Hardening Apache

TONY MOBILY

Apress™

ISBN: 1-59059-378-2

Printed and bound in the United States of America 10987654321

Lead Editor: Jim Sumser

Technical Reviewers: Ken Coar and Jonathan Hassell

Project Manager: Nate McFadden

Copy Manager: Nicole LeClerc

Copy Editor: Brian MacDonald

Production Manager: Kari Brooks

Production Editor: Kelly Winquist

Compositor: Molly Sharp, ContentWorks

Proofreader: Liz Welch

Indexer: Valerie Hanes Perry

Artist: Kinetic Publishing Services, LLC

Cover Designer: Kurt Krames

Manufacturing Manager: Tom Debolski

Distributed to the book trade in the United States by Springer-Verlag New York, Inc., 175 Fifth Avenue, New York, NY, 10010 and outside the United States by Springer-Verlag GmbH & Co. KG, Tiergartenstr. 17, 69112 Heidelberg, Germany.

In the United States: phone 1-800-SPRINGER, email orders@springer-ny.com, or visit http://www.springer-ny.com. Outside the United States: fax +49 6221 345229, email orders@springer.de, or visit http://www.springer.de.

For information on translations, please contact Apress directly at 2560 Ninth Street, Suite 219, Berkeley, CA 94710. Phone 510-549-5930, fax 510-549-5939, email info@apress.com, or visit http://www.apress.com.

The information in this book is distributed on an “as is” basis, without warranty. Although every precaution has been taken in the preparation of this work, neither the author(s) nor Apress shall have any liability to any person or entity with respect to any loss or damage caused or alleged to be caused directly or indirectly by the information contained in this work.

The source code for this book is available to readers at http://www.apress.com in the Downloads section.

CHAPTER 1

Secure Installation and Configuration

When you install a piece of software, you can usually just follow the instructions provided by the `README` or the `INSTALL` file. In a way, Apache is no exception. However, Apache is a very complex program, and needs to be compiled and installed with particular care, to make sure that it's reasonably secure in the short and in the long term.

In this chapter I will show you:

- How to download Apache making sure that you have a "genuine" package; I will also take the opportunity to describe how encryption works.
- The commands I used to install both Apache 1.3.*x* and Apache 2.*x*. I included this section mainly because I will use those installations throughout the book.
- How to test your installation with an automatic testing tool.
- How to configure Apache more securely.
- How to block particular requests and IP addresses.
- How to configure Apache 1.3.*x* and 2.*x* with Secure Sockets Layer (SSL).

Downloading the Right Apache

There are two major "branches" of Apache that are still fully supported: 1.3.*x* and 2.0.*x* (the latest ones at the time of writing are 1.3.29 and 2.0.48). Remember that by the time this book goes to print the versions will probably have been updated. You have two options for downloading Apache:

- **Download the Apache source from** `http://httpd.apache.org`. This is the only option available for maximum control.
- **Use a package from your favorite distribution.** In this case, you are bound to what your distribution gives you in terms of version and compiling options.

In this book I will only cover downloading and installing the "official" Apache server source distributed by the Apache Software Foundation.

Is It Safe to Download?

The very first step in installing Apache is downloading the Apache package from `http://httpd.apache.org/download.cgi`.

Downloading Apache is very straightforward. Unfortunately, there are dangerous conditions: the Apache web site (or, more possibly, one of its many mirror sites) might have been hacked, and a maliciously modified version of Apache might have replaced the real distribution file. This fake version could do exactly what it was supposed to do, plus open a back door on the server that was running it (and maybe somehow notify the person who originally wrote the code for the back door).

The Apache Software Foundation is well aware of this problem, so it *signs* its own packages. It is up to you to check that the signature of the package you downloaded is correct. In this section I will show you how to do that step by step.

Making Sure Your Apache Is Right Using GnuPG

Every official Apache package comes with a *digital signature*, aimed at ensuring that your package is genuine.

To *sign* a file, as well as verify the validity of an existing signature, you can use GnuPG (`http://www.gnupg.org`), a free clone of Pretty Good Privacy (PGP). If you are security-conscious, it's probably worth your while to study how GnuPG works.

NOTE GnuPG comes with a very well written manual, the *GNU Privacy Handbook*, The manual is at `http://www.gnupg.org/gph/en/manual.html`, and is an amazing introduction to cryptography in general.

In the next section, I will introduce the basic concepts behind cryptography, while showing what commands you can use to verify your Apache package. I will refer to these concepts to make sure that you know exactly what each command does.

A Short Introduction to Asymmetric Encryption and GnuPG

Encryption is the conversion of data into a form (called a *cipher text*) that can only be decoded by authorized people. The decoding process commonly needs a *key*—this means that only the people with the right key will be able to decrypt the information and have the original data available again.

The most basic encryption technique is one where the same secret word is used to both cipher and decipher the information. This is called *symmetric encryption* (or *secret key encryption*). Let's suppose that Adam wants to communicate with Betty. Adam will have to encrypt the data he wants to send using his key. The information will be safe while in transit. Then, Betty will have to use Adam's key to decode the information. The problem with this scheme is: how does Adam deliver his private key to Betty? There is no easy solution to this problem.

An alternative approach is *asymmetric encryption*, where Adam would have two related keys: a *private* key and a *public* key. If a piece of information is encrypted using a *private key*, the only way to decrypt it is by using the right *public key* (the two keys are generated at the same time). Adam's and Betty's public keys would be widely available through as many means as possible (such as through the Web). When Adam wants to send a message to Betty, he encrypts the message with Betty's public key. From that moment on, no one except Betty will be able to decrypt the message—not even Adam, who encrypted it in the first place!

NOTE Using a metaphor, the public key is a padlock, and the private key is the key for that particular padlock. It is in your best interest to give away as many padlocks as possible (your public key), and at the same time keep your padlock's key very secret (your private key).

After you've used encryption for a while, you will have several people's public keys stored somewhere in your computer (the key's exact location in the file system depends on the program you use). In GnuPG terminology, other people's public keys would be placed in your *public key ring*.

Another important application of asymmetric encryption is the ability to sign a block of data (a document, for example). If Adam wants to send a message to Betty, and wants her to be absolutely sure that the communication came from Adam and no one else, all Adam has to do is create a *digital signature* for that message. A digital signature is the *hash value* of the message, encrypted with Adam's private key. A hash value is a string generated using the message as the source of information, with the guarantee that two different messages will have two different hash values. All Betty has to do is to calculate the hash value of the received communication, decrypt the received hash value (that is, the signature) using Adam's public key, and compare the two.

Setting Up GnuPG

Most Linux distributions provide GnuPG, so I will assume that you have GnuPG installed on your system. The first time you run it, some basic configuration files will be created:

```
[merc@merc merc]$ gpg
gpg: /home/merc/.gnupg: directory created
gpg: /home/merc/.gnupg/options: new options file created
gpg: you have to start GnuPG again, so it can read the new options file
[merc@localhost merc]$
```

You can now create your own public and private keys using the option `--gen-key`:

```
[merc@merc merc]$ gpg --gen-key
[...]
Please select what kind of key you want:
   (1) DSA and ElGamal (default)
   (2) DSA (sign only)
   (5) RSA (sign only)
Your selection? 1
DSA keypair will have 1024 bits.
About to generate a new ELG-E keypair.
              minimum keysize is  768 bits
              default keysize is 1024 bits
    highest suggested keysize is 2048 bits
What keysize do you want? (1024) 1024
Requested keysize is 1024 bits
Please specify how long the key should be valid.
         0 = key does not expire
      <n>  = key expires in n days
      <n>w = key expires in n weeks
      <n>m = key expires in n months
      <n>y = key expires in n years
Key is valid for? (0) 0
Key does not expire at all
Is this correct (y/n)? y

You need a User-ID to identify your key; the software constructs the user id
from Real Name, Comment and Email Address in this form:
    "Heinrich Heine (Der Dichter) <heinrichh@duesseldorf.de>"

Real name: Tony Mobily
Email address: merc@mobily.com
Comment: Myself
You selected this USER-ID:
    "Tony Mobily (Myself) <merc@mobily.com>"
Change (N)ame, (C)omment, (E)mail or (O)kay/(Q)uit? o
You need a Passphrase to protect your secret key.
[...]
key marked as ultimately trusted.
pub  1024D/B763CD69 2003-08-03 Tony Mobily (Myself) <merc@mobily.com>
     Key fingerprint = A524 518E 9487 F66F C613  A506 0D8F C15F B763 CD69
sub  1024g/0C1049EE 2003-08-03
[merc@merc merc]$
```

Notice how I have chosen all the default options: “DSA and ElGamal” for my private key, 1024 bits for the encryption, and no expiration date. Also, I have entered all my personal details: name, surname, comments, and e-mail address. My ID will be “Tony Mobily (Myself) merc@mobily.com”.

