Intrusion Detection on a Large Network

GIAC Security Essentials Certification (GSEC)

Practical Assignment

Version 1.4b, Option #1 (amended August 29, 2002)

Author: Jason Botwick

Submitted: February 23, 2004

Table of Contents

Introduction

Abstract

This paper will describe in detail the steps for setting up and managing an intrusion detection system across a large corporate network. It will begin with a discussion of the potential problems and benefits of the use of a NIDS on a large network. The basics of installing, configuring and implementing the necessary software on a hypothetical network will be covered. Additional steps to automate, fail-safe and secure the system will be described. Finally, a brief discussion of the potential difficulties of tuning a rule-based system such as Snort that is deployed on a large, heterogeneous, well-secured network will be presented.

Conventions

To provide accurate instructions as to how to install and deploy a hypothetical NIDS in a large network setting, very specific instructions will be given. Many of these steps involve executing commands on a computer terminal and verifying the results of this command or editing configuration files in a text editor. Such steps will be framed as follows, following the convention of Red Hat Linux online documentation (Red Hat, Inc.):

```
% here is a unix command
and here is the output of the unix command
and here is some more output
```

Intrusion Detection on a Large Network

Problems with NIDS and large networks

Design

The problems of the design of a NIDS on a large network largely boil down to the decisions that have to be made regarding the placement of the “sensors”, or the machines on which the actual intrusion detection engines run (Packer, 2001). A corporate network may have many different gateways to untrusted networks such as the Internet, or even to other partially trusted networks within the corporation. These gateways may be highly secured through use of devices such as firewalls, they may be partially or poorly secured, such as is the case for many local wireless networks (Borisov, Goldberg, and Wagner), or they may not be secured at all.

Decisions need to be made about where to place sensors in each of these scenarios. For example, if a network gateway is secured by a firewall, the sensor could be placed either outside or inside that barrier. Ideally, a sensor could be placed both outside and inside the firewall (Roamer), but this may not be possible due to corporate security policies or cost considerations.

Inside a firewall, or within a DMZ, an IDS is much less susceptible to attacks itself, but being behind a properly configured firewall can reduce the utility of such a system, since many kinds of attacks would be blocked before ever reaching the sensor.

Outside the perimeter, a detection engine can help to analyze patterns of network activity, providing useful information for configuration of firewalls and proxies. However, any machine placed outside the protection of a firewall is much more vulnerable to attack.

In many very large companies, responsibility for administering networks is divided among departments of divisions. This results in a situation where a company is composed of several quite large networks connected to one another through gateways, as opposed to one huge, centrally administered network. Given the size of the networks involved, it is frequently impossible for the security analyst to reliably determine how well secured these sibling networks are, yet often the gateway from one department’s network to another’s is completely unsecured. Such a gateway is a prime candidate for an intrusion detection sensor, since the data passing through this gateway must be considered unfiltered, and thus should be monitored.

Though there are many sensible places to insert a NIDS sensor, the design problem is complicated by cost factors. Though some a NIDS can be installed at a very low cost by using open-source software, in a large network setting, there are still costs associated with purchasing, deploying and maintaining the hardware required. In a large network setting, there is likely to be very high bandwidth usage, especially at peak times during the workday, so a location or gateway may require several sensor machines arranged in some kind of load-balancing configuration to efficiently capture all incoming data (Desai).

The Pig That Cried Wolf

> “However effective they are, there's one thing that IDSes universally do well: generate data. Lots of it.”(Braue).

There is some debate about whether NIDS are worth the cost and effort required to deploy and maintain. Most of this debate centers on the costs and difficulty of analyzing the massive amount of data that such a system can generate. Of course, this problem is proportional to the size of the network, so security analysts charged with implementing a NIDS at large companies often find that they are quickly overwhelmed with the sheer amount of data thrown at them. In most cases, the majority of this data consists of false alerts, and it becomes a enormous undertaking to sift through all this data to find patterns of network activity that represent actual attacks against assets that are vulnerable to these attacks (indeed, the problem of maintaining an accurate topology of a large corporate network is a difficult task in and of itself). In fact, this problem is so daunting, that systems are often simply shut off within weeks of being activated (Braue).

“We’ve Been Hacked”

Many network intrusion detection systems advertise real-time detection of attacks. Theoretically this may be true for some systems, as they must be designed with an emphasis on performance to keep up with the large amount of data passing through a large network.

But even immediate detection of a threat is rarely any practical value if the response is not similarly as swift. And as attack vectors such as the recent spate of worms that attempted to exploit RPC/DCOM vulnerabilities, even a response time on the order of seconds or minutes may be to no avail. Realistically, any suspicious pattern of network activity must be analyzed and its gravity verified. Even for experienced security professionals, this takes time. Formulating and executing the appropriate response also takes time. Thus, though an attack may be detected and logged by intrusion detection software almost instantaneously, the time between an attack and a response would be on the order of hours, not seconds.

This problem is exacerbated in a large network environment for at least two reasons. As discussed above, the large amount of data generated by even a well-tuned NIDS makes the culling of real threats from the large amount of false alarms even more time-consuming.

Also, a large network is typically operated a large corporation. Rapid decision-making can be difficult to achieve in large corporate settings due to issues related to established processes and complicated or lengthy policies that must be adhered to. Further time can be wasted after a decision is made to respond to the threat because of these side effects of bureaucracy.

The end result is that the analyst is all too often left only with a data trail of evidence that an attack has occurred and the unpleasant responsibility of reporting this fact to superiors.

What is intrusion detection good for?

So why should a large organization spend the time and money to implement a NIDS? Intrusion detection is not a easily implemented panacea to protect corporate networks. But it can provide benefits:

Detection of Threats

Regardless of where sensors are placed with respect to devices such as firewalls and proxies, they can help security analysts to detect and analyze network-based attacks.

Inside perimeter firewalls, a pattern-based detection engine can detect backdoor or trojan activity as it makes its way out of the network, or the rare attack that does make it through the firewall.

Modern firewalls are highly effective at blocking attacks (Braue). However, attacks penetrating a firewall are still possible, and a sensor placed inside the firewall would be able to detect such attacks; and at the very least, such a system can help to verify the effectiveness of the firewall configuration (Fink et al. 2). Even if no attacks penetrate the corporate firewall, a sensor outside the firewall provides useful data for observing trends in network activity or attack patterns (Yegneswaran, Barford, and Ullrich 3). Further, IDS data can be useful to the firewall administrator in configuring the firewall itself in preparation for attacks directed at the corporate network.

There are other logical placements for NIDS sensors, such as within a DMZ, behind VPN gateways or at the entrance point to a subnet of particular interest, such as a cluster of servers (Ryon, Staggs, and Harlow 6).

Evidence

If an attack is successful, a properly deployed and configured intrusion detection system will likely have registered all of the network activity associated with the attack. Such forensic evidence is useful for securing the network against future attacks of this nature, and for tracing the origin of the attack. As privacy and data security concerns continue to be crystallized into legislation, such evidence has taken on heightened importance, since companies can be held liable for exposures caused by network-based attacks, even if they originate from outside the company (State of California).

Knowledge of Network

Intrusion detection engines are flexible network monitoring tools that can be configured to look for any kind of traffic. An often-overlooked feature of these engines is their ability to profile network usage patterns within the corporate network setting. This ability can be useful for detecting corporate policy violations, patterns of network activity related to illegal activity perpetrated by employees, or even simply time profiles of peak usage patterns, that can aid the network administrator in maintaining the health of the network.

Building The System

Design Requirements

To address the concerns presented above, a design is presented here is based on the actual design and implementation of an intrusion detection system by the author. Many of the characteristics of this actual network are duplicated on the hypothetical network used as the subject of this pseudo-case study. However, due to non-disclosure agreements and other company policies related to the maintenance of network security, implementation details of the actual network may be slightly altered or not disclosed at all.

A typical large corporate network has many points of entry that may be important to monitor for suspicious network traffic. As discussed above, some of these are quite well secured, but others may not be well secured, or they may not be secured at all. Thus, it is necessary to place detection engines at these gateways. Also, a central management console to monitor the sensors and collect data will be necessary.

Management Console

Each of the sensors could possibly generate a great deal of data. To ease management of this volume of data as well as the sensors themselves, a central computer will be placed inside the corporate network called the management console. The management console will have the following characteristics and functions:

Build and Deploy Sensor Software

The management console will be built with a full set of development tools, so that intrusion detection software can be built and deployed to the sensors. It would be undesirable to use the sensor machines themselves to build detection software, since this would require that a set of development software be installed on one or more sensors. This could be a security vulnerability, and it certainly increases the amount of software that must be updated as new versions are released. One simple way to maintain the security of systems is to reduce the amount of unneeded software installed on the system so as to reduce the amount of patch management required (Harper).

Monitoring of and Communication With The Sensors

The management console will monitor the status of sensor boxes that are placed at various locations on the network. It should have the ability to communicate securely with the sensors, which may or may not be within the corporate network and therefore may not be protected by firewalls, proxies, etc.

Data Repository

The management console will also serve as a database server that receives and stores data collected by the sensor machines.

Data Analysis and Presentation

Finally, the console will contain software that will act as an interface to the data stored in the database. This software should facilitate the viewing and analysis of data collected by the various sensors to selected personnel within the corporation.

Intrusion Detection

The NIDS software that is chosen must meet two criteria: It must not drop packets under network conditions where high throughput (>500 MB/s) is possible, and it must be able to perform stateful inspection of packets, so that suspicious activity can be tracked and analyzed.

Sensors

A large network intrusion detection system must be composed of several sensor machines placed at various points inside and outside the gateways of the network, since no one machine could span all these entry points. Further, it is unlikely that any single machine or detection software could reliably process all incoming packets without becoming overloaded.

Configuration

The sensor machines should be configured with a minimal operating system with only the software necessary to run the detection software and to communicate securely with the management console.

Security

Since some of these sensors will be placed outside the protection of the corporation's hardware-based security systems such as firewalls and proxies, it is necessary to reduce the number of possible security vulnerabilities as much as possible by reducing the amount of software installed. Also, each sensor machine should be "locked down" to as great a degree as possible, by limiting the number of open TCP and UDP ports such that only secure, encrypted communication can take place between the sensor and any other machine.

Placement

Our hypothetical network is a simplified set of connections and machines characteristic of many large networks. The hypothetical network under our control would be a circumscribed list of domains, analogous to a department within a larger company. These domains include the department network as a whole, as separated from the public internet and the remainder of the corporate network. A server cluster within the department contains servers that serve the department, but do not interface with the public internet or the corporate network directly, but instead through a router. Thus, the network consists of several gateways of interest.

The gateway between the public internet and the corporate network as a whole is typically guarded by a firewall. At this gateway, NIDS sensors would ideally be placed both inside and outside the firewall. A sensor placed outside the firewall would serve to monitor any and all traffic directed at the corporate network. Assuming that the firewall is adequately configured, many attacks will not ever reach past the firewall. However, the information gathered by this outer sensor can help network administrators and security analysts to anticipate attack trends and configure firewalls and other security devices.

A sensor placed inside the firewall may be the most important sensor in a distributed NIDS (Packer, Staggs and Harlow). This is the sensor that will be depended upon to accurate detect and report attacks that do reach through the firewall. For similar reasons, a sensor should be placed within the DMZ originating from the outer firewall.

