Global Information Assurance Certification Paper

Secure Configuration of Apache in the Mac OS X Environment

GSEC Gold Certification

Author: Neil Fryer, xyberpix@xyberpix.com

Adviser: Don C. Weber

Accepted: July 20th, 2006

© SANS Institute 2006, Author retains full rights.

Outline

Introduction

With OS X gaining more traction in the Operating System market and Apache still being the worlds widest deployed web server on the Internet, I felt that there was a need to combine the process of securing both of these into one paper. I have done considerable research on both OS X and Apache, and have found documents on how to secure them each individually, but never one tailored to running Apache on OS X in a secure manner.

For a long time OS X seemed to be invisible to hackers, as it was never one of the more widely deployed Operating Systems on the Internet. Ever since Apple released the iPod though, more and more people have become interested in their other offerings, namely their Operating System (OS X) and their hardware.

Within this paper I will attempt to show how to secure both OS X and Apache, so that it can be used as an Internet facing web server. I will be using OS X 10.4.7, Apache 1.3.33 and PHP 4.4.1 for all the examples in this paper. I shall also list the shortcomings of the default configuration of OS X and Apache on OS X from a security perspective.

Abstract

Since April 1996 the Apache web server has been the most popular web server on the Internet. (Apache, 2006) The November 2005 Netcraft survey showed that Apache was the web server hosting more than 70% of the web sites on the Internet (Netcraft, 2006), this shows that Apache is more widely deployed on the Internet than all other web servers combined.

The Apache web server is also Open Source Software (OSS); this has both good points and bad. The good points are that anyone with the relevant programming skills can modify it to suite their exact needs, and also view and understand the source code to find out exactly what the application does, and how it functions.

The bad side to this however is that this also makes it easier for hackers to find vulnerabilities in the application, and then exploit them, thus gaining control of your web server, or worse yet, your entire Operating System (OS).

Due to the vast amount of people constantly striving to make Apache more secure, the amount of vulnerabilities getting found are a lot less now than they were a few years ago, but this isn't to say that more won't get found. Also due to the fact that Apache is modular in

design, sometimes vulnerabilities are found in the modules that different groups or individuals develop.

Apple's OS X is the predecessor to OS 9, and has taken a complete turn for the better. OS X is built upon a UNIX foundation of the FreeBSD micro kernel (Apple, 2006). Which means that it has all the functionality and stability of some of the more widely known UNIX systems in use today, and also all the added security features that they posses.

Quite a few of the UNIX components that come with OS X are all OSS, and thus have their source code freely available on the Internet for anyone to view and scrutinize. So when a vulnerability gets found in say, the Bash shell on a Linux or FreeBSD machine, there is a good chance that if the version is the same as the one that is bundled with OS X that the vulnerability will also affect OS X.

It is because of these vulnerabilities that we have to make sure that our instances of Apache and OS X are as secure possible, and taking a layered approach to doing this is the most secure way possible. If we secure just Apache and not OS X, then an attacker may target OS X, if we secure OS X, but not Apache, then an attacker may target Apache. Hence why we will be securing both. This may not always stop a hacker from compromising your web server, but adding security by layers may just slow him down enough for you to realize

what's going on before it's too late, or better yet, be too challenging for the hacker, so he will move on to an easier target.

One thing that I think that everyone has to realize is that once you place a web server out on the wild frontier of the Internet, there is a very good chance that one day, you probably will get hacked. Throughout this paper we are going to do our best to make that job ever harder for the would-be hacker.

Current Security Posture

As we are going to be using OS X and Apache as a web server, this will be an Internet facing host, maybe sitting behind a firewall in the demilitarized zone (DMZ), or using the built in firewall and placing this host directly on the Internet. Even though we will be configuring the built in firewall, I would recommend placing it behind a separate firewall device on you network. It is also not advisable to use the web server to do all your daily tasks on, but this does happen. If you decide to do this, please be aware of the risk of having all your personal information stored on your publicly accessible web server.

Current Risk

Apple states on the OS X section of it's website that OS X is "Secure by default" (Apple, 2006), which technically is correct. OS X has been built on top of a FreeBSD micro-kernel, which means that underneath the hood OS X is UNIX. It also makes use of the Open Source packet filtering firewall IPFW for securing external connections to and from the host. The "root" account on OS X is also disabled in a default installation, this is because all "System" modifications are done under your OS X "Administrator" account, which by default is the first user account you create on your system.

This however does not mean that it has been secured as much as it could have been, there are countless security features included in the OS that are not enabled by default, which when we are planning on placing a web server on to the Internet, need to be taken into consideration and implemented. Apple is correct in saying that OS X itself is secure by default, but some of the sub-systems such as Apache, SSH, CUPS, SAMBA have not been secured, they have been left in a default state which is not secure. Each of these subsystems requires securing in it's own way, and should be researched thoroughly before enabling any of them. With a little bit of research searching online, you will be able to find papers on securing the various service that you want to enable, such as the following:

- Five-minutes to a more secure SSH. (Maxwell, 2006)

- Securing Samba. (Tridgell, 2003)

Even if the paper is not targeted to securing the service on OS X, so long as the underlying host is UNIX based, you should be able to apply the various security measures without causing any harm. As always though, test all your changes in a lab environment first, before testing them on a live host.

A few of the threats that you face with running an unsecured web server on the Internet are the following (OWASP, 2004):

- Loss of reputation

- Damage to a corporate brand

- Loss of earnings

- Loss of personal private information

As we are going to run a production Apache web server, we can't afford to have any security vulnerabilities present in our configuration. The risks that we will be addressing with our configuration are the following:

- Unnecessary ports listening
- No host based firewall
- No firewall logging
- Unencrypted home directories
- Automatic login enabled
- System preferences not secured
- No screensaver password
- Insecure swap space usage
- No automated security updates
- Core dumps enabled
- No Open Firmware/EFI password
- No login banners
- Weak passwords

- Default UMASK is insecure
- No file integrity checks
- No rootkit detection
- Application level attacks (XSS, SQL injection, buffer overflows)
- No HTTP encryption for sensitive data
- Unnecessary Apache modules loaded
- Apache information leakage (version info, error messages, etc)

Impact of SANS Training on the Situation

The SANS GSEC training course (SANS, 2006) highlighted the need to take a "Defense in depth" strategy towards security of any network and networked host within a network. The reasoning behind this, is that the more layers of defense that you put in place, the more difficult it makes it for anyone to penetrate those layers. If you just place a firewall in front of your host, and a vulnerability is found within your firewall then you effectively have no level of protection, as your firewall has just been rendered useless by the vulnerability.

If you build other security measures into the system that you are trying to secure, then you should be confident that if one of these measures fails, then there would be another one to back up the failed measure. All we are trying to do with the "Defense in depth" (Wikipedia, 2006) strategy is to make it as difficult for the attacker as possible. Hopefully the attacker will realize that compromising this host will both take too much time and effort and move on to an easier target. If this is not the case, then the attacker is after some of your resources for a very good reason, you obviously have something that they want.

This is why we will not only be covering a secure firewall policy for a web server, but we will also be implementing various security measures into the Apache web server itself, our ideal is to have a web server that would be able to survive on the Internet without a firewall in

front of it. This way, if for some reason an attacker does manage to bypass our firewall, we still have a secure host for them to try and compromise.

During

It is common knowledge that web servers are getting defaced on a daily basis (BBC, 2005) and in great numbers as well, some of these are only defacements, while others lead to a full system compromise. Even though Apache is the widest used web server on the Internet at present, this doesn't mean that it hasn't been without its share of vulnerabilities. If you have a look a the following link, you will see a list of all the vulnerabilities in the Apache 1.3.x tree that have been addressed by the Apache Security Team:

http://httpd.apache.org/security/vulnerabilities_13.html

If you do a keyword search on the Common Vulnerabilities and Exposures (CVE) website for Apache at present it lists a total of 201 entries, some of these have been fixed, some are in the Apache 2.x.x branch and some of them have yet to be fixed.

http://cve.mitre.org/cgi-bin/cvekey.cgi?keyword=Apache

Zone-H keep a daily record of website defacements, and so far today there have been a total of 904 confirmed defacements:

http://www.zone-h.org/component/option,com_attacks/Itemid,43/

The total listing of website defacements that Zone-H have currently totals 32722 defacements, you can view the complete archive here:

http://www.zone-h.org/component/option,com_attacks/Itemid,44/total%20deface

Proposed Solution

My proposed solution will be to configure all the security on the OS X host in a much more secure manner, by enabling and tightening some of the built in security controls that are not enabled by default on the OS. By doing this, we will them be able to place the host on the Internet and serve up web pages in a secure manner.

I will also be configuring the Apache web server in a secure manner, as well as adding extra security modules and extra functionality to the web server.

PHP will be getting added to add additional functionality to the web server, as using PHP on the web server means that there will be ways to communicate with databases, and the web server will also be able to provide more than just static content by adding scripting language functionality to the web server. Due to its powerful nature though, we will need to tighten up the security on PHP before using this as a production server.

Solution Implementation

All of the following configuration changes assume that your OS X installation is in it's default state, and were performed on a fresh installation of OS X. Basic familiarity with the OS X command line, and a command line text editor such as vi, nano or emacs is also assumed. All examples in this text will be designed for the vi editor.

