The

Securing Websites

Training Manual

Helping protect your website installations, code and sanity
Securing Websites
Training Manual

Helping protect your website installations, code and sanity

By Cassidy B. Larson
The Securing Websites Training Manual

by Cassidy B. Larson

No rights are reserved.

Any part of this work may be reproduced or transmitted, including copying, or other methods of transmission with or without the prior written permission of the author.

For information regarding this work, please contact the author.

Cassidy B. Larson
Chief Technical Officer
InfoWest, Inc
cbl@infowest.com
# Table of Contents

## SECTION ONE  
7  
**Introduction** ................................................................. 7  
**A Secure Foundation** ......................................................... 8  
**Intro to PCI Compliance** .................................................... 9  
  Requirements ................................................................. 9  
**LAB EXERCISE** ................................................................. 11  
  Are You PCI Compliant? ..................................................... 11  
  Exercise ....................................................................... 11

## SECTION TWO  
12  
**Securing the Operating System** ........................................... 12  
  Update, Update and Update .............................................. 12  
  Network and Application Strategy .................................... 13  
  Firewall Voodoo ............................................................ 14  
  Remote Access Lockdown ................................................. 14  
  SSH Suggestions ............................................................ 14  
  Remote Desktop Suggestions ........................................... 16  
  Be Aware ................................................................. 16

## SECTION THREE  
17  
**Basics of Securing Apache** ............................................... 17  
  The Basics ................................................................. 18  
  Hiding Sensitive Information about your Apache Install ............ 18  
  What user and group run Apache? ...................................... 19  
  Making sure access to files outside of the web root is DENIED .... 19  
  Directory Browsing ......................................................... 19  
  Turning off Server Side Includes ...................................... 20  
  Turn off CGI Execution .................................................... 20  
  Don’t allow Apache to follow Symbolic Links ......................... 20  
  Turning Multiple Options On and Off at a time ..................... 20  
  Disable Unnecessary Apache Modules .................................. 21  
  Who owns the Apache config files and binaries? ..................... 21  
  Tweaking the KeepAlive Settings ....................................... 22  
  Restricting Access by IP ................................................... 22  
  Restricting Access by Username/Password ........................... 23  
  Running Apache in a FreeBSD Jail .................................... 26  
  Additional Readings ......................................................... 26  
  Conclusion ................................................................. 26  
**LAB EXERCISES** ............................................................... 27  
  What are YOU showing the world? ..................................... 27  
  Setting up User Authentication ......................................... 28  
  Restricting Access by IP Address ...................................... 29

## SECTION FOUR  
30  
**Advanced Apache Security** ................................................ 30  
  Improving Cipher Suite Security ....................................... 30
Disable Trace Capability
Using the Files Option
mod_security
  Mod_Security Basics
  The Actions
  The Filters
  The Core Rules
HTTP:BL
  DNS Query Format
  Response Syntax
  No Results
  Implementation
  More Details

LAB EXERCISES ................................................................. 40
Will you pass a PCI Scan? 40
  Trace and Track Verifying
  Does your SSL Enabled Host Support SSLv2?
  Does your SSL Enabled Host Support the Weak Ciphers?
Mod_Security Exercises
  Basic Mod_Security Rules
  Mod_Security Core Rule Set

SECTION FIVE ................................................................. 43
Securing PHP ................................................................. 43
  Update to PHP 5.3.x Now!
  MailHeader Patch
  Random Number Generation Gotchas
  Salting your Passwords
  Validating E-Mail Addresses
  Using Suhosin
  XSRF Preventions
  Using Database Abstraction Layers
  Using Captchas
  phpMyAdmin Gotchas
    Locking phpMyAdmin Down

LAB EXERCISES ............................................................... 52
  Put Some Salt In Your Passwords
  Validate User-Submitted E-Mail Addresses
  Catchya..Captcha

SECTION SIX ................................................................. 53
Securing MySQL ........................................................... 53
  Adjusting The Bind Address
  Who's Running MySQL?
  Protecting the Socket File
  Setting the Root Password
  Cleaning out Anonymous Users and Test Database
  Using MySQL Prepared Statements

LAB EXERCISES ............................................................... 56
  Securing MySQL
Introduction

Since the inception of the Internet, web servers have been at the heart of its content distribution. For servers that are considered to be running 24x7x365.25, security needs to be a major focus. It seems that almost daily we’re hearing of some web application that has a new vulnerability, a new attack vector for hackers or perhaps crackers and the like to gain access to your important information.

Web servers are typically hooked into the backbones of your network. Now-a-days they’re typically not behind a connection on T1 speeds, nor located on an end-users home DSL or Cable connection. Instead, these machines are housed in high-speed data centers, on cloud systems or located in places where bandwidth is readily available to do what they do best, serve up their content. Ensuring that web servers are secure from outside attack should be a prime concern for any organization who relies on them.

In the past few years attacks on web severs have increased dramatically. It does not matter where your server is housed, as malicious code exploitation respects no boundaries of nations or countries. If your site is accessible from the Internet, you are vulnerable to exploitation.

Even though web servers are vulnerable due to their “open” nature, they can be protected. It requires a unique understanding by the administrator and programmers of the current trends and tactics that are being used in todays world. It requires constant vigilance of the software and packages installed to make sure that no new vulnerabilities exist. If your “open-to-the-public” website relies on an application that has a known vulnerability and exploit, it is only a matter of time before someone exploits it on your server.

Within this training manual, I will go through a number of different scenarios and situations. We will discuss best practices on how to keep our web servers secure. The appropriate ways to keep our applications up-to-date, how to secure our websites and installations and how to write great apps in order to be safe from possible exploitation. We will also discuss some of the basic things you’ll need to do in order to pass a PCI compliancy test on your web server.

There are countless discussions on how to correctly secure web sites on the Internet. It is my hope that I will be able to discuss with you some of the key things that have helped me in my day-to-day live as a Systems Administrator. I deal with customers that have had their code exploited and manipulated. My hope is that by using this training manual you’ll be better prepared against the problems that face us as Administrators of Web Sites.
A Secure Foundation

In order to assure that you have a secure website and web application, you must be absolutely sure you have a secure server to host it on. This means that one must build upon a solid foundation of an operating system that is fully patched, a web server software that is up-to-date and patched, as well as keeping your scripting languages of choice updated, patched and secure.

Below you will find a diagram of the layers that make up a web server. Each one of the blocks represents an attack vector.

![Diagram of web server layers]

Obviously, if a hacker were to gain access through a major vulnerability in the Operating System, then all the services, websites and data on the server could be accessed with ease. It is crucial that we keep our Operating Systems patched with the latest security patches, as typically there are exploits in the wild within days of a security patch being released (if not sooner).

The same goes for any of the services, applications and installations you run on your web server. There are numerous vulnerabilities that go un-patched on web servers around the world because their administrators are not vigilant or aware of the vulnerabilities.

I suggest you make a list of the software, services and applications you run on your web server, as well as the version numbers. This will help you in identifying when you might be “at risk”. If you get an email from the security-announcement-list about WordPress being vulnerable to a major security hole with versions 2.2 and below, you can quickly determine if you need to patch.

There are many threats an administrator of a web server needs to be vigilant for. I won’t be able to discuss everything in detail in this training manual, but ideally I’ll give you a few good pointers to start securing your websites against the most common exploitable threats there are on the Internet today.
Here is a list of a couple of suggestions for keeping your box secure:

- Know what is on your web server and keep it patched.
- Make a list of applications, services and versions you need to function.
- Find out what is not needed. Disable those services, libraries and applications.
- Lock your machine down tight via Firewall rules and user based permissions.
- Subscribe to the security announcement mailing lists for your Operating System, Web Server Software, Database Software and Downloaded Libraries or applications.
- Use an anti-virus and enable its automatic updates.
- Think like a hacker.

**Intro to PCI Compliance**

Unless you’ve been hiding under a rock for the last few years, or are brand new to the website administration game, you probably have already heard of PCI Compliance.

The Payment Card Industry (PCI), in the early 2000’s, decided to make a security standard that all organizations that accept credit or debit cards should follow. Ideally, this was to deter credit card theft, fraud and provide enhanced security to make everybody’s life more safe when buying things online.

On December 15th, 2004 the PCI Security Standards Council (PCI SSC) released the Data Security Standard (PCI DSS). The initial intention of this DSS was to create an additional level of protection for card issuers by ensuring that merchants meet minimum security standards when they store, process and transmit cardholder data.

Since 2004, there have been a few revisions to the PCI DSS. Most of these revisions were only for enhanced clarification and consistency in the already defined standard. The latest standard is at version 1.2.1.

**Requirements**

The current version (1.2.1) of the PCI Standard specifies 12 requirements for compliance:

1. Install and maintain a firewall configuration to protect cardholder data.
2. Do not use vendor-supplied defaults for system passwords and other security parameters.
3. Protect stored cardholder data.
4. Encrypt transmission of cardholder data across open, public networks.
5. Use and regularly update anti-virus software on all systems commonly affected by malware.
6. Develop and maintain secure systems and applications.
7. Restrict access to cardholder data by business need-to-know.
8. Assign a unique ID to each person with computer access.

