stunnel for this particular installation was stunnel-3.22 [4].

- Install dependencies: openssl and openssl-devel
- Install from rpm or compile.
- Download and unzip the source of Stunnel
- The source can be found at stunnel.org and packages at rpmseek.com.
- Install Stunnel: Do a **configure**, **make**, **make install** or a RPM installation in applicable.

Configuration:

1. Generate a server certificate if not completed during compilation

    - Needed for stunnel to start on the server.
    - Cd to /usr/share/ssl/certs and execute:

    # sh ./make-dummy-cert stunnel.pem

- Otherwise, if not installed via RPM:
- Go to the ssl certs directory, "/usr/share/ssl/certs" in redhat and execute:

  # openssl x509 -hash -noout -in stunnel.pem

- Point to the *.pem file form the server stunnel command

2. Run the following from a script, possibly "/etc/rc.d/rc.local". Make sure to replace xxx.xxx.xxx.xxx with the IP number of your loghost:

   - client: **# stunnel -c -d 127.0.0.1:syslog-ng -r xxx.xxx.xxx.xxx:syslog-ngs -N syslogng**
   - server: **# stunnel -p /usr/share/ssl/certs/stunnel.pem -d syslog-ngs -r 127.0.0.1:syslog-ng -N syslogng**
   - I actually found things to load more reliably with stunnel starting before syslog-ng. I recommend launching stunnel within the syslog-ng rc script as in the following which would reside on the server:

```
 . . .
start() {
        echo -n $"Starting system logger: "
       /usr/sbin/stunnel -p /usr/share/ssl/certs/stunnel.pem -d syslog-ngs -r
127.0.0.1:syslog-ng -N syslogng
        daemon /usr/local/sbin/syslog-ng $SYSLOGD_OPTIONS -f /etc/syslog-ng.conf
        RETVAL=$?
        echo
        [ $RETVAL -eq 0 ] && touch /var/lock/subsys/syslog-ng
        return $RETVAL
. . .
```

3. Ensure that your local firewall is setup to accept in the incoming traffic to port 5140 (or whatever port you decide to choose) and forward it to the loghost. The packet filter rules may look something like the following:

```
$IPTABLES -t nat -A PREROUTING -p tcp -d [local firewall IP] --dport 5140 -j DNAT --to
[loghost IP]:5140
$IPTABLES -t nat -A POSTROUTING -p tcp -d [loghost IP] --dport 5140 -s [loghost
subnet]/
255.255.255.0 -j SNAT --to [local firewall IP for loghost subnet]
```

Snort

Industry standard, open source Network Intrusion Detection System

Product Source: Martin Roesch – http://www.snort.org
Purpose: Real-time monitoring and logging of network attacks and probes
Setup Requirements: Standard Linux/BSD load, root-level access

Now that remote logging is available, let's start collecting information related to network attacks against our NIDS/firewall. Snort source can be downloaded from snort.org or a RPM version from rpmseek.com. A separate Snort rules package is available that is routinely updated and is the heart of Snort's content signature matching capability. The steps below were tested with version 1.9.0. You may have to get the libpq.so separately and add it to the system. An RPM form exists as part of the postgresql-libs which may be easy to find from rpmfind.net or rpmseek.com.

Installation:

For a source installation on the NIDS box, we will untar Snort and walk through the configuration [5].

 # pwd
/export/snort-1.9.0
 # ./configure && make && make install

Configuration:

We will want to make the following modifications to the Snort configuration file.

```
# you can specify the variable to be any IP address
# like this:
# var HOME_NET 192.168.0.1
# Or if you have a DMZ net included and would like to include the # subnets
var HOME_NET [192.168.0.0/24,192.168.1.0/24]
# Set up the external network addresses as well.
# A good start may be "any"
var EXTERNAL_NET any
# Path to your rules files
var RULE_PATH /etc/snort/rules
# Output plugins
output alert_syslog: LOG_AUTH LOG_INFO LOG_ALERT
# When vision18 rules from whitehat are renamed and placed in the # rule path
include $RULE_PATH/vision18.rules
```

Now we can initiate Snort logging to the local syslog facility by specifying "-s" on the command line:

 # adduser Snort
 # passwd Snort
 # ./snort –s –D –A full [–c /etc/snort/snort.conf] –d –D –e –u snort –g snort

(an optional –N will prevent the local /var/log/snort packet logging). This can be put in the rc script.