GnuPG and Apache Signatures

You now need to fetch the public key of the person who signed the Apache packages you downloaded. From the page `http://httpd.apache.org/download.cgi` on the web site you can read:

- `httpd-2.0.48.tar.gz` is signed by Sander Striker DE885DD3.
- `httpd-2.0.48-win32-src.zip` is signed by William Rowe 10FDE075.
- `httpd-1.3.28.tar.gz` is signed by Jim Jagielski 08C975E5.

If you want to check `httpd-2.0.48.tar.gz`'s signature, you will need to put Sander Striker's public key in your public key ring.

You can obtain his pubic key in two ways. The first is by downloading the `KEYS` file directly from Apache's web site (`http://www.apache.org/dist/httpd/KEYS`). The file contains the public keys of all Apache's developers. To import it, simply run `gnupg --import KEYS` (assuming that the file `KEYS` is in your working directory):

```
[merc@localhost merc]$ gpg --import KEYS
gpg: key 2719AF35: public key imported
gpg: /home/merc/.gnupg/trustdb.gpg: trustdb created
[...]
gpg: key DE885DD3: public key imported
gpg: key E005C9CB: public key imported
gpg: Total number processed: 41
gpg:           w/o user IDs: 3
gpg:               imported: 37  (RSA: 22)
gpg:              unchanged: 1
[merc@localhost merc]$
```

You can also download Sander's key by downloading it from a *public key server* using the following command:

```
[merc@merc merc]$ gpg --keyserver pgpkeys.mit.edu --recv-key DE885DD3
gpg: key DE885DD3: "Sander Striker <striker@apache.org>" 59 new signatures
gpg: Total number processed: 1
gpg:         new signatures: 59
[merc@merc merc]$
```

You now have Sander's public key in your key ring. This means that you can check if a message or a file was actually signed by him, by decrypting his signature (using his public key) and comparing the result with the hash value of the message you have received.

Verifying the Downloaded Package

You are now ready to check the package you downloaded. To do this, you will need the signature file from the Apache web site. Again, it is crucial to get the signature file (which is very small) from the main site, rather than from one of its mirrors. For example, if you downloaded version 2.0.48 of Apache, you will need the file `httpd-2.0.48.tar.gz.asc`.

Now, run the command `gpg --verify`, providing both the signature file and the downloaded file as parameters:

```
[merc@merc merc]$ gpg --verify httpd-2.0.48.tar.gz.asc httpd-2.0.48.tar.gz
gpg: please see http://www.gnupg.org/faq.html for more information
gpg: Signature made Mon 07 Jul 2003 22:56:49 WST using DSA key ID DE885DD3
gpg: Good signature from "Sander Striker <striker@apache.org>"
gpg:                 aka "Sander Striker <striker@striker.nl>"
gpg: checking the trustdb
gpg: checking at depth 0 signed=0 ot(-/q/n/m/f/u)=0/0/0/0/0/1
gpg: WARNING: This key is not certified with a trusted signature!
gpg:          There is no indication that the signature belongs to the owner.
Primary key fingerprint: 4C1E ADAD B4EF 5007 579C  919C 6635 B6C0 DE88 5DD3
[merc@merc merc]$
```

The signature is correct ("Good signature from ..."). However, GnuPG warns you not to trust the person, because someone hasn't signed the key with a trusted public key.

In GnuPG, a trusted public key is one that has been signed (and therefore verified) by another trusted key. As soon as you install GnuPG, the only trusted public key is your own. This means that you are able to sign (and therefore verify) Sander's signature. The question is: what makes you think that what you downloaded *really* is his signature? A cracker might have created a public key, called him or herself "Sander Striker," and signed the files you are about to use.

To check the authenticity of a public key, you can check the key's fingerprint. The *GNU Privacy Handbook* states:

> *A key's fingerprint is verified with the key's owner. This may be done in person or over the phone or through any other means as long as you can guarantee that you are communicating with the key's true owner.*

It is up to you to decide what you should do to check the authenticity of the fingerprint, as long as you can make absolutely sure that you are communicating with the real person. This could be a little hard: Sander Striker would have very little time left to develop Apache if he had to meet in person with every single system administrator who wants to check his or her copy of Apache.

The good news is that if you imported the `KEYS` file from Apache's main site, you will also have a collection of public keys owned by Apache's developers, who in turn signed Sander's public key after meeting him. This means that if you verify any one of them, your GnuPG will automatically trust Sander's public key as well. You can obtain a list of developers who signed Sander's public key by using this command:

```
[merc@merc merc]$ gpg --edit-key Sander
[...]
```

```
pub  1024D/DE885DD3  created: 2002-04-10 expires: never      trust: -/-
sub  2048g/532D14CA  created: 2002-04-10 expires: never
(1)  Sander Striker <striker@striker.nl>
(2). Sander Striker <striker@apache.org>
Command> check
uid  Sander Striker <striker@striker.nl>
sig!3       DE885DD3 2002-04-10   [self-signature]
[...]
sig!3       F88341D9 2002-11-18   Lars Eilebrecht <lars@eilebrecht.org>
sig!3       49A563D9 2002-11-23   Mark Cox <mjc@redhat.com>
sig!3       E04F9A89 2002-11-18   Roy T. Fielding <fielding@apache.org>
sig!3       08C975E5 2002-11-21   Jim Jagielski <jim@apache.org>
39 signatures not checked due to missing keys
Command>
```

In this case, you will pretend that you talked to or met Sander Striker in person. You can therefore sign his signature with your public key:

```
Command> sign
Really sign all user IDs? y

pub  1024D/DE885DD3  created: 2002-04-10 expires: never      trust: -/-
 Primary key fingerprint: 4C1E ADAD B4EF 5007 579C  919C 6635 B6C0 DE88 5DD3

     Sander Striker <striker@striker.nl>
     Sander Striker <striker@apache.org>

How carefully have you verified the key you are about to sign actually belongs
to the person named above?  If you don't know what to answer, enter "0".

   (0) I will not answer. (default)
   (1) I have not checked at all.
   (2) I have done casual checking.
   (3) I have done very careful checking.

Your selection? 0
Are you really sure that you want to sign this key
with your key: "Tony Mobily (Myself) <merc@mobily.com>"

Really sign? y

You need a passphrase to unlock the secret key for
user: "Tony Mobily (Myself) <merc@mobily.com>"
1024-bit DSA key, ID B763CD69, created 2003-08-03
```

```
Command> q
Save changes? y
[merc@merc merc]$
```

All done! You now have set GnuPG so that it trusts Sander's signature.

NOTE In a normal situation, most people would just sign Sander's public key, trusting that the KEYS file is original, or checking his key's fingerprints on newsgroups or mailing lists. If security is an issue, you could contact one of the developers who verified Sander's public key's fingerprint by e-mail (sent to the developer's official e-mail address) or by phone (if that's absolutely necessary). Finally, if security is a *real* issue (for example, if it's a bank's web site), they you may decide that you need to meet one of the developers in person. (I imagine this would rarely be necessary.)

Finally Checking Apache

You can now check if your Apache packages have been tampered with or not:

```
[merc@merc merc]$ gpg --verify httpd-2.0.48.tar.gz.asc  httpd-2.0.48.tar.gz
gpg: Signature made Mon 07 Jul 2003 22:56:49 WST using DSA key ID DE885DD3
gpg: Good signature from "Sander Striker <striker@apache.org>"
gpg:                 aka "Sander Striker <striker@striker.nl>"
[merc@merc merc]$
```

What would happen if there were problems? You would receive a warning message from GnuPG. For example:

```
[merc@localhost merc]$ cp httpd-2.0.48.tar.gz httpd-2.0.48.tar.gz.CORRUPTED
[merc@localhost merc]$ ls -l >> httpd-2.0.48.tar.gz.CORRUPTED
[merc@localhost merc]$ gpg --verify httpd-2.0.48.tar.gz.asc httpd-
2.0.48.tar.gz.CORRUPTED
gpg: Signature made Sat 10 Aug 2002 01:51:45 AM WST using DSA key ID DE885DD3
gpg: BAD signature from "Sander Striker <striker@apache.org>"
[merc@localhost merc]$
```

To generate the above warning, I created a spare copy of the Apache server and appended some garbage at the end of the file, making it slightly different. I then ran GnuPG to verify the package and found it faulty. If this error actually occurred, you would have to warn the webmaster immediately of the discrepancy.