9. Restrict physical access to cardholder data.

10. Track and monitor all access to network resources and cardholder data.

11. Regularly test security systems and processes.

12. Maintain a policy that addresses information security.

Some of these requirements are a given for any website administrator. Some, however, will require some documentation, work and stress in order to honestly be in full PCI Compliance.

Now-a-days there are many online companies capitalizing on the fact that online merchants need to be PCI compliant in order to accept credit cards. Numerous online organizations are offering “PCI Security Scans” of your web servers, for a nominal fee, in order to determine if you are “PCI Compliant”.

These organizations will typically run a program such as Nessus (www.nessus.org) against the external public IP address you request. Such scans will check for known insecure vulnerabilities, open ports, un-patched software installations, as well as SQL injection vulnerabilities, brute force attacks, and the like.

Supposedly, all merchants are required to do two things; run quarterly PCI Scans on all public IP addresses, and submit a yearly report on compliance.

In the lab section below, you will be able to ask yourself how you think you would stand against an initial PCI scan.
LAB EXERCISE

Are You PCI Compliant?
The purpose of this exercise is to simply ask yourself the twelve PCI requirements to see if you think you are PCI compliant. Typically, a PCI scan will reveal weaknesses in your installations that will need to be patched and addressed, but it is always interesting to see the differences in a before and after scenario.

Exercise
Truthfully answer these questions in relation to one of the websites/webservers in which you maintain/administer which may or may not accept credit or debit cards:

<table>
<thead>
<tr>
<th>Question</th>
<th>True</th>
<th>False</th>
</tr>
</thead>
<tbody>
<tr>
<td>The Web Server has a custom firewall and is actively maintained</td>
<td></td>
<td></td>
</tr>
<tr>
<td>We have changed all default system and application passwords</td>
<td></td>
<td></td>
</tr>
<tr>
<td>We have taken steps to protect cardholder data</td>
<td></td>
<td></td>
</tr>
<tr>
<td>We employ encryption methods when transmitting cardholder data across public networks</td>
<td></td>
<td></td>
</tr>
<tr>
<td>We use and regularly update our anti-virus software on our web server</td>
<td></td>
<td></td>
</tr>
<tr>
<td>When developing custom website applications we are security conscious and sanitize all user submitted data</td>
<td></td>
<td></td>
</tr>
<tr>
<td>We restrict card holder data to only those that need to know it</td>
<td></td>
<td></td>
</tr>
<tr>
<td>We have assigned a unique ID to each system and web user on the system</td>
<td></td>
<td></td>
</tr>
<tr>
<td>We have restricted physical access to the machine(s) with cardholder data</td>
<td></td>
<td></td>
</tr>
<tr>
<td>We track and monitor all access to card holder data and network resources</td>
<td></td>
<td></td>
</tr>
<tr>
<td>We regularly test our software for vulnerabilities and holes</td>
<td></td>
<td></td>
</tr>
<tr>
<td>We have a maintained policy for information security</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Totals: 11
Securing the Operating System

Depending on how you look at things, the Operating System is the bottom layer to what powers your web site. If it’s not secure, nothing else that rides on top of it can be considered secure.

There are many different Operating Systems which will serve up Web Sites. The most common are some sort of flavor of Windows and Linux. In this section, we will not go into the nitty gritty details of what to type, click or run. Instead, I will give a brief overview of a few of the important functions you will want to look out for on your Operating System in order to keep your web site installation secure.

Authors Note: There was a detailed training guide on Ubuntu Security that was provided at the 2010 UtahSAINT Conference during the “Securing Ubuntu” course. If you run Ubuntu for your web server Operating System, you will want to review the training material from that course for plenty of detailed steps on security.

Update, Update and Update

I can never stress enough the importance of keeping your systems up-to-date. It is one of the first thing the malicious miscreants will go after. It is the key to drive the whole ship. Once someone has broken in and gotten root or Administrator privileges on your web server, GAME OVER. You might never find out to what extent things have been compromised on the machine and what might have been installed without your knowledge. It is better to just back everything up, reformat and start re-installing from scratch.

This is why I always suggest to my clients to keep their Operating Systems up-to-date. All it takes is one root-escalation that is easily exploitable, and all of their credit card transactions, data and customer information is now in the hand of some kid in North Korea.

Keep a regularly scheduled maintenance window.

Just because you might not use it, doesn’t mean that you shouldn’t plan on it. 0-day bugs are becoming more and more common place, and if you already have a scheduled maintenance window for Wednesdays at 3am, then there is no need to re-schedule a maintenance window when is convenient for everybody.
**Keep on top of Security Announcements**
As I have stated in previous sections, it’s imperative that you are aware of the vulnerabilities in your Operation System as they become known. The perfect way of being aware of them is by subscribing to the Security Mailing Lists for your Operating System.

Here is a list of a few mailing lists or RSS feeds you may want to frequent depending on your installation:

- FreeBSD Security Announcements: [http://security.freebsd.org/](http://security.freebsd.org/)

I’m sure there are plenty of others, just ask Google and you will find them.

Once you receive a vulnerability you can gauge if it requires immediate patching of the Kernel or affected software, or if you should delay it accordingly.

**Network and Application Strategy**
Why is this under “Securing the Operating System”? Great question! In order to proceed with securing an OS you need to have some sort of idea what the layout of the application you plan on running on your web server is.

There are an infinite number of possible solutions to designing a website with databases, custom applications and the like. Here are a few of my suggestions for planing for proper network access:

- Keep the database on a separate host, on a private VLAN or **secondary** Network Interface that is NOT directly accessible from the public Internet. If your database instance needs to be hosted on the local machine, bind it to ‘localhost’ only. If it needs accessibility from other machines, bind it to the private IP address **only**. NEVER allow direct public access to your database, even if you have it firewalled off. This will help with your PCI Security Questions.

- **Log Everything!** This includes logging emails that get generated by your Web Application (PHP has a nice Mail Header patch, to log in the header what script was ran to generate the Email, useful for finding out later if someone exploited a script and determining which one).

- Running Memcached? If it’s run on the local machine then bind it to ‘localhost’. Otherwise, limit its access to private network interfaces only. There is no need to run this on a public IP address.

- Are users uploading files via the web or FTP? What plans do you have to scan them for viruses? Where are they stored? Are users locked down to only their directories, or can they see all other uploads?

- What scripting languages are needed? Why have Python on your web server if PHP is the only thing being used?

A lot of these questions and more are dealt with in a regular “Securing your Box” type of course. However, you do need to be watch out for simple things like these when building and securing websites in general.
Firewall Voodoo

As one of the main questions on the PCI Compliance List relates to Firewall.

My company runs a number of public hosting web server in the Private Sector. In the years past I have come to find out that many exploitations can be stopped dead in their tracks when a robust firewall is in place.

The biggest suggestion I can make for anybody running a public facing web server is using the following rule in their Firewall lists:

**Deny all requests initiated FROM your web server TO the public Internet.**

This may seem a bit “over the top” and paranoid, but trust me, it works wonders. One of the most popular ways for malicious script-kiddies to exploit your web applications is by getting your web server to download their custom code. After they have had your application download it, they can run it at which point you security is toast. Soon enough, they’ll have an IRC connection opened back to their server so they can control your server, and then they’ll start serving up Anime from your web server, or worse.

By denying requests from your web server to the Internet, it prohibits your machine from downloading these exploitable hacks. When they can’t download them, they can’t run them. You will always have end-users that write code that is exploitable. Especially in the private sector, I can’t monitor 24x7 all of the code which my users upload to their paid hosting accounts.

Obviously, you’ll want to allow traffic like DNS, NTP, Syslogs and such to make its way out through the firewall. This is also true for inbound requests from the Internet to your webservice. Those that already have a TCP state, you can let through just like normal. Just deny all outbound requests that don’t already have a session open.

Typically, you’ll have to make exclusions for this by whitelisting hosts/IP addresses that are allowed to be accessible. I have seen this for those websites that need to grab UPS/FedEx shipping rates directly or perhaps check if their software versions are running the latest patched copies. These should be valid exceptions you would allow requests out for.

Trust me, it’s much easier to add an IP to a whitelist than having to repair an exploited machine. You’ll also probably have to email all of your users because you don’t know how much of their credit card data has been exposed to the miscreants and in the private sector security breaches of credit card information will run you out of business.

Remote Access Lockdown

As a website administrator, you’ll more than likely need remote access to your web server. This is done, typically, via SSH on a UNIX machine or Remote Desktop on a Windows Server. In this section we will talk a little about locking down access to these remotely accessible processes.

**SSH Suggestions**

Since most UNIX installations use SSH for remote access, it is a common attack vector used for exploiting ones machine. I have seen countless coordinated attacks against some of my machines that have public accessible SSH instances.