(The –u and –g specify running snort as the snort user rather than root. The /var/log/snort directory will need ownership changes.)

* This sequence should be put in place of the default command string in /etc/init.d/snortd when that is created by a RPM load.

Upon initiation of Snort, our Syslog-ng will now be trapping those alerts and sending them over the wire to the remote Syslog-ng server, the loghost. You will most likely find the non-verbose alerts in the /var/log/HOSTS/machine_name/auth logfile on the loghost.

Additional Snort rules:

To increase the number of signature rules available, it is possible to merge the vision series rules from http://www.whitehats.com/ids [6]. These are complete conf files. Interface variables as well as redundancies must be configured/deleted. Adding an "include" at the bottom of the snort.conf and renaming vision18.conf to vision18.rules should make it a bit easier and more consistent.

Rules that don't exist, don't apply, or appear to result in error messages when starting Snort, must be removed from the snort.conf or Snort will refuse to start.

Automation of Snort rule updates:

There are some nice utilities that assist in the update of Snort rules. Some can even run as daily cron jobs to ensure you always have the most recent rules, although that's not the recommended way given the importance of the information. One of them is Oinkmaster, a simple perl script that can retrieve updates straight from the snort.org web site [7]. You can find the latest version and documentation at http://www.algonet.se/~nitzer/oinkmaster/.

Guardian

Automatically updates firewall rules based on Snort alerts in log files

Product Source: Anthony Stevens – http://www.chaotic.org/guardian
Purpose: Auto blocking of attacks based on local log alerts
Setup Requirements: Standard Linux/BSD load, root-level access, perl

To auto block sources that have generated Snort alerts, you can use Guardian Active Response [8]. It is a perl script available from http://www.chaotic.org/guardian. It can be loaded at the end of the rc.local script. This archived program collection will need the following executables moved into /usr/local/bin or another directory in the system PATH:

guardian_block.sh, guardian_unblock.sh, and guardian.pl

The block and unblock scripts need to be verified for use with the right packet filter, some of which are case sensitive in the chain names. Most of the time, the sample scripts for a particular packet filter will have to be renamed to match the names given above.

Guardian.pl turns itself into a daemon when run. It uses /etc/guardian.conf, /etc/guardian.ignore, and /etc/guardian.target to determine its actions. Guardian.ignore contains host IP's that are ignored as far as alert generation; this should include your monitoring host/loghost or the firewall it resides behind. There may be a need to modify the alert word that Guardian looks for in the log file if you don't find it to work as expected. Look for "/snort/" in guardian.pl and replace it with the common word that is used under network attack in your particular system's "AlertFile". Guardian.target lists the IP's of the local interfaces.

Installation:

No special steps required. The scripts need to be available in the PATH environment setting and executable under perl.

Configuration:

The guardian.conf will need the following verified:

```
Interface     eth0 or ex0
#(If this is the outside interface)  It should match the interface in
/etc/init.d/snortd.
LogFile       /var/log/messages
```

```
AlertFile     /var/log/messages
TimeLimit     86400  (Time listed is in seconds, so this translates to 1 day)
```

Guardian.pl may require the interface discovery line to be changed from “FreeBSD” to “NetBSD” if installed on some BSD systems.

