Global Information Assurance Certification Paper

GSEC Practical V.1.4
Paul Fontenot
June 23, 2002

Introduction

We will be utilizing the freely available operating system OpenBSD ver. 3.1 for centralization of Unix syslog data. We will be using the MySQL SQL database engine for collection and storage of the syslog data, the Apache web server for displaying the web pages, Perl with DBI and DBD for collection and insertion of syslog data, and the Apache web server with PHP for display of syslog data.

We will *not* be changing the syslogd configuration (/etc/syslog.conf) on the central server; this is to avoid possible syslog "bombing" which could render our centralization efforts nil. The MySQL database engine can handle the flow of large amounts of data much better than the syslog facility. We will be changing the syslogd configuration on the remote servers to allow for logging of syslog data to another file for processing by Perl. The DBI / DBD scripts will pick up and insert data in the MySQL database.

OpenBSD 3.1

We will be doing an initial installation of OpenBSD from a network source. After we have downloaded the boot disk installation image and written it to a formatted floppy disk. I used a Linux machine for the creation for the floppy disk with the following command:

```
# dd if=floppy31.fs of=/dev/fd0 bs=126k
```

The above command says that we will using an input file called floppy31.fs. This is an *image file* so just copying it to a disk doesn't work. We are telling it to use an output file of /dev/fd0 or the floppy disk, and bs=126k means to write the image in 126k blocks.

During the installation of OpenBSD you will need to label your disks. If you are not familiar with UNIX disk labeling, think of it as partitioning. I set my disks up in this example with a simple scheme, a /, swap, and /usr partition. After the disk partitioning is done OpenBSD will make the file systems, format them in other words, and move to configuring the network. After the network configuration you will be asked to enter a root password. The installation program wants to know if you will be running the X window system, for my installation I answered no. The installer then wants to know if you are doing an ftp, http, tape, cd-rom, or local installation. I chose ftp and none for my proxy, as I don't have one. After answering the rest of the questions the installer moves into the actually installation phase. It will reboot when complete.

After the initial installation of OpenBSD we will utilize the cvs program that is included with OpenBSD to update source and ports trees. We will need to set the variables for CVS_RSH and CVSROOT as well as issue the cvs commands to

update the source and ports trees. The default shell for OpenBSD is csh so at the command prompt type the following:

```
# setenv CVSROOT anoncvs@anoncvs1.usa.openbsd.org:/cvs
# setenv CVS_RSH /usr/bin/ssh
# cvs -q get -rOPENBSD_3_1 src ← updates source code
# cvs -q get -rOPENBSD_3_1 ports ← updates the ports collection
```

At this point we will compile a new kernel and reboot the system to utilize the new kernel. After reboot we will recompile the entire operating system. This will allow us to incorporate all the latest changes, security patches and updates to the system. You can either roll your own kernel, by copying /usr/src/sys/arch/i38/conf/GENERIC to /usr/src/sys/arch/i386/conf/WHATEVER and editing the copy or you can simply "cd /usr/src/sys/arch/i386/conf" and type "make GENERIC". For my example I only recompiled the GENERIC kernel, this kept my system in a known state and makes it easier to get support from openbsd's mailing lists. It also applied all the up to date kernel source code patches. The steps for doing this are outlined below:

```
# cd /usr/src/sys/arch/i386/conf
# config GENERIC
# cd ../GENERIC
# make depend
# make clean
# make
# make install
# shutdown -r now
```

When the system comes back up after the reboot, log back in as root and type:

```
# cd /usr/src
# make World ←Takes a LONG time
# shutdown -r now
```

This is the actual "full system rebuild" that will rebuild all the operating system binaries from the patched source code.

After reboot the base operating system will be ready for you to start turning things on and will be patched for all known vulnerabilities up to this point. We will begin utilizing the PF facility in OpenBSD to control access to the server. This is a very capable packet filter that we will configure to allow only ports 22 (ssh for administration), 443 (https), 3306 (MySQL server).