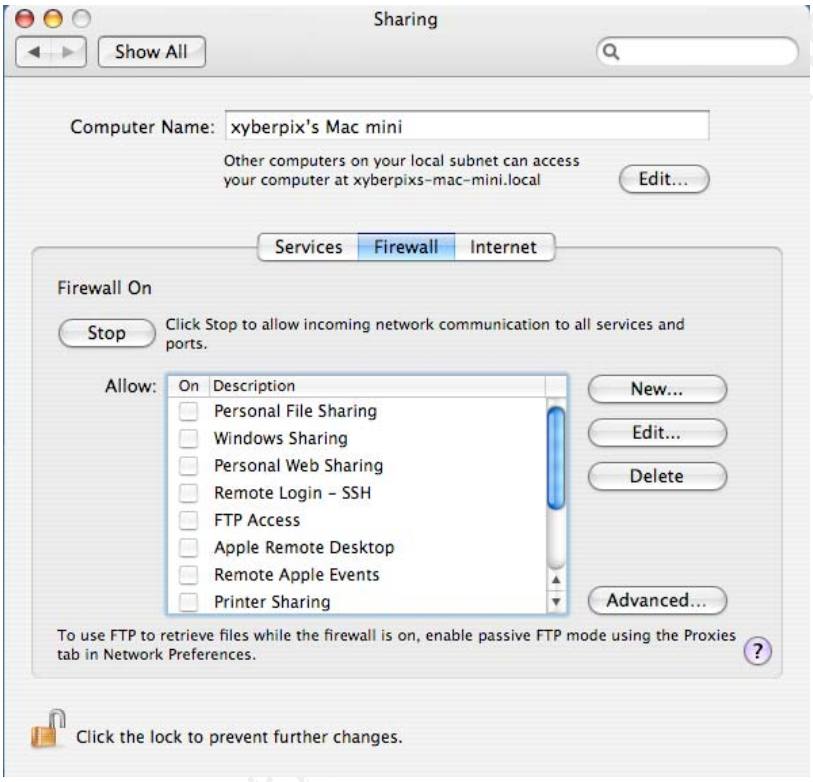
Host Security

Firewall Configuration

The first thing that we are going to do in regard to securing OS X is going to be locking down the firewall (IPFW) within OS X. The firewall is pretty secure by default, but we are going to tighten up the security and firewall policy even more, so that we can put our Mac on the Internet, and feel safe about it.

To access the Firewall settings go to System Preferences by clicking the icon on your dock. You should then see the following screen:

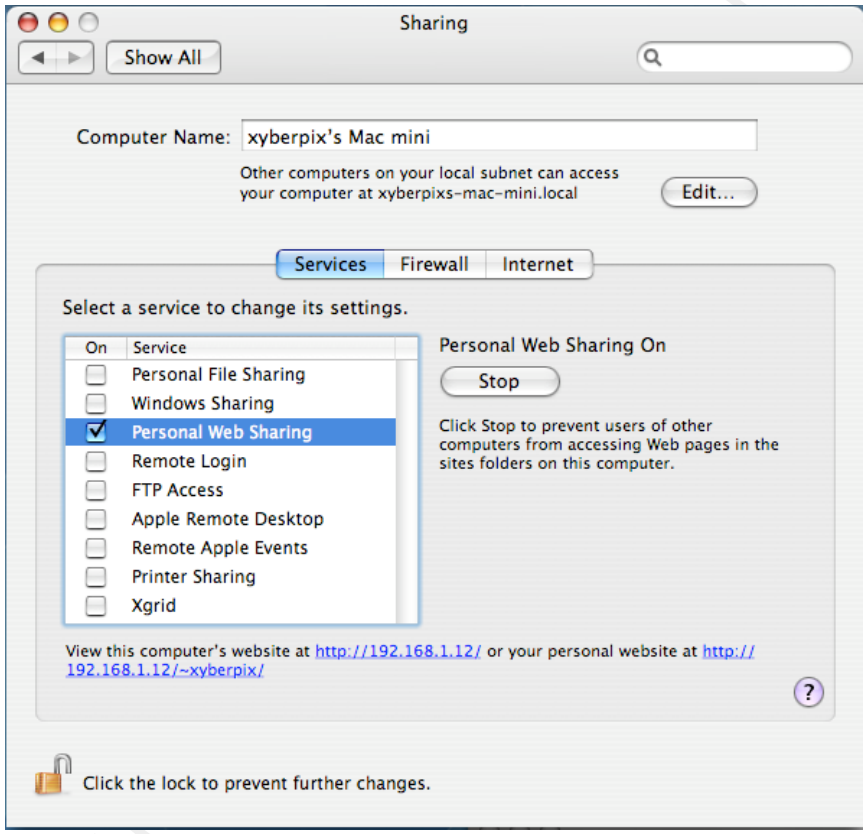
Select the Sharing option. Once inside the sharing preferences, select the Firewall tab; you should now be seeing the same thing as the screenshot below:



On the default OS X firewall policy, only the network time port (UDP/123) is open, all other ports are closed. As we are going to be serving web pages from this host, we are going to have enable the "Personal Web Sharing" option in the firewall, this will allow traffic to our host on ports:

- 80(HTTP), for normal web traffic
- 443(HTTPS), for secure web traffic
- 427(SLP), required for Bonjour network discovery.

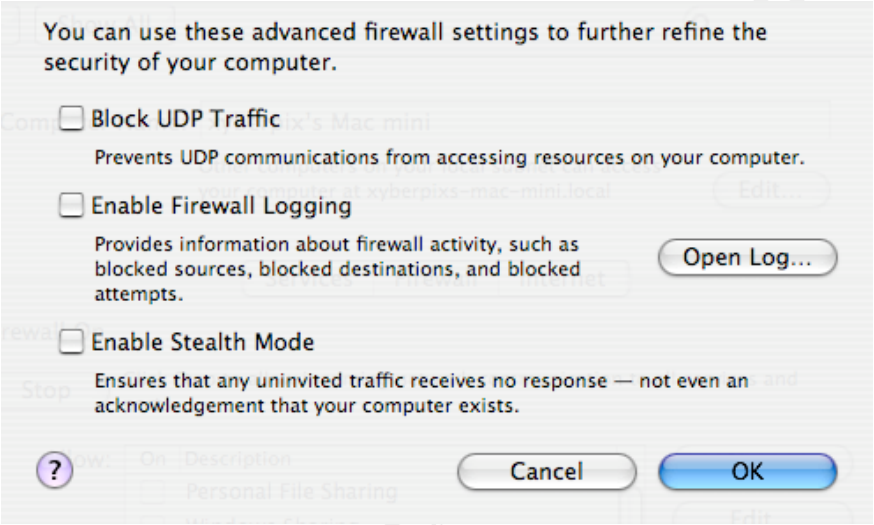
To do this, click on the “Services” tab and then click in the block on the left hand side of “Personal Web Sharing”.



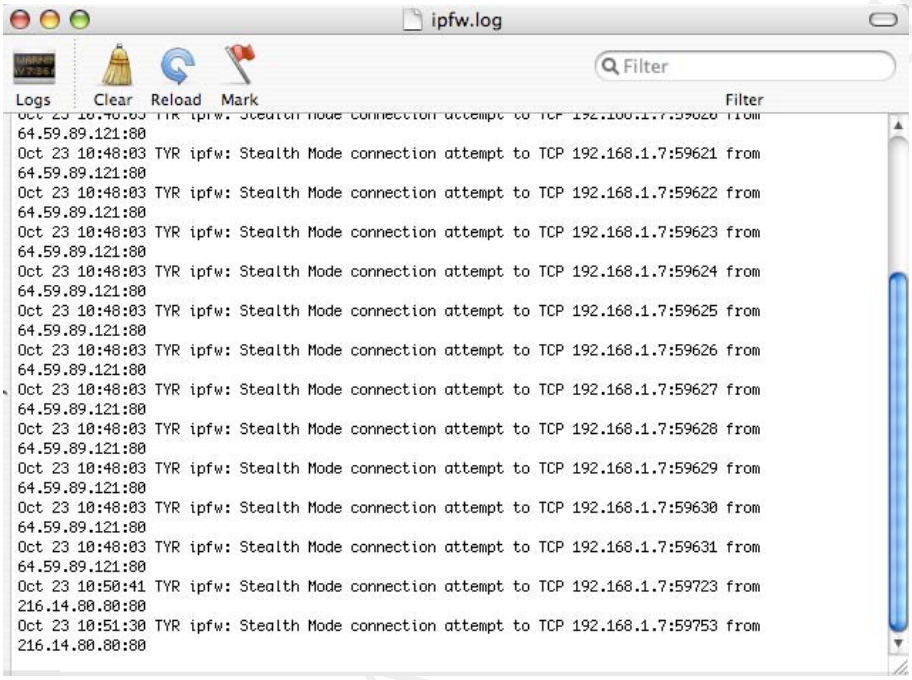
Now if you fire up a web browser and point it at you Mac’s IP address, you should see the default Apache web page. The next thing that we want to do to our firewall configuration is to enable logging, block UDP traffic and enable stealth mode.

Now, click the “Firewall” tab again, and in the bottom right hand corner you will see a button labeled “Advanced”, that’s where we want to go, click on that one. That will then

present you with a screen like the one below:



We want to enable all three of these options, to do this, click once inside each of the three boxes. We have now added a few more layers of security to the default firewall configuration, and most importantly enabled logging.



A few good web sites to learn more about the capabilities of IPFW, adding custom rules on the command line are, and the default OS X services are:

- Exploring the Mac OS X Firewall. (Hickman, 2005)
- Configuring IPFW firewalls on OS X. (Hays,2005)
- Manual page for IPFW(8). (Apple, 2002)
- Firewall Configuration For OS X 10.3. (University Of Melbourne, 2006)
- Mac OS X: What are all those processes. (Davisson, 2005)

FileVault and system security preferences

FileVault automatically encrypts and decrypts all the files and folders in your home directory on OS X on the fly (Apple, 2006). The encryption is done using 128-bit AES encryption. What this means is that everything stored within your home directory is only readable by supplying OS X with the correct user credentials, for the home directory at logon time, no other user on the system will be able to gain access to these files and folders.

By default though, this is not enabled on OS X, so we need to enable this to add to our defense in depth strategy.

To enable FileVault, go to System Preferences, and then select the Security item, this should bring you to the following screen:

Firstly we are going to set a master password, this is not the same as a UNIX root account password, and this is purely to unlock FileVault in an emergency. It is important that you make this password as difficult to guess, as possible, and that you don't forget this password, as there may come a time when you need this! It is advisable to also set a hint to help you remember this password in case you do forget it, do not put the actual password in the hint field, just something to jog your memory.