NOTE At the address `http://httpd.apache.org/dev/verification.html` you will find a short guide that describes how to check your Apache packages

GNUPG: Is All This Necessary?

At this point, you should have successfully downloaded Apache and ensured that the package is an authentic copy distributed by the Apache Software Foundation. You should also be familiar with GnuPG and have a glimpse of its potential.

Running such thorough checks might seem a bit meticulous, but for a professional system administrator, there is no room for being slack. The main web server or the local mirror may have been hacked, and the downloaded Apache package may have been modified. This scenario, that seemed to be science fiction a few months ago, became reality when the main Debian web server was cracked, and nobody was absolutely sure if *any* of the distribution's packages had been modified. This episode has opened a lot of eyes to the value of signature checking.

Some system administrators consider the MD5 checksum a safe enough method for checking the validity of a package. MD5 is an algorithm that aims to return a truly unique integer number when given a list of bytes in input. This means that the MD5 checksum for two different files is guaranteed to be different. The `md5sum` command can be used to calculate the MD5 checksum of a file, and the result is printed on the standard output. Although MD5 checksums can be useful in checking that a file was downloaded correctly (you can easily run `md5sum` and compare your checksum to what it should be), it should not be used to check that an Apache package is genuine.

Installing Apache

In this section I provide a short explanation on how I installed the Apache servers (both 1.3.*x* and 2.*x* versions) that I will use in the rest of the book.

Apache and Dynamic Modules

Apache comes with a wide set of modules that are not part of the core server, and can be compiled as dynamic modules (they can be loaded into the main server if they are needed). An example of a module in Apache that may be compiled as loadable is autoindex, which is responsible for generating a directory index in HTML (and is therefore well formatted if seen through a browser). This may seem totally useless to your server, but it could be useful later on.

Apache can be built as a static server, or as a dynamic server; it depends on what options you set when you run configure. Apache can actually be built as a mix, with some of the modules built in the main server, and others available as loadable modules.

As far as security is concerned, I believe it is a good idea to compile most of the modules dynamically, and leave the main server stripped to the bones. There are several advantages to doing so:

- You can compile all the modules available, but leave them out of the server to save some memory.
- You can add modules later, without having to recompile the whole server.
- If a security problem is discovered in one of the modules, you can easily disable it until the problem is dealt with. Therefore, you need to configure your Apache so that your web site won't be defaced if you disable any of the modules.
- If a new version of a module comes out (such as PHP), you can easily upgrade it without having to recompile the whole server.

You can get a detailed description of the modules from http://httpd.apache.org/docs-2.0/mod/ or from http://httpd.apache.org/docs/mod/.

Apache 1.3.x

The following are the commands I used to install Apache 1.3.*x* on my server. The options --enable-module=most --enable-shared=max compile most modules as shared objects ("most" excludes mod_auth_db, which is sometimes considered to be problematic to compile, and mod_log_agent and mod_log_referer, which are both deprecated). This Apache's directory will be /usr/local/apache1.

```
 [root@merc apache_source]# tar xvzf apache_1.3.29.tar.gz
apache_1.3.29/
apache_1.3.29/cgi-bin/
apache_1.3.29/cgi-bin/printenv
apache_1.3.29/cgi-bin/test-cgi
[...]
apache_1.3.29/src/support/suexec.8
apache_1.3.29/src/support/suexec.c
```

```
apache_1.3.29/src/support/suexec.h
apache_1.3.29/src/Configuration
[root@merc apache_source]# cd apache_1.3.29
[root@merc apache_1.3.29]# ./configure --prefix=/usr/local/apache1
  --enable-module=most  --enable-shared=max
Configuring for Apache, Version 1.3.29
 + using installation path layout: Apache (config.layout)
[...]
Creating Makefile in src/modules/standard
Creating Makefile in src/modules/proxy
[root@merc apache_1.3.29]# make
===> src
make[1]: Entering directory `/root/apache_source/apache_1.3.29'
make[2]: Entering directory `/root/apache_source/apache_1.3.29/src'
===> src/regex
sh ./mkh  -p regcomp.c >regcomp.i
[...]
make[2]: Leaving directory `/root/apache_source/apache_1.3.29/src/support'
<=== src/support
make[1]: Leaving directory `/root/apache_source/apache_1.3.29'
<=== src
 [root@merc apache_1.3.29]# make install
make[1]: Entering directory `/root/apache_source/apache_1.3.29'
===> [mktree: Creating Apache installation tree]
./src/helpers/mkdir.sh /usr/local/apache1/bin
mkdir /usr/local/apache1
mkdir /usr/local/apache1/bin
[...]
Thanks for using Apache.       The Apache Group
                                http://www.apache.org/
 [root@merc apache_1.3.29]#
```

Apache 2.x

Here is the transcript of the commands I used to install Apache 2.*x* on my server. The option `--enable-mods-shared=most` compiles all the standard modules, and leaves out the ones that are considered experimental:

- `mod_mime_magic`
- `mod_cern_meta`

- mod_user_track
- mod_unique_id

All the modules are compiled dynamically. Also, Apache's main directory will be /usr/local/apache2/.

```
[root@merc apache_source]# tar xvzf httpd-2.0.48.tar.gz
httpd-2.0.48/
httpd-2.0.48/os/
httpd-2.0.48/os/os2/
httpd-2.0.48/os/os2/os.h
 [...]
httpd-2.0.48/include/util_cfgtree.h
httpd-2.0.48/acconfig.h
[root@merc apache_source]# cd httpd-2.0.48
[root@merc httpd-2.0.48]# ./configure --prefix=/usr/local/apache2
  --enable-mods-shared=most
checking for chosen layout... Apache
checking for working mkdir -p... yes
 [...]
config.status: creating build/rules.mk
config.status: creating include/ap_config_auto.h
config.status: executing default commands
[root@merc httpd-2.0.48]# make
Making all in srclib
make[1]: Entering directory `/root/apache_source/httpd-2.0.48/srclib'
Making all in apr
[...]
config.status: creating include/ap_config_auto.h
config.status: include/ap_config_auto.h is unchanged
config.status: executing default commands
 [root@merc httpd-2.0.48]# make install
Making install in srclib
make[1]: Entering directory `/root/apache_source/httpd-2.0.48/srclib'
Making install in apr
[...]
mkdir /usr/local/apache2/manual
Installing build system files
make[1]: Leaving directory `/root/apache_source/httpd-2.0.48'
[root@merc httpd-2.0.48]#
```

Apache is now installed.

Running Apache

You can now start the server and check that everything has worked properly. The best way of doing this is through the script called `apachectl`, located in the `$PREFIX/bin` (in my case, `/usr/local/apache2/bin`). This script is "designed to allow an easy command-line interface to controlling Apache" (quoting the script itself). By running it you will see the options it accepts:

```
[root@localhost ~]# /usr/local/apache2/bin/apachectl start
```

In order to check that the server has actually started, you can run a `ps` command:

```
[root@merc httpd-2.0.48]# ps ax | grep httpd
17072 ?        S      0:00 /usr/local/apache2/bin/httpd -k start
17073 ?        S      0:00 [httpd]
17074 ?        S      0:00 [httpd]
17075 ?        S      0:00 [httpd]
17076 ?        S      0:00 [httpd]
17077 ?        S      0:00 [httpd]
17079 pts/2    S      0:00 grep httpd
[root@merc httpd-2.0.48]#
```

A better way of checking it is through its log file:

```
[root@merc httpd-2.0.48]# tail -f /usr/local/apache2/logs/error_log
[Sun Aug 03 14:30:24 2003] [notice] Digest: generating secret for digest authen-
tication ...
[Sun Aug 03 14:30:24 2003] [notice] Digest: done
[Sun Aug 03 14:30:24 2003] [notice] Apache/2.0.48 (Unix) DAV/2 configured
-- resuming normal operations
```

The server is now listening to port 80 on your computer, and waiting for connections...

Testing Your Apache with Nikto

You should periodically check whether your Apache server is secure or not. To do this by hand can be very hard, as there can be problems that simply slip through. Fortunately, there are several tools for Unix whose sole purpose is testing a server from a security point of view. For example:

- Nessus (`http://www.nessus.org`). This is probably the best known and most powerful vulnerability assessment tool existing today.

- SAINT (`http://www.saintcorporation.com/products/saint_engine.html`). A commercial assessment tool for Unix.

- SARA (`http://www-arc.com/sara/`). An assessment tool derived from SATAN (a now obsolete system security tool that came out in 1995). It's free.