I have a couple suggestions and recommendations that have worked for me in the past for preventing my SSH instances from getting exploited.
Disable Root Login via SSH
Make sure in your `sshd_config` configuration file that you have the following set:

```
PermitRootLogin no
```

This line will prohibit users from being able to login as the root user via SSH. Typically, it is disabled already on most installations, but it’s always an important thing to watch for. If you really do need root login for scripts or some other random purpose, use SSH keys and only allow the root user to login via key login. To do this, use the same line as above but change the ‘no’ to ‘without-password’ as shown here:

```
PermitRootLogin without-password
```

Decrease Login Grace Time
Anybody that’s trying to login to your machine does NOT need to have a login prompt for two minutes before your SSH instance will disconnect them. It is crazy that the default on some installations is so high. I recommend you set it to 10 seconds with the following line in your `sshd_config` file:

```
LoginGraceTime 10s
```

10 seconds might still seem a bit high, but for those with extra long passwords, it should be plenty. Unless they hunt and peck. Increase or decrease the value accordingly. Note that for minutes you’ll change the “s” back to a “m”.

Decrease Default Maximum Authentication Tries
How many times do you want to give a user, or a password guesser the opportunity to retry a password before the SSH process disconnects him? By default the value is set to six on most installations. I recommend setting it to four to be a little bit more restrictive. Set this value in the same `sshd_config` file as the above examples:

```
MaxAuthTries 4
```

Tweaking the hosts.allow or hosts.deny file
Depending on your UNIX distribution, you may want to use a hosts.allow or hosts.deny file for limiting and restricting SSH access to your web server. If you are not able to firewall your SSH service in your OS firewall, this would be the next logical place to allow or deny specific IP addresses or networks access to the SSH service.

To set it up just follow this example:

```
sshd : A.B.C.D : allow      (If you want to allow a single IP address)
sshd : A.B.C.0/255.255.255.0 : allow (If you want to allow a whole network)
sshd : A.B.C.D : deny      (If you want to deny a single IP address)
sshd : A.B.C.0/255.255.255.0 : deny (If you want to deny a whole network)
```

Replace 255.255.255.0 with the appropriate subnet mask in the network you want to allow or block. If you’re restricting access to your SSH process via the hosts.allow/deny file, you’ll want a final line that determines what should happen for all that have not matched to this point.

```
sshd : ALL : deny (If you want to deny all IPs that did not match)
```
Change the ‘deny’ in the line above to ‘allow’ if you would rather want to allow all hosts.

**Fail2Ban**

In my organization, I have a silly requirement to keep SSH open to the public, so my customers can access their websites via SFTP. This, however, causes all the bot-nets in the world to try and guess passwords on these servers. This can be a minor problem, as I don’t want people that are guessing accounts to keep trying until they break in. A perfect solution for such a problem is **Fail2Ban**.

I have tried many programs that watch the output from the SSH logs for possible break-in attempts and block the IP addresses at the firewall level. **Fail2Ban** so far is my absolute favorite. It is written in python and comes with a number of default plugins for monitoring abuses to DNS, Apache, SSH, Mail, etc. The administrator of the web server is able to set thresholds, white list IP addresses and networks, custom times for blocking, sending custom emails upon blocking or un-blocking of an abuser. It meets my needs perfectly.

I highly suggest anybody needing to secure their web server implement this application. Have it scan your SSH logs. Have it watch your Apache Error logs for people trying to guess where your application resides. If there are too many errors, it’ll block them for the time you specify.

Check it out at: [http://www.fail2ban.org](http://www.fail2ban.org). Most distributions should have a package or port already made for easy installation.

**Remote Desktop Suggestions**

While I am in no ways saying I am a Windows Server expert, I have found that Remote Desktop is one of the most popular attack vectors for malicious users to gain access to your web servers.

Here is a short list of common security practices you should remember when dealing with Remote Desktop on a Windows based Web Server:

- Require all members of Remote Desktop Users group to logon with strong passwords.
- Restrict **WHO** can access Remote Desktop and from **WHERE**.
- Don’t login as **Administrator**.
- Configure Disconnect and Reset Timeouts

**Be Aware**

To summarize, if you can keep your Operating System, as a whole, secure, patched and off the radar from being exploited, it’s one less thing you’ll have to worry about. When an OS is exploited, **everything** is accessible. However, when a website is exploited through a coding weakness, the exploit is typically limited to only the data that the website user or application has access to.
Basics of Securing Apache

Whenever you talk about web sites and security, you must mention the web server software that serves up your pages. In this section, we will discuss the appropriate methods for securing the web server software, Apache.

Apache came about in 1995 after development of the much popular NCSA httpd had stalled. Using NCS httpd 1.3 as a base, the group of core founders applied all of the published bug fixes and enhancements and made their first official public release of the Apache web server.

Since that time, Apache has grown to be the number one web server software for over 14 years.

As of September 2010, the latest stable release of Apache is version 2.2.16.

You should note that each minor release of Apache (ie 2.2.16 to 2.2.17) is something to look into as soon as it is released. Typically, there is at least one security fix implemented into the code base, if not multiple.

For a list of what vulnerabilities are fixed in the latest Apache 2.2 line, pull up:

http://httpd.apache.org/security/vulnerabilities_22.html

The Basics

As I’ve stated above, the first step you need to do is make sure you have the latest Apache revision installed. Since there are many security fixes involved in each release, the further you are behind the live release, the more vulnerable you might become to potential security threats against your Apache installation.

After you’ve secured your Apache, by updating it, the next trick is to hide the Apache version number, and other sensitive information.

Hiding Sensitive Information about your Apache Install

Why bother with telling the whole world what version of Apache you’re running? It just opens the door and broadcasts to the world to let anybody know what versions you’re vulnerable to! By default, Apache will also tell what operating system you’re running as well as what modules are installed (PHP, mod_python, mod_perl, etc).

There are two lines in the Apache configuration file (httpd.conf) that you need to add or edit:

ServerSignature Off
ServerTokens Prod

The ServerSignature line appears at the bottom of each page generated by Apache, pages such as 404 errors, directory listings, etc. The ServerTokens directive is used determine what Apache will put in the Server HTTP response header. The options you could choose are the following:

Full | OS | Minor | Minimal | Major | Prod

Where Full displays the most information and Prod displays the least. By setting the value to Prod, this tells Apache to set the response header to the following:

Server: Apache

If you are extra security paranoid, you can change the “Apache” in the response above to be something totally unique (like “Microsoft IIS/4.0”) by using mod_security (we’ll talk about this in detail later) or by modifying the Apache source before you compile.

PCI Security Notice:
You need to apply the configuration change above, in order to pass a PCI Security Assessment Scan.
What user and group run Apache?
Some Apache installations have their Apache process run as the user nobody. This is a big no-no when securing Apache from baddies. Suppose your Apache install, and your mail server are both running as nobody, an exploitable bug from Apache may allow access to your mail server as well. The same is true in reverse.

Typically, you will not need to change these values as your binary or package install will be sufficiently secure and not ran as nobody or root.

User apache
Group apache

On FreeBSD port installations you may see the following:

User www
Group www

If, however, you see that Apache has its User or Group lines set to root or nobody from the configuration files, you will want to create a new user and group specifically for Apache to run as.

Making sure access to files outside of the web root is DENIED
In any sort of sane world, we do not want to allow Apache to be able to access files outside of its web root. Doing so, would be a bit insane. Assuming all of your web sites are placed under a single directory (/var/www in this case), you would set Apache up as follow:

```
<Directory />
  Order Deny,Allow
  Deny from all
  Options None
  AllowOverride None
</Directory>
<Directory /var/www>
  Order Allow,Deny
  Allow from all
</Directory>
```

As you may have noticed above, we set Options None and AllowOverride None. This turns off all options and extra overrides for the entire server. This does, however, require that you add them back explicitly for each directory that requires them.

Turning off Directory Browsing

Directory Browsing
Sometimes you’ll want to allow (or deny) users (and search engines) to browse all of the contents in a particular directory. This can easily be done by using the Options directive within a Directory tag. You can set it to either None or -Indexes, to disable. To Enable this feature, simply use Indexes (without the -). In the example below we are disabling Directory Browsing on a particular sub-directory (private) of a site:

```
<Directory /var/www/mysite.domain.com/private>
  Options -Indexes
</Directory>
```
I recommend you don’t put anything within a browsable directory that you don’t want to show up in Google searches. I know of someone who put a MySQL dump of their customer contact database on their home web server for after-hours working. That server where they put the file had Directory Indexes enabled. After Google got ahold of that information, it became easy to see who else was customers of the same organization. This is not something you want when trying to secure your website.

**PCI Security Notice:**
PCI Compliance Scans will recommend that Web Directories should NOT be browsable. You’ll be recommended to disable Directory Indexes for the site to avoid any warnings on your PCI Scan Summary.