Assuming that the server clusters contain more important data than general-use workstations, a NIDS sensor should be placed inside the router that switches traffic from the rest of the corporate network to individual servers within the cluster. This NIDS provides something of a level of redundancy, but also may report on attacks launched against the most critical network nodes from machines within the corporate network, attack vectors that may have slipped past a firewall, or against which firewalls are not always helpful, such as email-based worms.

Finally, it is always desirable to place security devices such as firewalls and NIDS sensors at the juncture between the managed network of a department and the remainder of the corporate network; since the security level of this part of the network is unknown, it should be treated almost as if it is part of the public internet.

Choice of Tools

Tools were chosen based on the obvious criterion of being able to accurately report suspicious network activity patterns. Beyond that, cost was a primary factor. Given the continuing debate over the efficacy of NIDS, especially those deployed on large corporate networks, it makes sense to avoid spending a great deal of money on such a system. Thus, open source software tools were used to build the system.

Operating System

The operating system chosen for each of the machines was Red Hat Linux. This choice was made for several reasons. Firstly, there are ample internet resources available to aid in configuring and securing the operating system itself (Red hat). Similarly, there are also many resources available that discuss, or even enumerate the steps necessary for building a NIDS on this distribution of Linux (Harper). Finally, among open source operating systems, the use of Red Hat Linux in a corporate setting has become quite widely accepted (Ge, Scott, VanderWiele).

Development Tools

Additionally, an advanced set of development tools necessary for building other open source software and developing others is included as an optional package with the Red Hat distribution.

Software

Although virtually all of the tools listed below are available as packages with the Red Hat Linux distribution, it was decided to build all of the packages from scratch in order to be able to use the most recent (and therefore presumably most well-patched and secure) versions of these tools. Also, building the packages individually allows rapid deployment when a new patch for any of the components is released.

Communication Between Console and Sensor

Secure Shell (ssh) is also included with most Linux distributions, and is a highly secure, flexible and reliable method of communicating between networked machines, since it encrypts all communications over a channel, allowing remote login to the sensors for maintenance purposes, secure transfer of updated executables or configuration files, and simple data transmission through tunneling from the sensor to the management console (Atcheson).

Intrusion Detection

Snort is a well-supported, robust tool for intrusion detection. It performs exceptionally well under load given sufficient hardware to run on (Antonatos et al.), and is highly configurable. At this writing, the current version of Snort is 2.0.5, and this version will be used in this hypothetical study. The latest release of Snort can be downloaded at the following URL:

http://www.snort.org/dl

Data Repository and Management

MySQL

MySQL is a well-supported, robust database management system with excellent performance characteristics, even when the volume of stored data becomes quite large. Also, Snort can be compiled with a plug-in that allows it to transmit data to a MySQL database. For this study, version 4.0.16 will be used. The most recent stable version of MySQL can be downloaded here:

http://www.mysql.com/downloads/index.html

Data Analysis and Presentation

In order to make intrusion detection data readily available to all security personnel as well as management, a primarily web-based system was designed, consisting of a web server, compiled with server-side scripting ability and an interface to the data generated by Snort.

Apache

Apache is the most widely used, stable and secure open source web server available (FreePhile). The version of Apache used for this study is 2.0.47. The most recent version can be downloaded here:

http://httpd.apache.org/download.cgi

PHP

Apache can be compiled with PHP, a server-side scripting tool that allows sophisticated, database-enabled web applications to be implemented. The version of PHP used for this study is 4.3.2. The most recent stable version can be obtained here:

http://www.php.net/downloads.php

ACID

ACID, or Analysis Console for Intrusion Databases, is a web-based front-end tool for analyzing the data generated by intrusion detection systems. The version used for this project is 0.9.6b23. The most recent version can be downloaded here:

http://acidlab.sourceforge.net/

Other Software

ACID requires two other packages in order to function. ADODB, or Active Data Objects Database, is an abstraction layer written using PHP for database access that ACID uses to communicate with MySQL (and many other databases). The version of ADODB used for this project is 3.30. The most recent version can be obtained here:

http://php.weblogs.com/ADODB

JPGraph is another PHP-based package that allows PHP applications to easily create graphics, such as graphs and charts to be displayed in a web browser. The version of JPGraph used for this project is 1.13. The most recent version can be downloaded here:

http://www.aditus.nu/jpgraph/

Infrastructure and other files

The following two libraries are required to build Snort and MySQL.

zLib

zlib is a library that provides functionality for algorithmic compression of files to save disk space. The version used in this study is 1.1.4. The most recent version is available for download here:

http://flow.dl.sourceforge.net/sourceforge/libpng/zlib-1.1.4.tar.gz

libpcap

libpcap is a library that is required by Snort in order to process packets from a network interface. For this project, the version used will be 0.7.2. The latest version can be downloaded here:

http://www.tcpdump.org/

Hardware

Another advantage of using open source software is that it permits flexibility in choosing hardware. In our case, we can use relatively pedestrian computers for both the management console and the sensor boxes. Dell Optiplex GX270s, built with 2.4 Ghz Pentium 4 processors, 512 MB of RAM, and 40 GB hard drives. These machines are otherwise configured with standard options, except that each sensor box will be augmented by three network interface cards (NICs). One of these cards is configured normally, and is connected to the corporate network behind any and all existing network perimeter security devices, such as firewalls and proxies.

The other two of these NICs are configured for "stealth" or promiscuous mode; that is, they are not bound to any network protocol stack, and thus have no IP address making them extremely difficult to detect on the network. This allows us to place sensor machines at unsecured locations inside or outside of our network and still be very confident that our sensors can record all passing network traffic without falling under attack themselves since they would be difficult to detect.

Note that a NIC in promiscuous mode does not mean that a sensor in this system is invulnerable to attack. Even a system is built and patched with the latest releases of the software suggested here, there have been cases where vulnerabilities in Snort-based NIDS have been discovered (Roberts), so it is still necessary to secure these systems to as a great a degree as possible.

The NICs used for the Snort sensors should be connected to the network using taps. Taps are chosen for connections over SPAN ports or placing the sensors inline for reasons of performance and ensuring the continuity of network operation. If an inline NIDS sensor fails, it can disrupt the operation of the network, while NIDS sensors connected via router SPAN ports have been known to miss packets due to hard throughput limits and/or affect network performance negatively under high loads (Cole et al. 758-9).

Finally, a Windows-based computer must be available to use as an interface to the management console. This machine only needs to have a web browser to serve this purpose, however additional rule-management software will also be installed here.

Building the System, Step-By-Step

There are many excellent guides available online that explain in detail how to build, configure and install a unix-based NIDS. Of particular interest is Patrick Harper's excellent "Snort Install Manual" (Harper), on which much of the following step-by-step material is based. However, Harper's manual explains how to build a single-sensor, self-contained intrusion detection system, where all the components are installed on one machine, as opposed to a distributed NIDS, which is what is presented here.

Operating System

Red Hat Linux version 9 will be used for the operating system. It will be installed as simply as possible, with only the necessary packages. Again, there are numerous resources that give step-by-step instructions for installing Linux, and even Red Hat Linux. Those will not be repeated here in their entirety; however, there are important choices that do need to be made when installing the operating system on the management console and on the sensor machines. Those steps will be described here.

Sensors

Basic Settings

For the beginning screens of the installer, most of the default choices should be selected. However, there are some exceptions:

- For "Install Type", choose "Custom", in order to have complete control over the packages installed.
- The network settings should **not** be configured with DHCP. Static IP addresses should be assigned to the sensors, because it makes configuration of the intrusion detection software more difficult.
- The "Firewall" settings should be set to "High", and there should be no "trusted devices" selected. Finally, the only ports allowed open should be for SSH.

Packages

When you choose to customize the packages installed with Linux, a screen will eventually be presented that allows you to choose which packages will be installed. These packages are categorized on this screen, and some of the notable categories and important choices are shown below. In some cases, certain packages are selected by default, and most of these need to be deselected. Again, for sensor machines, very few packages should be installed so that patch management is as easy as possible.

Desktops

No desktop environments should be installed.

Applications

An editor such as vi or emacs will come in handy. But no other applications in this section should be selected.

Server Section

No packages in this section should be selected.

Development

Obviously, no development tools should be installed on the sensors.

System

No extra system tools should be installed.

Miscellaneous:

No miscellaneous tools should be selected.

Post-Installation

User Accounts

Make sure to add a user account to use for logging into the box for general maintenance purposes, so that it is not necessary to log in directly as the root user.

Services

Any non-essential services should be disabled. In the case of a sensor machine, that means virtually all services should be disabled. In the case of the distribution installed for this study, the following non-essential services were among those enabled by default and needed to be disabled with the command-line tool chkconfig:

- apmd
- cups
- netfs
- nfslock
- pcmcia
- portmap

To turn these off, use chkconfig like so:

```
chkconfig –level 345 apmd off
```

SSH

Since SSH will be used as the exclusive method of communication between the sensor and console, it must be set up properly. This is a simple matter of modifying the configuration file for SSH, /etc/ssh/sshd_config. Make sure the following lines read as follows (Harper).

```
Protocol 2

PermitRootLogin no

PermitEmptyPasswords no
```

Management Console

The management console is built exactly the same as the sensors with the following exception:

Packages

Development Tools

Select all the development tools except for those related to desktop or XWindows environments.

Management Console Software

No additional software needs to be installed on the sensor machines—except for Snort itself; this will be covered in a later section. However, Snort and its required components will be built on the management console.

Note: Snort will be statically linked. This is in order to ease deployment of the of the application from the management console to the various sensors, since only one file needs to be deployed. Also, this provides an additional benefit in terms of security, since there is also only one file that can be compromised outside of the core operating system files (Hicks, Weirich, and Crary 1). Finally, in most cases statically linked executables exhibit faster performance than those that are dynamically linked (Levine 10).

Downloaded archive files for each tool or library should be downloaded to a central directory, such as /usr/local/src, to simplify development. All of the following instructions assume that the user is logged in as root or has "su'd" to root privilege, and that the current directory is the central source directory (/usr/local/src). As an additional note, these build instructions are adapted from Harper's comprehensive instructions (Harper, 2003), and also from the installation instructions included in each package. Finally, the results of each command are not printed here for the sake of clarity, as they can be voluminous.