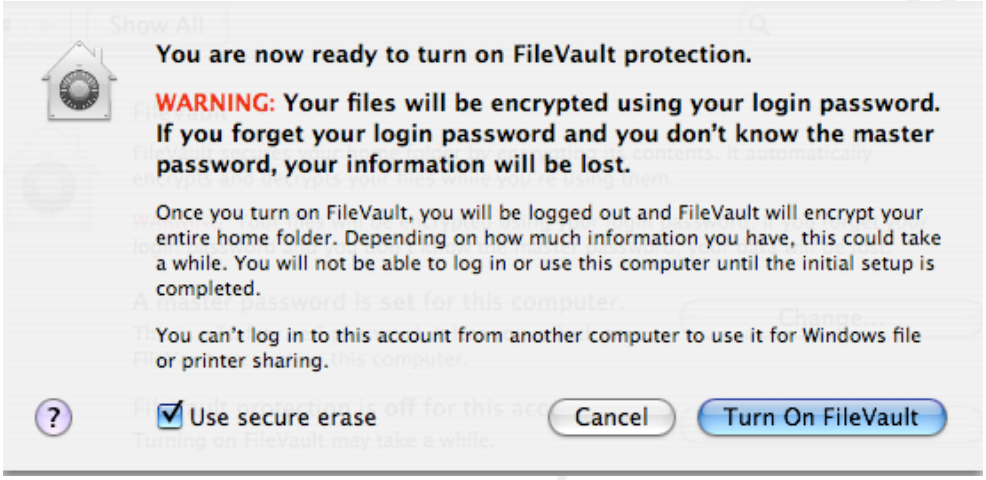
DO NOT SET THIS PASSWORD TO THE SAME AS YOUR LOGIN PASSWORD

Once you have set the master password, we are now ready to turn on the FileVault,

please bear in mind that this will take a while depending on the size of your home folder on your system.

Your Security settings screen should now look like the following once you have set your master password:

To enable FileVault for this account, click on the "Turn On FileVault…" button. You will then be asked to type in the password for your currently logged in user's account. Once you have done this, you will be presented with the following screen:



Make sure that you put a tick in the block labeled “Use secure erase”, this will take a bit longer, but it encrypts all the files before erasing them. This is to make sure that there are no un-encrypted private files left on your system after FileVault has encrypted your home folder. Once you have done this, click the “Turn On FileVault Button”. You will then be logged out of the system while FileVault encrypts your home folder. Now would be a good time to go and make a cup of tea, as I mentioned before, this does take a while.

Once FileVault has finished encrypting your home folder, you can then log back into your system and you should notice that the icon for your home directory has now changed to an icon that looks like a combination of a safe and a house.

Congratulations, you have just enabled FileVault. Be aware though that any files that are stored within your home folder will not be able to be accessed by applications such as Anti-Virus, or any of the system daemons, as they will be encrypted (Apple, 2006). Make sure that any files that applications or daemons rely on to work properly are stored outside of your home directory.

Now we are going to go back into the System Preferences, and select the Security item again, this time to set some extra security measures for the host.

We are now going to enable quite a few extra security features, place ticks in the following boxes:

- Require password to wake this computer from sleep or screen saver
- Disable automatic login
- Require password to unlock each secure system preference
- Log out after 60 minutes of inactivity. We want to set this one to 10 minutes.
- Use secure virtual memory.

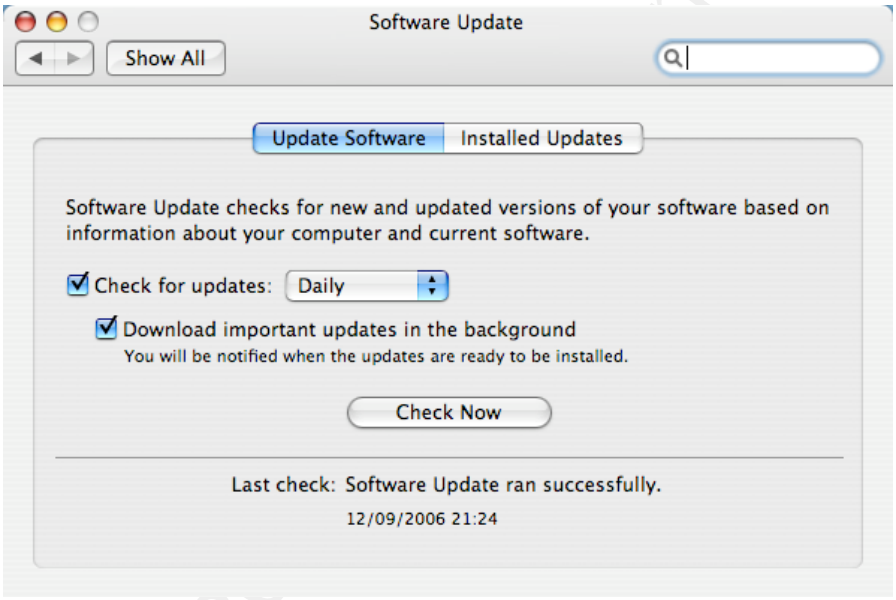
Once you have done this, you should see the following:

Once you have enabled all of these then click on the lock in the bottom left hand corner of the Security preference pane. This will prevent other users from changing these settings. You can then close the Security preferences pane.

Automatic Updates

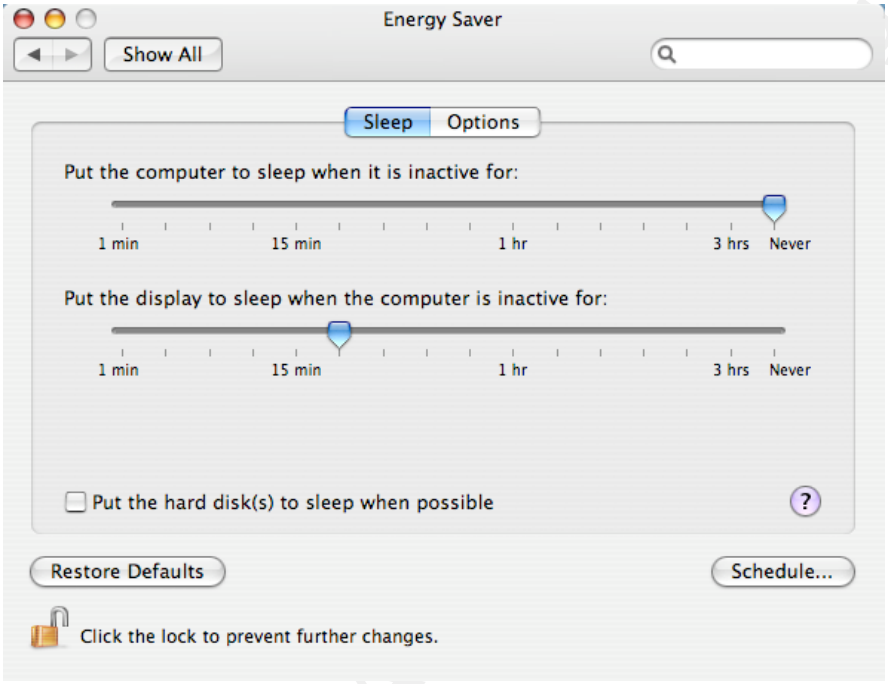
We are going to enable Automatic updates, as on an Internet facing host, the last thing that you want is a security hole due to the fact that you didn't apply the latest security updates. To do this, go to System Preferences, and then select the Software Update item. As this is going to be an Internet facing host, we are going to set it to check for updates on a

daily basis, and to download the important updates in the background. You will be notified by a pop-up when new updates are ready to be installed. Your Software Update screen should now look like the following:

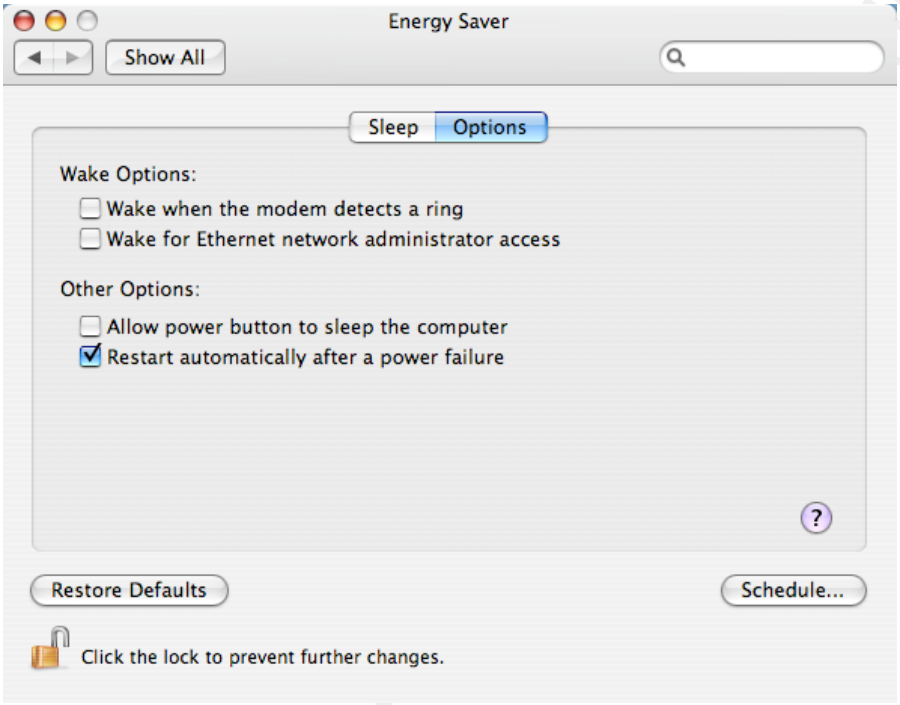


Energy Saver Options

As this is a web server, we do not want it to shut down after it hasn't been in use for a while, so we will be disabling the Energy Saver options. Open System Preferences and select the Energy Saver item. We are going to set it so that the computer does not go to sleep, and also so that the hard disk(s) never go to sleep as well. To do this, set your settings to the following:



Next go into the Options item at the top of the Energy Saver preference pane, and change your settings to match the following:



Once you have done this, once again click the lock in the bottom left hard corner of the preference pane to stop other users modifying these preferences, then close the preference pane.

Disable Core Dumps

A “core dump” (Wikipedia, 2006) is a dump of the contents in memory at the time that the dump happened, this is extremely useful for debugging purposes, but also helpful to hackers, as sometimes these dumps can reveal sensitive information such as usernames and passwords (Rogers, 1999).

To disable core dumps on OS X you need to open the Terminal.app and perform the following actions:

- The Terminal.app is located under /Applications/Utilities.
- Type the following into the Terminal.app window:

sudo vi /etc/sysctl.conf

You will them be prompted for your user password, enter this in. You have now created a new file, and you now need to edit this file to add some information into it. Insert the following into the newly created file:

kern.coredump=0

Then save the file, and restart your Mac for the change to take effect.