**Turning off Server Side Includes**
Using the `Options` directive inside of a `Directory` tag you can disable Server Side Includes. Set `Options` to either `None` or `-Includes`. In the example below we are disabling them for my blog site which happens to require CGI execution:

```html
<Directory /var/www/blog.cbl.us>
  Options -Includes ExecCGI
</Directory>
```

**Turn off CGI Execution**
If you’re not going to be using CGI on a site or in a directory, you can turn it off by using the `Options` directive inside of a `Directory` tag. Simply set `Options` to either `None` or `-ExecCGI`. In this example I decided my photos directory does NOT need CGI, but I do want to allow Directory Indexes:

```html
<Directory /var/www/blog.cbl.us/photos>
  Options -ExecCGI Indexes
</Directory>
```

**Don’t allow Apache to follow Symbolic Links**
If, perhaps, you do not want Apache to follow Symbolic links (or maybe you do in one sub-directory and your global site `Options None` is preventing it) you can enable or disable such feature by using an `Options` directive within a `Directory` tag. Just throw a `-FollowSymLinks` after the `Options` tag to disable symbolic links following (Get rid of the hyphen to enable symbolic link followings). In the example below I disabled symbolic links, whilst enabling Directory Indexes:

```html
<Directory /var/www/blog.cbl.us/movies>
  Options -FollowSymLinks Indexes
</Directory>
```

**Turning Multiple Options On and Off at a time**
As you may have already guessed from the above examples, you can enable and disable multiple options using the `Options` directive. I know I prefer the flexibility to have finer grained control over a per-directory (or per-site) customizations. It all allows us to use a few, many or none of the options wherever we so desire.
As you might have already come to realize, in order to disable multiple Options just throw a None at the end of an Options directive:

```xml
<Directory /var/www/blog.cbl.us/movies>
  Options None
</Directory>
```

In order to enable or disable multiple Options at once, just put a space between the different Options:

```xml
<Directory /var/www/blog.cbl.us/movies>
  Options -Includes -Indexes -ExecCGI
</Directory>
```

Combine them however you see fit.

**Disable Unnecessary Apache Modules**

By default, Apache will typically come with a load of modules already configured and ready to load. Sometimes they will be loaded into the Apache config for you, depending on your package maintainer. If this is the case, this tends to add excess memory needs and opens you up to potential security holes if there are un-patched bugs in a particular module.

Look for lines in your `httpd.conf` configuration file that start with `LoadModule`. These are the lines that you can comment out if you find a particular module that you are sure you don’t need for your installation.

Here are a couple that you may not use and could safely comment out (again, all depending on your application needs):

```
mod_autoindex, mod_dav, mod_dav_fs, mod_speling, mod_userdir,
mod_include, mod_imap, mod_imagemap, mod_info, mod_cgi
```

Some package maintainers on Debian and Ubuntu will give you a minimalistic approach and only load the modules you actually need in the default `httpd.conf` configuration file. This is great! Others, especially if you compile from source, will throw a big long list in there.

Either way, it will still be wise for the security-aware web server administrator to check out the configuration file and see which modules are being loaded. Familiarize yourself with what they do and which you might be able to live without being loaded.

**Who owns the Apache config files and binaries?**

Who owns your Apache configs and binaries? Normally, this is the root user. If you installed from a pre-bundled package, you normally do not need to worry about this. However, it is still wise to check none-the-less. Depending on your distribution and OS your config files may reside in a number of different places: `/etc/apache2` or `/usr/local/apache/conf` or `/usr/local/etc/apache2` or somewhere else. The same goes for the binaries.

Once you find the Apache configs, make sure that nobody but the root user and group (or wheel group) has access to read and modify these configuration files. Letting scripts access these configs is like giving someone a blue print to how to exploit your system. I suggest a chmod of 740 on all the Apache configuration files.

The same goes for the Apache binaries. Make sure the root user and group (or wheel group) are the owners for the Apache binaries. Make sure that nobody but root has write permissions over
these files, otherwise you might find that someone may have overwritten your Apache binary with a custom version.

**Tweaking the KeepAlive Settings**

If you have a busy site, you will want to play with the KeepAlive settings. It will require some fine tuning to get the best result, but could help fend off potential DOS attacks by limiting how long Apache processes are tied up waiting for responses.

The default `KeepAliveTimeout` is 15 seconds. Apache documentation says that setting this to a high value may (will!) cause performance problems in heavily loaded servers. The higher the timeout, the more server processes will be kept occupied waiting on connections with idle clients. By checking the `server-status` (we’ll talk more about this later), you’ll be able to fine tune accordingly to your load and need.

The other directive to look into for DOS prevention is the `MaxKeepAliveRequests`. It limits the number of requests allowed per connection when the `KeepAlive` directive is set to on. If it is set to 0, unlimited requests will be allowed. The default is set to 100. You’ll want to keep it high for maximum performance, but not too high that you might get DOSed and your web server becomes unresponsive. Using `ab` (Apache Benchmark Tool) you can help get a better idea of what is the best setting to use.

On one major site I run, I have the `MaxKeepAliveRequests` set to 100, while I modified the `KeepAliveTimeout` to 10 seconds. On the other hand, on another server I turned `KeepAlive Off` because of the requirements of the static site (memcached, mod_python and Django). This server was pushing 95Mbps when it was mentioned on primetime TV, without breaking a sweat.

**Restricting Access by IP**

At times you will be required to restrict a particular site or directory down to a specific IP address or block of addresses. Using the `Directory` tag and some `Allow from` and `Deny from` lines, it can’t be easier!

```
<Directory /var/www/blog.cbl.us/admin>
  Order Deny,Allow
  Allow from 192.168.10.0/255.255.255.0
  Allow from 10.0.0.0/8
  Allow from 209.3.1.3
  Deny from All
</Directory>
```

In the above example you’ll notice I am restricting access to my admin directory of my blog for everybody that does not match the `Allow from` lines. If the user is accessing Apache from a 192.168.10.0/24 or 10.0.0.0/8 or 209.3.1.3 network range (or IP), then they’ll be permitted to pull up the folder otherwise they’ll get an error. You have the flexibility of using a CIDR notation (/24) or a subnet mask depending on your preference. Other options include matching on hostname or domain name, but I still prefer the IP network approach for finer grained lock downs.

The `Order` line can be altered with the following results:

```
Order, Deny: First, all Allow directives are evaluated; at least one must match, or the request is rejected. Next, all Deny directives are evaluated. If any matches, the request is rejected. Last, any requests which do not match an Allow or a Deny directive are denied by default.
```
Deny, Allow: First, all Deny directives are evaluated; if any match, the request is denied unless it also matches an Allow directive. Any requests which do not match any Allow or Deny directives are permitted.

Restricting Access by Username/Password
There are a couple of different ways to restrict access to a site (or directory) by username and password authentication. You can build it into your web application (PHP, Perl, etc) to require authentication before displaying the page, or you can use Apache’s built-in authentication modules.

Apache has a couple of different stock modules for authentication, depending on your needs and security desires. These two are named mod_auth_basic and mod_auth_digest. Ideally, you should have both available to you for your authentication selections. If not, you'll typically be missing mod_auth_digest from your install as some package maintainers don't include it (as it is still considered ‘experimental’).

mod_auth_basic with mod_authn_file
Apache’s simplest authentication module is mod_auth_basic. This module is what most users use when needing to lock down a directory or site quick and easily. It just works.

Here’s how to get it to work.

First off, you’ll need to create a password file. This file should be placed somewhere that is inaccessible from the web. This prevents people from downloading the file, cracking it, and then gaining access to your site due to insecurity. I typically create a sub-directory below the root of the website for any password files and global includes that should NOT be web accessible.

To create the password file you will need to use the htpasswd utility that came with your Apache installation. Using the following syntax you will tell Apache to create the new password file for the specified username:

    htpasswd /path/to/password/file <username>

After pressing enter, htpasswd will prompt you for the password twice to make sure it is correct and then it will create the file with the encrypted password (if it did not already exist). Note that if you are running this command to add another user to the same password file, htpasswd will append any new usernames entered to the end of the password file.

By default htpasswd will use a CRYPT encryption of the password you enter. In this case Apache uses the traditional UNIX crypt style function with a randomly generated 32-bit salt (however only 12 bits are used) and the first 8 characters of the password entered. Instead, you can specify the “-m” or “-s” option when running htpasswd to use an MD5 or SHA-1 encryption method. MD5 will produce a 128-bit hash from a combination of a random 32-bit salt and the password whereas SHA-1 will produce a 160-bit digest of the hash (note that Base64 encoded version of the SHA-1 digest is stored in the password file).

Now that you’ve created a password file, the next step is to tell Apache what directory you want to password protect. To do so, you’ll need to use the following example and enter it into your Apache configuration file:
Let's break this down line by line. **AuthType Basic** tells Apache which method will be used to authenticate the user. In this case we're using Basic. This is the simplest method to authenticate a user, however you should realize that passwords will be sent unencrypted. If you wish to send the password encrypted you'll need to use another Authentication Type: **AuthType Digest**. We'll cover digest authentication in the next sub-section.