You will need to enable pf in /etc/rc.conf. You should read the man page for pf to understand all the options available to you. Change the NO to YES on the following line in /etc/rc.conf:

```
pf=NO                   # Packet filter / NAT
```

This activates the kernel pf module on reboot. If you need it now, and have a working /etc/pf.conf file then type the following:

```
# pfctl -R /etc/pf.conf -e
```

The “R” tells pfctl what configuration file to load for the packet filter and the “e” enables packet filtering. The /etc/pf.conf file that was used in this configuration is located here.

MySQL 3.23.49

The easiest way to get MySQL installed, up and running is by using the ports collection. We “fetched” the ports collection when we where issuing cvs commands above. We need the MySQL server package for our example so we will need to pass some environment variables to the make command, specifically the “–server” tag. Type the following at the command prompt:

```
# env SUBPACKAGE=”-server” make ← makes the MySQL server package
```

When complete, you pass the same variables to the make install command:

```
# env SUBPACKAGE=”-server” make install ← installs MySQL package
```

We now have MySQL installed and now need to start it up and configure it for our application. By default MySQL comes with no administrative passwords and a “blank” account. We’ll be setting the administrative password and restricting the administrative access to the localhost, removing the blank account and setting up the SYSLOG database with a user and password.

Setting the administrative password in MySQL is done with the following command:

```
# /usr/local/bin/mysqladmin –u root password ‘YOUR_PASSWORD’
```

You will see nothing if successful. If you see an error like:

```
# /usr/local/bin/mysqladmin –u root password ‘YOUR_PASSWORD’
mysqladmin: connect to server at ‘localhost’ failed
error: ‘Can’t connect to local MySQL server through socket
‘/var/run/mysql/mysql.sock’ ‘(2)’
Check that MySQL is running and that the socket:
‘/var/run/mysql/mysql.sock’ exists!
```

This means that MySQL has not been started. You start mysql by issuing this command:

```
# /usr/local/bin/safe_mysqld &
[1] 28883 ← Process id for the server
# Starting mysqld daemon with databases from /var/mysql
```

After mysqladmin changes the password, you will see nothing and mysqladmin will give back the console. Now you should be able to log into the mysql server and check the default permissions. You do this with the following command:

```
# mysql -u root -p mysql
testbox# mysql -u root -p mysql
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 5 to server version: 3.23.49

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql>
```

From here issue this query to see what access is set:

```
mysql> select user, host, password from user;
+------+-----------+------------------+
| user | host      | password         |
+------+-----------+------------------+
| root | localhost | 011bf66a2f709a2d |
| root | testbox   |                  |
|      | localhost |                  |
|      | testbox   |                  |
+------+-----------+------------------+
4 rows in set (0.00 sec)
```

Now we will begin restricting the administrative access to the localhost only, this prevents personnel from gaining admin access to our server via the network:

```
mysql> delete from user where user = 'root' and password ='';
Query OK, 1 row affected (0.08 sec)
```

Now we have restricted access to the localhost for administrative functions. Once again issue the query to verify that we have restricted administrative access to localhost only.

```
mysql> select user, host, password from user;
+------+-----------+------------------+
| user | host      | password         |
+------+-----------+------------------+
| root | localhost | 011bf66a2f709a2d |
|      | localhost |                  |
|      | testbox   |                  |
+------+-----------+------------------+
3 rows in set (0.00 sec)
```

Now we will remove the blank accounts from the system. This will allow only the administrative account on the localhost to have any access to the system.

```
mysql> delete from user where user = '';
Query OK, 2 rows affected (0.00 sec)
```

Once again issue the query to verify that only the administrative account has access to our server.

```
mysql> select user, host, password from user;
+------+-----------+------------------+
| user | host      | password         |
+------+-----------+------------------+
| root | localhost | 011bf66a2f709a2d |
```

```
+------+----------+-----------------+
1 row in set (0.00 sec)
```

Once this is done it is time to create the SYSLOG database. The is accomplished with the following command:

```
mysql> create database SYSLOG;
Query OK, 1 row affected (0.01 sec)
```

You will now grant privileges on the database and its tables. We will be granting select and insert privileges on the SYSLOG database to the user syslog. For this example we will be using the password 'syslog'. This is ONLY an example and not to be used in production.

```
mysql> grant select, insert on SYSLOG.* \
    -> to syslog identified by 'syslog';
Query OK, 0 rows affected (0.01 sec)
```