Open Firmware or EFI Password

The Open Firmware and EFI are similar to the BIOS on a PC, we are going to set a password on this to stop other people forcing your Mac to boot from another device such as a Firewire drive or CD/DVD drive and to stop others from booting into single user mode, as if they were able to do this, they would be able to bypass all your security settings. Single user

mode automatically logs the user in as root, as you are aware the root account has the highest privileges on a UNIX system, and with this account the user would be able to also bypass all your security measures.

To set an Open Firmware or EFI password:

- On your OS X Tiger installation disc, copy the Open Firmware Password application from /Applications/Utilities on the DVD to /Applications/Utilities on you OS X hard drive.

- Open the Open Firmware Password application

- Place a tick in the box next to “Require password to change Open Firmware Settings, then type in the password that you wish to set for the Open Firmware into the Password and Verify fields.

- Click the OK button; you will then be prompted for your user password.

- You should them receive a confirmation that the Open Firmware or EFI password has been set. Select Quit from the Open Firmware Password application menu.

- Reboot your Mac for this change to take effect.

Login Banner

Login banners are useful to notify users of unauthorized access to inform them that anything that they do on the system may be monitored. This also covers your bases from a legal perspective as well, so in general it is always a good idea to set login banners. Here we will be setting a login banner on the OS X login screen. I will be using a section of the D.O.D

login banner (ICODES Web, 2006), as it is one of the more complete ones out there, you can feel free to edit this one to suit your needs.

Before you use this policy though, check with your organizations legal department to make sure that this banner complies with all their requirements.

- Open Terminal.app
- Type the following to change your logon window access-warning banner:

sudo defaults write /Library/Preferences/com.apple.loginwindow LoginwindowText "Use of this computer system, authorized or unauthorized, constitutes consent to monitoring of this system. Unauthorized use may subject you to criminal prosecution. Evidence of unauthorized use collected during monitoring may be used for administrative, criminal, or other adverse action. Use of this system constitutes consent to monitoring for these purposes."

- Everything between the quotation marks is what will be displayed.
- Log out to test that you can see the login banner.

Passwords

One of the most important things to do on your OS X system is to make sure that you set strong passwords for all users (AUSCERT, 1993). At a minimum these should be 8 characters long, a mix of both upper and lowercase letters, alphanumeric, and also contain special characters (!@£$%^&*<>?). Also these passwords should not be based on dictionary words in any language. Consider using passphrases instead of passwords, a passphrase is any memorable phrase that you use the first letters from to make up a phrase to use instead of a password. For instance “Two cows flew over the Eiffel Tower together?!”, could become “2cFot3Tt?!”. Now you have a passphrase that you can remember, that meets all the minimum password requirements.

Default UMASK

The default umask value is what determines the users permissions on any newly created files (Maple, 2000). The default value for this on OS X is 022. This means that all users are granted read access to all newly created files.

We are going to change this to the value 027, this will limit the file permissions so that only users in the same group can access newly created files.

To change the default OS X umask open Terminal.app, and type the following:

sudo defaults write /Library/Preferences/.GlobalPreferences NSUmask 23

File Integrity

Now that you have the host OS configured securely, you need to make sure that it stays that way, and that if anything changes you are informed about it.

The best tool to do this with is Tripwire, configuring Tripwire is a complex subject in itself so I have included a couple of links here on how to do this.

http://www.macguru.net/~frodo/Tripwire-osx.html

http://www.clusters.umaine.edu/blog/2006/01/30/installing-tripwire-on-os-x-using-a-plist-file/

Rootkits

"A rootkit is a set of software tools intended to conceal running processes, files or system data, thereby helping an intruder to maintain access to a system whilst avoiding detection. Rootkits are known to exist for a variety of operating systems such as Linux, Solaris and versions of Microsoft Windows. Rootkits often modify parts of the operating system or install themselves as drivers or kernel modules." – Wikipedia

Even though we have increased the security of OS X, we still need to make sure that if your server does get compromised, and a rootkit left behind, that we detect it as soon as possible. Tripwire will let you know if anything changes, but we need to have something that will run in the background and check for rootkits on a daily basis. For this I would recommend chkrootkit. More information on chkrootkit, as well as installation instructions is available from: http://www.chkrootkit.org

Resources for securing OS X further

The following is a list of useful web sites on the Internet in regard to securing OS X even further. It would be a good idea to read through these if you are serious about locking down your system.

http://www.corsaire.com/white-papers/050819-securing-mac-os-x-tiger.pdf

http://images.apple.com/server/pdfs/Tiger_Security_Config.pdf

Apache Configuration

Now that we have covered securing OS X, we are ready to configure the Apache web server and build all the necessary components to secure it and turn it into a Internet facing web server. I will walk you through all the necessary steps to get this done below, so what are

we waiting for? Let's get moving!

Install OS X Developer Tools

We need to install the developer tools on our host so that we can compile some of the modules that Apache requires. The OS X developer tools package contains vital libraries and the gcc compiler, which we will need for compiling our new applications.

To install the OS X Developer tools first insert your OS X DVD.

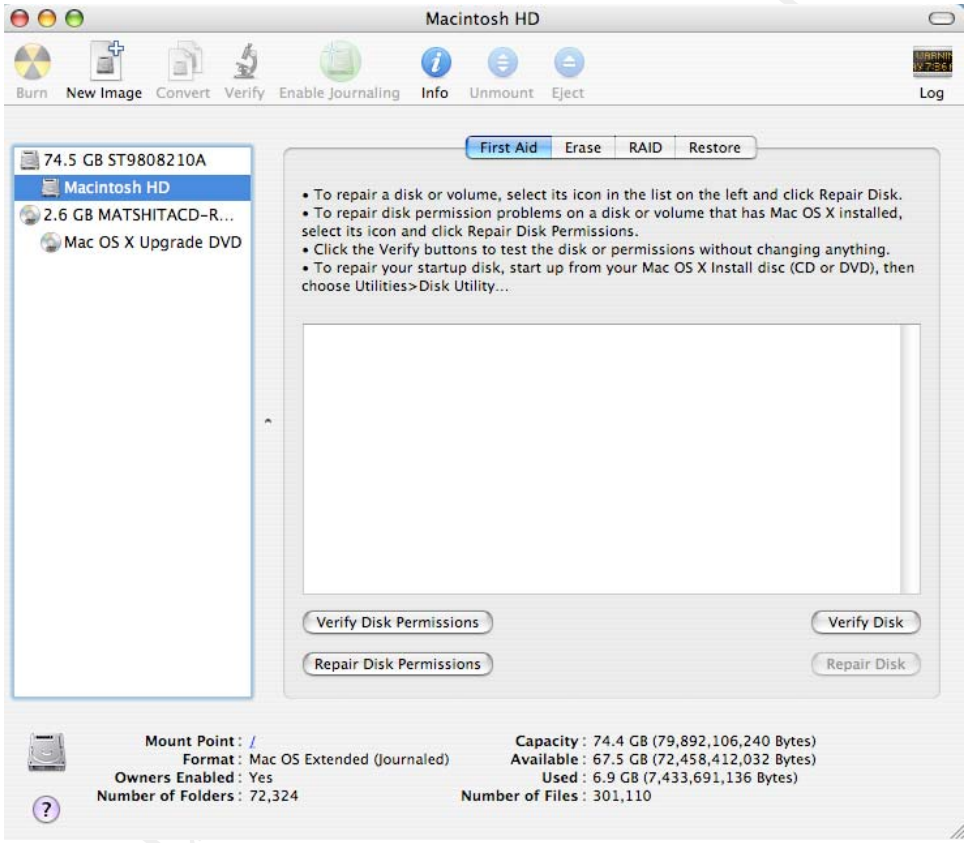
On your OS X DVD you will see a folder called Xcode Tools, open this and then install the XcodeTools.mpkg package.

Once the installation has finished, we are going to repair the OS X file permissions just to make sure that nothing went wrong with our installation. It is good practice to always repair the file permissions on an OS X volume, as sometimes the permissions can get a bit mixed up and cause system problems.

Repair File Permissions

It is good practice to repair the file permissions whenever you install any new security updates as well.

- Go to /Applications/Utilities and open the Disk Utility application, then select your Macintosh HD drive from the list on the left hand side.



- Click on the Repair Disc Permissions button, once this has completed, restart your Mac.

Repairing the file permissions makes sure that all the file permissions are set in the correct manner (Apple, 2005).

Downloading mod_security

Mod_Security is a web application firewall (Modsecurity, 2006), which can be run either as an Apache module or as a standalone instance, and is used to further protect web applications from the various attacks that they can be vulnerable to. We are including this module into our Apache build to build upon our layered defense strategy.

Before we can begin, we have to go and get the new module that we are going to add to our Apache configuration and verify its authenticity. Below are the links to the module, the current rule set, as well as the MD5 checksum for verification.

Mod_Security: http://www.modsecurity.org/download/modsecurity-apache_1.9.4.tar.gz

Mod_Security MD5 signature: http://www.modsecurity.org/download/modsecurity-apache_1.9.4.tar.gz.md5

Mod_Security Current Rule Set: http://www.modsecurity.org/download/modsecurity-rules-current.tar.gz

Please go and download the module, the rules and the MD5 checksum.

Apache Modules

All of the Apache configuration changes and the adding of mod_security will all be

done via the Terminal.app, so open this one up, and then let's get started on our quest to secure Apache on OS X.

Firstly we are going to move everything that we downloaded for mod_security to our /tmp folder on OS X. To move these three files from your Desktop to the /tmp directory using the Terminal.app, type the following:

mv Desktop/modsecurity- /tmp*

Now change to your /tmp directory, and make sure that all the files are there.

Verifying MD5 Checksums

Now to verify the MD5 checksum on the modsecurity-apache_1.9.4.tar.gz.tar file using the MD5 checksum modsecurity-apache_1.9.4.tar.gz.md5.txt run the following commands in Terminal.app:

cat modsecurity-apache_1.9.4.tar.gz.md5.txt

md5 modsecurity-apache_1.9.4.tar.gz.tar

As you can see we first issued the command "cat modsecurity-apache_1.9.4.tar.gz.md5.txt", this was used to read the MD5 checksum. The MD5 checksum

that was issued was 74d2317781bab619cd7b6b376b978107.

We then ran the MD5 utility against the file modsecurity-apache_1.9.4.tar.gz.tar, by issuing the command “md5 modsecurity-apache_1.9.4.tar.gz.tar”, which gave us the output value of 74d2317781bab619cd7b6b376b978107.