Infrastructure

libpcap

```
export CFLAGS=-static
tar –xvzf libpcap-0.7.2.tar.gz
cd libpcap-0.7.2
./configure
make
make install
cd ..
```

zlib

```
export CFLAGS=-static
tar -xvzf zlib-1.1.4.tar.gz
cd zlib-1.1.4
./configure; make test
make install
cd ..
```

Apache and PHP:

Build Apache

```
tar -xvzf httpd-2.0.47.tar.gz
cd httpd_2.0.47
export CFLAGS=(blank)
./configure—enable-so –enable-layout=Red Hat
make
make install
```

Build PHP

```
tar -xvzf php-4.3.2.tar.gz
cd php-4.3.2
./configure --prefix=/var/www/php --with-
apxs2=/var/www/bin/apxs --with-config- filepath=/etc/php --
enable-sockets --with-mysql=/usr/local/mysql --with-
zlibdir=/usr/local --with-gd
make
make install
cp php.ini-dist /var/www/php/php.ini
```

Configure Apache with PHP

The following directives should be added to /etc/httpd/conf/httpd.conf:

```
LoadModule php4_module /usr/lib/apache/libphp4.so
AddType application/x-httpd-php .php
DirectoryIndex index.php index.html index.html.var
```

Configure Apache to run as a service/daemon

```
cp /var/www/sbin/apachectl /etc/init.d/httpd
cd /etc/rc3.d
ln -s ../init.d/httpd S85httpd
ln -s ../init.d/httpd K85httpd
cd /etc/rc5.d
ln -s ../init.d/httpd S85httpd
ln -s ../init.d/httpd K85httpd
```

Database

MySQL:

MySQL user and group

Create the user and group for MySQL with the following commands:

```
groupadd mysql

useradd -g mysql mysql
```

Install and configure MySQL.

```
tar –xvzf mysql-4.0.16.tar.gz

cd mysql-4.0.16

export CFLAGS=”-O3 –pipe –fomit-frame-pointer”

./configure –-prefix=/usr/local/mysql --with-mysqld-user=mysql
--without-debug --with-client-ldflags=-all-static --with-
mysqld-ldflags=-all-static –disable-shared --with-extra-
charsets=none --enable-assembler

make

make install

scripts/mysql_install_db

chown -R root /usr/local/mysql

mkdir /usr/local/mysql/var

chown -R mysql /usr/local/mysql/var

chgrp -R mysql /usr/local/mysql

cp support-files/my-medium.cnf /etc/my.cnf
```

Configure the system link loader

These lines must be added to /etc/ld.so.conf:

```
/usr/local/mysql/lib/mysql

/usr/local/lib to the /etc/ld.so.conf file.
```

After you add the lines, run

```
ldconfig –v
```

Configure MySQL to run as a service/daemon

```
cp support-files/mysql.server /etc/init.d/mysql

cd /etc/rc3.d

ln -s ../init.d/mysql S85mysql

ln -s ../init.d/mysql K85mysql

cd /etc/rc5.d

ln -s ../init.d/mysql S85mysql
ln -s ../init.d/mysql K85mysql
cd ../init.d
chmod 755 mysql
```

Setting up the snort and snort_archive databases in MySQL

Two databases should be created. The main one, which we will name “snort”, will be the database to which the Snort instances running on sensors will log data. The archive database, which will name “snort_archive”, will be used by ACID to save notable or interesting cases of suspicious network activity for further analysis.

Set database privileges

```
/usr/local/mysql/bin/mysql
mysql> SET PASSWORD FOR
root@127.0.0.1=PASSWORD('your complex password’);
>Query OK, 0 rows affected (0.25 sec)
mysql> create database snort;
>Query OK, 1 row affected (0.01 sec)
mysql> create database snort_archive;
>Query OK, 1 row affected (0.01 sec)
mysql> grant INSERT,SELECT on root.* to snort@127.0.0.1;
>Query OK, 0 rows affected (0.02 sec)
mysql> SET PASSWORD FOR
snort@127.0.0.1=PASSWORD('your complex password’);
>Query OK, 0 rows affected (0.25 sec)
mysql> grant CREATE, INSERT, SELECT, DELETE, UPDATE on snort.*
to snort@127.0.0.1;
                          >Query OK, 0 rows affected (0.02 sec)
mysql> grant CREATE, INSERT, SELECT, DELETE, UPDATE on snort.*
to root;
>Query OK, 0 rows affected (0.02 sec)
mysql> grant CREATE, INSERT, SELECT, DELETE, UPDATE on
snort_archive.* to snort@127.0.0.1;
>Query OK, 0 rows affected (0.02 sec)
mysql> grant CREATE, INSERT, SELECT, DELETE, UPDATE on
snort_archive.* to root;
>Query OK, 0 rows affected (0.02 sec)
mysql> exit
```

```
>Bye
```

Create database structure

In the Snort source directory, there is a directory called “contrib” with important scripts and tools that must be run. Navigate to that directory.

Main snort database

```
/usr/local/mysql/bin/mysql -u root -p < . /create_mysql snort
Enter password:
zcat snortdb-extra.gz |/usr/local/mysql/bin/mysql -p snort
Enter password:
```

snort_archive database

```
/usr/local/mysql/bin/mysql -u root -p < . /create_mysql snort
Enter password:
zcat snortdb-extra.gz |/usr/local/mysql/bin/mysql -p snort
Enter password:
```

phpMyAdmin

phpMyAdmin is a web-based interface to the MySQL DBMS. It will not be mentioned again in this document. It is only mentioned here as a recommended way to view and manage the state of the databases created for this project (phpMyAdmin).

Unpack and build

```
tar -xzvf phpMyAdmin-2.5.3-php.tar.gz
mv phpMyAdmin-2.5.3 /var/www/html/phpMyAdmin
```

Configure the Application

```
cd var/www/html/phpMyAdmin
```

Edit the file config.inc.php, changing lines:

```
$cfg['PmaAbsoluteUri'] = 'http://<ip of the console
box>/phpMyAdmin/';
$cfg['Servers'][$i]['controluser']   = 'pma';
$cfg['Servers'][$i]['controlpass']   = 'pmapass';
$cfg['Servers'][$i]['auth_type']     = 'http';
$cfg['Servers'][$i]['user']          = 'root';
$cfg['Servers'][$i]['password']      = '';
```

Database Permissions

From a MySQL interface, execute the following commands:

```
GRANT SELECT ON mysql.db TO 'pma'@'localhost';

GRANT SELECT ON mysql.host TO 'pma'@'localhost';

GRANT SELECT (Host, Db, User, Table_name, Table_priv,
Column_priv) ON mysql.tables_priv TO 'pma'@'localhost';
```

Set up the control user

```
GRANT SELECT (Host, User, Select_priv, Insert_priv,
Update_priv, Delete_priv, Create_priv, Drop_priv, Reload_priv,
Shutdown_priv, Process_priv, File_priv, Grant_priv,
References_priv, Index_priv, Alter_priv) ON mysql.user TO
'pma'@'localhost';
```

Test

```
Point a web browser to http://<ip of console>/phpMyAdmin
```

IDS

Snort

The sensor tools (Snort) will be built on the console box, because no development tools or supplemental libraries will be available on the sensor machines. Furthermore, Snort needs to be built as a static executable for the reasons discussed above. So Snort will be built on the console box, and then securely copied with its configuration/rules files to each sensor machine.

Setup user/group for snort

```
groupadd snort

useradd -g snort snort
```

Create necessary directories (STATIC BUILD)

```
mkdir /etc/snort

mkdir /var/log/snort
```

Build Snort

```
tar -xvzf snort-2.0.1.tar.gz

cd snort-2.0.1

export CFLAGS=-static

./configure—with-mysql=/usr/local/mysql

make

make install
```

Note on building Snort

Under some conditions, the following line may need to be changed in the Makefile generated by the configure command in the src directory (the first line is how it originally appears, the second is how it must be written in order for Snort to link statically.

```
LIBS =  -lz  -lpcap -lm -lnsl  -lmysqlclient
LIBS =  -lpcap -lm -lnsl  -lmysqlclient -lz
```

Rules and configuration

The rule and configuration files are copied from the source directory to a directory on the console that is analogous to where they will be located when copied to the sensor machines. Note that no Snort instance will run on the management console, and the primary reason for this step is to establish a "staging" environment of sorts for the rule and configuration files for later deployment.

```
cd rules
cp * /etc/snort
cd ../etc
cp snort.conf /etc/snort
cp *.config /etc/snort
chown snort /etc/snort/*
chgrp snort /etc/snort/*
```

snort.conf

The main configuration file snort.conf is now in /etc/snort. This version of the file will be used as a base configuration for all Snort instances to be run on the sensor machines, so any changes specific to the console environment or communication from sensor to console should be made to this copy:

```
var RULE_PATH /etc/snort/
output database: log, mysql, user=snort
password=your_complex_password dbname=snort host=127.0.0.1
```

Data Analysis Tools

JPGraph

Instructions taken from the original distribution (JPGraph):

```
cp jpgraph-1.13.tar.gz /var/www/html
cd /var/www/html
tar –xvzf jpgraph-1.13.tar.gz
rm –rf jpgraph-1.13.tar.gz
```

```
mv jpgraph-1.13 jpgraph
cd jpgraph
rm -rf README
rm -rf QPL.txt
```

ADODB:

Instructions taken from the original distribution (ADODB):

```
cp adodb370.tgz /var/www/html/
cd /var/www/html
tar -xvzf adodb370.tgz
rm –rf adodb370.tgz
mv adodb370 adodb
chmod –R 755 adodb
```

ACID

Instructions taken from the original distribution (ACID):

Installing

```
cp acid-0.9.6b23.tar.gz /var/www/html
cd /var/www/html
tar –xvzf acid-0.9.6b23.tar.gz
rm –f acid-0.9.6b23.tar.gz
```

Configuring the main ACID application

Edit the ACID configuration file

The configuration file is located in /var/www/html/acid and is called acid_conf.php file. Make sure the following lines look as follows:

```
$DBlib_path = "/var/www/html/adodb";
$DBtype = "mysql";
$alert_dbname = "snort";
$alert_host = "127.0.0.1";
$alert_port = "";
$alert_user = "snort";
$alert_password = "your_complex_password";
$archive_dbname = "snort_archive";
```

```
$archive_host = “127.0.0.1”;
$archive_port = “”;
$archive_user = “snort “;
$archive_password = “your_complex_password”;
$ChartLib_path = “/var/www/html/jpgraph/src”;
$chart_file_format = “png”;
```

Complete the building of the snort database

Navigate to the ACID home page, which should be something like http://<IP of management console>/acid/acid_main.php.

Click on the button that says “Setup Page”, then on the resulting page, click on the button that says “Create Acid AG”.

Create ACID tables for the snort databases

```
cd /var/www/html/acid
mysql –p snort < create_acid_tbls_mysql.sql
```

Configuring the archive ACID application

Edit the ACID configuration file

The configuration file is located in /var/www/html/acid_archive and is called acid_conf.php file. Change only the following line look as follows:

```
$alert_dbname = “snort_archive”;
```

Create pages to view the snort_archive data

```
cp -R /var/www/html/acid /var/www/html/acid_archive
```

Complete the building of the databases

Navigate to the ACID home page for the snort_archive database, which should be something like http://<IP of management console>/acid_archive/acid_main.php.

Click on the button that says “Setup Page”, then on the resulting page, click on the button that says “Create Acid AG”.

Create ACID tables for the snort_archive databases

```
cd /var/www/html/acid
mysql –p snort_archive < create_acid_tbls_mysql.sql
```

Securing the Web Interface

```
mkdir /var/www/passwords
```

/var/www/bin/htpasswd -c /var/www/passwords/passwords acid

(acid will be the username you will use to get into this directory, along with the password you choose)

It will ask you to enter the password you want for this user, this is what you will have to type when you want to view your acid page

Edit the httpd.conf and include the following under the section that starts with

```
</Directory>

<Directory "/var/www/html/acid">

AuthType Basic

AuthName "SnortIDS"

AuthUserFile /var/www/passwords/passwords

Require user acid

</Directory>
```

Now restart Apache:

```
/etc/init.d/httpd restart
```

Next time you go to the acid webpage you will get a prompt for a username and password.

Rule Management Tools

IDS Policy Manager

Communications Between Console and Sensors

Since all communications between the console and sensors will be encrypted and authenticated via SSH, these connections must be set up beforehand. The following instructions are adapted from Guy Davis' succinct tutorial on secure communications between MySQL databases using SSH (Davis, 2003).