The next line up is the **AuthName**. This is shown in the browser password popup box that the user sees when a username and password is required. It is also used by the browser as a realm to re-use the same username and password combo for any area on the same server with the same AuthName. This allows you to secure more than one directory with the same password file, by using the same realm name.

After AuthName is **AuthUserFile**. This is the path to the password file that we already created.

The Require line allows you to restrict access to this directory to a single user, a group of users, or simply any valid-user. To limit it to a single user change the line to read:

```
Require user john.smith
```

This will prevent any users in the password file, even thought they might successfully authenticate from gaining access to the directory. You can see we can get very detailed in what users are allowed or denied access to specific directories.

Site note: If you plan on doing your authentication in an `.htaccess` file, you'll need to make sure your Apache config has the following **AllowOverride directive** enabled:

```
AllowOverride AuthConfig
```

If you have disabled all overrides in Apache (which is typically most secure), you will not be able to set any authentication parameters in the `.htaccess`.

**mod_auth_digest**

Using **mod_auth_digest** is very similar to using Apache's **mod_auth_basic** from above, it only requires a few tweaks but the same standard idea persists. Using the Directory example above, you simply will need to replace **AuthType Basic** with **AuthType Digest** and replace AuthUserFile with AuthDigestFile (only if you are running Apache prior to 2.2). You will also need to add an **AuthDigestDomain directive**, which we'll discuss after the example:

```
<Directory /var/www/blog.cbl.us/admin>
    AuthType Digest
    AuthName “Blog Admin”
    AuthUserFile /path/to/digest/password
    AuthDigestDomain /admin/ http://admin.blog.cbl.us/
    Require valid-user
</Directory>
```
In the example above, you’ll notice that things are quite similar to the mod_auth_basic ways of doing things. One of the new changes, however, is the fact that we’re using a new program to create our password digest file, htdigest. Instead of using htpasswd, like we previously used, we’ll want to use htdigest for creating our password users.

When using htdigest, you’ll need to specify three things. The location to the password file, the realm to which the username belongs and finally, the username to create or update in the password file.

```
htdigest /path/to/digest/passwords <realm> <username>
```

If we wanted to create a password for an ‘admin’ user using the Directory example from above we’d run the following command:

```
htdigest /path/to/digest/passwords "Blog Admin" admin
```

Note that because our realm in the example from above had a space, we should encapsulate it with quotation marks.

If your password file does NOT already exist, you will need to specify a “–c” option before the digest password file path. This will create the file for you. If you specify the “–c” option on an already existing file, it will wipe out all of the existing contents and replace it with the new username you specify.

The AuthDigestDomain directive allows you to specify one or more URIs which are in the same protection space. This means that they’ll use the same realm along with the same username and password. They can be absolute URIs or relative. You should always at least specify the root URI for the directory on which you wish to protect. Otherwise, your browser will send an Authorization header for every request sent to the server.

Digest authentication provides a more secure password system than the basic method. However, digest auth only works with newer-age browsers. If you’re using IE 5 or 6, there are workarounds to getting it to work (see the Apache Manual). Any browser since 2002 should work fine, however because of the old browsers still in existence you should be prepared that some might not work. It is a good reason to get your users updated to a modern browser though.

In Apache version 2.2, the AuthDigestFile directive was moved back to the AuthUserFile. If you are running version 2.2.x and are getting an error similar to: Invalid command 'AuthDigestFile'. Then you’ll want to change your AuthDigestFile directive to AuthUserFile.

Others
Besides the above two authentication modules in Apache, there are a number of other modules you can add and use in Apache. Here is a list of a couple that might peak your interest:

    mod_auth_ldap
    mod_auth_mysql
    mod_auth_radius
    mod_auth_pgsql
There should be at least one potential Apache module that will let you authenticate your users in the way you want. If not, you can always write your own.

**Running Apache in a FreeBSD Jail**

Not everybody is familiar with FreeBSD Jails, but it does offer a very secure alternative for segregating different websites into their own secure locked-down environment.

Within FreeBSD Jails, each users are imprisoned into their own process tree and directory structure. Similar to chroot, but on steroids. If someone were to hack the Apache process or exploit a website vulnerability and gain root access within the jail, they would not be able to modify the main server processes outside of the jail, nor any other jails. Thus, it offers the perfect solution for securing multiple sites controlled by multiple people on the same machine without having the pesky overhead of creating multiple full Virtual Machines for each user and website.

For more detailed information check out my slides from the UtahSAINT 2009 presentation on FreeBSD Jails at:

[http://cbl.us/utahsaint09/freebsd/](http://cbl.us/utahsaint09/freebsd/)

Chroot-ing Apache is another way you can secure your Apache installation, but I **highly favor** the FreeBSD jail method over using chroot. It’s cleaner, easier and allows for more flexibility.

**Additional Readings**

For more in-depth discussions and examples relating to Apache security and lockdown, I suggest you pick up a copy of *Apache Security*, from O'Reilly. It's a great book that goes into details that we can’t cover in this training manual.

**Conclusion**

By now you are probably well aware that there are plenty of knobs and tweaks in Apache that can lead to big security nightmares. Why do we continue to use it? Simply because of these tweaks and knobs as well as the ability to customize everything! You can even write your own Apache modules if you want to authenticate someone out of a MySQL database because the existing mod/mysql modules don’t cut it (I’m facing this exact dilemma right now). Apache takes a bit of getting-used-to in order to customize it fully, but as you apply some of the items mentioned in this section, you’ll be sure and keep it secure and safe from those you want to keep out.
LAB EXERCISES

What are YOU showing the world?
By default, Apache shows a lot about your operating system and installation. What exactly are you showing the world about your machine?

In order to figure these things out you can run a set of commands from a telnet prompt to grab the Apache headers.

1. Find out what IP your Apache is listening on
2. Open up a UNIX Command line
3. Type the following command on your open CLI:
   
   \texttt{telnet <ip\_address or hostname> 80}

4. You should get a response similar to this:
   
   \texttt{telnet <your ip address> 80}
   Trying <your ip address>...
   Connected to <your ip address>.
   Escape character is '^]'.

5. After you get the above prompt type the following into your open telnet session:
   
   \texttt{HEAD / HTTP/1.0} \textit{(then press enter twice)}

6. You should get a response similar to the following:
   
   HTTP/1.1 200 OK
   Date: Tue, 21 Sep 2010 20:35:56 GMT
   Server: Apache/2.2.15 (FreeBSD) mod_ssl/2.2.15 OpenSSL/0.9.8k PHP/5.3.2
   Last-Modified: Thu, 16 Sep 2010 17:19:09 GMT
   ETag: "2e647a-11ac-49063a4ddbd40"
   Accept-Ranges: bytes
   Content-Length: 4524
   Connection: close
   Content-Type: text/html

What exactly does your web server return?

In the example above, you’ll notice Apache, and PHP are a revision behind the most stable released versions. You’ll also notice that FreeBSD is the reported Operating System. If there’s a known vulnerability in Apache/2.2.15 for FreeBSD, you can bet it sure makes life easier for those trying to exploit those known vulnerabilities.
**Setting up User Authentication**

In this Lab Exercise we will lock down a directory by using username and password authentication. We will setup both the Basic and Digest methods in order to gain experience by using both methods. Use the instruction details in the section above for details on how to complete the following requirements.

**Basic Authentication Steps**

1. Open up your Apache configuration file

2. Add a new Directory handler with the following information:
   
   ```
   Directory = /var/www/section3-basic
   Password File = /var/www-passwords/basicauth.passwd
   ```

3. Create a password file by running the htpasswd command.

4. Verify the file has your entry in it from the command line:
   
   ```
   cat /var/www-passwords/basicauth.passwd
   ```

5. Restart Apache from the command line using: apachectl graceful

6. Verify the authentication works by pulling up the IP of your Apache process in your browser along with the /section3-basic/ directory:
   
   ```
   http://AA.BB.CC.DD/section3-basic/
   ```

Try using an incorrect username and password combo first. Once that denies you, try using the correct username and password combination to verify it validates your account correctly.

**Digest Authentication Steps**

1. Open up your Apache configuration file

2. Add a new Directory handler with the following information:
   
   ```
   Directory = /var/www/section3-digest
   Password File = /var/www-passwords/digest.passwd
   ```

3. Create a password file by running the htdigest command

4. Verify the file has your entry in it from the command line:
   
   ```
   cat /var/www-passwords/digest.passwd
   ```

5. Restart Apache from the command line using: apachectl graceful

6. Verify the authentication works by pulling up the IP of your Apache process in your browser along with the /section3-basic/ directory:
   
   ```
   http://AA.BB.CC.DD/section3-digest/
   ```

Try using an incorrect username and password combo first. Once that denies you, try using the correct username and password combination to verify it validates your account correctly.
Restricting Access by IP Address
In this Lab Exercise we will lock down a directory by restricting access to specific IP addresses.

1. Open up your Apache configuration file

2. Add a new Directory handler with the following information:
   
   Directory = /var/www/section3-lockdown

3. Lock down access to this directory to only allow from 127.0.0.1 primarily. Then verify it is not letting you connect to it. Restart Apache after making the change.