If all is as it should be, both MD5 values should match, if this is not the case, delete both files and download them again, as this will mean that they are not the correct files, and they may have been compromised or corrupted.

Compiling and installing mod_security

Now that we have verified that the module that we downloaded is a legitimate module, we are now ready to start compiling it, and installing it.

The first thing that we want to do is uncompress the mod_security module, to do this on within the Terminal.app, use the following command:

tar zxvf modsecurity-apache_1.9.4.tar.gz.tar

Once this has been uncompressed, then change into the mod_security directory.

Before we go any further we are going to back up our copy of Apache’s configuration

file httpd.conf. To do this type the following into your command line:

sudo cp /private/etc/httpd/httpd.conf /private/etc/httpd/httpd.conf.backup

This makes a copy of the Apache configuration file /private/etc/httpd/httpd.conf called httpd.conf.backup in the same location as your original httpd.conf file. This is just in case we ever need to revert back to this one.

Now we need to compile mod_security, type the following on the command line:

/usr/local/apache/bin/apxs -cia mod_security.c

You now have Apache running with mod_security installed, but now you need to add some default rules to mod_security, to do this you'll have to edit the Apache configuration file httpd.conf.

Open the file /private/etc/httpd/httpd.conf with super user privileges in you text editor. in the Terminal.app:

sudo vi /private/etc/httpd/httpd.conf

As you can see from the comments in this text below, we will be configuring mod_security to block various attacks such as SQL injection, XSS, input validation and buffer

overflows.

Once you have this file open, scroll down to the bottom of the file and then paste the text below into it, once you have done this save the file.

```
<IfModule mod_security.c>
# Turn the filtering engine On or Off
SecFilterEngine On


# Make sure that URL encoding is valid
SecFilterCheckURLEncoding On
# Unicode encoding check
SecFilterCheckUnicodeEncoding Off
# Only allow bytes from this range
SecFilterForceByteRange 0 255
# Only log suspicious requests
SecAuditEngine RelevantOnly
# The name of the audit log file
SecAuditLog /var/log/audit_log
# Debug level set to a minimum
```

```
SecFilterDebugLog /var/log/modsec_debug_log

SecFilterDebugLevel 0

# Should mod_security inspect POST payloads

SecFilterScanPOST On



# By default log and deny suspicious requests

# with HTTP status 500

SecFilterDefaultAction "deny,log,status:500"



# Prevent drop table SQL injection

SecFilter "drop[[:space:]]table"



#SQL injection attacks

SecFilter "delete[[:space:]]+from"

SecFilter "insert[[:space:]]+into"

SecFilter "select.+from"



# Only inspect dynamic requests
```

```
# (YOU MUST TEST TO MAKE SURE IT WORKS AS EXPECTED)

SecFilterEngine DynamicOnly



# Prevent OS specific keywords

SecFilter /etc/passwd

SecFilter /bin/ls



# Prevent path traversal (..) attacks

SecFilter "\.\./"



# Weaker XSS protection but allows common HTML tags

SecFilter "<[[:space:]]*script"



# Prevent XSS atacks and do input validation (HTML/Javascript injection)

SecFilter "<(.|\n)+>"



# Reject requests with status 403

SecFilterDefaultAction "deny,log,status:403"
```

```
SecFilterSelective SCRIPT_FILENAME "export\.php$" chain

SecFilterSelective ARG_what "\.\."

SecFilterSelective "HTTP_Via" "pinappleproxy"

</IfModule>
```

We now need to restart Apache for these rules to take effect, to do this, type the following:

sudo apachectl graceful

If all went well Apache should return something similar to the following:

/usr/sbin/apachectl graceful: httpd gracefully restarted

You have now installed and configured mod_security. If you would like to learn more about mod_security head over to the following website, as it has all the documentation you could want on mod_security:

http://www.modsecurity.org/documentation/modsecurity-apache/1.9.3/modsecurity-manual.html#N1008C

Apache SSL (Self Signed Certificate)

There are two ways to enable SSL on Apache and both of these require a valid SSL certificate, using the one way we will create an SSL certificate ourselves and the other way is to purchase one from a valid Certificate Authority such as Verisign.

I would recommend that if you are going to be doing online sales or running a corporate website, then you should purchase a SSL certificate, but if this will be just to host a personal site, you can get away with using a self-signed certificate.

First I will show you how to create a self-signed SSL certificate, then I will tell you the steps involved in purchasing one and installing it on your web server.

The following steps closely follow the steps outlined by legacyb4 on the MacOS X hints website here:

http://www.macosxhints.com/article.php?story=20041129143420344

There are basically 5 steps involved in creating a self-signed SSL certificate for Apache (legacyb4, 2004), and they are the following:

1. Creating a Certificate Authority

2. Generate a private key for the web server

3. Generate a Certificate Request

4. Sign the Certificate Request

5. Add the SSL components to Apache

1. Creating a Certificate Authority

Open up the Terminal.app and change your directory to /Documents directory.

Next create a new directory called certs and move into this directory:

mkdir certs; cd certs

We now need to create a new Certificate Authority, to do this we are going to run the CA.pl script from the /System/Library/OpenSSL/misc directory. From the command line type the following:

/System/Library/OpenSSL/misc/CA.pl -newca

You will then see the following within the Terminal.app:

```
xyberpix-mac-mini:~/Documents/certs xyberpix$ System/Library/OpenSSL/misc/CA.pl
```

```
-newca

        CA certificate filename (or enter to create)



        Making CA certificate ...

        Generating a 1024 bit RSA private key

        ..................................................++++++

        .........++++++

        writing new private key to './demoCA/private/cakey.pem'

        Enter PEM pass phrase:

        Verifying - Enter PEM pass phrase:

        -----

        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) [AU]:GB

State or Province Name (full name) [Some-State]:Berkshire

Locality Name (eg, city) []:Reading

Organization Name (eg, company) [Internet Widgits Pty Ltd]:Neil's CA

Organizational Unit Name (eg, section) []:IT

Common Name (eg, YOUR name) []:Neil Fryer

Email Address []:xyberpix@xyberpix.com
```

As you get prompted for information enter it accordingly, until you are returned back to the command prompt.

2. Generate a private key for the web server

To generate a private key for the web server type the following in the ~/Documents/certs directory:

openssl genrsa –des3 –out webserver.key 1024

This will then create an encrypted, private key for your web server. Make sure that you use a secure password here. The output that you will see on the screen should be similar to the following:

```
xyberpix-mac-mini:~/Documents/certs xyberpix$ openssl genrsa -des3 -out webserver.key
1024

Generating RSA private key, 1024 bit long modulus

.....++++++

.....++++++

e is 65537 (0x10001)

Enter pass phrase for webserver.key:

Verifying - Enter pass phrase for webserver.key:

xyberpixs-mac-mini:~/Documents/certs xyberpix$
```

The next thing that we need to do is generate a non-password protected copy of the key. This is done because the user can not enter a password for the certificate when OS X is booting. To do this, type the following into the Terminal:

openssl rsa –in webserver.key –out webserver.nopass.key

This will then generate a non-password protected copy of the private key that you just created:

```
xyberpixs-mac-mini:~/Documents/certs xyberpix$ openssl rsa -in webserver.key -out
webserver.nopass.key

Enter pass phrase for webserver.key: (This will be the pass phrase that you set earlier)
```

```
writing RSA key

xyberpixs-mac-mini:~/Documents/certs xyberpix$
```

3. Generate A Certificate Request

The next step in our process is to generate a certificate request for our web server; this will be based on the private key that we generated back in step 2.

You should still be in the ~/Documents/certs directory in Terminal.app, if not go there and then type the following all on the same line:

openssl req –config /System/Library/OpenSSL/openssl.cnf –new –key webserver.key –out newreq.pem –days 3650

Once you have hit enter you will be prompted for your certificate pass phrase.

What you have done here is to tell the system to generate a new certificate request "newreq.pem", using the default openssl configuration "openssl.cnf", using the web server key "webserver.key" and give it a validity period of 10 years "3650" days.

You should then get the following in your Terminal.app:

```
Enter pass phrase for webserver.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) [AU]:GB

State or Province Name (full name) [Some-State]:Berkshire

Locality Name (eg, city) []:Reading

Organization Name (eg, company) [Internet Widgits Pty Ltd]:Neil's CA

Organizational Unit Name (eg, section) []:IT

Common Name (eg, YOUR name) []:Neil Fryer

Email Address []:xyberpix@xyberpix.com



Please enter the following 'extra' attributes

to be sent with your certificate request

A challenge password []:^C
```

```
xyberpixs-mac-mini:~/Documents/certs xyberpix$ openssl req -config
/System/Library/OpenSSL/openssl.cnf -new -key webserver.key -out newreq.pem -days 3650

Enter pass phrase for webserver.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) [AU]:GB

State or Province Name (full name) [Some-State]:Berkshire

Locality Name (eg, city) []:Reading

Organization Name (eg, company) [Internet Widgits Pty Ltd]:Neil's CA

Organizational Unit Name (eg, section) []:IT

Common Name (eg, YOUR name) []:ssl.xyberpix.com

Email Address []:xyberpix@xyberpix.com
```

```
Please enter the following 'extra' attributes

to be sent with your certificate request

A challenge password []: leave this blank

An optional company name []: leave this blank
```

4.Sign The Certificate Request

The next thing that we need to do is to sign the certificate request “newreq.pem” with the Certificate Authority that we created way back in step one. Make sure that you are still in the ~/Documents/certs directory and enter the following into the Terminal.app:

/System/Library/OpenSSL/misc/CA.pl –signreq

This tells the system to sign the newreq.pem file we created in step 3.