- Nikto (`http://www.cirt.net/code/nikto.shtml`). A scanner that concentrates exclusively on web servers.

NOTE A more comprehensive list of tools is available at `http://www.insecure.org/ tools.html`.

As far as Apache is concerned, the most interesting free solution is Nikto, a tool based on LibWisker(`http://www.wiretrip.net/rfp/`). In this section I will show you how to install Nikto, and run it against the Apache server you just installed.

First of all, you will need to install Net_SSLeay (`http://search.cpan.org/author/SAMPO/Net_SSLeay.pm-1.23`), used by Nikto to establish SSL connections. The installation procedure is the same as with any other Perl module:

```
[root@merc root]# tar xvzf Net_SSLeay.pm-1.23.tar.gz
Net_SSLeay.pm-1.23/
Net_SSLeay.pm-1.23/ptrcasttst.c
[...]
Net_SSLeay.pm-1.23/Credits
Net_SSLeay.pm-1.23/typemap
[root@merc root]# cd Net_SSLeay.pm-1.23
[root@merc Net_SSLeay.pm-1.23]# perl Makefile.PL
Checking for OpenSSL-0.9.6j or 0.9.7b or newer...
[...]
Writing Makefile for Net::SSLeay::Handle
Writing Makefile for Net::SSLeay
[root@merc Net_SSLeay.pm-1.23]# make
cp ptrtstrun.pl blib/lib/Net/ptrtstrun.pl
[...]
chmod 644 blib/arch/auto/Net/SSLeay/SSLeay.bs
```

```
Manifying blib/man3/Net::SSLeay.3pm
[root@merc Net_SSLeay.pm-1.23]# make install
make[1]: Entering directory `/mnt/hda6/home/merc/Net_SSLeay.pm-1.23/Net-SSLeay-
Handle-0.50'
make[1]: Leaving directory `/mnt/hda6/home/merc/Net_SSLeay.pm-1.23/Net-SSLeay-
Handle-0.50'
Files found in blib/arch: installing files in blib/lib into architecture depen-
dent
library tree
Writing /usr/lib/perl5/site_perl/5.8.0/i386-linux-thread-
multi/auto/Net/SSLeay/.packlist
Appending installation info to /usr/lib/perl5/5.8.0/i386-linux-thread-multi/perl-
local.pod
[root@merc Net_SSLeay.pm-1.23]#
```

You will need OpenSSL (http://www.openssl.org) for this module to install. You will then need to download and uncompress Nikto:

```
[root@merc root]# tar xvzf ../nikto-current.tar.gz
nikto-1.30/
nikto-1.30/config.txt
nikto-1.30/docs/
nikto-1.30/docs/CHANGES.txt
[...]
nikto-1.30/plugins/servers.db
[root@merc root]#  cd nikto-1.30/
 [root@merc nikto-1.30]# ls -l
total 20
-rw-r--r--         1 root     sys          2999 May 31 06:52 config.txt
drwxrwxrwx    2 root     sys          4096 Jun 19 06:17 docs
-rwxr-xr-x       1 root     sys          5997 May 31 06:21 nikto.pl
drwxrwxrwx    2 root     sys          4096 Jun 19 06:17 plugins
[root@merc nikto-1.30]#
```

Nikto doesn't need to be installed: it's ready to go as soon as you uncompress it. However, two steps are recommended. The first one is downloading the latest LibWisker from http://www.wiretrip.net/rfp/lw.asp. (Although Nikto comes with LibWisker, it may not be the latest version available).

LibWisker comes as a single LW.pm file. Assuming that you downloaded it and placed it in your home directory, you can copy the new LW.pm file over the existing one in Nikto:

```
[root@merc nikto-1.30]# cd plugins/
[root@merc plugins]# cp ~merc/LW.pm .
```

```
cp: overwrite `./LW.pm'? y
[root@merc plugins]#
```

The second step is to update Nikto's database with the latest database and vulnerability files available from Nikto's web sites. You can do this automatically with Nikto's `-update` option:

```
[root@merc nikto-1.30]# ./nikto.pl -update
+ Retrieving 'realms.db'
+ Retrieving 'server_msgs.db'
+ Retrieving 'nikto_headers.plugin'
+ Retrieving 'nikto_httpoptions.plugin'
+ Retrieving 'servers.db'
+ Retrieving 'nikto_core.plugin'
+ Retrieving 'scan_database.db'
+ Retrieving 'outdated.db'
+ Retrieving 'CHANGES.txt'
getting:/nikto/UPDATES/1.30/CHANGES_nikto.txt
+ www.cirt.net message: Please report any bugs found in the 1.30 version
[root@merc nikto-1.30]#
```

You can now run Nikto, specifying your freshly installed Apache server as the target. In my case, this is the result:

```
[root@merc nikto-1.30]# ./nikto.pl -host localhost
---------------------------------------------------------------------------
- Nikto 1.30/1.13     -     www.cirt.net
+ Target IP:       127.0.0.1
+ Target Hostname: localhost
+ Target Port:     80
+ Start Time:      Sat Aug 16 21:38:43 2003
---------------------------------------------------------------------------
- Scan is dependent on "Server" string which can be faked, use -g to override
+ Server: Apache/2.0.48 (Unix) DAV/2
+ IIS may reveal its internal IP in the Content-Location header. The value is
"index.html.en". CAN-2000-0649.
+ Allowed HTTP Methods: GET,HEAD,POST,OPTIONS,TRACE
+ HTTP method 'TRACE' may allow client XSS or credential theft. See
http://www.cgisecurity.com/whitehat-mirror/WhitePaper_screen.pdf for details.
+ /icons/ - Directory indexing is enabled, it should only be enabled for specific
directories (if required). If indexing is not used all, the /icons directory should
be removed. (GET)
+ /index.html.ca - Apache default foreign language file found. All default files
should be removed from the web server as they may give an attacker additional
```

```
system information. (GET)
[...]
+ /index.html.var - Apache default foreign language file found. All default files
should be removed from the web server as they may give an attacker additional
system information. (GET)
+ /manual/images/ - Apache 2.0 directory indexing is enabled, it should only be
enabled for specific directories (if required). Apache's manual should be removed
and directory indexing disabled. (GET)
+ / - TRACE option appears to allow XSS or credential theft. See
http://www.cgisecurity.com/whitehat-mirror/WhitePaper_screen.pdf for details
(TRACE)
+ /manual/ - Web server manual? tsk tsk. (GET)
+ 1688 items checked - 30 items found on remote host
+ End Time:        Sat Aug 16 21:41:08 2003 (145 seconds)
----------------------------------------------------------------------------
[root@merc nikto-1.30]#
```

Apache was only just installed, and it already has several problems! Nikto pointed out the following issues:

- The method TRACE is enabled; this could lead to cross-site scripting attacks. (Chapter 4 in this book covers this type of attack.) The report points to http://www.betanews.com/whitehat/WH-WhitePaper_XST_ebook.pdf (if the link doesn't work, you should search on Google using the keywords *WH-WhitePaper_XST_ebook.pdf* or *WhitePaper_screen.pdf*). It's a very clear document that explains the issue in detail. Many system administrators consider it a non-issue, because it's used to tell the clients what they already know. If you want extra peace of mind, disable TRACE (the next section describes how to do this).

- The directories /icons and /manual/images allow indexing. This should be disabled for production servers (remember that both these directories reside under ServerRoot, and not in the machine's root directory /).

- Apache's manual is still installed. It should be deleted on production servers for three reasons. The main one is that you don't want to give away too much information on the server you are running (a cracker can work out at least if you are running Apache 1.3.*x* or Apache 2.0.*x* by seeing the manual). Also, you want to be fully in control of what your web server is actually serving. Finally, it's a matter of style: the presence of the manual often means that the system administrator didn't spend too long configuring the server properly.

- Several default HTML index pages were found.

Most of the problems came from the fact that absolutely nothing was done after installing Apache—the manual and the default `index.html` files weren't even deleted.

Nikto has several more options. You can, for example, enable one or more intrusion detection system (IDS) evasion techniques, or scan a predefined port range, use a proxy server, and so on. Please refer to Nikto's documentation for more information.

You should keep Nikto handy, and run it periodically (and after any server upgrades). You should also consider using other vulnerability assessment tools as well as Nikto (see Appendix A for a list of some of these tools).

Secure Configuration

Every server program comes with a prepackaged configuration file that can often be left nearly intact (think of FTP or Sendmail). Apache is different; its configuration is rather complicated. The standard configuration file provided with it is meant to show most of its capabilities, rather than a perfectly configured server. Many (if not most) system administrators only apply minor changes to the standard `http.conf` file; therefore, capabilities such as WebDav and multilanguage support (for example) are often found in English-only sites, which have no intention of offering WebDav functionalities. I will now show you an alternative approach to Apache configuration.