On console

As user root:

```
ssh-keygen –t rsa (keys only have to be generated the first
time this procedure is executed from this box.)

scp .ssh/id_rsa.pub <sensor ip>:
```

As user snort:

```
ssh-keygen –t rsa (keys only have to be generated the first
time this procedure is executed from this box.)

scp ~snort/.ssh/id_rsa.pub root@<sensor
```

```
ip>:~root/id_rsa.pub.snort
```

On sensor

As user root

```
cd ~
mkdir .ssh
cat id_rsa.pub >> .ssh/authorized_keys
cat id_rsa.pub.snort >> .ssh/authorized_keys
groupadd snort
useradd -g snort snort
mkdir ~snort/.ssh
cat id_rsa.pub >> ~snort/.ssh/authorized_keys
cat id_rsa.pub.snort >> ~snort/.ssh/authorized_keys
chown –R snort:snort ~snort/.ssh
chmod –R go-rwx .ssh
chmod –R go-rwx ~snort/.ssh
rm –f id_rsa.pub*
mkdir /etc/snort
mkdir /var/log/snort
chown –R snort:snort /var/log/snort
```

Supplementing Open Source Tools

As discussed in an earlier section, the maintenance and monitoring of NIDS tools and data in a large networked environment are time-consuming chores; so much so that analysts become overwhelmed and unable to keep up with alerts produced by the system. One consequence of this might be that alerts are not attended to in a timely fashion, or are missed altogether. Another response to this problem is that a system may be disregarded or shut off.

One way to alleviate this problem is to produce customized tools that partially or fully automate some of the tasks related to the maintenance of system availability, incident alerting, data and tool management.

Making sure system stays up

The first goal of any NIDS should be that it is continuously able to detect alerts. In the case of the system design presented in this paper, the number of components comprising the system complicates this goal: A database, a web server, a secure communications protocol, sensor agents, etc. One possible solution would be do implement daemon processes or services that would monitor, maintain and coordinate the various pieces. However, such processes are far from trivial to implement, and sometimes can be tricky to operate. Thus, a simple shell script is presented here which can be scheduled through cron or similar auto-scheduling utility to run as frequently as is necessary to ensure the continuous operation of the system. The transparency of a shell script allows for faster debugging of system problems and also for easier customization as the environment changes.

Sensormon

The script is named “sensormon” and has three main functions:

1) It starts and stops the system by initiating SSH tunnels and remotely executing Snort as a daemon process on the sensor.
2) It records the state of the system at regular intervals and stores this information to a database. Besides allowing administrators to track the state of the system, it also provides a simple audit trail of system readiness.
3) It sends alerts via email to administrators if an unrecoverable error is detected in the state of the system. For example, if a sensor machine is no longer available, and therefore cannot receive or send Snort data to the console, an alert would be quickly communicated to an administrator so that the problem could be investigated and remedied.

The code for a sample, working sensormon script is presented in Appendix A. The script requires the following modifications to the snort database:

Snort database mods

Sensor.run Sensor.interface columns

```
DROP TABLE IF EXISTS sensor;
CREATE TABLE sensor (
sid int(10) unsigned NOT NULL auto_increment,
hostname text,
interface text,
filter text,
description varchar(50),
detail tinyint(4) default NULL,
encoding tinyint(4) default NULL,
last_cid int(10) unsigned NOT NULL default ‘0’,
conf_file varchar(60) default ‘/etc/snort/snort.conf’,
run enum(‘on’,’off’) NOT NULL default ‘on’,
PRIMARY KEY  (sid)
) TYPE=MyISAM;
```

Sensor_status table

```
CREATE TABLE sensor_status (
sid int(11) NOT NULL default '0',
status enum('alive','dead','started','stopped') NOT NULL
default 'alive',
timestamp timestamp(14) NOT NULL,
description
enum('NORMAL','STARTED_SSH','RESTARTED_SNORT','CANNOT_START_SS
H','CANNOT_START_SNORT','CANNOT_PING_IP','STARTING','STOPPING'
) default NULL
) TYPE=MyISAM COMMENT='Current status of all sensors';
```

Sensor_status_history table

```
CREATE TABLE sensor_status_history (
sid int(11) NOT NULL default '0',
status enum('alive','dead','started','stopped') NOT NULL
default 'alive',
timestamp timestamp(14) NOT NULL,
description
enum('NORMAL','STARTED_SSH','RESTARTED_SNORT','CANNOT_START_SS
H','CANNOT_START_SNORT','CANNOT_PING_IP','STARTING','STOPPING'
) default NULL
) TYPE=MyISAM COMMENT='Status history of all sensors';
```

User sensormon must be created

Create user sensormon@localhost, with privs to select from sensor, select/update sensor_status and insert on sensor_status_history

Configuration files

To lessen the need to modify the sensormon script itself, it is designed to read a configuration file, which defines variables such as database connection parameters and file system paths.

```
mkdir /etc/sensormon
chgrp snort /etc/sensormon
chown snort /etc/sensormon
chmod o+rx /etc/sensormon
```

The file ids.conf (example in Appendix B) should go in this directory and should be readable/writable by user/group snort

Also, the file sensormon_cron (see below) should go in there, which is just a crontab script. This should also be readable/writable by user/group snort.

Finally, a directory should be created under the snort userid to store logs:

```
mkdir ~snort/logs
```

Set sensormon to run automatically

To run sensormon on a regular basis, simply use the crontab program to import the cron script called sensormon_cron given in Appendix B as follows:

```
/usr/bin/crontab /etc/sensormon/sensormon_cron
```

Alerting for High-Priority Attacks

Since all data logged by the NIDS is stored in a MySQL database, another simple script can be implemented to monitor this data and send alerts of a given priority via email to a list of security analysts. The sample crontab file given in Appendix B sets the script mail_snort_alert.sh (listing in Appendix A) to run every half hour, but obviously this could be adjusted.

Data Management

As discussed in the early sections of this paper, a NIDS can generate a great deal of data very quickly, especially when it is first deployed. In the system proposed here, there are two destinations for data, the database collecting Snort data, and the log file that tracks the state of the sensormon script. Both of these data repositories must be "cleaned" and/or archived regularly in order to avoid overloading the filesystem of the console and degrading MySQL database performance.

Preventing Database Overloading

Again, a simple script should be sufficient for saving the oldest data from the snort database to a archive file and removing it from the database. In this case, the script works by examining the size of the disk partition where MySQL writes the snort database files on a periodic basis. When a predetermined limit is reached, the database cleaning and archiving is commenced. Obviously, the particulars of this method would have to be adjusted according to the layout of the target system. A listing of the script called clean_snort_db.sh is given in Appendix A. It is set to run every hour in the sample cron file in Appendix B.

Managing Log Files

The sensormon script prints to stdout each time it runs in order to give the administrator a trace of the system's state. The sample crontab script presented in Appendix B redirects this output to a log file. A Perl script is presented here called clean_sensormon_log.pl that manages the size of this log file, truncating data according to its age.

System Control

The availability of a convenient way to monitor and control the status of sensors in the system, a simple active web page named sensormon.php is given in Appendix A that simply queries the snort database for sensors, reports on their status and gives the option to activate or deactivate each sensor.

Data Presentation

Given that the chosen data analysis tool (ACID), the database administration tool (phpMyAdmin) and the sensor control panel are all web-based, it is sensible to create a “portal” or home page of sorts to allow easy access to these components, as well as other resources that may be of aid to the intrusion detection analyst, such as documentation, or frequently used links. Called index.php, this script is also given in Appendix A.

Deploying

Install Snort

Installing Snort involves installing the Snort executable itself--recall that Snort was statically linked, so only one file must be moved to the sensor--and Snort's configuration and rules files. The following two commands will install both Snort and its associated files.

```
scp /usr/local/bin/snort <sensor ip>:/usr/local/bin

scp /etc/snort/* <sensor ip>:/etc/snort
```

Configure Sensors

Though they will be running exactly the same operating system and software, each sensor must be configured.

snort.conf changes

On each sensor, the /etc/snort/snort.conf file must be edited such that the sensor_name parameter of the "output to mysql" line is the IP of the sensor box. Last parameter of that line should read "sensor_name=<sensor IP>", like so:

```
output database: log, mysql, user=snort password=yzerman19
dbname=snort host=localhost sensor_name=101.102.103.104
```

Also, the sensor is configured so that it is not writing out log files to the sensor machine itself, so that the administrator is not responsible for cleaning them up. To do this, make sure these lines are in the "Command Line Options" section of snort.conf:

```
config alertfile: null

config disable_decode_alerts

config logdir: /dev
```

Permission/ownership changes

Finally, the permissions on the configuration files and executable are changed such that user snort has administrative access for later upgrades and editing.

```
chown –R snort:snort /etc/snort

chown snort:snort /usr/local/bin/snort

chmod ugo+rx /etc/snort
```

Testing

Testing

Before relying upon a NIDS to help protect a corporate network, a test plan should be devised and carried out to verify that the system is functioning as expected. It is possible to execute some simple tests to verify that the various services comprising the system are running on the expected machines.

Test MySQL/Snort/SSH connection

Before activating the system, it is a good idea to manually test the connectivity between the management console and each of the sensors. If ssh, Snort or MySQL are not properly configured according to the above directions, the system will not operate. The following test commands can be executed from the management console as user snort:

Test ssh connectivity

Console

First, execute the following commands:

```
ssh -f -q -N -R3306:127.0.0.1:3306 <root@sensor IP>

ssh -f -q <root@sensor IP> /usr/local/bin/snort -i eth1 -I -c
/etc/snort/snort.conf

ssh -f -q <root@sensor IP> /usr/local/bin/snort -i eth0 -I -c
/etc/snort/snort.conf
```

Then, check to see that the expected processes are running, by executing these commands:

```
ps -ef | grep ssh

ps -ef | grep snort
```

If all goes well, a process listing should appear, indicating that one ssh process and two snort processes are running for each sensor.

Sensors

Finally, log into each sensor, and execute:

```
ps -ef | grep snort
```

This will verify that Snort is indeed running on the sensor machine.

Services

Management Console Services

Once it is established that all of the components function when invoked manually, it must be verified that the system will start automatically. To test this, first we reboot the system. Once the system has rebooted, the crontab file listed in Appendix B will invoke the various scripts to automate the system, including sensormon, which will start the system. Once sensormon has run, the following ps commands should show the status of the system on the console:

```
ps –ef | grep mysql
ps –ef | grep httpd
ps -ef | grep ssh
```

All of these should generate a listing of the respective processes. Though the listings will vary depending on the system, they should look something like this:

```
0   359     /usr/bin/httpd
0   400     /usr/bin/ssh –f –q –N –R3306:127.0.0.1:3306
0   405     /usr/local/mysql/bin/mysql_safe
```

Sensor Services

Again, log into each sensor, and execute:

```
ps –ef | grep snort
```

This will verify that Snort is indeed running on the sensor machine.