The above grant statement says to "*grant select, insert, update and delete privileges on the SYSLOG database and all tables associated with it to the user syslog with the password syslog*". After issuing the grant statements we should verify the syslog user exists and there is an entry in the password field:

```
mysql> select user, host, password from user;
+--------+----------+------------------+
| user   | host     | password         |
+--------+----------+------------------+
| root   | localhost | 011bf66a2f709a2d |
| syslog | %        | 77f2beb776ba725d |
+--------+----------+------------------+
2 rows in set (0.01 sec)
```

Note the '%' in the host column. We did not specify a host to connect from in the grant statement. If no host is specified, the system assumes any host and will allow syslog to connect from any machine on the network. For a more secure system you should specify a host entry for each of the servers that will be sending data to the database. This will further prevent a compromise of the central server by refusing connections that maybe an attempt to overload the database engine. The schema for our database is located here.

Apache & SSL

Apache is included in the OpenBSD distribution and has SSL support compiled in by default. You will need to create your digital certificate before you can utilize the SSL encryption compiled into Apache. The man page for ssl(8) has the procedure to use when creating the RSA digital certificate to activate the SSL module in the Apache webserver.

To support https transactions in httpd you will need to generate an RSA certificate. The following command says to "generate an rsa key /etc/ssl/private/server.key and make it 1024bits long"

```
# openssl genrsa -out /etc/ssl/private/server.key 1024
```

Or, if you wish the key to be encrypted with a pass phrase that you will have to type in when starting the server:

```
# openssl genrsa -des3 -out /etc/ssl/private/server.key 1024
```

The next step is to generate a Certificate Signing Request, which is used to get a Certifying Authority (CA) to sign your certificate. To do this use the command:

```
# openssl req -new -key /etc/ssl/private/server.key \
   -out /etc/ssl/private/server.csr
```

This server.csr file can then be given to a Certifying Authority, which will sign the key. One such CA is Thawte Certification which you can reach at http://www.thawte.com/. Thawte can currently sign RSA keys for you. A procedure is being worked out to allow for DSA keys.

You can also sign the key yourself, using the following command:

```
# openssl x509 -req -days 365 -in /etc/ssl/private/server.csr \
  -signkey /etc/ssl/private/server.key -out /etc/ssl/server.crt
```

With /etc/ssl/server.crt and /etc/ssl/private/server.key in place, you should be able to start httpd(8) with the -DSSL flag, enabling https transactions with your machine on port 443.

You will most likely want to generate a self-signed certificate in the manner above along with your certificate signing request to test your server's functionality even if you are going to have the certificate signed by another Certifying Authority. Once your Certifying Authority returns the signed certificate to you, you can switch to using the new certificate by replacing the self-signed /etc/ssl/server.crt with the certificate signed by your Certifying Authority, and then restarting Apache with:

```
# /usr/sbin/apachectl restart
```

Use the htpasswd utility that came with Apache to create a password file to further protect the syslog data from un-authorized viewing. This will be located in the bin directory of wherever you installed Apache. In our example the htpasswd utility is located at /usr/bin/htpasswd. This is the default location on OpenBSD.
To create the password file, type:

```
# htpasswd -c /etc/httpd.passwords pfontenot
```

htpasswd will ask you for the password, and then ask you to type it again for confirmation:

```
# htpasswd -c /etc/httpd.passwords pfontenot
New password: mypassword
Re-type new password: mypassword
Adding password for user pfontenot
```

Note that in the example shown, a password file is being created containing a user called pfontenot, and this password file is being placed in the location /etc/httpd.passwords. We have chosen a location outside the webuser accessible directory tree. This makes compromising the password list for Apache more difficult.