Big Sister

System monitoring through a local web/log host

Product Source: Thomas Aeby – http://bigsister.graeff.com
Purpose: Monitoring of networked systems, compilation of logged activity, generation of alarms
Setup Requirements: Standard Linux/BSD load, root-level access, perl

A valuable monitoring tool, that is a free alternative to Big Brother, another popular monitoring tool, is Big Sister which can be acquired from http://bigsister.graeff.com [9]. This can be the one-stop-shop for checking system health, process counts, and log information. It can also be configured to send alerts based upon various criteria. In this case, we will assume that your network does not require 24x7 support and that you can easily take a peek at the console screen and log alerts whenever you like. The following steps will help in configuration:

Ensure that port 1984 traffic is allowed and forwarded to the appropriate DMZ monitoring host/loghost through local firewall rules. Alternate ports can of course be used if necessary. It may appear like the following:

```
$IPTABLES -t nat -A PREROUTING -p tcp -d [Local firewall IP] --dport 1984 -j DNAT --to
[loghost IP]:1984
$IPTABLES -t nat -A POSTROUTING -p tcp -d [loghost IP] --dport 1984 -s [loghost
subnet]/255.255.255.0 -j SNAT --to [local firewall IP for loghost subnet]
```

The software should be installed to use the user and group bs:bs.

Installation (client):

Install routine should resemble the following:

 # ./configure –prefix /etc/bs
 # /usr/sbin/useradd –d /home/bs –s /bin/false –c “Big Sister” bs
 # make install-client
 # chown –R /etc/bs bs:apache *

Configuration (client):

On the client side, you will need the base software and the agent software. The file /etc/bigsister/uxmon-net is the only one that needs modification to point to the real Big Sister server rather than localhost and to specify which tests will be run. Additional specific syslog tests, which will help with attack alert monitoring, will be in Big Sister’s etc/syslog config file, usually in /usr/share/bigsister/etc. Make sure the hostname of the localhost and the remote host resolves in the local /etc/hosts.

Modify /etc/bigsister/uxmon-net (or /etc/bs/adm/uxmon-net in the above install settings) to include the following (BSD systems may not function with any of the disk related settings and may need to be removed):

```
localhost     load diskload memory cpuload network \
              fs=all(^%-10%),all-ufs(6%-10%),all-ext2(6%-10%) diskfree \
              procs=init(1-1),stunnel(1-2),syslog-ng(1-1),sshd(1-1),\
              perl(2-3),snort(1-1) procs \
              frequency=1 syslog
              [IP address of monitoring host]   bsdisplay
```

Modify /usr/share/bigsister/etc/syslog (or /etc/bs/etc/syslog in the above install settings) to include the following:

```
.snort([^,]+)       yellow   1    Attack detected by snort: $1
```

Now launch the client with:

 # /etc/bs/bin/bb_start start

Installation (server):

Install routine should resemble the following:

 # ./configure –prefix /etc/bs
 # /usr/sbin/useradd –d /home/bs –s /bin/false –c "Big Sister" bs
 # make install-server
 # chown –R /etc/bs bs:apache *

Configuration (server):

The server side uses the base software plus the server package and most edits required exist in the /etc/bigsister/bb-display.cfg file or alternately under /etc/bs/adm/bb-display.cfg with the above installation parameters. The first thing you will probably want to do is change the %skin variables, which default to "static_lamps structured_bg" to "bigbro13" to get the look made popular by Big Brother.

You will also need to make a symlink, preferably called "bs" in your main web directory to point to the Big Sister HTML files, probably in /var/lib/bigsister/www or in /etc/bs/www.

The cgi's may end up in /usr/share/bigsister/cgi or under /etc/bs/cgi and may need to be relocated into a cgi-bin directory as allowed by the script aliases defined in Apache's httpd.conf, or through whatever web server configuration you are running locally. In my case, I copied the files to the existing /var/www/cgi-bin directory and added another script alias to cover the /cgi prefix expected by Big Sister. It should look like this:

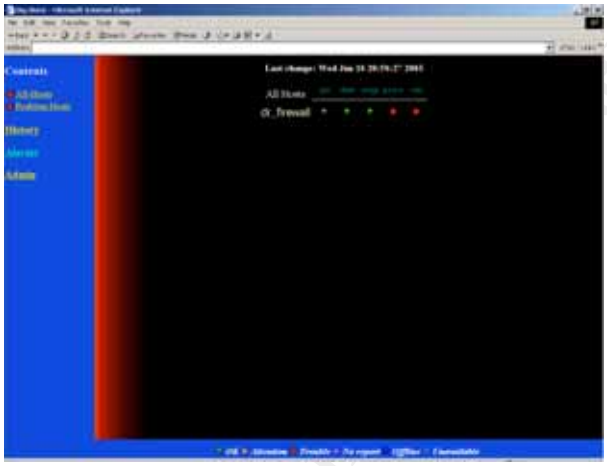
```
ScriptAlias /cgi/ "/var/www/cgi-bin/"
```