A Basic httpd.conf File

The idea is to configure Apache starting from an empty `httpd.conf` file, and add only the modules that are strictly necessary. In this example I will cover Apache 2.*x* running on Linux, but the same concepts can be applied to Apache 2.*x* and Apache 1.3.*x* on any platform.

To do this, you should create a backup copy of the default `httpd.conf` file first, which can be used as a reference in the future. You should then delete all the module-dependent MPM options that don't apply to you. MPM stands for Multi-Processing Module, and is a mechanism used by Apache to manage multiple threads accepting connections at the same time. Your Apache server will need at least one MPM module. A list of available modules is available at `http://httpd.apache.org/docs-2.0/mod/` in the "Core Features and Multi-Processing Modules" section. Normally, newly installed servers use the standard and well-established `prefork` MPM. If you are not sure what MPM you are using, you can use the `httpd -l` command, like this:

```
[root@merc root]# /usr/local/apache2/bin/httpd -l
Compiled in modules:
  core.c
  prefork.c
  http_core.c
  mod_so.c
[root@merc root]#
```

You can delete from your `httpd.conf` file entries such as:

```
<IfModule perchild.c>
NumServers           5
StartThreads         5
MinSpareThreads      5
MaxSpareThreads     10
MaxThreadsPerChild  20
MaxRequestsPerChild  0
</IfModule>
```

You can also delete all the `<ifModule prefork.c>` directives from your `httd.conf` file.

You should then:

1. Comment out all the `LoadModule` directives. Note that it is best to keep these directives in your `httpd.conf` file. You will need some of them, but for now just comment them all out and add the ones you need later.

2. Delete all the options that you are not going to use in the short term. For example: `IndexOptions` (and all the `AddIcon` directives), multi-language support (assuming that you are not going to use it), and anything else you are not likely to use.

The goal should be to have a clear, easily readable `httpd.conf` file that is fully understandable and easy to maintain, by you and by the system administrators who will change it in the future. Apache's documentation is a very precious aid. These links are especially useful: `http://httpd.apache.org/docs-2.0/mod/` and `http://httpd.apache.org/docs-2.0/mod/directives.html`. You should use this as an opportunity to fully understand what each option does, and why you should keep it on your server. When you are deciding whether to keep something or not, remember that the shorter your file is, the better it is, and that you can always copy lines over from the backup `httpd.conf` file.

This is what your `httpd.conf` file should look like after all the trimming (remember that you can, and should, add more comments for clarity):

```
ServerRoot "/usr/local/apache2"

# Server's options
Timeout 300
KeepAlive On
MaxKeepAliveRequests 100
KeepAliveTimeout 15

# This is a prefork server
StartServers         5
MinSpareServers      5
MaxSpareServers     10
MaxClients         150
MaxRequestsPerChild  0

Listen 80

#LoadModule access_module modules/mod_access.so
#LoadModule auth_module modules/mod_auth.so
#LoadModule auth_anon_module modules/mod_auth_anon.so
#LoadModule auth_dbm_module modules/mod_auth_dbm.so
#LoadModule auth_digest_module modules/mod_auth_digest.so
#LoadModule ext_filter_module modules/mod_ext_filter.so
#LoadModule include_module modules/mod_include.so
#LoadModule log_config_module modules/mod_log_config.so
#LoadModule env_module modules/mod_env.so
#LoadModule expires_module modules/mod_expires.so
#LoadModule headers_module modules/mod_headers.so
#LoadModule setenvif_module modules/mod_setenvif.so
#LoadModule mime_module modules/mod_mime.so
#LoadModule dav_module modules/mod_dav.so
#LoadModule status_module modules/mod_status.so
#LoadModule autoindex_module modules/mod_autoindex.so
#LoadModule asis_module modules/mod_asis.so
#LoadModule info_module modules/mod_info.so
#LoadModule cgi_module modules/mod_cgi.so
#LoadModule dav_fs_module modules/mod_dav_fs.so
#LoadModule vhost_alias_module modules/mod_vhost_alias.so
#LoadModule negotiation_module modules/mod_negotiation.so
#LoadModule dir_module modules/mod_dir.so
#LoadModule imap_module modules/mod_imap.so
#LoadModule actions_module modules/mod_actions.so
#LoadModule speling_module modules/mod_speling.so
```

```
#LoadModule userdir_module modules/mod_userdir.so
#LoadModule alias_module modules/mod_alias.so
#LoadModule rewrite_module modules/mod_rewrite.so

# Change the server's owner
User nobody
Group nobody

# Server info
ServerAdmin me@mobily.com
ServerName www.server.com:80
UseCanonicalName Off

DocumentRoot "/usr/local/apache2/htdocs"

# Minimal permissions for any directory
<Directory />
    Options -FollowSymLinks
    AllowOverride None
</Directory>

# More permissive options for sub-directories.
<Directory "/usr/local/apache2/htdocs">
    AllowOverride None
    Order allow,deny
    Allow from all
</Directory>

DirectoryIndex index.html

# Security filters, saves .htaccess files
<Files ~ "^\.ht">
    Order allow,deny
    Deny from all
    Satisfy All
</Files>

# Mime types information
TypesConfig conf/mime.types
DefaultType text/plain
```

```
# Logging
HostnameLookups Off
ErrorLog logs/error_log
LogLevel warn
LogFormat "%h %l %u %t \"%r\" %>s %b" common
CustomLog logs/access_log common

# Info given out. It can be Full,OS,Minor,Minimal,Major,Prod
ServerTokens Prod
ServerSignature Off

# CGI SCRIPTS
ScriptAlias /cgi-bin/ "/usr/local/apache2/cgi-bin/"
<Directory "/usr/local/apache2/cgi-bin">
    AllowOverride None
    Options None
    Order allow,deny
    Allow from all
</Directory>

# Set the default charset, prevents XSS
AddDefaultCharset ISO-8859-1

# Ugly but important hacks
BrowserMatch "Mozilla/2" nokeepalive
BrowserMatch "MSIE 4\.0b2;" nokeepalive downgrade-1.0 force-response-1.0
BrowserMatch "RealPlayer 4\.0" force-response-1.0
BrowserMatch "Java/1\.0" force-response-1.0
BrowserMatch "JDK/1\.0" force-response-1.0
BrowserMatch "Microsoft Data Access Internet Publishing Provider" redirect-carefully
BrowserMatch "^WebDrive" redirect-carefully
BrowserMatch "^WebDAVFS/1.[012]" redirect-carefully
BrowserMatch "^gnome-vfs" redirect-carefully
```

This file as it is won't work. When you try to restart Apache, you will get a message like this:

```
[root@merc htdocs]# /usr/local/apache2/bin/apachectl start
Syntax error on line 73 of /usr/local/apache2/conf/httpd.conf:
Invalid command 'Order', perhaps mis-spelled or defined by a module not included
in the server configuration
[root@merc htdocs]#
```

The reason is simple: Apache is lacking the module that is responsible for making `Order` an acceptable configuration directive.

You can find `Order` at `http://httpd.apache.org/docs-2.0/mod/directives.html`, and after clicking on it you are taken to a page that will tell you what module defines it. In this case, you can read

```
Module: mod_access
```

You then have to uncomment the following line:

```
# LoadModule access_module modules/mod_access.so
```

You will need to repeat this process a number of times. Eventually, you will probably uncomment the following lines:

```
LoadModule access_module modules/mod_access.so
LoadModule log_config_module modules/mod_log_config.so
LoadModule setenvif_module modules/mod_setenvif.so
LoadModule mime_module modules/mod_mime.so
LoadModule dir_module modules/mod_dir.so
LoadModule alias_module modules/mod_alias.so
```

Finally, you should also uncomment the following line to enable `mod_rewrite`, an important module for security:

```
LoadModule rewrite_module modules/mod_rewrite.so
```

Other Recommendations

It is hard to summarize in a few points what to do to make your Apache configuration more secure. The most important advice is to carefully read the available documentation, find out exactly what each directive does, and simply do not use anything unless it's necessary.

In this section I will highlight configuration options that you should be aware of to keep your Apache server more secure. Some of them are from the page `http://httpd.apache.org/docs-2.0/misc/security_tips.html`.