Once you have created the password file, you need to tell Apache about it, and tell Apache to use this file in order to require user credentials for admission. This configuration is done with the following directives:

AuthType	Authentication type being used. In this case, it will be set to Basic
AuthName	The authentication realm or name
AuthUserFile	The location of the password file
AuthGroupFile	The location of the group file, if any
Require	The requirement(s) which must be satisfied in order to grant admission

These directives may be placed in an .htaccess file in the particular directory being protected, or it may go in the main server configuration file, in a <Directory> section, or other scope container. We are placing them in the .htaccess file in the /var/www/htdocs directory. You will then need to make a change to the httpd.conf file telling it to use the .htaccess file. Change the following line:

```
AllowOverride None
```

to

```
AllowOveride All
```

The example shown below defines an authentication realm called ``By Invitation Only''. The password file located at /etc/httpd.passwords will be used to verify the user's identity. Only users named pfontenot will be granted access, and even then only if they provide a password that matches the password stored in the password file.

```
AuthType Basic
AuthName "By Invitation Only"
AuthUserFile /etc/httpd.passwords
Require user pfontenot
```

The phrase ``By Invitation Only'' will be displayed in the password pop-up box, where the user will have to type their credentials. You will need to restart your Apache server in order for the new configuration to take effect, if these directives

were put in the main server configuration file. Directives placed in .htaccess files take effect immediately, since .htaccess files are parsed each time files are served.

The next time that you load a file from that directory, you will see the familiar username/password dialog box pop up, requiring that you type the username and password before you are permitted to proceed.

Note that in addition to specifically listing the users to whom you want to grant access, you can specify that any valid user should be let in. This is done with the valid-user keyword:

```
Require valid-user
```

PHP & MySQL

We are going to be using the ports collection for the installation of PHP with support for the MySQL database engine. This is the fastest, easiest way to get PHP, MySQL, and Apache working together. When choosing the ports collection, you have the option of using arguments to the make commands that will tailor the software to what you need. Looking in the Makefile for PHP4 you will notice the FLAVOR lines:

```
FLAVORS+=       gdbm gettext imap ldap mhash mm recode snmp
FLAVORS+=       gd no_x11 pdflib mcrypt curl
FLAVORS+=       dbase filepro mysql mysql_bundled postgresql iodbc
# FLAVORS+=     freetds
```

We only need support for mysql, because when we built our server we chose not to have X11. So we need to pass environment variables to make that tell it we don't have X, we want support for the gd library so that later on we might utilize the "on the fly" image generation capabilities of PHP and we need MySQL support. Our command would be:

```
# env FLAVOR="no_x11 gd mysql" make
```

When the make has completed we will install the software with a similar command:

```
# env FLAVOR="no_x11 gd mysql" make install
```

After we have PHP installed we have to tell the Apache server it is there and what extensions require its use. Luckily for us, the PHP port comes with a script that makes this easy. You activate the PHP4 module by using the command:

```
# /usr/local/sbin/php4-enable
```

After activation of the module, you still need to edit the /var/www/httpd.conf file and tell apache how to use the .php extension and that index.php is a valid directory index file like index.html. The lines that need to be added to the /var/www/httpd.conf file are:

```
# DirectoryIndex: The name of the file or files to use as a directory
# index
DirectoryIndex    index.php index.html

# AddType allows you to make certain files by certain types, and
# determine what to do with them
AddType application/x-httpd-php .php
```

Now for some fun. Create a php script with your favorite editor and place it in the web tree to test for PHP capability. The default location for html documents on OpenBSD is /var/www/htdocs. I have created a simple php script that can be found here and placed it in the /var/www/htdocs directory as test.php. Verify the webserver is running by using the ps command and piping the output to grep to search for httpd:

```
# ps -ax | grep httpd
16305 ??  Is      0:00.02 /usr/sbin/httpd
12596 ??  I       0:00.01 /usr/sbin/httpd
20153 ??  I       0:00.02 /usr/sbin/httpd
  216 ??  I       0:00.01 /usr/sbin/httpd
31816 ??  I       0:00.01 /usr/sbin/httpd
16616 ??  I       0:00.01 /usr/sbin/httpd
```

If the output looks like that above, then apache is running. You should restart apache using the /usr/sbin/apachectl restart command. It will tell you that apache has restarted.