You should receive the following output in Terminal.app:

```
Using configuration from /System/Library/OpenSSL/openssl.cnf

Enter pass phrase for ./demoCA/private/cakey.pem:

Check that the request matches the signature

Signature ok

Certificate Details:
```

```
Serial Number:
    86:23:d2:db:4f:5b:c7:9a
Validity
    Not Before: Sep 18 00:10:39 2006 GMT
    Not After : Sep 18 00:10:39 2007 GMT
Subject:
    countryName               = GB
    stateOrProvinceName       = Berkshire
    localityName              = Reading
    organizationName          = Neil's CA
    organizationalUnitName    = IT
    commonName                = ssl.xyberpix.com
    emailAddress              = xyberpix@xyberpix.com
X509v3 extensions:
    X509v3 Basic Constraints:
        CA:FALSE
    Netscape Comment:
        OpenSSL Generated Certificate
```

```
        X509v3 Subject Key Identifier:

            0C:15:85:02:5B:B8:B0:8B:CD:77:DA:A0:E4:00:49:CF:BD:4C:71:C3

        X509v3 Authority Key Identifier:

            keyid:1D:B7:AE:8C:AE:CF:6D:7F:81:BD:D0:F6:9E:65:B6:47:50:D0:47:B8

            DirName:/C=GB/ST=Berkshire/L=Reading/O=Neil's CA/OU=IT/CN=Neil
Fryer/emailAddress=xyberpix@xyberpix.com

            serial:86:23:D2:DB:4F:5B:C7:99



Certificate is to be certified until Sep 18 00:10:39 2007 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

Signed certificate is in newcert.pem
```

5. Add The SSL Components To Apache

Well now we have all the SSL pieces that we require, so all that's left is to move the

pieces and add them to our Apache configuration.

To move the entire “certs” directory out of your home folder to the /private/etc/httpd/ directory and change it’s permissions in Terminal.app, type the following:

sudo mv /Users/xyberpix/Documents/certs /private/etc/httpd/

sudo chown –R root:wheel /private/etc/httpd/certs

To add the SSL configuration to the Apache configuration file, using your text editor within Terminal.app, open the file /private/etc/httpd.conf with super user privileges.

Paste the following at the end of httpd.conf file (legacyb4,2004), taking care to replace all the highlighted entries with those of your own:

```
<IfModule mod_ssl.c>

Listen 80

Listen 443

SSLRandomSeed startup builtin

SSLRandomSeed connect builtin
```

```
<VirtualHost _default_:443>

SSLEngine on

ServerName ssl.xyberpix.com

ServerAdmin xyberpix@xyberpix.com

ErrorLog /var/log/httpd/error_log

SSLCipherSuite ALL:!ADH:!EXPORT56:RC4+RSA:+HIGH:+MEDIUM:+LOW:+SSLv2:+EXP:+eNULL

SSLCertificateFile /private/etc/httpd/certs/newcert.pem

SSLCertificateKeyFile /private/etc/httpd/certs/webserver.nopass.key

SSLCACertificateFile /private/etc/httpd/certs/demoCA/cacert.pem

SSLCARevocationPath /private/etc/httpd/certs/demoCA/crl

</VirtualHost>

</IfModule>
```

Once you have pasted this in, scroll up until you find the following couple of lines that look like this:

```
#LoadModule ssl_module        libexec/httpd/libssl.so
```

```
#AddModule mod_ssl.c
```

The # in front of these lines means that they have been commented out. We need to remove these #'s to enable SSL. To do this, go to the beginning of each of these two lines and delete the unwanted #'s.

Once you have done this, save and exit the file.

We now need to restart Apache for our changes to take effect, to do this type the following:

sudo apachectl stop;sudo apachectl start

You have now successfully got SSL configured into Apache with a self-signed SSL certificate.

Apache SSL (3rd Party Certificate)

So you now know how to use a self signed SSL certificate, but if you want to use this server for a production web site, then you're going to want to use a valid SSL certificate that's been issued by a valid Certificate Authority.

Adding certificate from a valid CA is quite a simple process, firstly you will need to

purchase an SSL certificate, and the two most well know issuers are Verisign and Thwate:

http://www.verisign.com

http://www.thawte.com/

Basically all you need to do to install your new certificate once you have received it, is to place it in a secure directory on your web server and then set the following directives in your Apache configuration file to point to your new certificate, key file and chain file respectively:

SSLCertificateFile

SSLCertificateKeyFile

SSLCertificateChainFile

Both Verisign and Thwate have some well written instructions on their web sites for installing SSL certificates, so I am not going to duplicate them here, instead I will provide the links to their instructions.

http://www.verisign.com/support/ssl-certificates-support/page_dev019509.html

http://www.thawte.com/ssl-digital-certificates/technical-support/apache_install.html

Enabling PHP within Apache

Apache on OS X already has PHP support built into it; only it is not enabled by default.

To enable PHP on Apache all we have to do is edit the httpd.conf file slightly. Using your text editor in Terminal.app open the following file with super user privileges /private/etc/httpd/http.conf.

Find the following two lines:

```
#LoadModule php4_module        libexec/httpd/libphp4.so
#AddModule mod_php4.c
```

Then delete the #'s from the beginning of these lines, then save the file.

You should now be back to your command line, and have saved the changes that you made to the httpd.conf file. All that's left now is to restart Apache, and you have now enabled PHP. To restart Apache from the Terminal.app:

sudo apachectl stop;sudo apachectl start

Securing the Apache configuration file

Now that we have all of the components installed, we now need to trim a lot of the unnecessary features out of Apache, and thus increase its security. Firstly we are going to comment out any modules that we are not likely to need in our instance of Apache, if you feel that you will require any of these then don't comment them out.

Full documentation on all the Apache modules for the 1.3.x Apache branch can be found here:

http://httpd.apache.org/docs/1.3/mod/

To comment out a line in the Apache configuration file, all you need to do is insert a # at the beginning of the line.

Let's get started, the modules that you are going to comment out of the httpd.conf file are the following:

autoindex_module	For Directory Indexing
imap_module	For IMAP connecivity
userdir_module	To allow users to host their own web sites
mod_bonjour	Used for Apple networking

Once again in Terminal.app open the Apache configuration (/private/etc/httpd/httpd.conf) file in your text editor using super user privileges.

Once you have this file open go and place #'s in front of the following lines in the configuration file:

In the LoadModules section:

```
LoadModule autoindex_module   libexec/httpd/mod_autoindex.so
LoadModule imap_module        libexec/httpd/mod_imap.so
LoadModule userdir_module     libexec/httpd/mod_userdir.so
LoadModule bonjour_module     libexec/httpd/mod_bonjour.so
```

In the AddModules section:

```
AddModule mod_autoindex.c
AddModule mod_imap.c
AddModule mod_userdir.c
AddModule mod_bonjour.c
```

By default Apache will give out it's version information, so we want to stop this from happening, as we want to make it as hard as possible for any attackers, so what we are going to do here is turn this off. Find the directive named ServerSignature, and set it to off, so that it

should now look like the following:

```
ServerSignature Off
```

Below the above directive we are going to add a new directive, this directive tells Apache to only send “Apache” in it’s header information.

```
ServerTokens Prod
```

Apache in its default configuration also has the Apache manual available as well (Apache, 2006), so we are going to remove this, comment out the entire section below:

```
# This Alias will project the on-line documentation tree under /manual/
    # even if you change the DocumentRoot. Comment it if you don't want to
    # provide access to the on-line documentation.
    #
    Alias /manual/ "/Library/WebServer/Documents/manual/"

    <Directory "debug2: channel 0: window 32687 sent adjust 32849
        Options Indexes FollowSymlinks MultiViews
        AllowOverride None
        Order allow,deny
```

```
    Allow from all
</Directory>
```

Apache also ships with quite a few languages as well, so we are going to comment out all the entries that we don't need. In your httpd.conf file you will find entries like the following:

```
AddLanguage da .dk
AddLanguage nl .nl
AddLanguage en .en
AddLanguage et .ee
```

Comment out all the entries you don't need, but make sure to leave the AddCharset entries.

The last line that you will want to comment out is the directive to allow users to host their own sites from out of their home directories. The line to comment out here is:

```
Include /private/etc/httpd/users/*.conf
```

Once you have commented all of those lines out, you can then save the file.

We now need to remove the Apache manual pages from the server, to do this, type the following at the command line:

sudo rm –rf /Library/WebServer/Documents/manual

You have now finished securing your Apache server configuration.

Securing PHP

Even though we have secured Apache itself, we still need to make sure that PHP has been secured. PHP is a very powerful programming language, and in the wrong hands it can wreak havoc.

The main PHP configuration file is located at /private/etc/php.ini.default, and this is the file that we will be editing in Terminal.app.

Open the file /private/etc/php.ini.default in Terminal.app with super user privileges in your text editor:

We are going to set the following parameters in this file to the values specified here.

safe_mode = On

"By enabling safe_mode, PHP scripts are only able to access files when their owner is the owner of the PHP scripts. This is one of the most important security mechanisms built into PHP, and prevents unauthorized users from accessing system resources. " – Security Focus

safe_mode_gid = Off

"With safe_mode_gid turned off, PHP scripts are able to access files not only when the user id's are the same, but also when the group of the PHP script owner, is the same as that

of the group of the owner of the file." – Security Focus

```
disable_functions = "dl,phpinfo,shell_exec,passthru,exec,popen,system,
proc_get_status,proc_nice,proc_open,proc_terminate,proc_close"
```

By disabling certain PHP functions that shouldn't be needed we are making the system more secure and less vulnerable to attack.

```
expose_php = Off
```

Turning off expose_php prevents PHP from sending information about itself in the HTTP headers.

```
display_errors = Off
```

We are disabling all errors, as these can give away vital information about our PHP installation.

```
log_errors = On
```

We are going to log all PHP errors, as your log files can sometimes be your only means of finding out how a compromise happened.

Once you have changed these parameters, save the file.

For these changes to take effect you have to rename your php.ini.default file to php.ini

and then restart Apache, to do this type the following:

sudo cp /private/etc/php.ini.default /private/etc/php.ini

And:

sudo apachectl restart

You have now secured PHP, all that's left to do is clean up the system a bit. till

Clean Up

Removing the Apple developer tools

Now that we have finished installing everything, there's no need to leave the developer tools on our system for attackers to use as these can be used to run and compile custom exploits on our system, so we're going to remove them.

In Terminal.app type the following:

sudo /Developer/Tools/uninstall-devtools.pl

Removing the downloaded application source code

We also need to remove the source code for mod_security, to do this, type the following into Terminal.app:

*rm –rf /tmp/modsecurity**

Repair disk permissions and reboot

The final thing that we need to do is to repair the disk permissions and reboot our Mac. The reason for repairing the permissions now, is that we have done a lot of work on our system, and the last thing that we would want is for something to go wrong at this point. For a

complete list of everything that repairing permissions does see:

http://docs.info.apple.com/article.html?artnum=25751

To repair the disk permissions, open /Applications/Utilities/Disk Utility, then select "Repair Disk Permissions". Once this has completed, reboot your Mac, and you now have a secure web server running on OS X.