Alert Detection

Once the system is fully operational, a final set of tests should be run to determine whether the sensors are detecting traffic as they should according to a basic or default rule set, and whether these alerts are being recorded in the database. Any number of tools can be used to perform such a test, from the simple use of a ping program, to portscan or vulnerability testing tools (Caswell et al. 185), to tools that are specifically designed to light up--or even defeat--intrusion detection systems such as Snort. For example, a simply way to verify that the system is function is to use nmap (nmap) to do a simple TCP connect scan against a machine or machines that are known to be “behind” a NIDS sensor running a default Snort rule set with all rules active. For example, if a group of servers receive network traffic from a router, and this router has a NIDS sensor connected to a SPAN port or network tap, then a simple set of nmap scans could be executed from a workstation on the same network but outside the server group. The end result of such a test should be that these scans should be recorded in the Snort/ACID database installed as described above.

A step-by-step description of how to execute a more comprehensive test of a NIDS is beyond the scope of this paper; putting a NIDS through a sufficient set of tests to verify the conditions under which it can operate is a significant task in itself. And since the results of the such testing greatly depend on the particular attacks and extant network background traffic (Mell et al.), a test plan must be tailored to the network the NIDS is being tested against in order for test results to be valid. But for a NIDS built to handle the potentially very large bandwidth of a large corporate network, it would be negligent to depend on intrusion detection as a means of securing or monitoring the network without ensuring that the system does not drop an unacceptable number network packets, and that it successfully detects any traffic that meets any criteria defined by the active rule set (Caswell et al. 185-91).

Negligible Packet Loss

In order to determine whether Snort drops packets under load, simple tests can be carried out comparing the output of a NIDS sensor as described in this document with that of a simple network logging tool such as tcpdump (tcpdump) installed on a similar system connected to the network at the same access point. For example, two network taps could be installed at the same logical location on the network, one with an instance of Snort running, one with tcpdump. Since Snort can be configured to run as a packet logger that generates log files in tcpdump format (Roesch and Green), this comparison should be very straightforward. If the packet logs of Snort are not capturing the same packets as shown in the tcpdump log, it can normally be deduced that the installation of Snort is dropping packets.

Accurate Detection and False Alerts

To get more detailed data regarding the effectiveness of a NIDS, Mell et al. (2000) recommend a general test methodology to assess IDS accuracy, whose data can be distilled into a receiver operating characteristic (ROC) graph. An ROC for intrusion detection systems typically plots accuracy of detection as a function of the probability of false alerts. Such a graph can simplify the determination of whether a NIDS is of practical use for a given network topography and load. That is, if a system is detecting a high number of alerts, but the vast proportion of those are found to be false alerts, the system is providing little in the way of security, yet much in the way of busy work for the analyst. Normally, a steep curve indicates a NIDS that is performing acceptably, as it indicates a high percentage of attacks detected with a low percentage of these events being false alarms(Lippman et al.). A graph like this can also be useful for tuning purposes, as it can provide a straightforward way to compare the effectiveness and precision of different rule sets.

Managing and Making Use of the Data

Preparing for the Flood: Activating the NIDS

Finally, the moment of truth has arrived. The system is operational, and appears to be operating within the design parameters set out above. But sadly, the work of implementing a NIDS in a large network environment has only begun. Even with properly configured firewalls, DMZs and other tools of a well-secured network in place and operating, when real network bandwidth is directed at an operating NIDS configured with a default rule set, there is likely to be a flood of data into the alert database. This is why it is necessary to have tools in place prior to activation that can manage the potentially massive influx of data, in order to avoid overloading the system causing a failure.

But even with the help of some automated tools, the data must still be analyzed, and decisions as to what actions to take based on the large number of alerts initially recorded.

The Neverending Story: Tuning The Large-Network IDS

As discussed in the Design section above, one of the great challenges of implementing a NIDS is to reduce the amount of incoming data that needs to be analyzed by tuning the system; in other words, reducing the amount of false alarms reported by the system. A typical corporate network will have a very large amount of network traffic passing across any given access point, some of which is bound to resemble known attack signatures. A default Snort rule set is very likely to generate a very large number of alerts, possibly on the order of thousands per minute. This volume of alerts (whether false or not) make it difficult, if not impossible for even a team of security analysts to be able to respond in a timely manner to serious threats. Also, a well-tuned system is necessary for the system to perform optimally. Tuning is also important to reduce the likelihood of the system or the analyst being intentionally flooded with data by an attacker (Packer, Staggs and Harlow).

There are several different ways of tuning a NIDS. The first, and most obvious, is to begin the task of wading through the alert data, investigating each incident, or possibly groups of related incidents to determine if they are actual threats or innocuous traffic. Often, a rule can be modified or disabled based on the result of the analysis of a group of related incidents. For example, there are many cases of machines running Microsoft Windows utilizing ICMP to communicate with one another across a network. These ping signatures often resemble the signatures of other known information-gathering attempts that may come from outside a trusted network. Depending on the topography of the network, the location of the sensor reporting the bogus alert and the source and destination IP addresses, this rule may be disabled on that sensor, thereby eliminating that source of false alerts.

Similarly, an analyst may disable alerts of low interest or priority. With the possibility of thousands of rules being triggered every minute, it would seem prudent to attend to the most serious potential threats in the environment, such as signatures designed to detect fast-spreading worms or Trojan horse activity before investigating alerts triggered by rules written to detect corporate email policy violations. Instead of wading through thousands of these lower priority alerts and/or compromising the system's reliability due to the amount of network traffic it must analyze, such rules can be disabled, at least temporarily to allow the system and the analyst to concentrate on incidents that are potentially more serious.

By the same token, a thorough knowledge of the network may allow the analyst to disable rules that are irrelevant for the environment. For example, if there is a rule signature designed to detect attacks attempting to exploit a vulnerability in Apache web servers, yet the company operates only Microsoft IIS web servers, this rule could safely be turned off, since even a successful penetration of this kind of attack could not do any damage.

Another way to tune the system is to use a traffic filter such as Berekely Packet Filters (BPF), which is built-in to Snort, to filter out traffic that we know to be innocuous before it gets to the parsing and alerting engine (Caswell et al. 189-91). This can result in tremendous reductions in processing time, and thus a more efficient sensor. It would be appropriate to use this strategy when a particular false alert or type of traffic can be reliably identified (and differentiated from other, possibly unknown traffic) using packet header information only.

Finally, the rules themselves may be tuned. Sometimes rules that trigger many false alerts can be modified based on an analysis of the differences between a false and true alert for that rule. With this information, the specificity of the signature can be heightened, resulting in fewer false alarms.

Maintaining the System

In order to maintain system security and keep the many software components of a robust NIDS operating at peak efficiency, the software must be patched periodically. The directions for patching the components enumerated in this document vary by package and by patch, so those details cannot be included here. However, once software is patched and rebuilt on the management console, a general process for updating the executables on sensors is described in brief here.

The first step in any upgrade of sensor software is to make sure sensor is stopped on destination box.

Upgrade sensor software

```
scp /etc/snort/* <sensor ip>:/etc/snort
```

Upgrade sensor rule set

```
scp /usr/local/bin/snort <sensor ip>:/usr/local/bin
```

Suggestions for Further Research

Though this paper has tried to be comprehensive in its coverage of the steps necessary to implement and the issues associated with operating a large-scale distributed network intrusion detection system, many of the topics covered here could be investigated more deeply. In addition, there are other related aspects of NIDS that were not discussed in this paper.

Wireless

Large- and small-scale wireless networks are becoming more common in the corporate workplace. By their very nature, wireless gateways and the machines connecting to the corporate network are volatile. A “war-driving” attacker may be able to confuse or subvert a NIDS by attacking from a series of different wireless access points. Since the success of some attacks or information gathering attempts require persistence on the part of the attacker, these types of attacks would be harder to recognize if the source of the attack changed repeatedly. It would be useful to formalize methods to correlate events originating on separate wireless access points.

Dialup Gateways

Though the technology is much older, dialup access points to corporate networks are sometimes among the least secured. Between network-connected fax machines, to built-in modems that are absent-mindedly connected to live phone lines, there are potentially many ways that an attacker could gain access to a corporate network without encountering a firewall or a NIDS.

Event Correlation

The most important avenue of future research impacting the viability and value of NIDS is that of event correlation. Network administrators and security analysts have many sources of data at their disposal, including firewall logs, host intrusion detector data, network monitoring tools and vulnerability assessment tools. In most cases, this data is usable only within the tool that generated it. A NIDS with a customizable design such as that presented here could be augmented so that it could be used as a central data repository for all these data, and “meta tools” could be written to analyze patterns of data collected with this suite of tools, correlating it by time, relevance, extant vulnerabilities, etc. Such tools exist; for example, the Simple Event Correlator (Dillis) and ntop (Deri and Suin), but research into best practices of use and their effectiveness is in its nascent stages.

Still, this is the most promising avenue of research in the field of intrusion detection, and would seem to represent the next step in the evolution of NIDS. A comprehensive event correlation engine could negate for all practical purposes the biggest problems in operating a NIDS by reducing the number of false alerts reported and by accurately adjusting the priority of real alerts so that the security analyst can attend to serious events in the most practical and timely manner possible.

Conclusions

Deploying a NIDS on a corporate network is a large undertaking. The goal of this paper is to describe the steps necessary to implement a distributed network intrusion detection system such that it is manageable and adds value and security to a network rather than simply occupying a small army of security analysts.

The value of intrusion detection systems was discussed, as were the potential pitfalls and problems associated with operating these systems.

The steps necessary for implementing a distributed NIDS in a large corporate environment were detailed. The potential problems of data management and tuning the system in this environment were discussed and some strategies for reducing the impact of these problems were explained.

An effective intrusion detection system can be a valuable tool for the network administrator and the security analyst charged with the security and health of a large network--but only if it is properly designed, tested and tuned.

References

ACID. Version 0.9.6b23. 27 November 2003. <http://acidlab.sourceforge.net>.

ADODB. Version 3.30. 27 November 2003. <http://php.weblogs.com/ADODB>.

Antonatos, S., K. G. Anagnostakis, E. P. Markatos, M. Polychronakis. “Performance Analysis of Content Matching Intrusion Detection Systems”. Proceedings of the International Symposium on Applications and the Internet 2004. 17 December, 2003 <http://www.ics.forth.gr/carv/papers/2003.SAINT04.idsperf.pdf>.

Apache. Version 2.0.47. 27 November 2003. <http://www.mysql.com/downloads/index.html>

Apache HTTP Server Documentation Project. “Compiling and Installing Apache HTTP Server Version 2.0”. 14 November 2003 <http://httpd.apache.org/docs-2.0/install.html>.

Atcheson, Steve. “The Secure Shell Frequently Asked Questions”. Version 1.4. 16 February 2001. 23 October 2003 <http://www.employees.org/~satch/ssh/faq/ssh-faq.html>.

Borisov, Nikita, Ian Goldberg and David Wagner. “The Security of the WEP Algorithm”. Internet Security, Applications, Authentication and Cryptography. Computer Science Division at the University of California at Berkeley. 17 October 2003 <http://www.isaac.cs.berkeley.edu/isaac/wep-faq.html>.

Braue, David. “Intrusion detection: caught in its own web?” Technology & Business Magazine Australia. 04 September 2003. 10 November 2003 <http://www.zdnetcom.au/newstech/security/story/0,2000048600,20278214,00.htm>.

Caswell, Brian, Jay Beale, James C. Foster and Jeffrey Posluns. Snort 2.0 Intrusion Detection. Rockland, MA: Syngress Publishing, Inc., 2003.

Cole, Eric, Jason fossen, Stephen Northcutt and Hal Pomeranz. SANS Security Essentials with CISSP CBK. Volume 1, Version 2.1. SANS Press, 2003.