Now point a web browser at the server and verify that is has picked up the php extension you told it about in the /var/www/conf/httpd.conf file. The php script used for the test.php file will display many things about the way php was compiled and where certain files are installed. Remove this file as soon as you have determined that PHP is in fact being utilized by apache. An attacker could possible guess the name of this file and use its contents against your site.

Perl with DBI and DBD

The ports collection has the next two pieces of software that are required to make this work. They are the Perl DBI and the DBD for MySQL. They use the same make and make install commands as the rest of the ports collection and they are found under the /usr/ports/databases directory. The Perl DBI is /usr/ports/databases/p5-DBI and the Perl DBD is /usr/ports/databases/p5-DBD-Msql-Mysql. Upon completion of compilation and installation we can begin writing the perl script that does the work. Logger.pl is located here.

Making things happen

At this point in time we have all the components installed and configured. What now? This is the point where we start doing some configuration changes to syslog. As I said earlier, syslog, on the central server, is NOT doing the logging for the remote servers. Though some minor changes will need to be made to the syslog.conf files on the remotes and the main server.

The changes that are being made to syslog.conf are the same for all the servers. We are basically deciding what to log and sending that to a file of our choosing so that the perl script, logger.pl, can parse it and insert the data into the MySQL server that is running the SYSLOG database. Essentially we are adding one (1) line to the syslog.conf. I am going to copy this line:

```
*.notice;auth,authpriv,cron,ftp,kern,lpr,mail,user.none /var/log/messages
```

So that I get the same data that would be going into the messages file, going into the SYSLOG database. I do this by copying the above line and pasting it back into /etc/syslog.conf, but then I change the file to a different name. Now syslog.conf looks like the following:

```
*.notice;auth,authpriv,cron,ftp,kern,lpr,mail,user.none /var/log/messages
*.notice;auth,authpriv,cron,ftp,kern,lpr,mail,user.none /var/log/mymessages
```

I make those changes to all the servers that will be sending data to the SYSLOG database.

Now that syslog is writing duplicate data, what do I do with it? Remember the logger.pl script? This script get placed in the /sbin directory, or any other directory you chose on your system. This script gets run from cron every 60 seconds. It parses through the /var/log/mymessages file and if this is the SYSLOG database server, it then inserts the data into the SYSLOG database. If it is a remote server, then it sends the data via the perl dbd for mysql to the mysql server that is running on port 3306 on the SYSLOG database server. After logger.pl has parsed the data it clears the file and writes a line into it so that syslog will continue writing to the file.

Now we're getting somewhere. We have data flowing into our database. Now what? This is a very basic system that has many possibilities. We have the ability to utilize the graphing capability of PHP to draw "on the fly" graphics to visually display our data trends, notices, warnings, or any information that is held in our secured remote location.

We will utilize PHP to access the database information and tell us what is happening. We can graph failures of certain types, daemons, logins, etc… We can even have PHP draw pictures, we had that support compiled in when we added the gd tag to the make command. Or, as in this example we will be using PHP to display the data in a syslog type format. So that it is displayed like this:

```
Jul 14 04:29:35 laptop /bsd: mainbus0 (root)
```

Except on a web page. Now this helps the administrator immensely, if we suspect that our servers have been compromised then we can select everything on any given date that is in the database.

You will need to code the PHP front end to the SYSLOG database. I have included a simple example of what this code could look like. With relative easy you

should be able to take this example and expand it into a very nice web driven font end to a syslog database. The database schema is going to look just like the layout for the syslog logfile. You will be able to query SYSLOG for all events on any given day in the database:

```
mysql> select * from SYSLOG where month =”July” and day = “15”;
```

Will return all events logged for all hosts on July 15, or you will be able to query for specific events:

```
mysql> select  * from SYSLOG where Host = “localhost” and Description
like “syslogd%”;
```

Will return all events logged for the localhost that deal with syslogd.

By wrapping a web front end around our SYSLOG database using PHP we have created a very effective monitoring station. By utilizing Perl and the DBI and DBD we can set the syslog database to auto age data, or flush aged data to disk. We could expand on this by using Perl and the DBI and DBD to query the database every 60 seconds and if certain criteria are in the database, Perl can then email the administrator to alert them of the condition that has been met. The possibilities are endless once the data is in the database.