All your web pages that you want to publish should all be placed into the directory /Library/WebServer/Documents, for CGI's there is /Library/WebServer/CGI-Executables/.

After

We now have a web server that is suitable for doing secure transactions over the Internet, so you could now have an online ordering site, or set up access to your e-mail server via web mail, without worrying about sending sensitive details over a clear link.

We have also added the mod_security module to Apache, and defined a policy for this, this will help to protect any web applications that we now publish, from the likes of certain XSS or SQL injection attacks.

PHP was added into the mix as well, so that we can have web pages accessing databases, and being a lot more active than static HTML would ever allow, due to the power of PHP as a programming language we have secured the default PHP installation as well.

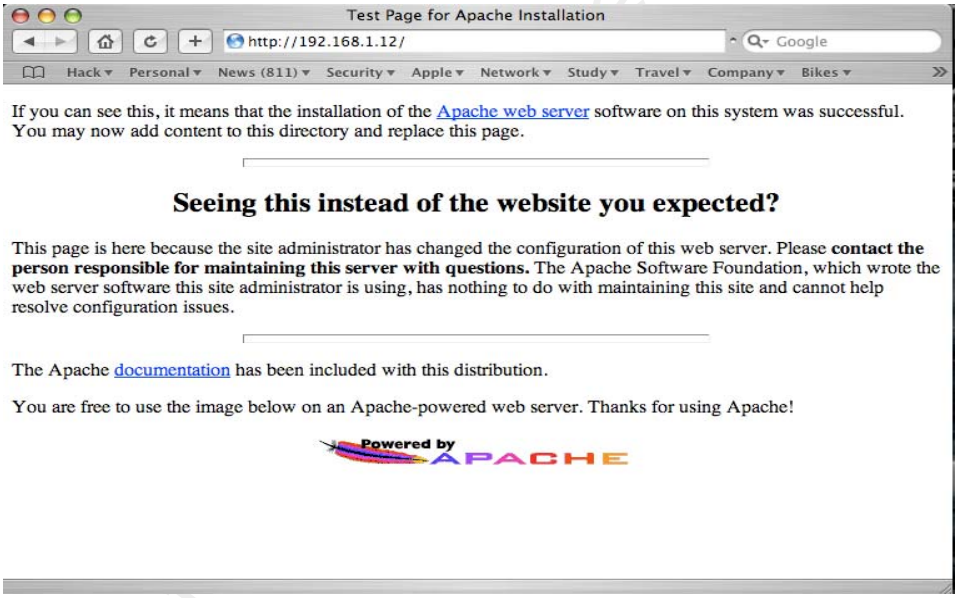
We also tightened up the security on the Apache configuration file as well, in it's default state it is too open, and gives out too much information, so we have now secured this as much as possible without loosing any performance or functionality.

During this process, I only ran into one issue, which was not removing the SSL certificates out of my home directory, and thus having them be encrypted by FileVault and break the SSL functionality of Apache. Thankfully this was easily solved.

We have now secured our instance of Apache, and added functionality for processing sensitive date, and added prevention mechanisms to prevent web site attacks such as XSS and SQL injection. We have also firewalled and locked down our OS X host, so that we now have a defense in depth strategy which is exactly what we were working towards here.

Solution Testing and Validation

We now need to test that everything is working, as it should be. So we are going to start with testing that Apache is running, to do this, point your web browser to the IP address of your web server, and you should then be presented with the default Apache server screen:



Now that we have Apache serving up web pages we also need to test that the PHP functionality is working properly as well. To do this copy the following script into a file called test.php in your /Library/WebServer/Documents folder on your web server.

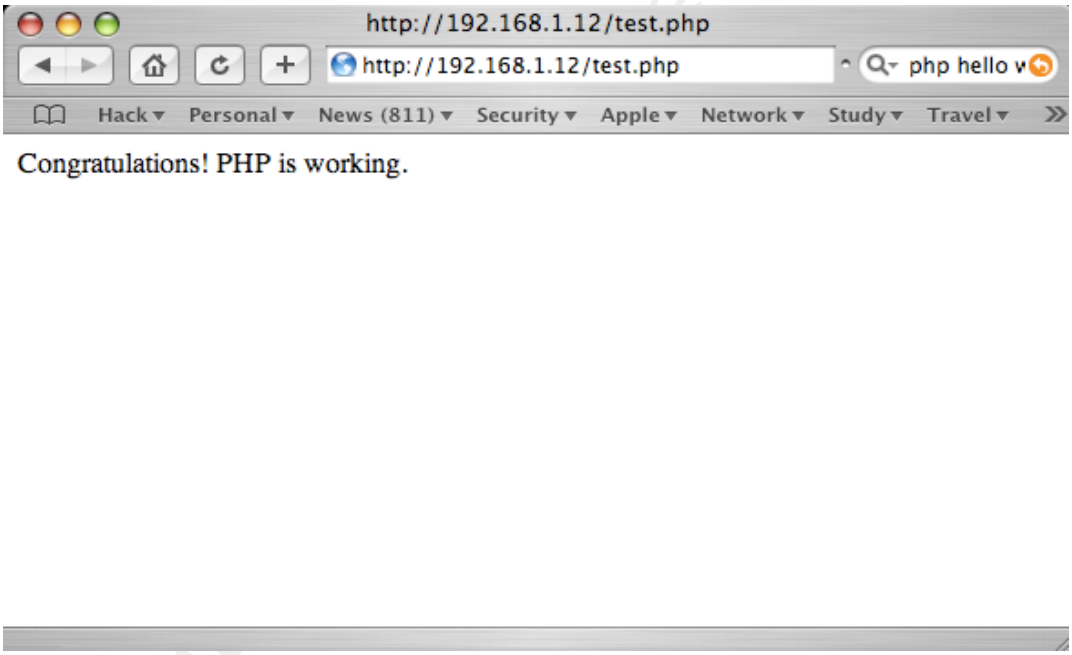
```
<?php

echo 'Congratulations! PHP is working.';
```

```
?>
```

The point your web browser at your web server's IP address again, but this time add test.php to the end of the URL, like so http://192.168.1.12/test.php

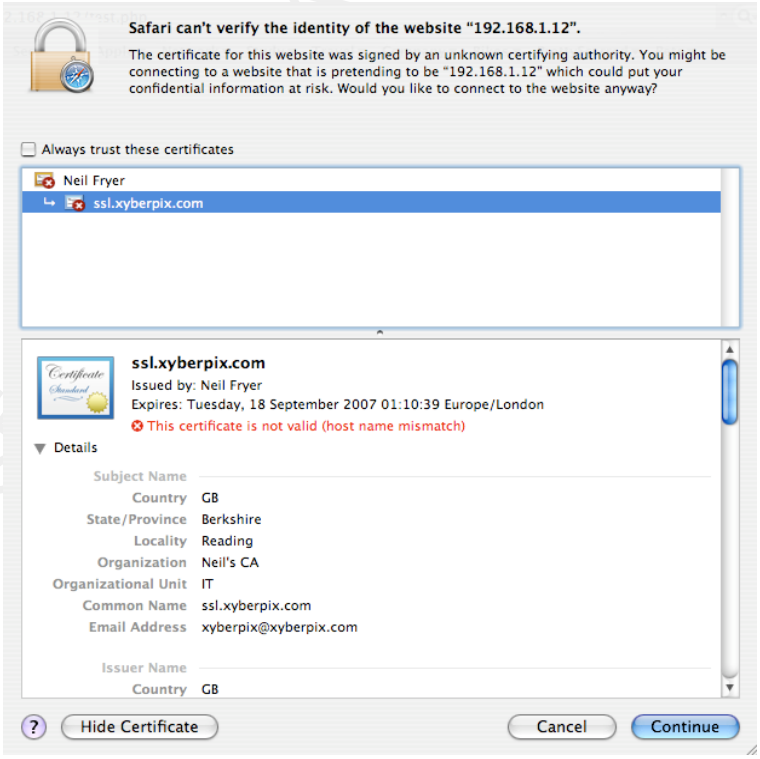
You should then be presented with the following screen:



To test that we have SSL running smoothly, we are going to point our web browser at are servers IP address again, but this time, we are going to add HTTPS to the front of the URL, like so https://192.168.1.12.

You should then be presented with a SSL certificate screen, like the following:

You should then click on the "Show Certificate" button, and that will take you to screen showing you the details of the certificate, click on the marker next to the details heading, and you should then be able to view all the details about the certificate that you created or bought.



Once you are happy that this is your certificate you can then click on the "Continue"

button to gain access to you HTTPS web site.

Risk Assessment

The level of risk associated with this host now is considerably less than it was initially. If we look at the initial risks that we had earlier, and how they have now been mitigated, you can see that we now have a more secure host.

Unnecessary ports listening

Mitigation: enabled the system firewall.

No host based firewall

Mitigation: enabled the system firewall.

No firewall logging

Mitigation: enabled the system firewall logging feature.

Unencrypted home directories

Mitigation: Enabled FileVault.

Automatic login enabled

Mitigation: disabled Automatic Login.

System preferences not secured

Mitigation: secured the important system preferences.

No screensaver password

Mitigation: Enabled a screensaver password.

Insecure swap space usage

Mitigation: enabled secure system memory.

No automated security updates

Mitigation: enabled automatic updates.

Core dumps enabled

Mitigation: disabled core dumps.

No Open Firmware/EFI password

Mitigation: Enabled the Open Firmware/EFI password.

No login banners

Mitigation: set login banners.

Weak passwords

Mitigation: used secure minimum length alphanumeric passphrases containing special characters.

Default UMASK is insecure

Mitigation: changed the default UMASK value.

No file integrity checks

Mitigation: installed Tripwire.

No rootkit detection

Mitigation: installed chkrootkit.

Application level attacks (XSS, SQL injection, buffer overflows)

Mitigation: installed mod_security.

No HTTP encryption for sensitive data

Mitigation: Enabled mod_ssl.

Unnecessary Apache modules loaded

Mitigation: Removed unnecessary Apache modules.

Apache information leakage (version info, error messages, etc)

Mitigation: configured Apache securely.

We have reduced the vulnerabilities in our web server by configuring it in a secure manner; we cannot reduce the threat level due to the fact that a lot of web site attacks are automated, but in reducing the vulnerability of our web server, we have succeeded in reducing the level of risk that we now face.