Danyliw, Roman. “ACID: Frequently Asked Questions”. 6 March 2003. 22 September 2003 <http://www.andrew.cmu.edu/~rdanyliw/snort/acid_faq.html>.

Danyliw, Roman. “ACID: Installation and Configuration”. 9 October 2002. 22 September 2003 <http://www.andrew.cmu.edu/~rdanyliw/snort/acid_config.html>.

Davis, Guy. “Secure Remote Database Access”. 7 October 2003 <http://www.guydavis.ca/projects/oss/docs/ssh_mysql.jsp>.

Deri, Luca and Stefano Suin. “Improving Network Security Using Ntop”. RAID 2000: Third International Workshop on the Recent Advances in Intrusion Detection, October 2-4, 2000. 3 February 2004. <http://www.raid-symposium.org/raid2000/Materials/Abstracts/13/fp.13.pdf>

Desai, Neil. “Optimizing NIDS Performance”. SecurityFocus. 6 June 2002. 16 November 2003 <http://www.securityfocus.com/infocus/1589>.

Dillis, Christopher D. “IDS Event Correlation with SEC - The Simple Event Correlator”. 2003. 14 January 2004. <http://www.giac.org/practical/GCIA/Christopher_Dillis_GCIA.pdf>

Fink, G. A., B. L. Chappell, T. G. Turner and K. F. O’Donoghue. “A Metrics-based Approach to Intrusion Detection System Evaluation for Distributed Real-Time Systems”. Information Transfer Technology Group, Code B35, Naval Surface Warfare Center, Dahlgren Division. 15 April 2002. 12 December 2003 <http://csgrad.cs.vt.edu/~finkga/published/WPDRTS-paper-Jan02.pdf>

FreePhile. “More than ever, more is better.” 02 December 2002. 17 December 2003 <http://freephile.com/compare/apache.php>.

Gartner, Inc. “Gartner Information Security Hype Cycle Declares Intrusion Detection Systems a Market Failure”. Press Release. 11 June 2003. 7 November 2003 <http://www3.gartner.com/5_about/press_releases/pr11june2003c.jsp>.

Ge, Li, Linda Scott and Mark VanderWiele. “Putting Linux reliability to the test”. The Linux Technology Center, IBM. 17 December 2003. 27 December 2003 <http://www-106.ibm.com/developerworks/library/l-rel/?ca=dgr-lnxw01LTP>.

Green, Chris. “Snort FAQ”. Version 1.14. 25 March 2002. 12 September 2003 <http://www.snort.org/docs/faq.html>.

Harper, Patrick. “Snort Install Manual: Snort, Apache, PHP, MySQL and Acid Install on RH9.0”. Version 4. 06 October 2003. 11 October 2003 <http://www.snort.org/docs/snort_acid_rh9.pdf>.

Hicks, Michael, Stephanie Weirich, and Karl Crary. “Safe and Flexible Dynamic Linking of Native Code”. 21 September 2000. 29 November 2003 <http://www.cs.cornell.edu/sweirich/papers/dynlink/taldynlink.pdf>.

JPGraph. Version 1.13. 27 November 2003. <http://www.aditus.nu/jpgraph>.

Kendall, Kristopher. “A Database of Computer Attacks for the Evaluation of Intrusion Detection Systems.” 1999. 7 January 2004. <http://citeseer.nj.nec.com/rd/55034624%2C598202%2C1%2C0.25%2CDownload/http%3AqSqqSqwww.kkendall.orgqSqfilesqSqthesisqSqkrkthesis.pdf>

Levine, John R. Linkers and Loaders. San Francisco: Morgan-Kauffman, 1999.

libpcap. Version 0.7.2. 27 November 2003. <http://www.tcpdump.org/>.

Lippman, Richard P., David J. Fried, Isaac Graf, Joshua W. Haines, Kristopher R. Kendall, David McClung, Dan Weber, Seth E. Webster, Dan Wyschogrod, Robert K. Cunningham, and Marc A. Zissman. “Evaluating Intrusion Detection Systems: The 1998 DARPA Off-line Intrusioin Detection Evaluation”. 2000. 4 January 2004. <http://citeseer.nj.nec.com/rd/34785115%2C326231%2C1%2C0.25%2CDownload/http%3AqSqqSqwww.ll.mit.eduqSqISTqSqpubsqSqdiscex2000-rpl-paper.pdf>.

Mell, Peter, Vincent Hu, Richard Lippman, Josh Hines and Marc Zissman. “An Overview of Issues in Testing Intrusion Detection Systems”. 2000. 3 December 2003. <http://csrc.nist.gov/publications/nistir/nistir-7007.pdf>.

MySQL. Version 4.0.16. 27 November 2003.
<http://www.mysql.com/downloads/index.html>.

nmap. Version 3.50. 27 November 2003. <http://www.insecure.org/nmap/>

Packer, Ryon. "Maximizing the Value of Network Intrusion Detection." Intrusion.com. 2001. 28 November 2003.
<http://www.intrusion.com/products/download/MaximizingValueIDS.pdf>

Northcutt, Stephen and Judy Novak. Network Intrusion Detection. 3rd ed. Indianapolis: Que Publishing, Inc., 2003.

Packer, Ryon, Michael J. Staggs and Darren Harlow. "Deploying and Tuning Network Intrusion Detection Systems". Intrusion.com. 2001. 28 November 2003
<http://www.intrusion.com/products/download/Deploying_and_Tuning_NIDS.pdf>
.

PHP. Version 4.3.2. 27 September 2003. <http://www.php.net/downloads.php>.

phpMyAdmin. Version 2.5.3. 27 November 2003.
<http://www.phpmyadmin.net/home_page/>.

Red Hat, Inc. "Red Hat Linux Customization Guide". 2002. 15 November 20003
<http://www.redhat.com/docs/manuals/linux/RHL-9-Manual/custom-guide/>.

Red Hat, Inc. "Red Hat Linux Security Guide". 2002. 16 November 20003
<http://www.redhat.com/docs/manuals/linux/RHL-9-Manual/security-guide/>.

Roamer. "Network IDS Sensor Placement". 25 January 2001. 13 November 2003
<http://www.securityhorizon.com/whitepapers/technical/IDSplace.html>

Roberts, Paul. "ISS Reports Snort Vulnerability". InfoWorld. 4 March 2003. 14 Nov 2003
<http://www.infoworld.com/article/03/03/04/HNsnort_1.html>.

Roesch, Martin and Chris Green. "Snort Users Manual", Version 2.0.0. 9 September 2003 <http://www.snort.org/docs/writing_rules/>.

Snort. Ver. 2.0.5. 19 Nov 2003. <http://www.snort.org/dl>.

State of California. "Notice of Security Breach - Civil Code Sections 1798.29 and 1798.82 - 1798.84". California Department of Consumer Affairs. 24 June 2003. 17 December 2003 <http://www.privacy.ca.gov/code/cc1798.291798.82.htm>.

tcpdump. Version 3.8. 27 November 2003. <http://www.tcpdump.org/>.

Yegneswaran, Vinod, Paul Barford and Johannes Ullrich. "Internet Intrusions: Global Characteristics and Prevalence". SIGMETRICS'03, June 10-14, 2003. 9 December 2003 <http://www.cs.wisc.edu/~pb/dshield_paper.pdf>.

zLib. Version 1.1.4. 27 November 2003.
<http://flow.dl.sourceforge.net/sourceforge/libpng/zlib-1.1.4.tar.gz>.