File Permissions

Make sure that Apache's files and directories are only writable by root. These are the commands suggested by Apache's web site:

```
# cd /usr/local/apache
# chown 0 . bin conf logs
# chgrp 0 . bin conf logs
# chmod 755 . bin conf logs
# chown 0 /usr/local/apache/bin/httpd
# chgrp 0 /usr/local/apache/bin/httpd
# chmod 511 /usr/local/apache/bin/httpd
```

Having the wrong permissions set could allow a malicious user to replace the `httpd` binary, and therefore execute a script as root.

Understand How Options Are Applied

Most directives can be defined in different sections of your `httpd.conf` file:

1. `<Directory>` sections
2. `<DirectoryMatch>` sections
3. `<Files>` and `<FilesMatch>` sections
4. `<Location>` and `<LocationMatch>` sections
5. `<VirtualHost>` sections
6. `.htaccess` (if it is allowed)

Apache merges the directives found in these sections following this particular order. Also, each section is processed in the order that it appears in the configuration file, except `<Directory>`, where the shortest directories are processed first. Also, `<Directory>` refers to actual directories on the file system, whereas `<Location>` refers to a Web location.

The configuration files I proposed earlier in the chapter read:

```
<Directory />
    Options -FollowSymLinks
    AllowOverride None
</Directory>
```

The directive `<Directory />` refers to the actual root directory of the server.

Immediately after, you can see the following directive, which is processed *after* the previous one, as it refers to a longer path:

```
<Directory "/usr/local/apache2/htdocs">
     AllowOverride None
    Order allow,deny
    Allow from all
</Directory>
```

In this case, I set very restrictive permissions for the server's root directory, and then I allowed looser security for the document root (`/usr/local/apache2/htdocs`). I could have defined different options for a directory inside `htdocs`:

```
<Directory "/usr/local/apache2/htdocs/insecure">
    Options +FollowSymLinks +AllowOverride
    AllowOverride None
    Order allow,deny
    Allow from all
</Directory>
```

For the directory `insecure`, the options `FollowSymLinks` (which allows the following of symbolic links, and should be turned off) and `AllowOverride` (which enables the .htaccess file mechanism, and should be disabled if possible) are *added* (thanks to the + symbol) to the inherited options.

You should be aware of how this mechanism works. The document `http://httpd.apache.org/docs-2.0/sections.html` explains exactly how Apache reads its configuration section, and `http://httpd.apache.org/docs-2.0/mod/core.html#options` explains what options can be set. A complete understanding of both these documents is vital to ensure the security of your Apache installation.

Don't Expose Root's Home Page

If you need to allow users' directories (such as `http://www.site.com/~username`) with the directive `Userdir`, remember to have this line in your configuration:

```
UserDir disabled root
```

Delete Any Default Files

The CGI scripts `printenv` and `test-cgi` are installed by default. They have caused problems in the past, because standard Apache scripts that came with the server were inherently vulnerable, and a source of security breaches and problems. Now, when Apache is installed they are not executable. However, in a production environment they should be deleted:

```
[root@merc root]# cd /usr/local/apache2/
[root@merc apache2]# ls cgi-bin
printenv  test-cgi
[root@merc apache2]# rm cgi-bin/*
rm: remove regular file `cgi-bin/printenv'? y
rm: remove regular file `cgi-bin/test-cgi'? y
[root@merc apache2]#
```

The same applies to the default web site, which should be deleted:

```
[root@merc root]# rm -rf /usr/local/apache2/htdocs/*
```

You should then place your own web site in the `htdocs` directory.

Don't Give Extra Information Away

Don't let people know what version of Apache you are running by the HTTP response header. You can easily do this with this option (this will only display `Apache`, rather than `Apache/2.0.48 (Unix)`):

```
ServerTokens Minimal
```

Disable the Method TRACE

This is a controversial issue at the moment, because many system administrators are ready to swear that it's only a client-side issue. However, it's better to be safe than sorry. `TRACE` is supposed to make it possible to execute cross-site scripting attacks (see Chapter 4). You can disable `TRACE` with the following `mod_rewrite` directives (more on `mod_rewrite` in the next section):

```
RewriteEngine on
RewriteCond %{REQUEST_METHOD} ^TRACE
RewriteRule .*   [F]
```

Remember to place these directives in a `<Location />` container, or outside all containers.

Running Nikto Again

You should now run Nikto again and see if the report is any different. Here is the result in my case:

```
[root@merc nikto-1.30]#  ./nikto -h localhost
---------------------------------------------------------------------------
- Nikto 1.30/1.15     -     www.cirt.net
+ Target IP:       127.0.0.1
+ Target Hostname: localhost
+ Target Port:     80
+ Start Time:      Sun Aug 17 15:19:05 2003
---------------------------------------------------------------------------
- Scan is dependent on "Server" string which can be faked, use -g to override
+ Server: Apache/2.0.48 (Unix)
+ No CGI Directories found (use -a to force check all possible dirs)
+ Allowed HTTP Methods: GET,HEAD,POST,OPTIONS,TRACE
+ HTTP method 'TRACE' is typically only used for debugging. It should be disabled.
+ 1137 items checked - 0 items found on remote host
+ End Time:        Sun Aug 17 15:21:08 2003 (123 seconds)
---------------------------------------------------------------------------
[root@merc nikto-1.30]#
```

Blocking Access to Your Site

Sometimes, you'll need to block access to your web site (or sites) from particular IP addresses. Generally, there are two ways of doing this: using mod_access directives, or using mod_rewrite. mod_access is easier to use, but it's limited. On the other hand, mod_rewrite is very powerful, but it's notoriously hard to use.

Using mod_access

mod_access is described in detail at http://httpd.apache.org/docs-2.0/mod/mod_access.html. It has three options: Allow, Deny, and Order. Here is an example taken from the httpd.conf file I presented earlier in this chapter:

```
<Directory "/usr/local/apache2/htdocs">
     AllowOverride None
    Order allow,deny
    Allow from all
</Directory>
```

The Allow and Deny directives are always followed by from, and then either an IP address, or a string like env=*env-variable* (where *env-variable* is an environment variable such as REMOTE_HOST; the full list is at http://hoohoo.ncsa.uiuc.edu/cgi/env.html). Remember that you can also define arbitrary environment variables using setenvif, browsermatch, and rewriterule, and these can all be used in the env= clauses.

Order can have two parameters: Deny,Allow or Allow,Deny. The use of this directive is sometimes a source of confusion. The following is from the Apache documentation (the emphasis is mine):

- Deny,Allow. The Deny directives are evaluated before the Allow directives. **Access is allowed by default.** Any client which does not match a Deny directive or does match an Allow directive will be allowed access to the server.

- Allow,Deny. The Allow directives are evaluated before the Deny directives. **Access is denied by default.** Any client which does not match an Allow directive or does match a Deny directive will be denied access to the server.

If you are used to configuring firewalls, you must remember that here *all* the Allow and Deny directives are evaluated before making a decision. In the short example earlier, the order is allow,deny. This means that *all* the Allow directives will be evaluated first, and then *all* the Deny directives are evaluated (in this case, there aren't any), even if some of the Allow directives are matched.

In the example the order is allow,deny. This means that the default is deny. Apache first evaluates the Allow directive, which is followed by all—that is, everybody is allowed. Because there are no Deny directives to evaluate, the access is granted.

Now consider the following code, taken from the same httpd.conf file:

```
<Files ~ "^\.ht">
    Order allow,deny
    Deny from all
    Satisfy all
</Files>
```

This filter applies to files that match the pattern "^\.ht", that is, files whose names start with ".ht". The order is Allow,Deny; therefore, the default status is Deny. The Allow directives are processed first—in this case, there aren't any. Then, the Deny directive is processed: Deny from all. Consequently, the default status remains deny, and access to the resource is never granted.

To block any connection from hosts outside the network 192.168.1.0, you can write:

```
Order Deny,Allow
Deny from all
Allow from 192.168.1.0/24
```

To block a particular IP address you can use something like this:

```
Order Allow, Deny
Allow from all
Deny from bad_ip_address_here
```

For more comprehensive information about `mod_access`, consult `http://httpd.apache.org/docs-2.0/mod/mod_access.html`. Remember also that it's always wiser to have access control from a firewall, rather than from Apache directly. However, you can use `mod_access` as a temporary solution in case of emergencies.