Conclusion

This guide is intended to help anyone who is planning on running a production web server on an OS X host. We have covered securing OS X, enabling SSL for web transactions, enabling mod_security as a web application firewall, securing the Apache configuration and enabling and securing PHP on our host.

This paper does not go into depth in areas such as where on your network to place your web server, or secure network architecture design. You can find out more on these topics from the following sources:

http://www.iu.hio.no/~mark/WebCourse/node4.html

http://www.sans.org/reading_room/whitepapers/hsoffice/1645.php

Acronyms

AES - Advanced Encryption Standard, also known as Rijndael

CA - Certificate Authority

DMZ - Demilitarized Zone

DOD - Department Of Defense (U.S)

EFI - Extensible Firmware Interface, found on Intel Macs

GCC - The GNU Compiler Collection

GUI - Graphical User Interface

HFS - Hierarchical File System

HTTP - Hyper Text Transfer Protocol

HTTPS - Hyper Text Transfer Protocol Secure sockets

IDS - Intrusion Detection System

IP - Internet Protocol

IPS - Intrusion Prevention System

ISP - Internet Service Provider

MP3 - Mpeg Layer 3

MPAA- Motion Picture Association of America

OS - Operating System

OSS - Open Source Software

OS X - Operating System 10, Apple's current Operating System

PHP - Hypertext Preprocessor

PPC - Power PC

RIAA - Recording Industry Association of America

SLP - Service Location Protocol, used by OS X for Bonjour networking

SQL - Structured Query Language

SSL - Secure Sockets Layer

URL - Uniform Resource Locator

References

Apple - Mac OS X - UNIX. Retrieved October 23, 2006, from Apple Web site: http://www.apple.com/macosx/features/unix/

Apache Software Foundation. Retrieved October 23, 2006, from Apache software foundation Web site: http://www.apache.org/

About the Apache HTTP Server Project. Retrieved October 23, 2006, from Apache software foundation Web site: http://httpd.apache.org/ABOUT_APACHE.html

Apache Modules. Retrieved October 23, 2006, from Apache software foundation Web site: http://httpd.apache.org/docs/1.3/mod/

Apache httpd 1.3 vulnerabilities. Retrieved October 23, 2006, from Apache software foundation Web site: http://httpd.apache.org/security/vulnerabilities_13.html

Search results. Retrieved October 23, 2006, from Common vulnerabilities and exposures Web site: http://cve.mitre.org/cgi-bin/cvekey.cgi?keyword=Apache

Web server survey archives. Retrieved October 23, 2006, from Netcraft Web site: http://news.netcraft.com/archives/web_server_survey.html

Google. Retrieved October 23, 2006, from Google Web site: http://www.google.com

The apache interface to OpenSSL. Retrieved October 23, 2006, from mod_ssl Web site: http://www.modssl.com

Open source web application firewall. Retrieved October 23, 2006, from ModSecurity Web site: http://www.modsecurity.org/

Hypertext Preprocessor. Retrieved October 23, 2006, from PHP Web site: http://www.php.net

The FreeBSD project. Retrieved October 23, 2006, from The FreeBSD Project Web site: http://www.freebsd.org

(2006,7,6). Threats prompt Mac switch advice. Retrieved October 23, 2006, from BBC Web site: http://news.bbc.co.uk/1/hi/technology/5150508.stm

Burton, Richard (2002,5,3). Mac OS X command line 101. Retrieved October 23, 2006, from The Mac Observer Web site: http://www.macobserver.com/tips/macosxcl101/2002/20020503.shtml

Common vulnerabilities and exposures. Retrieved October 23, 2006, from CVE Web site: http://cve.mitre.org

Home. Retrieved October 23, 2006, from Zone-H Web site: http://w.zone-h.org

Digital attacks archive: today's verified attacks. Retrieved October 23, 2006, from Zone-H Web site: http://www.zone-h.org/component/option,com_attacks/Itemid,43/

Digital attacks archive: Special Defacements. Retrieved October 23, 2006, from Zone-H Web site: http://www.zone-h.org/component/option,com_attacks/Itemid,44/total%20deface

Hickman, Peter (2005,03,15). Exploring the Mac OS X Firewall. Retrieved October 23, 2006, from MacDevCenter Web site: http://www.macdevcenter.com/pub/a/mac/2005/03/15/firewall.html

Hays, Bill (2005,8,23). Configuring IPFW firewalls on OS X. Retrieved October 23, 2006, from Ibiblio Web site: http://www.ibiblio.org/macsupport/ipfw/

(2002,8,13). Manual page for IPFW(8). Retrieved October 23, 2006, from Apple Developer Connection Web site: http://developer.apple.com/documentation/Darwin/Reference/ManPages/man8/ipfw.8.html

FileVault. Retrieved October 23, 2006, from Apple Web site: http://www.apple.com/macosx/features/filevault/

de Vries, Stephen (2005,8,19). Securing Mac OS X. Retrieved October 23, 2006, from Corsaire Web site: http://www.corsaire.com/white-papers/050819-securing-mac-os-x-tiger.pdf#search=%22OS%20X%20disable%20core%20dumps%22

(2006). Mac OS X security configuration for version 10.4 or later. Retrieved October 23, 2006, from Apple Web site: http://images.apple.com/server/pdfs/Tiger_Security_Config.pdf

Pelow, James (2005,3,6). Enabling Apple's supplied PHP in OS X 10.4 Tiger. Retrieved October 23, 2006, from PHPmac Web site: http://www.phpmac.com/articles.php?view=225

Pelow, James (2005,6,1). Installing mod_security on OSX 10.4 Tiger. Retrieved October 23, 2006, from Studi2f Web site: http://www.studio2f.com/misc/2005/06/01installing_mod_security_on_osx_104_tiger.php

Modsecurity (mod_Security) - Open Source Web Application Firewall. Retrieved November 12, 2006, from ModSecurity Web site: http://www.modsecurity.org/

Download. Retrieved October 23, 2006, from ModSecurity Web site: http://www.modsecurity.org/download/index.html

Ristic, Ivan (2006,4,10). ModSecurity for Apache user guide. Retrieved October 23, 2006, from ModSecurity Web site: http://www.modsecurity.org/documentation/modsecurity-apache/1.9.3/modsecurity-manual.html#N1008C

B4, Legacy (2004,12,1). How to create a secure (HTTPS) OS X webserver. Retrieved October 23, 2006, from MacOSXhints Web site: http://www.macosxhints.com/article.php?story=20041129143420344

Security, communications and information services. Retrieved October 23, 2006, from Verisign Web site: http://www.verisign.com/

Installation Instructions- Apache Server with modssl. Retrieved October 23, 2006, from Verisign Web site: http://www.verisign.com/support/ssl-certificates-support/page_dev019509.html

Thwate. Retrieved October 23, 2006, from Thwate Web site: http://www.thawte.com/

Apache-SSL / Apache ModSSL Certificate installation instructions. Retrieved October 23, 2006, from Thwate Web site: http://www.thawte.com/ssl-digital-certificates/technical-support/apache_install.html

Maj, Artur (2003,6,23). Securing PHP: Step-by-Step. Retrieved October 23, 2006, from Security Focus Web site: http://www.securityfocus.com/infocus/1706

K, Chuck (2002,10,23). Removing the developer tools. Retrieved October 23, 2006, from Security Focus Web site: http://www.macosxhints.com/article.php?story=20021023062113136

Maxwell, Doug (2006,4,19). Five-minutes to a more secure SSH. Retrieved October 23, 2006, from Geek Pit Web site: http://geekpit.blogspot.com/2006/04/five-minutes-to-more-secure-ssh.html

Tridgell, Andrew (2003,5,26). Securing Samba. Retrieved October 23, 2006, from Samba Web site: http://samba.org/samba/docs/man/Samba-HOWTO-Collection/securing-samba.html

Firewall Configuration For OS X 10.3. Retrieved October 23, 2006, from The University Of Melbourne Web site: http://www.infodiv.unimelb.edu.au/lansg/osx/x3-firewall-configuration.html#command

Davisson, Gordon (2005). Mac OS X: What are all those processes. Retrieved October 23, 2006, from Westwind Computing Web site: http://www.westwind.com/reference/OS-

X/background-processes.html

frodo (2005,5,1). Tripwire on OS X. Retrieved October 24, 2006, from Macguru Web site: http://www.macguru.net/~frodo/Tripwire-osx.html

Chris (2006,1,30). Installing Tripwire on OS X using a .plist file. Retrieved October 24, 2006, from UMaine HPC Web site: http://www.clusters.umaine.edu/blog/2006/01/30/installing-tripwire-on-os-x-using-a-plist-file/

OWASP Top Ten. Retrieved November 12, 2006, from OWASP Web site: http://www.owasp.org/index.php/OWASP_Top_Ten_Project

GIAC security essentials certification (GSEC). Retrieved November 12, 2006, from www.sans.org Web site: http://www.giac.org/certifications/security/gsec.php

Defence in depth - Wikipedia. Retrieved November 12, 2006, from www.wikipedia.org Web site: http://en.wikipedia.org/wiki/Defence_in_depth

(2005, 4, 25). Web server attacks 'growing fast'. Retrieved November 12, 2006, from www.bbc.com Web site: http://news.bbc.co.uk/1/hi/technology/4480689.stm

Core Dump - Wikipedia. Retrieved November 12, 2006, from Wikipedia Web site: http://en.wikipedia.org/wiki/Core_dump

Rogers, Russ (1999,10,02). Security Horizon Computer and Network Security. Retrieved November 12, 2006, from Exploiting the FTP PASV vulnerability Web site: http://www.securityhorizon.com/whitepapersHacking/PASV.php

ICODES Web: DoD Login Banner. Retrieved November 12, 2006, from ICODES Web Web site: https://www.icodesweb.com/dod.icw

Guidelines For Developing A Sensible Password Policy. Retrieved November 12, 2006, from AUSCERT Web site: http://www.auscert.org.au/render.html?it=1832

Maple, Ryan (2000, 6, 24). Using umask. Retrieved November 12, 2006, from Linux Security Web site: http://www.linuxsecurity.com/content/view/117255/

Rootkit - Wikipedia. Retrieved November 12, 2006, from Wikipedia Web site:

http://en.wikipedia.org/wiki/Rootkit

(2005, 9, 20). About Disk Utility's Repair Disk Permissions feature. Retrieved November 12, 2006, from Apple Computers Web site:

http://docs.info.apple.com/article.html?artnum=25751