Appendices

A. Code Listings

sensormon

```
#!/bin/bash

usage="usage: sensormon [-dv] [start|stop|restart]"

ACCEPTABLE_DOWNTIME=15
cronfile=/etc/sensormon/sensormon_cron

# Get configuration
. /etc/sensormon/ids.conf

# Process command line options
while getopts ":dv" opt
do
  case $opt in
      d  )
          DEBUG=TRUE;;
      v  )
          VERBOSE=TRUE;;
      \? )
          echo "$usage"
          return 1;;
  esac
done
shift $(($OPTIND -1 ))

# Assume start if no action specified
action=${1:-start}
```

```
if [[ $action = "restart" ]]
then
  ./sensormon ${DEBUG:+-d} ${VERBOSE:+-v} stop
fi


# Find out our IP
#OUR_IP=$(/sbin/ifconfig | sed -n -e "s/^.*addr://g" -n -e "s/
*Bcast.*//p")
OUR_IP=3.97.104.114
echo
"************************************************************"
echo "SENSORMON: $(date) epoch_seconds:$(date +%s)"
echo
"************************************************************"


# If MySQL is down, send alert mail, disable cron
if ! $mysql -u $username --password=$password $db -h $host -s
<<-EOF
  SELECT user();
  exit
  EOF
then
  if [[ -n $DEBUG ]]; then
      echo "MySQL is down"
      sleep 5
  fi

  mail $RECIPIENTS -s "IDS Console MySQL is down" < /dev/null
  if [[ -x $cronfile ]]
  then
      mv -f $cronfile ${cronfile}.bak
  fi
  crontab -l > $cronfile
  crontab -r
  exit
fi

if [[ -n $DEBUG ]]; then
  echo "action: $action"
  echo "OUR_IP: $OUR_IP"
fi

{
  $mysql -u $username --password=$password $db -h $host -s <<-
EOF
  SELECT sid, hostname, conf_file, interface, run
  FROM sensor
  ORDER BY sid, hostname, interface;
  exit
  EOF
} | while read id ip config_file interface should_be_running
do
  if [[ -n $DEBUG ]]; then
      echo
"**************************************************************
*******************"
      echo "**************** Checking sensor #$id IP: $ip
**************************"
      echo
"**************************************************************
*******************"
  fi
  # If snort is running on this box don't treat it as a sensor
  if [[ $OUR_IP != $ip ]]
  then
      # Set our command strings
      port_fwd_cmd="ssh -f -q -N -R3306:$host:3306 root@$ip"
      get_port_fwd_pid="ps -ew -o pid,etime,command | grep -e
\"$port_fwd_cmd\" | grep -v 'grep' | tr -s ' ' | sed -e 's/^
```

```
//' | cut -f1 -d' '"
     kill_port_fwd_cmd='kill $port_fwd_pid'
     snort_cmd="/usr/local/bin/snort -i $interface -I -c
$config_file"
     start_snort_cmd="ssh -f -q root@$ip $snort_cmd"
     if [[ -z $VERBOSE ]];then
         start_snort_cmd="$start_snort_cmd >/dev/null 2>&1"
     fi
     get_snort_pid="ssh -f root@$ip ps -ew -o
pid,etime,command | grep -e \"$snort_cmd\" | grep -v 'grep' |
tr -s ' ' | sed -e 's/^ //' | cut -f1 -d' '"
     get_any_snort_pid="ssh -f root@$ip ps -ew -o
pid,etime,command | grep -e /usr/local/bin/snort | grep -v
'grep' | tr -s ' ' | sed -e 's/^ //' | cut -f1 -d' '"
     kill_snort_cmd='ssh -f root@$ip kill -9 $snort_pid'

     # Reset error flag for each sensor
     errFlag="NORMAL"
     status="alive"

     if [[ -n $DEBUG ]]; then
         echo "port_fwd_cmd: $port_fwd_cmd"
         echo "start_snort_cmd: $start_snort_cmd"
         echo "get_port_fwd_pid: $get_port_fwd_pid"
         echo "get_snort_pid: $get_snort_pid"
     fi

     if [[ ( $action = "start" || $action = "restart" ) &&
$should_be_running = "on" ]]
     then
         touch /var/lock/subsys/sensormon
         echo "sensormon: Starting sensor $ip"
         # Make sure sensor box is up
         #if ping -q -c 2 -w 2 $ip
         if ! /usr/bin/nmap -sT -P0 -p 22 --host_timeout 1000
```

```
$ip | grep timeout >/dev/null
         then
             if [[ -n $DEBUG ]]; then
                 echo "Able to ping sensor $ip"
             fi

             # Check for existing ssh port forward to this
machine
             port_fwd_pid=$(eval $get_port_fwd_pid)
             if [[ ! -n $port_fwd_pid ]]
             then
                 if [[ -n $DEBUG ]]; then
                     echo "Did not find existing SSH
connection to sensor $ip"
                     echo "Attempting to start with -
$port_fwd_cmd . . ."
                 fi

                 # If it's not there, try to start it
                 $port_fwd_cmd

                 # Now check it
                 port_fwd_pid=$(eval $get_port_fwd_pid)
                 if [[ ! -n $port_fwd_pid ]]
                 then
                     if [[ -n $DEBUG ]]; then
                         echo "Could not start port forwarding
for sensor $ip"
                     fi
                     # If it's not there, after attempting
restart, update log
                     errFlag="CANNOT_START_SSH"
                     status="dead"
                 else
```

```
                    if [[ -n $DEBUG ]]; then
                        echo "Started port forwarding for
sensor $ip"
                    fi
                    errFlag="STARTED_SSH"
                fi
            else
                if [[ -n $DEBUG ]]; then
                    echo "Port forwarding for sensor $ip has
already been established"
                fi
            fi

            # If port forwarding is up, start snort
            if [[ -n $port_fwd_pid ]]
            then
                snort_pid=$(eval $get_snort_pid)
                if [[ ! -n $snort_pid || $errFlag =
"STARTED_SSH" ]]
                then

                    if [[ $errFlag = "STARTED_SSH" && -n
$snort_pid ]]
                    then
                        if [[ -n $DEBUG ]]; then
                            echo "Found existing snort
process on sensor $ip: $snort_pid"
                            echo "But needs to be restarted
since SSH was"
                            echo "Attempting to kill with
$kill_snort_cmd . . ."
                            eval $kill_snort_cmd
                            echo "Attempting to start with -
$start_snort_cmd . . ."
                            errFlag="RESTARTED_SNORT"
```

```
                          fi
                      else
                          if [[ -n $DEBUG ]]; then
                              echo "Did not find existing snort
process on sensor $ip"
                              echo "Attempting to start with -
$start_snort_cmd . . ."
                              errFlag="NORMAL"
                          fi
                      fi

                      # If it's not running there, try to start
it
                      $start_snort_cmd

                      # Now check it
                      snort_pid=$(eval $get_snort_pid)
                      if [[ ! -n $snort_pid ]]
                      then
                          if [[ -n $DEBUG ]]; then
                              echo "Could not start snort on
sensor $ip"
                          fi
                          # If it's not there, after attempting
restart, update log
                          errFlag="CANNOT_START_SNORT"
                          status="dead"
                          # Make sure to kill the port
forwarding
                          eval $kill_port_fwd_cmd
                      else
                          if [[ -n $DEBUG ]]; then
                              echo "Started snort on sensor
$ip"
                          fi
                          status="started"
                      fi
                  else
                      if [[ -n $DEBUG ]]; then
                          echo "Snort on sensor $ip is already
running"
                      fi
                  fi
              else
                  if [[ -n $DEBUG ]]; then
                      echo "Did not try to start snort, because
port forwarding failed"
                  fi
              fi
          else
              # update that the box is not up
              if [[ -n $DEBUG ]]; then
                  echo "Sensor $ip is unreachable via ICMP"
              fi

              errFlag="CANNOT_PING_IP"
              status="dead"

              # make sure to tear down any ssh connection to
that box
              port_fwd_pid=$(eval $get_port_fwd_pid)
              if [[ -n $port_fwd_pid ]]
              then
                  eval $kill_port_fwd_cmd
              fi
          fi
      elif [[ $action = "stop" || $should_be_running = "off" ]]
      then
          echo "sensormon: Stopping sensor #$id IP $ip"
          rm -f /var/lock/subsys/sensormon


          #if ping -q -c 2 -w 2 $ip
          if ! /usr/bin/nmap -sT -P0 -p 22 --host_timeout 1000
$ip | grep timeout >/dev/null
          then
              snort_pid=$(eval $get_snort_pid)
              if [[ -n $snort_pid ]]; then
                  if [[ -n $DEBUG ]]; then
                      echo "Killing snort PID $snort_pid"
                  fi
                  eval $kill_snort_cmd
                  status="stopped"
              else
                  if [[ -n $DEBUG ]]; then
                      echo "No snort instance running on $ip"
                  fi
                  status="dead"
              fi
          else
              if [[ -n $DEBUG ]]; then
                  echo "Sensor $ip is unreachable via ICMP"
              fi

              errFlag="CANNOT_PING_IP"
          fi

          # Always check port forwarding
          port_fwd_pid=$(eval $get_port_fwd_pid)
          any_snort_running_pid=$(eval $get_any_snort_pid)

          if [[ -n $port_fwd_pid ]]; then
              if [[ $action = "stop" || $should_be_running =
"off" && -z $any_snort_running_pid ]]; then
                  if [[ -n $DEBUG ]]; then
                      echo "Killing port forwarding PID
$port_fwd_pid "
                  fi
                  eval $kill_port_fwd_cmd
                  status="stopped"
              else
                  if [[ -n $DEBUG ]]; then
                      echo "Not killing port forwarding PID
$port_fwd_pid -- found snort running on $ip with PID
$any_snort_running_pid"
                  fi
              fi
          else
              if [[ -n $DEBUG ]]; then
                  echo "No port forwarding running for $ip"
              fi
              status="dead"
          fi

          # Turn it off if we're stopping
          #if [[ $should_be_running = "on" ]]
          #then
          #   $mysql -u $username --password=$password $db -h
$host -s <<-EOF
          #   UPDATE sensor
          #   SET run = 'off'
          #   WHERE sid = $id;
          #   exit
          #   EOF
          #fi
      else
          if [[ -n $DEBUG ]]; then
```

```
              echo "Unknown action specified: $action"
              echo "$usage"
          fi
      fi

      # Log to the snort database
      if [[ -n $DEBUG ]]; then
          echo "LOG: $id, $ip, $status, $errFlag"
      fi



      $mysql -u $username --password=$password $db -h $host -s
<<-EOF
      INSERT INTO sensor_status_history
      ( SELECT * FROM sensor_status WHERE sid=$id);
      DELETE FROM sensor_status WHERE sid=$id;
      INSERT INTO sensor_status
      ( sid, status, description )
      VALUES
      ( $id, '$status', '$errFlag' );
      exit
      EOF

      {
          $mysql -u $username --password=$password $db -h $host
-s <<-EOF
          SELECT COUNT( * )
          FROM sensor s, sensor_status ss,
              sensor_status_history ssh
          WHERE ssh.status = 'dead'
          AND DATE_FORMAT(ssh.timestamp, '%Y%m%d%H%i') >=
              DATE_FORMAT( DATE_SUB(NOW(), INTERVAL
$ACCEPTABLE_DOWNTIME MINUTE), '%Y%m%d%H%i' )
          AND ssh.sid = ss.sid
          AND ss.status = 'dead'
          AND ss.description != 'STARTING'
          AND ssh.sid = s.sid
          AND s.run = 'on' ;
          exit
          EOF
      } | while read count
      do
          if (( $count >= $ACCEPTABLE_DOWNTIME ))
          then
              if [[ -n $DEBUG ]]; then
                  echo "Sensor $id has been dead for $count 
minutes"
              fi
              mail $RECIPIENTS -s "NIDS Sensor #$id($ip) dead 
for $count minutes" < /dev/null

              # Update run to off, so inboxes aren't flooded
              $mysql -u $username --password=$password $db -h 
$host -s <<-EOF
              UPDATE sensor
              SET run = 'off'
              WHERE sid = $id;
              exit
              EOF

              errFlag="ERROR_SHUTDOWN"
          fi

      done
  else
      if [[ -n $DEBUG ]]; then
          echo "IP found is console box; do not process as 
sensor"
```

```
     fi
  fi

done
```

mail_snort_alert.sh

```
#!/bin/bash
#
# mail_snort_alert.sh
#
#    Send alert mails based on new entries in the database
#deleting old event data (after backing it up)

exec 1>~/logs/mail_alerts 2>&1

# Get configuration
. /etc/sensormon/ids.conf

# Set some variables
lowest_priority=1
today=$(date)
cid_file=~/logs/mail_alerts_last_cid

# Find out the last cid that we looked at
max_cid=`echo "select max(cid) from event;" | $mysql -h $host
-u $username --password=$password $db -s`
last_cid=`cat $cid_file 2>/dev/null`
echo $last_cid
echo $max_cid
echo $max_cid >$cid_file

# In case we've never run before
if [[ -z $last_cid ]]
```

```
then
  exit 0
fi


# This is the query that will retrieve recent alerts of a
given severity
alert_query="
SELECT  e.sid, e.cid,
  DATE_FORMAT(timestamp, '%Y%m%d_%H:%i:%s'),
  REPLACE(s.sig_name, ' ', '_'),
  i.ip_src, i.ip_dst
FROM signature s, event e
JOIN iphdr i
USING ( sid, cid )
WHERE e.cid > $last_cid
AND e.cid <= $max_cid
AND e.signature = s.sig_id
AND s.sig_priority <= $lowest_priority
"


# First count to see if there are any
count=`$mysql -h $host -u $username --password=$password $db -
s <<-EOF
  SELECT COUNT(*)
  FROM signature s, event e
  JOIN iphdr i
  USING ( sid, cid )
  WHERE e.cid > $last_cid
  AND e.cid <= $max_cid
  AND e.signature = s.sig_id
  AND s.sig_priority <= $lowest_priority
  ORDER BY e.timestamp DESC;
  exit
  EOF`

echo "C: $count"

# If there are, send out an email
if [[ $count > 0 ]]
then
(
  $mysql -h $host -u $username --password=$password $db -s <<-
EOF
  $alert_query;
  exit
  EOF

) | while read sid cid timestamp signature src_raw dst_raw
tag_close
do
  src=$(perl -e "use Socket;print
inet_ntoa(pack(\"N\",$src_raw));")
  dst=$(perl -e "use Socket;print
inet_ntoa(pack(\"N\",$dst_raw));")
  echo "($timestamp)$signature src=$src dst=$dst"
  echo
"http://3.97.104.114/acid/acid_qry_alert.php?submit=%230-
%28$sid-$cid%29"
done | mail $RECIPIENTS -s "Snort Alert $today"
fi
```