4. Once you have verified that you no longer have access to the section3-lockdown directory, add a second rule to permit your IP to connect to it. Restart Apache after making the change and then verify it'll let you access it again.

5. For bonus points, combine this directory with a user authentication method from the previous lab exercise.
Advanced Apache Security

Improving Cipher Suite Security

Most of us already use Secure Certificates with Apache for securely transferring information between our browser and our server. An important note to take reference of is the appropriate settings Apache is configured to use for this SSL transactions.

By default, Apache has many weak ciphers enabled to allow for a lot of the older (less secure) browsers to work right. This, however, is an unfortunate problem when you’re doing PCI scans. The PCI scan will throw an exception if it encounters insecure and weak cipher suites in-use on your SSL enabled host.

In order to make your SSL enabled host more secure you will want to find the “SSLCipherSuite” line in the appropriate Apache configuration file (on FreeBSD it’s in the /usr/local/etc/apache22/extra/httpd-ssl.conf file). The default looks something like this:

```
```

The way to correctly pass a PCI security scan is by commenting out the line above (put a # in front of it) and changing it to the following:

```
SSLCipherSuite HIGH:!aNULL:+SHA1:+MD5:+HIGH
```

This line is a bit over-the-top and extra paranoid. I have been able to pass PCI security scans by using this line instead:

```
SSLCipherSuite ALL:!aNULL:!ADH:!EXPORT56:RC4+RSA:+HIGH:+MEDIUM:!LOW:!EXP
```

You’ll notice the differences in the ciphers that Apache will be allowed to use to transmit the information.

The other line you’ll need to watch out for with PCI scans in this same file is the SSLProtocol line. By default the line will be:

```
SSLProtocol all
```
This, however, is a shortcut for the following:

```
SSLProtocol +SSLv2 +SSLv3 +TLSv1
```

The nitty gritty details with the PCI scan is that they don’t want you to use SSLv2 with your SSL enabled host. In order to pass the scan, you’ll want to disable SSLv2 on this line. This line will enable you to pass your PCI scan:

```
SSLProtocol All -SSLv2
```

The above line will enable SSLv3 and TLSv1 but disable SSLv2, which is exactly what the PCI scans want.

**PCI Security Notice:**
You will need to apply the discussed changes above for any SSL enabled host in order to pass a PCI Security Assessment Scan. SSLProtocol and SSLCipherSuite NEED to be modified in order to pass.

---

**Disable Trace Capability**

Recent PCI security scans are complaining about the TRACE/TRACK option being enabled. See [http://httpd.apache.org/docs/2.2/mod/core.html#traceenable](http://httpd.apache.org/docs/2.2/mod/core.html#traceenable) for more details. Using TraceEnable in your Apache configurations will override the default behavior for both the server and mod_proxy. By default, the TraceEnable is set to on. This disallows any request body to accompany the request. Disabling this feature, causes the core server and mod_proxy to return a 405 error (Method not allowed) to the client.

In order to disable this feature simply put this line in your Apache configuration:

```
TraceEnable off
```

Another way of disabling this feature as well as the Track feature is by using mod_rewrite:

```
RewriteEngine On
RewriteCond %{REQUEST_METHOD} ^(TRACE|TRACK)$ [NC]
RewriteRule ^.*$ - [F]
```

This is what I’ve used on my sites in order to pass the PCI security scan.

**PCI Security Notice:**
You will need to apply the discussed changes above for your Apache configurations. Every PCI scan will complain if Trace is enabled.

---

**Using the Files Option**

By default Apache installs the following lines to prevent users from accessing the contents of .htaccess and .htpasswd files:
<FilesMatch "^\\.ht">
    Order allow,deny
    Deny from all
</FilesMatch>

Obviously, we'll want to restrict access to these files that start with .ht. There are other files, perhaps that we don't want people being able to access. These might include files that end with ".log":

<Files ~ "^\.*\.(Ll)[Oo][Gg])">
    Order allow,deny
    Deny from all
    Satisfy All
</Files>

The above lines will restrict access to files ending with *.log.

You are able, with the Files directive, to insert multiple file types into each rule, simply separate them by a pipe "|".

One more example is for the MacOSX folks that upload files. Sometimes you'll have .DS_Store files in the directories you'll be serving Apache from. These should not be accessible by the public and will likely fail a PCI scan. In order to remove access to them from the web add this:

<Files .DS_Store>
    Order allow,deny
    Deny from all
    Satisfy All
</Files>

The PCI “powers-that-be” have determined that they do not like any hidden files in a public viewable directories. They'll end up viewing it as an attack vector, and as such they'll flag you as non-compliant if they find any. Here's the final example on blocking access from the web to hidden directories or files using Apache:

<FilesMatch "^\.">
    Order allow,deny
    Deny from all
</FilesMatch>

**mod_security**

Mod_Security is an Apache Module just like mod_rewrite that you can compile and load into Apache on runtime. It allows you to basically run a Web Application Firewall, allowing you access to every bit of the HTTP connection. This includes HTTP Headers, Cookies, POST payloads, XML-RPC calls from AJAX, connection information and more! Mod_Security uses Regular Expressions (Regex) and directives in .htaccess or httpd.conf files just like mod_rewrite.

Mod_Security can get very complex very quickly with the amount of rules you decide to add. There are plenty of sites with a bunch of pre-written rules you can copy/paste into your Apache config files that'll give you the extra layer of security that you'll want when running a web server that is viewable from the public.

Let's start with the basics.
Mod_Security Basics

In order to start writing mod_security rules, you’ll first have to make sure and have the module loaded. To do so, if it’s not already loaded add this line to your modules section in the Apache configuration:

```
LoadModule security2_module libexec/apache22/mod_security2.so
```

After adding the line above, you’ll have to restart Apache to get the module loaded. In order to start writing rules for mod_security, there are a few basic lines you’ll want to set in order to determine the defaults you wish for your web server.

I recommend putting all of your mod_security rules within these tags:

```
<IfModule mod_security.c>
    (mod_security rules and statements go inside here)
</IfModule>
```

If you encounter a problem with mod_security, you can easily just remove the LoadModule line from above and restart Apache without having to comment out (or remove) all of your lines you added with the mod_security rules. This way you can diagnose the rules later by adding the LoadModule back in.

Here are a couple of more defaults (with explanations) you’ll want to add in between the above IfModule lines:

```
# Turn the filtering engine On or Off
SecRuleEngine On

# The audit engine works independently and
# can be turned On of Off on the per-server or
# on the per-directory basis. This will log only suspicious requests
SecAuditEngine RelevantOnly

# The name of the audit log file
SecAuditLog /var/log/modsec_audit_log

# Debug log file and debug log level:
SecDebugLog /var/log/modsec_debug_log
SecDebugLogLevel 0

# Configures whether request bodies will be buffered
# and processed by ModSecurity by default
SecRequestBodyAccess On

# Default action is set to deny, it will log suspicious requests
# with a HTTP status of 406
SecDefaultAction "log,deny,status:406,phase:2"
```
Even with the relaxed configuration, mod_security will still provide two benefits. First, it will perform a series of anti-evasive techniques and will normalize the input. This will help later when you start adding custom filtering rules to the configuration. Imagine you want to prevent people from executing a `ps` binary on the server, using a regular expression such as `/bin/ps ax`. This expression would catch simple invocations but perhaps not `/bin//ps ax` or `/bin/ps%20ax` or `/bin/./ps ax`. Here is a list of what mod_security does here:

- Remove multiple forward slashes (`//`).
- Remove self-referenced directories (`./`).
- Treat `\` and `/` equally (on Windows only).
- Perform URL decoding.
- Replace null bytes (`%00`) with spaces.

**The Actions**
Whenever a rule match occurs a series of actions are performed. The default action list (configured through `SecDefaultAction`) is used in most cases. It is also possible to specify per-rule actions by supplying a third parameter to `SecRule`. Supported actions are:

- `deny`, deny the request
- `drop`, drop the transaction without an error
- `allow`, stop rule processing and allow the request
- `status: nnn`, respond with a HTTP status `nnn`
- `redirect: url`, redirect the request to the absolute URL `url`
- `exec: cmd`, execute a script `cmd`
- `log`, log the request to the error log
- `pass`, ignore the current rule match and go to the next rule
- `pause: nnn`, stall the request for `nnn` milliseconds. Be very careful with this action; one Apache instance will be busy stalling the request. You could actually help the attackers in creating a denial of service attack.

Other actions affect the flow of the rules, similarly to how `mod_rewrite` works:

- `chain`, go to evaluate the next rule in the chain. When one rule fails to trigger an alert the remaining rules from the chain will be skipped.
- `skipnext: n`, skip the next `n` rules.