Using mod_rewrite

`mod_rewrite` is a module that lets you manipulate URLs. To use `mod_rewrite`, you need to master *regular expressions*. Explaining regular expressions is outside the scope of this book. They are very powerful, and if you don't have much experience with them, it's certainly worth your while to study them, because they are used extensively in Apache configuration. A search for "regular expressions tutorial" returned these interesting results:

- `http://gnosis.cx/publish/programming/regular_expressions.html`
- `http://www.english.uga.edu/humcomp/perl/regex2a.html`
- `http://etext.lib.virginia.edu/helpsheets/regex.html`

After reading some tutorials, you could also extend your knowledge by studying the official documentation for Perl's regular expressions: `http://www.perldoc.com/perl5.8.0/pod/perlre.html`. Remember that the library used by Perl for regular expressions is normally more powerful than the one used by Apache, and some of the features mentioned at that site might be missing in Apache. However, this URL above is an excellent read and fully explains regular expressions.

The most basic use of mod_rewrite is the following:

```
RewriteEngine on
RewriteRule ^/test$  /index.html
```

Any request to /test is "rewritten" into /index.html (in regular expressions, ^ and $ are the beginning and the end of a string, respectively). You can test this on your server: you should receive the index.html page when you request http://localhost/test.

Another common use of mod_rewrite is to deny access to a specific resource using a third parameter, [F], in the rewriting rule. Consider the following:

```
RewriteEngine on
RewriteRule \.htaccess$ - [F]
```

In this case, access to any URL ending with .htaccess (.htaccess, foo.htaccess, anything.htaccess, and so on) will be forbidden. I had to escape the period character (.) with a backslash, because in a regular expression, a dot means "any character." In this case what the URL is rewritten into (-) is irrelevant.

You can put a condition to your rule rewriting using the RewriteCond directive. For example, you could deny access to your site only to a specific IP address using REMOTE_ADDR:

```
RewriteEngine on
RewriteCond %{REMOTE_ADDR} ^192.168.0.200$
RewriteRule .* - [F]
```

You can also base such a check on any environment variable (that is, the ones that aren't predefined and known to mod_rewrite) using the syntax %{ENV:any_arbitrary_variable}.

The RewriteRule directive that follows a RewriteCond is only executed if the condition in RewriteCond is satisfied. You can have several conditions:

```
RewriteCond %{REMOTE_ADDR} ^192.168.0.201$
RewriteRule /shop\.html - [F]

RewriteCond %{REMOTE_ADDR} ^192.168.0.200$
RewriteRule .* - [F]
```

192.168.0.201 will only be denied access to the page shop.html, whereas the IP 192.168.0.200 will be denied access to any URL (in regular expressions, . means "any character," and * means "0 or more of the previous character").

You can concatenate several RewriteCond directives as follows:

```
RewriteCond %{REMOTE_ADDR} ^192.168.0.200$ [OR]
RewriteCond %{REMOTE_ADDR} ^192.168.0.201$
RewriteRule .* - [F]
```

This will prevent access to any URL from the IPs 192.168.0.200 and 192.168.0.201. Using mod_rewrite, you can prevent other people from using images from your web site:

```
RewriteEngine on
RewriteCond %{HTTP_REFERER} !^$
RewriteCond %{HTTP_REFERER} !^http://www.your_site.com/images/.*$ [NC]
RewriteRule .*\.gif$ -[F]
```

The first RewriteCond directive is satisfied if the HTTP_REFERER variable is not empty (the ! means "not," and ^$ is an empty string). The second condition is satisfied if the HTTP_REFERER variable is not http://www.your_site.com/images/ (NC stands for "case insensitive"). If both conditions are satisfied, the request is likely to be coming from a browser that is not visiting www.your_site.com. Therefore, any request of files ending with .gif is rejected. (Unfortunately the referrer is often spoofed or obscured, so this is not a very useful technique.)

This is only the very tip of the iceberg: mod_rewrite can do much more, and its rules and conditions are very elaborate. You can find extensive documentation on mod_rewrite here: http://httpd.apache.org/docs-2.0/mod/mod_rewrite.html. When you feel brave, you can have a look at http://httpd.apache.org/docs-2.0/misc/rewriteguide.html, which provides a number of practical solutions using mod_rewrite. At the URL http://www.promotiondata.com/sections.php?op=listarticles&secid=1, you will find a simple tutorial that should help you get started.

Apache and SSL

At the beginning of this chapter I introduced cryptography as a means of checking that the Apache package I downloaded was correct (digital signatures).

SSL (Secure Sockets Layer) is a protocol used by a web browser (and therefore Apache) to establish an encrypted connection. It is common to see SSL on sites that accept confidential information from their client (for example, credit card numbers or personal details). In this section, I will explain how to compile Apache with mod_ssl, generate the relevant certificate, and have it signed.

To understand SSL in general, you can (and should) read the documentation for mod_ssl at http://www.modssl.org/docs/2.8/. The second chapter, `http://www.modssl.org/ docs/2.8/ssl_intro.html`, is an excellent article based on Frederick Hirsch's paper "Introducing SSL and Certificates using SSLeay." Remember that this documentation is valid for `mod_ssl` as a stand-alone third-party module, and therefore it may not be perfectly applicable to the `mod_ssl` bundled with Apache 2.*x*.

Installation for Apache 1.3.x

You will first need to download `mod_ssl` from `http://www.modssl.org/`, making sure that you select the right package for your version of Apache (in my case, mod_ssl-2.8.14-1.3.29.tar.gz for Apache 1.3.29).

You should then read the `INSTALL` file, which comes with the package and details all the installation options. I would recommend following the instructions marked as "The flexible APACI-only way," which show you how to install any third-party modules in Apache (as well as SSL).

Here is the transcript of my installation, which should have exactly the same result as the one I showed at the beginning of the chapter (in this case, OpenSSL was already installed on the target system):

```
[root@merc apache_source]# tar xvzf apache_1.3.29.tar.gz
apache_1.3.29/
apache_1.3.29/cgi-bin/
apache_1.3.29/cgi-bin/printenv
[...]
apache_1.3.29/src/support/suexec.c
apache_1.3.29/src/support/suexec.h
apache_1.3.29/src/Configuration
[root@merc apache_source]#
[root@merc apache_source]# tar xvzf mod_ssl-2.8.14-1.3.29.tar.gz
mod_ssl-2.8.14-1.3.29/ANNOUNCE
mod_ssl-2.8.14-1.3.29/CHANGES
mod_ssl-2.8.14-1.3.29/CREDITS
mod_ssl-2.8.14-1.3.29/INSTALL
[...]
mod_ssl-2.8.14-1.3.29/pkg.sslsup/mkcert.sh
mod_ssl-2.8.14-1.3.29/pkg.sslsup/sslsup.patch
[root@merc apache_source]#
[root@merc apache_source]# cd mod_ssl-2.8.14-1.3.29/
[root@merc mod_ssl-2.8.14-1.3.29]# ./configure --with-apache=../apache_1.3.29
Configuring mod_ssl/2.8.14 for Apache/1.3.29
 + Apache location: ../apache_1.3.29 (Version 1.3.29)
 [...]
```

```
[root@merc mod_ssl-2.8.14-1.3.29]# cd ..
[root@merc apache_source]# cd apache_1.3.29
[root@merc apache_1.3.29]# SSL_BASE=/usr ./configure --enable-module=ssl
--prefix=/usr/local/apache1 --enable-module=most  --enable-shared=max
[...]
Creating Makefile in src/modules/extra
Creating Makefile in src/modules/proxy
Creating Makefile in src/modules/ssl
[root@merc apache_1.3.29]# make
===> src
make[1]: Entering directory `/root/apache_source/apache_1.3.29'
make[2]: Entering directory `/root/apache_source/apache_1.3.29/src'
===> src/regex
[...]
+----------------------------------------------------------------------+
make[1]: Leaving directory `/root/apache_source/apache_1.3.29'
<=== src
[root@merc apache_1.3.29]# make install
make[1]: Entering directory `/root/apache_source/apache_1.3.29'
===> [mktree: Creating Apache installation tree]
./src/helpers/mkdir.sh /usr/local/apache1/bin
mkdir /usr/local/apache1
mkdir /usr/local/apache1/bin
./src/helpers/mkdir.sh /usr/local/apache1/bin
./src/helpers/mkdir.sh /usr/local/apache1/libexec
[...]
| Thanks for using Apache. The Apache Group         |
|                       http://www.apache.org/  |
+--------------------------------------------------------------------------------+
[root@merc apache_1.3.29]#
```

Your Apache installation should now be ready to go.

Installation for Apache 2.x

mod_ssl is included in Apache 2.*x*; this makes its installation very simple. All you have to do is add two options to the ./configure script: --enable-ssl (to enable SSL) and -with-ssl=/openssl_directory (to specify OpenSSL's base directory).