clean_snort_db.sh

```
#!/bin/bash
#
# clean_snort_db.sh
#
#    perform maintenance on MySQL db "snort" by
# deleting old event data (after backing it up)
```

```
usage="$(basename $0) [-f]"

# Process command line options
while getopts ":df" opt
do
        case $opt in
               d  )
                       DEBUG=TRUE;;
               f  )
                       FORCE=TRUE;;
                \? )
                       echo "$usage"
                       return 1;;
        esac
done
shift $(($OPTIND -1 ))

# If the filesystem isn't that full, no need to delete
percentFull=`df | grep sda2 | perl -ne '/(\d+)%/;print $1;'`
if (( $percentFull < 85 )) && [[ -z $FORCE ]]
then
 exit 0
fi

# Get configuration
. /etc/sensormon/ids.conf

backupdir=~snort/archive/mysql_snort
timestamp=`date +'%Y%m%d%H%M%S'`
backupfile=$backupdir/snort_$timestamp

# Just in case, for first time runs
```

```
if [[ ! -d $backupdir ]]
then
 mkdir $backupdir
fi


# First, do backup
tables="data event icmphdr iphdr opt tcphdr udphdr signature
sig_class sig_reference reference reference_system acid_event
acid_ip_cache"
$mysqldump $db -u $username -h $host --password=$password
$tables > $backupfile
gzip $backupfile


# Now do the big D
$mysql -h $host -u $username --password=$password $db <<-EOF
DELETE
FROM sensor_status_history
WHERE timestamp < DATE_SUB( NOW(  ) ,  INTERVAL 7 DAY  ) ;
DELETE FROM data;
DELETE FROM event;
DELETE FROM icmphdr;
DELETE FROM iphdr;
DELETE FROM opt;
DELETE FROM tcphdr;
DELETE FROM udphdr;
DELETE FROM signature;
DELETE FROM sig_class;
DELETE FROM sig_reference;
DELETE FROM reference;
DELETE FROM reference_system;
DELETE FROM acid_event;
DELETE FROM acid_ip_cache;
EOF
```

```
$mysql -u root --password=$rootpassword <<-EOF
RESET MASTER;
EOF


mail $RECIPIENTS -s "Cleaned Snort database at $(date)"
</dev/null
```

clean_sensormon_log.sh

```
#!/usr/bin/perl -i


#     clean_sensormon_log.pl
#
#     Cleans the logs generated by sensormon,
#     removing all log entries before a given date

$now = `date +%s`;
$ok = 0;
$keep_days = 3;

while (<>)
{
  # This searches for the first date in the logfile
  # that is as old as we want to keep, and sets a flag
  if (/epoch_seconds:(\d+)/ && ! $ok)
  {
      $diff_days = ($now - $1)/(60 * 60 * 24);

      if ($diff_days <= $keep_days)
      {
          $ok = 1;
      }
  }
```

```
 # If the flag is true, print the line
 print if ($ok);
}
```

sensormon.php

```
<?php
//
 $title = "IDS Sensor Monitor";


 // This DB link will be persistent and is used later in the
script
 $dbLink = mysql_connect("localhost", "sensormon",
"sensormon");
 mysql_select_db("snort", $dbLink);


 // First update sensor table in case user chose to turn
on/off a sensor
 $sql =  "SELECT sid, run FROM sensor";
 $dbResult = mysql_query($sql, $dbLink);


 while($row = mysql_fetch_array($dbResult, MYSQL_ASSOC))
 {
     if ($_POST[$row["sid"]] == "Stop" || $_POST[$row["sid"]]
== "Start")
     {
         $_POST[$row["sid"]] == "Stop" ? $desc = 'STOPPING' :
$desc = 'STARTING';
         $_POST[$row["sid"]] == "Stop" ? $runVal = 'off' :
$runVal = 'on';


         $updSQL = "UPDATE sensor SET run = '" . $runVal . "'
WHERE sid = " . $row['sid'];
         $insHistorySQL = " INSERT INTO sensor_status_history
" .
                        "( SELECT * FROM sensor_status WHERE
sid=" . $row['sid'] . ") ";
```

```
                 $updStatusSQL = "UPDATE sensor_status " .
                 "SET description = '" . $desc . "', " .
                 "timestamp = NOW() " .
                 "WHERE sid=" . $row['sid'];


         $dbResultOfUpdate = mysql_query($updSQL, $dbLink);
         $dbResultOfInsHistory = mysql_query($insHistorySQL,
$dbLink);
         $dbResultOfupdStatus = mysql_query($updStatusSQL,
$dbLink);


         // This is necessary to avoid the "duplicate post"
problem
         header("Location: " . $_SERVER["PHP_SELF"]);
     }
 }


?>
<HTML>
<HEAD>
<LINK HREF="/common/real.css" REL="stylesheet"
TYPE="text/css">
<TITLE><?=$title?></TITLE>
<META HTTP-EQUIV="Refresh" CONTENT="30;URL=<?php
$_SERVER["PHP_SELF"] ?>">
</HEAD>
<BODY>
<CENTER>
<H2> sensormon </H2>
<?php
 print "<FORM NAME=turn_on_off ACTION=" . $_SERVER["PHP_SELF"]
. " METHOD=POST>";


 // Now retrieve the status of the sensors
 $sql =  "SELECT s.run action, s.sid id, s.description, " .
```

```
      "CONCAT(s.hostname, '(', s.interface, ')') sensor_ip, " .
      "ss.status, DATE_FORMAT(ss.timestamp, '%m/%d %I:%i %p') 
updated, " .
      "ss.description reason " .
      "FROM sensor s " .
      "LEFT JOIN sensor_status ss ON ss.sid = s.sid ";

  $dbResult = mysql_query($sql, $dbLink);
  print "<TABLE BORDER=0 CELLSPACING=0 CELLPADDING=3>";
  print "<TR BGCOLOR=#770000>";
  for($i = 0; $i < mysql_num_fields($dbResult); $i++)
  {
      if (mysql_field_name($dbResult, $i) == "action")
      {
          print ("<TD>&nbsp;</TD>");
      }
      else
      {
          print ("<TH><FONT COLOR=white>" . 
mysql_field_name($dbResult, $i) . "</TH>");
      }
  }
  print "</TR>";
  while($row = mysql_fetch_array($dbResult, MYSQL_ASSOC))
  {
      if ($row["status"] == "alive" && $row["reason"] == 
"NORMAL")
      {
          $count++ % 2 == 1 ? $bcolor = "#DCFFDC" : $bcolor = 
"#CDFFCD";
      }
      else if ($row["status"] == "dead" && $row["reason"] == 
"NORMAL")
      {
          $count++ % 2 == 1 ? $bcolor = "#FFDCDC" : $bcolor =
"#FFCDCD";
      }
      else
      {
          $count++ % 2 == 1 ? $bcolor = "#FFFFCC" : $bcolor =
"#FFFFBB";
      }
      print "<TR BGCOLOR=$bcolor>";
      for($i = 0; $i < mysql_num_fields($dbResult); $i++)
      {
          if (mysql_field_name($dbResult, $i) == "action")
          {
              $buttonVal = $row["action"] == "on" ? "Stop" :
"Start";
              print ("<TD><INPUT TYPE=SUBMIT NAME=" .
$row["id"] );
              print(" VALUE=\"" . $buttonVal .
"\">&nbsp;</TD>");
          }
          else
          {
              print ("<TD><FONT SIZE=-1 FACE=Times>" .
$row[mysql_field_name($dbResult, $i)] . "&nbsp;</TD>");
          }
      }

      print "</TR>";
  }
  print "</TABLE>";
?>
</FORM>
</CENTER>
</BODY>
</HTML>
```

index.php

```
<?php
//
$title = "Intrusion Detection Console";
?>
<head>
<link href="/common/real.css" rel="stylesheet"
type="text/css">
<title><?=$title?></title>
<script type="text/javascript">
function changeSrc(obj)
{
  document.getElementById("target").src=obj.href;
}
</script>
</head>
<body>
<DIV ID="head">
<H1> Intrusion Detection Console </H1>
</DIV>
<IMG SRC=/icons/sso_header13.gif WIDTH=850>
<DIV ID="menu">
Applications
<UL>
<LI><A HREF=acid onClick="changeSrc(this);return
false;">ACID</A></LI>
<LI><A HREF=acid_archive onClick="changeSrc(this);return
false;">ACID Archive</A></LI>
<LI><A HREF=phpMyAdmin onClick="changeSrc(this);return
false;">phpMyAdmin</A></LI>
<LI><A HREF=main/info.php onClick="changeSrc(this);return
false;">phpInfo()</A></LI>
<LI><A HREF=main/networkquery.php
onClick="changeSrc(this);return false;">Network Query</A></LI>
<LI><A HREF=http://www.infobear.com/nslookup.shtml
```

```
onClick="changeSrc(this);return false;">nslookup</A></LI>

<LI><A HREF=main/sensormon.php onClick="changeSrc(this);return
false;">Sensormon</A></LI>

</UL>

Docs

<UL>

<LI><A TARGET=new
HREF=http://www.andrew.cmu.edu/~rdanyliw/snort/snortacid.html>
ACID</A></LI>

<LI><A HREF=/manual onClick="changeSrc(this);return
false;">Apache Manual</A></LI>

<LI><A HREF=jpgraph/docs onClick="changeSrc(this);return
false;">JPGraph Docs</A></LI>

<LI><A HREF="http://www.mysql.com/doc/en/index.html"
onClick="changeSrc(this);return false;">MySQL</A></LI>

<LI><A HREF="http://www.php.net/manual/en/"
onClick="changeSrc(this);return false;">PHP</A></LI>

<LI><A HREF="http://www.redhat.com/docs/manuals/linux/RHL-9-
Manual/" onClick="changeSrc(this);return false;">RedHat
Linux</A></LI>

<LI><A HREF="http://www.snort.org/docs/writing_rules/"
onClick="changeSrc(this);return false;">Snort</A></LI>

</UL>

Security Links

<UL>

<LI><A TARGET=new HREF=http://www.cert.org>CERT</A></LI>

<LI><A TARGET=new
HREF=http://www.securitypronews.com>SecurityPro</A></LI>

<LI><A TARGET=new HREF=http://www.securityfocus.com>Security
Focus</A></LI>

<LI><A TARGET=new
HREF=http://www.whitehats.com>Whitehats</A></LI>

</UL>


<CENTER><IMG SRC=/icons/icon.jpg WIDTH=80 HEIGHT=80></CENTER>

</DIV>

<iframe id=target src=main/sensormon.php></iframe>
```

```
</body>
</html>
```

B. Configuration Files

Sample ids.conf

```
mysql_home=/usr/local/mysql
username=snort
password=snort_password
host=127.0.0.1
db=snort
mysql="$mysql_home/bin/mysql"
mysqldump="$mysql_home/bin/mysqldump"
RECIPIENTS="Security@company.com"
rootpassword=mysql_db_root_password
```

Sample sensormon_cron

```
# DO NOT EDIT THIS FILE - edit the master and reinstall.
# (Cron version -- $Id: crontab.c,v 2.13 1994/01/17 03:20:37
vixie Exp $)
* * * * * ~snort/bin/sensormon -dv >> ~snort/logs/sensormon
2>&1
18,48 * * * * ~snort/bin/mail_snort_alert.sh >>
~snort/logs/mail_snort_alert 2>&1
00 06 * * * ~snort/bin/clean_sensormon_log.pl ~/logs/sensormon
05 * * * * ~snort/bin/clean_snort_db.sh
>>~/logs/clean_snort_db 2>&1
07 12 * * 1 ~snort/bin/backup.sh 1>/dev/null 2>&1
```