**The Filters**
Using combinations from the Actions in the section above we can create any number of concoctions to filter, deny or allow specific requests to your web server. Most of these filters will use the `SecRule`. Below I'll give a few examples I've picked up in my research and those that might be of interest to you:

```
SecRule REMOTE_ADDR "^AA\BB\CC\DD$" nolog,allow
```
If you want to allow a specific IP address into the server (a trusted IP), without processing any more rules in mod_security you can use the above line. If you want to restrict a specific IP change the nolog, allow in the line above to log, deny.

SecRule REQUEST_COOKIES:sessionid "!^([0-9]{1,9})$"
The line above will prevent SQL injection in a cookie session id. It will restrict the cookie id to a series of 1 to 9 digits. If it contains anything else (such as SQL injection code), the default rule from The Basics section will be applied (in our case log,deny,status:406,phase:2)

SecRule &REQUEST_HEADERS:User-Agent "@eq 0"
SecRule &REQUEST_HEADERS:Host "@eq 0"
The above lines require User-Agent and Host headers in every request. Attackers often investigate using simple tools (even telnet) and don't send all headers as browsers do. Such requests can be rejected, logged, and monitored.

SecRule REQUEST_URI "login_failed\.php" chain
SecRule ARGS:username "^admin\$" log,exec:/home/apache/bin/alertme.pl
The lines above are a pair of rules that will get executed when someone hits the login_failed.php page. In the application if someone fails a login, they’ll pull up the login_failed.php page, and the first rule will match. The second line will check to see if a username argument was used and if its value was admin. If so, it’ll log the request and execute the /home/apache/bin/alertme.pl script which sends me an email letting me know that someone failed a login trying to use the admin username and password. Sure you could do this in the login_failed.php file, but you’d have to code in specific failed values, and what if the login_failed.php is a commercial file that is encoded and you are unable to modify?

SecServerSignature "Microsoft-IIS/5.0"
As we detailed in a previous section, we can use mod_security to change the Server Signature that gets reported in the HTTP headers. In this case, we’re changing it to be IIS/5.0. (Note that you can use the module mod_headers with Apache 2.x to change the server signature as well.) You can’t use the above line within a Directory tag, it has to be in the main Apache configuration.

The Core Rules
There are a plethora of rules that the community maintains to prevent XSS attacks, SQL injection vulnerabilities, comment spam, and more. These rules are found in the mod_security Core Rule Set (CRS) section at: owasp.org with the download at Source Forge:


In order to setup the Core Rule Set on your Apache installation follow the following instructions:

1. The modsecurity_crs_10_config.conf file includes management rules and directives that can control important CRS (Core Rule Set) functions. Pay attention to the SecRuleEngine setting (On by default) and that the SecDefaultAction directive is set to "pass". The 49 inbound blocking and 59 outbound blocking rules files use the "block" action which inherits this setting. The effectively means that you can toggle the SecDefaultAction setting to decide if you would like to deny on an anomaly scoring/correlation match.

Update the PARANOID_MODE variable setting if you want to become more aggressive in your detection. Caution though as this will cause more false positives.
Should also update the appropriate anomaly scoring levels that will be propagated to the inbound/outbound blocking files.

Update the TX policy settings for allowed Request Methods, File Extensions, etc...

2. Add the following line to your httpd.conf (assuming you've placed the rule files into conf/Includes/mod_security2 directory):

```c
<IfModule mod_security.c>
  Include conf/Includes/mod_security2/*.conf
  Include conf/Includes/mod_security2/base_rules/*.conf
</IfModule>
```

3. Restart web server.
4. Make sure your web sites are still running fine.
5. Simulate an attack against the web server. Then check the attack was correctly logged in the Apache error log, ModSecurity debug log (if you enabled it) and ModSecurity audit log (if you enabled it).

We will go over setting up the Core Rule Set in one of the Lab Exercises.

For plenty of more mod_security details, knobs, tweaks and documentation visit the official Mod_Security Documentation page at:

http://www.modsecurity.org/documentation/

**HTTP:BL**

How many of us have used DNSRBL's to fight spammers. My organization pays for the rights to use a couple of more popular RBLs with our spam scanners, in order to help identify which IP addresses are already known spammers. This allows us to assign more points (or outright block) messages from these mail server IP addresses that are known for their abusive behavior. Http:BL is a similar idea, but designed for web traffic instead of mail traffic. The data feed allows you, the website administrator, the ability on how to integrate it with your site.

To use the service, a host simply needs to make a DSN lookup of the visitors IP address. The DNS system will return a value which indicates the status of the visitor. Visitors may be identified as search engines, suspicious, harvesters, comment spammers, or a combination. The response is a simple DNS query which indicates what type of visitor is accessing your page.

You will be required to register to use this service. Once you register you will be provided an Access Key that is passed to the Http:BL DNS servers when you make the DSN lookup. Each user is required to register with Project Honey Pot (www.projecthoneypot.org), after which you can request an access key.

**DNS Query Format**

Queries to the Http:BL DNS servers are performed using the standard DNS syntax. The only trick is, you'll be required to reverse the visitors IP when passing it to the DNS servers for an answer.

The format of the queries must be precisely setup in order for accurate responses. All queries must include the assigned Access Key followed by the IP address you want to know about (in reverse-octet format) followed by the List-Specific Domain you are querying.
For example, if you are querying information on the IP 209.33.192.3, and your access key is abcdesghijkl, the format of your query will be as follows:

```
abcdesghijkl.3.192.33.209.dnsbl.httpbl.org
```

Remember! The IP address you are asking about is the IP of the visitor to your website and it has to be reversed!

**Response Syntax**

The DNS response you receive will provide details about your IP. Sometimes, there will not be any information about the IP you are asking about, other times there will be. The format of the response is specific, just like the query. Each of the octets in the IP response will designate a particular portion of the information that is on-hand for this IP. This allows for greater flexibility rather than “GOOD” or “BAD”.

Example Response: 127.3.5.1

The **first octet** in the response will always be 127. It does not have a specific meaning, other than to let you know that the response was good. If the first octet is something other than 127, then there’s an error condition and your query was probably not formatted correctly.

The **second octet** (3 in the example above) represents the number of days since the last activity. In the example above, it has been 3 days since the last time the queried IP address saw activity on the Project Honey Pot network. This value ranges from 0 days to 255 days. This value is useful in helping you assess how "stale" the information provided by http:BL is and therefore the extent to which you should rely on it.

The **third octet** (5 in the example above) gives you a threat score for this IP. This score is assigned internally by the project based on a number of different factors, including the number of honey pots that the IP has been seen visiting, damage done, etc. The range of the score is from 0 to 255, where 255 is extremely threatening and 0 indicates no threat score has been assigned. From the example above, the threat score would be 5 which is fairly low. Ideally, this threat score will help you in determining a possible threat the visitor IP has against your site.

The **fourth octet** (1 in the example above) represents the type of visitor. Defined types include: "search engine," "suspicious," "harvester," and "comment spammer." Because a visitor may belong to multiple types (e.g., a harvester that is also a comment spammer) this octet is represented as a bitset with an aggregate value from 0 to 255. In the example above, the type is listed as 1, which means the visitor is merely "suspicious." A chart outlining the different types appears below. This value is useful because it allows you to treat different types of robots differently.

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Search Engine</td>
</tr>
<tr>
<td>1</td>
<td>Suspicious</td>
</tr>
<tr>
<td>2</td>
<td>Harvester</td>
</tr>
<tr>
<td>4</td>
<td>Content Spammer</td>
</tr>
<tr>
<td>8</td>
<td>[future use]</td>
</tr>
</tbody>
</table>
Sometimes you will find that some IP addresses are classified in multiple categories. Project Honey Pot accounted for this by making the fourth octet a bitset. This allows you to add a combination of the values that match together to get a combo result. Here is a table of the current possible values:

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Search Engine (0)</td>
</tr>
<tr>
<td>1</td>
<td>Suspicious (1)</td>
</tr>
<tr>
<td>2</td>
<td>Harvester (2)</td>
</tr>
<tr>
<td>3</td>
<td>Suspicious and Harvester (1+2)</td>
</tr>
<tr>
<td>4</td>
<td>Comment Spammer (4)</td>
</tr>
<tr>
<td>5</td>
<td>Suspicious and Comment Spammer (1+4)</td>
</tr>
<tr>
<td>6</td>
<td>Harvester and Comment Spammer (2+4)</td>
</tr>
<tr>
<td>7</td>
<td>Suspicious, Harvester and Comment Spammer (1+2+4)</td>
</tr>
<tr>
<td>&gt;7</td>
<td>[future use]</td>
</tr>
</tbody>
</table>

IP addresses are labelled suspicious if they engage in behavior that is consistent with a malicious robot, but malicious behavior has yet to be observed. For example, on average it takes a harvester nearly a week between when it finds an email address and when it send the first spam message to that address. In the meantime, the as-of-yet-unidentified harvester’s IP address in the meantime is seen hitting a number of honey pots, not obeying rules such as those set forth by robots.txt, and otherwise behaving suspiciously. In this case, the IP may be listed as suspicious.

Here is a list of different responses you might see from the Honey Pot network and a detailed explanation.