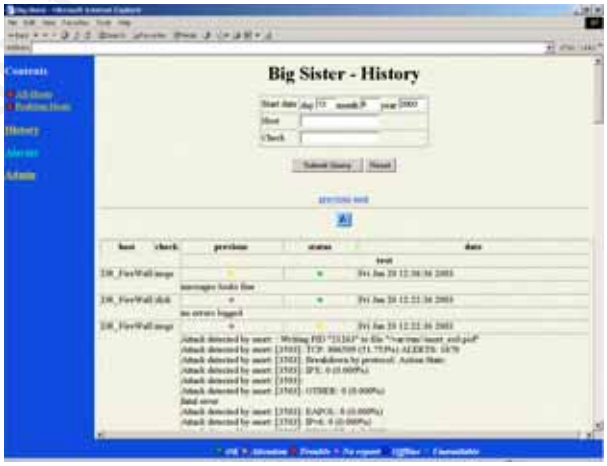
The binary to start may end up as /usr/share/bigsister/bin/bb_start or /etc/bs/bin/bb_start. A rc script may be better:

 # cat /etc/init.d/bigsister

```
#!/bin/sh
#
# chkconfig: 2345 98 99
# description: Big Sister System and Network Monitor
#
# Wrapper script for bb_start: the init directories should only
# contain shell scripts, no perl scripts ...

bs_instdir=/usr/share/bigsister

[ -x "$bs_instdir/bin/bb_start" ] || exit 0

exec $bs_instdir/bin/bb_start "$@"
```

The end result should look something like this:





Conclusion

With the simple tools presented in this document, you should be able to provide a good, lightweight, secure environment that routinely provides you with valuable system and IDS information from your remote installations. Remote locations with small networks will gain an added level of protection from outsiders, using a methodology that roughly mirrors many large network installations. You will gain the advantage of being able to exercise some level of immediate control and response to attacks. The systems themselves can be built from inexpensive PC's and free, GPL licensed, software.

Incidentally, this configuration has been tested on both Linux and BSD platforms with only minor changes required between the two.

Focus on SOHO and small school networks has been somewhat lacking until recently, so there still may be a sizeable need, and market, for the deployment of solutions such as the one presented in this paper. With 24x7 broadband Internet connections spreading throughout the country at an increasing rate, but very few people well aware of the potential security consequences, security experts may have a window of opportunity for providing powerful network protection and monitoring solutions for the ever expanding Internet community.

References:

1. Lasser, J., Beale, J. "Bastille Linux". 2003. URL: http://www.bastille-linux.org/

2. Barrett, D.J., and Silverman, R.E. SSH The Secure Shell: The Definitive Guide, O'Reilly & Associates, Inc., 2001, p. 4.

3. Zwicky, E.D., Cooper, S., and Chapman, B.D. Building Internet Firewalls (Second Edition), O'Reilly & Associates, Inc., 2000.

4. Zerr, Jeremy. "Secure Remote Logging with syslog-ng and stunnel HOWTO". URL: http://venus.ece.ndsu.nodak.edu/~jezerr/linux/secure-remote-logging.html

5. Hines, Eric "Loki". "Flying Pigs: Snorting Next Generation Secure Remote Log Servers over TCP". 29 May 2002. URL: http://www.linuxsecurity.com/feature_stories/snortlog-part1.html

6. Whitehats, Inc., 2001. URL: http://www.whitehats.com/ids

7. Oinkmaster. Written by Andreas Östling. 2003. URL: http://www.algonet.se/~nitzer/oinkmaster/

8. Stevens, Anthony. "Guardian Active Response for Snort". 26 March 2002. URL: http://www.chaotic.org/guardian/

9. Aeby, Thomas. 2001. URL: http://bigsister.graeff.com/pdoc/CONFIG.html