In Conclusion

This system is a collection of database clients that are updating syslog information stored in a MySQL database. This avoids the problems of traditional sysloging, such as syslog bombing, to a central server by utilizing the security model in MySQL to only allow certain hosts to connect, and the pf utility in OpenBSD to only allow certain ports to be available for connection.

References:

Simon Garfinkel with Gene Spafford. Web Security and Commerce. Cambridge: O’Reilly, June 1997. 293 – 309.

Tom Christiansen and Nathan Torkington. Perl Cookbook. Cambridge: O’Reilly, August 1998.

AUTHOR. Programming the Perl DBI. CITY: O’Reilly, DATE

“OpenBSD frequently asked questions”
URL: http://www.openbsd.org/faq/index.html (17 June 2002)

“MySQL | Documentation”
URL: http://mysql.org/documentation/index.html (1995 - 2002)

“Apache HTTP Server Version 1.3 Documentation”
URL: http://httpd.apache.org/docs/

"PHP:FAQ: Frequently Asked Questions"
URL: http://www.php.net/manual/en/faq.php (10 Jun 2002)

Dennis Jr, Israel and Eugene Otto. "The Soothingly Seamless Setup of Apache, SSL, MySQL, and PHP". 7 June 2002
URL:
http://zope1.devshed.com/zope.devshed.com/Server_Side/PHP/SoothinglySeamless/page1.html

Code

PF.CONF

```
# Define useful variables
ExtIF="xl0"                     # External Interface
NoRouteIPs="{127.0.0.0/8, 192.168.0.0/16, 172.16.0.0/12}"
Services="{ssh, https, mysql}"

# Clean up fragmented and abnormal packets
scrub in all

# Don't allow anyone to spoof non-routable addresses
block in  quick on $ExtIF from $NoRouteIPs to any
block out quick on $ExtIF from any to $NoRouteIPs

# by default, block all incoming packets, except those explicitly
# allowed by further rules
block in on $ExtIF all

# allow others to use http and https
pass in on $ExtIF inet proto tcp from any to any port \
          $Services flags S/SA keep state

# and let out-going traffic out and maintain state on established
# connections pass out all protocols, including TCP, UDP, ICMP, and
# create state, so that external DNS servers can reply to our own DNS
# requests (UDP).
block out on $ExtIF                 all
pass  out on $ExtIF inet proto tcp  all flags S/SA keep state
pass  out on $ExtIF inet proto udp  all keep state
pass  out on $ExtIF inet proto icmp all keep state
```

MySQL Schema

```
#
# The following is the create table schema for our database
# for simplicity to we will only show one host schema.
#
CREATE TABLE messages (
      Month char(5),
      Day char(2),
```

```
        Hms char(8),
        Host char(30),
        Description text
        );
```

Perl scripts (logger.pl)

```
#!/usr/bin/perl
#
#
use strict;
use DBI;

my
($database,$driver,$dsn,$user,$passwd,$log,$dbh,$sth,$ref,$month,$day
,$hms,$host,$system,$description);

$database = "SYSLOG";
$driver = "mysql";
$dsn = "DBI:$driver:$database;localhost;3306";
$user = "syslog";
$passwd = "CHANGE_THIS";
$log = "/var/log/mymessages";

$dbh = DBI->connect($dsn, $user, $passwd);

open(LOG, "< $log") || die "Can't open $log: $! \n";
while (<LOG>)
{
        next if /^#/;

        ($month, $day, $hms, $host, $description) = split(/\s+/, $_,
5);

        $description =~ s/'//gs;

        $dbh->do("INSERT INTO messages (month, day, hms, host,
description) VALUES ('$month', '$day', '$hms', '$host', '$description
'

)");
}
close(LOG) || die "Can't close $log: $! \n";

system("/bin/cat /dev/null > /var/log/mymessages");
system("echo \"### IGNORE THIS LINE ###\" > /var/log/mymessages");

$dbh->disconnect;
```

Simple PHP script (test.php)

```
// This is a simple script that we display all the configuration
// variables and the default locations for files that were compiled
// into PHP
<?php

phpinfo();

?>
```