Here is the installation transcript:

```
[root@merc httpd-2.0.48]# ./configure --prefix=/usr/local/apache2 --enable-mods-
shared=most --enable-ssl --with-ssl=/usr
checking for chosen layout... Apache
checking for working mkdir -p... yes
checking build system type... i686-pc-linux-gnu
checking host system type... i686-pc-linux-gnu
checking target system type... i686-pc-linux-gnu

Configuring Apache Portable Runtime library ...

checking for APR... reconfig
[...]
checking for SSL/TLS toolkit base... /usr
checking for SSL/TLS toolkit version... OpenSSL 0.9.7a Feb 19 2003
checking for SSL/TLS toolkit includes... /usr/include
checking for SSL/TLS toolkit libraries... /usr/lib
  adding "-I/usr/include/openssl" to INCLUDES
  setting LIBS to "-lssl -lcrypto"
checking for SSL_set_state... no
checking for SSL_set_cert_store... no
checking whether to enable mod_ssl... shared (most)
[...]
config.status: executing default commands
 [root@merc httpd-2.0.48]# make
Making all in srclib
make[1]: Entering directory `/root/apache_source/httpd-2.0.48/srclib'
Making all in apr
[...]
make[2]: Leaving directory `/root/apache_source/httpd-2.0.48/support'
make[1]: Leaving directory `/root/apache_source/httpd-2.0.48'
[root@merc httpd-2.0.48]# make install
make install[root@merc httpd-2.0.48]# make install
Making install in srclib
make[1]: Entering directory `/root/apache_source/httpd-2.0.48/srclib'
Making install in apr
make[2]: Entering directory `/root/apache_source/httpd-2.0.48/srclib/apr'
Making all in strings
Installing build system files
make[1]: Leaving directory `/root/apache_source/httpd-2.0.48'
[...]
[root@merc httpd-2.0.48]#
```

Generating the Certificates

Before you start Apache, you need to generate the server's private key. You can use the following command:

```
[root@merc root]# openssl genrsa -des3 -out server.key 1024
Generating RSA private key, 1024 bit long modulus
...++++++
.++++++
e is 65537 (0x10001)
Enter pass phrase for server.key: ******
Verifying - Enter pass phrase for server.key: ******
[root@merc root]#
```

You then need to create a Certificate Signing Request (CSR), using your server's private key:

```
[root@merc root]# openssl req -new -key server.key -out server.csr
Enter pass phrase for server.key:
You are about to be asked to enter information that will be incorporated
[...]
-----
Country Name (2 letter code) [GB]:AU
State or Province Name (full name) [Berkshire]:WA
Locality Name (eg, city) [Newbury]:Fremantle
Organization Name (eg, company) [My Company Ltd]:Mobily.com
Organizational Unit Name (eg, section) []:
Common Name (eg, your name or your server's hostname) []:www.mobily.com
Email Address []:my_address@mobily.com
 Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:
An optional company name []:
[root@merc root]#
```

You should send the generated file, `server.csr`, to a Certificate Authority (CA). After verifying your details, they will reply with a proper certificate (the file would be probably called `server.crt`).

If you want to test your server, you will need to create your own CA first:

```
[root@merc root]# openssl genrsa -des3 -out ca.key 1024
Generating RSA private key, 1024 bit long modulus
.........++++++
........++++++
e is 65537 (0x10001)
Enter pass phrase for ca.key: ******
Verifying - Enter pass phrase for ca.key: ******
```

You now need to create a self-signed CA certificate:

```
[root@merc root]# openssl req -new -x509 -days 365 -key ca.key -out ca.crt
Enter pass phrase for ca.key: ******
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [GB]:AU
State or Province Name (full name) [Berkshire]:Test
Locality Name (eg, city) [Newbury]:Test
Organization Name (eg, company) [My Company Ltd]:Test
Organizational Unit Name (eg, section) []:Test
Common Name (eg, your name or your server's hostname) []:
Email Address []:
[root@merc root]#
```

You should now use the script `sign.sh` to sign your `server.csr` file with your newly created certifying authority:

```
[root@merc root]# apache_source/mod_ssl-2.8.14-1.3.29/pkg.contrib/sign.sh
server.csr
CA signing: server.csr -> server.crt:
Using configuration from ca.config
Enter pass phrase for ./ca.key:
Check that the request matches the signature
Signature ok
The Subject's Distinguished Name is as follows
countryName           :PRINTABLE:'AU'
stateOrProvinceName   :PRINTABLE:'WA'
localityName          :PRINTABLE:'Fremantle'
organizationName      :PRINTABLE:'Mobily.com'
commonName            :PRINTABLE:'www.mobily.com'
emailAddress          :IA5STRING:'merc@mobily.com'
Certificate is to be certified until Aug 17 04:42:23 2004 GMT (365 days)
Sign the certificate? [y/n]:y
 1 out of 1 certificate requests certified, commit? [y/n]y
```

```
Write out database with 1 new entries
Data Base Updated
CA verifying: server.crt <-> CA cert
server.crt: OK
```

You now have the files `server.crt` (your certificate) and `server.key` (your server's private key).

Configuration

You need to place the files `server.crt` and `server.key` in your conf directory:

```
[root@merc root]# cp server.crt server.key /usr/local/apache2/conf/
```

For simplicity's sake, I will place all the SSL directives in a different file. Normally, they would reside in the main `httpd.conf` file, inside `<VirtualHost>` directives. In this example, I will place an `include` directive in the `httpd.conf` file:

```
Include conf/ssl.conf
```

You can now set your `ssl.conf` file. Here is an example:

```
# Some mime types
AddType application/x-x509-ca-cert .crt
AddType application/x-pkcs7-crl    .crl

# Server-wide options
Listen 443

SSLPassPhraseDialog  builtin
SSLSessionCache         dbm:logs/ssl_scache
SSLSessionCacheTimeout  300
SSLMutex  file:logs/ssl_mutex
SSLRandomSeed startup builtin
SSLRandomSeed connect builtin

<VirtualHost _default_:443>
    DocumentRoot "/usr/local/apache2/htdocs"
    ServerName new.host.name:443
    ServerAdmin you@your.address
    ErrorLog logs/error_log
    TransferLog logs/access_log

    # Enable SSL with specific encryption methods
    SSLEngine on
    SSLCipherSuite ALL:!ADH:!EXPORT56:RC4+RSA:+HIGH:+MEDIUM:+LOW:+SSLv2:+EXP:
       +eNULL

   # Set the server's key and certificate
   SSLCertificateFile /usr/local/apache2/conf/server.crt
   SSLCertificateKeyFile /usr/local/apache2/conf/server.key

   # Set specific options
   <Files ~ "\.(cgi|shtml|phtml|php3?)$">
       SSLOptions +StdEnvVars
   </Files>
   <Directory "/usr/local/apache2/cgi-bin">
       SSLOptions +StdEnvVars
   </Directory>

   # Ugly hack
   SetEnvIf User-Agent ".*MSIE.*" \
         nokeepalive ssl-unclean-shutdown \
         downgrade-1.0 force-response-1.0

   # ssl logging
   CustomLog logs/ssl_request_log \
          "%t %h %{SSL_PROTOCOL}x %{SSL_CIPHER}x \"%r\" %b"

</VirtualHost>
```

You can choose a different server key and certificate for each virtual server you manage. Also, note that Apache will ask you to key in the `server.key`'s passphrase every time it starts.

Remember that this is only a basic configuration, and shouldn't be used in a production server. For more detailed information about configuring `mod_ssl`, please read Chapter 3 of the official documentation at `http://www.modssl.org/docs/2.8/ssl_reference.html`. In this section I explained briefly how to install and configure `mod_ssl`. I didn't get into details mainly because the available documentation is excellent.

Checkpoints

- Obtain the Apache package from a secure source (such as `http://httpd.apache.org`), or your distribution's FTP site or CD-ROM.
- Check the integrity of the package you obtain (using GnuPG, MD5, or using the tools provided by your distribution).
- Be aware of exactly what each directive does, and what possible consequences the directives have for your server's security. You should configure Apache so that `httpd.conf` contains only the directives you actually need.
- Apply all the basic security checks on your configuration: file permissions, protection of root's home page, deletion of any default files, disabling of any extra information on your server, and disabling of the `TRACE` method.
- Make sure that you have protected important files (such as `.htaccess`) using `mod_access`; and make sure that you need to make minimal modifications to your `httpd.conf` file (uncomment specific, prewritten lines) to block a particular IP address.
- Learn a little about `mod_rewrite`, and use it to prevent people from using your web site's images.
- Install and configure SSL (when required) using the latest SSL implementation available; obtain a valid certificate from a Certificate Authority.
- Test your installation's strength using an automatic auditing program (such as Nikto, Nessus, SAINT, or SARA).