Example PHP front end for SYSLOG database

Sample index.php script

```
<?php

include ("html.inc.php");

$title = "SYSLOG database test server";
$header = "SYSLOG Data handled by MySQL";

html_begin($title,$header);

print ("<hr>\n");

db_connect()
        or exit();

$result = mysql_query ("SELECT COUNT(*) FROM messages")
        or exit();
if ($row = mysql_fetch_array ($result))
        echo "<a href=\"testbox.php\">There are " . $row[0] . "
records in testboxs' message database<BR></a>\n";
mysql_free_result ($result);

$result = mysql_query ("SELECT COUNT(*) FROM demarc")
        or exit();
if ($row = mysql_fetch_array ($result))
        echo "<a href=\"demarc.php\">There are " . $row[0] . "
records in demarc's message database<BR></a>\n";
mysql_free_result ($result);

$result = mysql_query ("SELECT COUNT(*) FROM scnmgmt")
        or exit();
if ($row = mysql_fetch_array ($result))
        echo "<a href=\"scnmgmt.php\">There are " . $row[0] . "
records in scnmgmt's message database<BR></a>\n";
mysql_free_result ($result);

print ("<hr>\n");

index_end();

?>
```

Sample html.inc.php

```
<?php

function html_begin($title,$header)
{
        print ("<HTML>\n");
        print ("<HEAD>\n");
        print ("<TITLE>$title</TITLE>\n");

        print ("<meta http-equiv=\"Content-Type\"
content=\"text/html; charset=iso-8859-1\">\n");
        print ("<link rel=\"stylesheet\" type=\"text/css\"
href=\"syslog.css\">\n");

        print ("</HEAD>\n");
        print ("<BODY>\n");
```

```
        print ("<DIV ALIGN=\"CENTER\">\n");
        print ("<TABLE WIDTH=800 BORDER=1>\n");
        print ("<TR>\n");
        print ("<TD>\n");

        print ("<H1>$header</H1>\n");
}

function index_end()
{
        $date = date("H:i:s");
        print ("<DIV ALIGN=\"CENTER\">");
        print ("Local time in my cube is: $date");
        print ("</DIV>\n");
        print ("</TD>\n");
        print ("</TR>\n");
        print ("</TABLE>\n");
        print ("</DIV>\n");
        print ("</BODY>\n");
        print ("</HTML>\n");
}

function html_end()
{
        $date = date("H:i:s");
        print ("<HR>\n");
        print ("<DIV ALIGN=\"CENTER\">");
        print ("Local time in my cube is: $date<BR>");
        print ("<A HREF=\"index.php\">Return</A>\n");
        print ("</DIV>\n");
        print ("</TD>\n");
        print ("</TR>\n");
        print ("</TABLE>\n");
        print ("</DIV>\n");
        print ("</BODY>\n");
        print ("</HTML>\n");
}

function db_connect()
{
        $link = @mysql_pconnect ("localhost", "syslog", "syslog");
        if ($link && mysql_select_db ("SYSLOG"))
                return ($link);
        return (FALSE);
}

?>
```

Sample localhost.php script

```
<?php

include ("html.inc.php");

$title = "";
$header = "Testbox's Data";

html_begin($title,$header);

print ("<hr>\n");

db_connect()
        or exit();
```

```
$result = mysql_query ("SELECT * FROM messages");

if ($row = mysql_fetch_array($result)) {

        print ("<TABLE>\n");
        do
        {

                print ("<TR VALIGN=\"TOP\">\n");
                print ("<TD>\n");
                print $row["month"];
                print ("</TD>\n");
                print ("<TD>\n");
                print $row["day"];
                print ("</TD>\n");
                print ("<TD>\n");
                print $row["hms"];
                print ("</TD>\n");
                print ("<TD>\n");
                print $row["host"];
                print ("</TD>\n");
                print ("<TD>\n");
                print $row["description"];
                print ("</TD>\n");
                print ("</TR>\n");
        }

        while($row = mysql_fetch_array($result));
        print ("</TABLE>\n");
}
else
{
        print "Sorry, no records were found!";
}

html_end();

?>
```