<table>
<thead>
<tr>
<th>Response</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>127.3.16.3</td>
<td>This response means that the IP visiting your site is known as a Suspicious and Harvester. The threat score assigned is 16 and it was last seen by the project 3 days ago.</td>
</tr>
<tr>
<td>Response</td>
<td>Meaning</td>
</tr>
<tr>
<td>--------------</td>
<td>-----------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>127.88.36.4</td>
<td>This response means that the IP visiting your site is a known Comment Spammer. The threat score assigned is 36 and it was last seen by the project 88 days ago.</td>
</tr>
<tr>
<td>127.1.12.1</td>
<td>This response means that the IP visiting your site is engaged in Suspicious behavior. It has a threat score of 12 and last seen by the project 1 day ago.</td>
</tr>
</tbody>
</table>

Some of your applications may not need to figure out “last seen days” or “threat scores” you can simply reference the last octet. It all depends on how you want to deal with potential threats against your server and site.

**No Results**
A majority of IP addresses do not appear in http:BL’s records. If a IP you query for does not appear, http:BL will return a non-result (NXDOMAIN).

A non-result does not mean an IP address is certified in any way to be non-malicious. Instead, a non-result simply indicates that no malicious behavior has been observed by the IP in the recent past. You should continue to exercise caution when letting any visitor onto your website, even if the visitor is not listed in http:BL's records.

**Implementation**
There are already many plugins for some of the most-popular CMS systems that work with this application. Check out the Project Honey Pot Implementation page for more details:

http://projecthoneypot.org/httpbl_implementations.php

Some of the implementations include an Apache module, phpBB, Drupal, Joomla, Zope and more!

**More Details**
There are plenty of more details available at the Project Honey Pot page:

http://projecthoneypot.org/httpbl_api.php
LAB EXERCISES

Will you pass a PCI Scan?

Trace and Track Verifying
If you web server is listening on port 80, by far the easiest (and universal) way to determine whether it is vulnerable or not is using telnet. Simply open up your telnet application and connect to your web site/web server over port 80, (telnet <hostname | ip> <port>). If you are using the Microsoft telnet client, be careful because it doesn’t echo back what you were typing in. Once you connect, type the following:

```
TRACE / HTTP/1.0
Host: <hostname_you_are_testing>
TestA: Hello
TestB: World
```

Press enter twice and if trace is enabled, you should see output similar to the following:

```
HTTP/1.1 200 OK
Server: Apache
Date: Tue, 04 Aug 2009 20:17:15 GMT
Content-Type: message/http
Content-Length: 76

TRACE / HTTP/1.0
Host: <hostname_you_are_testing>
TestA: Hello
TestB: World
```

Request and Response over telnet for the HTTP TRACK method is identical, for testing purposes, as it is for TRACE. Simply substitute TRACK for TRACE. If you need to test a host that is listening on SSL port 443 (and does not have an HTTP port exposed), use openssl’s s_client. Simply type "openssl s_client -connect <ipaddress:SSLPORT> " . You will connect and then you can enter the above request the same as you would for telnet.

What are you showing? Is your Apache install responding to the TRACE and TRACK options? If so, try disabling it following the details in this Section, restart Apache (apachectl reload), and try the test again.

Does your SSL Enabled Host Support SSLv2?
You will need to have OpenSSL installed on the system that you will perform the tests from. Once installed, use the following command to test your web server, assuming port 443 is where you’re providing https connections:

```
openssl s_client -ssl2 -connect IPADDRESS:443
```

If the server does not support SSLv2 you should receive an error similar to the following:
What are you showing? Is your test throwing an error when using SSLv2? If so, try disabling it following the details in this Section, restart Apache (`apachectl reload`), and try the test again.

**Does your SSL Enabled Host Support the Weak Ciphers?**
You will need to have OpenSSL installed on the system that you will perform the tests from. Once installed, use the following command to test your web server, assuming port 443 is where you’re providing https connections:

```
openssl s_client -connect IPADDRESS:443 -cipher LOW:EXP
```

If the server does not support weak ciphers you should receive an error similar to the following:

```
CONNECTED(00000003)
```

What are you showing? Is your test throwing an error when using the weak ciphers in openssl? If so, try disabling it following the details in this Section, restart Apache (`apachectl reload`), and try the test again.

---

**Mod_Security Exercises**

**Basic Mod_Security Rules**
In this lab section we’ll setup our basic mod_security rules on our own in order to block some basic attacks against our web site. This will include enabling mod_security setting up a rule or two and testing its effectiveness on our own.

Using a Directory rule for `/var/www/section4-modsec` setup rules to block access for the following and verify they work:

- If the User-Agent is Mozilla
- If the Operating System in the User-Agent is Windows
- If the Requested URL includes `c99.php`
- Allow all access from your IP Address
- Run the `/tmp/section4-runme.sh` script whenever you pull up the `/section4-modsec/login_failed.php` URL.

  - Check that the output of `/tmp/section4-modsecoutput.txt` changes each time you pull it up.

**Mod_Security Core Rule Set**
In this lab section we’ll include the Core-Rule Set in our Apache config and see if we can break something.
The CRS are already installed, we just need to activate the configuration file to use them by doing this:

```
cd /usr/local/etc/apache22/Includes
mv mod_security2.conf.disabled mod_security2.conf
```

After you’ve run the above lines, you’ll want to restart Apache using the `apachectl restart` command.

We’ll be testing out some basic gotchas to make sure that our Core Rule Set is working. For this example we’ll be using the files in the directory: `/var/www/section4-crs/

Pull up the following in your browser to make sure it gets blocked after activating the mod_security2 CRS:

```
/section4-crs/get.php?action=&type=view&s=&id=-1' union select 0,concat(char(85),char(115),char(101),char(114),char(110),char(97),char(109),char(101),char(58),name,char(32),char(124),char(124),char(32),char(80),char(97),char(115),char(115),char(119),char(111),char(114),char(100),char(58),pass),0,0,0,0 from phpdesk_admin/*
```

In order to really validate that each rule is working as expected, you’ll have to look at the RegEx and validate them one by one. We may pick a few to do by hand as examples during the presentation of this subject.
Securing PHP

Update to PHP 5.3.x Now!

PHP 5.3 has a bunch of big changes that it brought into the mainstream code base. It also removed a lot of the old php.ini insecurities. The following is a list of a few of the old php.ini values that are now deprecated and will throw an error if they’re found in the php.ini file:

```
register_globals safe_mode magic_quotes_gpc register_long_arrays
```

A couple of functions have also been deprecated in 5.3.x for favor of their preg* counterparts. These include: `ereg()`, `ereg_replace`, `eregi()`, `eregi_replace()`, `split()` and `spliti()`.

Yes, the security in PHP5 just got a whole lot better since the release of 5.3.x. It may require you to rewrite some of your insecure code to avoid the above php.ini deprecated settings, or avoid the deprecated functions, but you’ll be that much more secure because of it.

Check out the migration information for going from PHP5.2.x to 5.3.x here:

```
```

MailHeader Patch

I’m a fan of knowing when someone has used PHP code to send out spam. The trick is, when you have a busy site and many many PHP scripts (and users modifying them) to identify exactly what script is the culprit.

Typically, when you start getting spam alerts those pesky spambcops will send your email administrator the full email headers so it’ll contain all the information you should need in order to stop the spamming. Unfortunately, the default PHP `mail()` function does not add any headers to the message to identify the script ran to generate the `mail()` command nor who ran it. This is where the PHP Mail Header patch comes in.
The PHP Mail Header patch is available at: http://choon.net/php-mail-header.php. It has been regularly updated for the latest PHP 5.3.x sub-releases (as well as previous releases) so it’ll patch cleanly to whatever version you’re wanting to run. What it does is it adds a nice little line to the message headers of all messages that are sent using the PHP built in `mail()` function. This line provides details of what script was ran to generate the email as well as who ran it. The header will look like this:

```
X-PHP-Script: <servername><php-self> for <remote-addr>
```

Now I’m not recommending that everybody go and patch their PHP installations to include this, but you should be aware of its existence and need.

I have had a few cases where some of my customers code was exploited to start to spam out thousands of emails. Because I run a very active group of hosting servers, it would be a big pain to sort through logs and identify who exactly was running what at the time of the message origination. With the PHP Mail Header patch, I just had to find out which script was ran on which site and then I could “chmod 000” it until I had a chance to let the customer know and to possibly fix their vulnerability.

On the flip side, this patch has also caused me grief one time. Another customer of mine has a cron job that pulls up a script via CURL that generates emails. One of the recipients of that email looked at the header and saw the URL from which that nightly email gets originated and clicked on it. It proceeded to start to send out the nightly email at 3pm to the same users from 3am the night before. My customer didn’t protect who could access that script, thinking that it was hidden because nobody knew about it and didn’t put steps to prevent who could access it (and from where). Once they protected the script from casual access, all was better.

I recommend you give it a go on your multi-user environment. If you don’t care if a header has what script generates an email, it could get you to the root of an abuse issue more quickly than not.

Random Number Generation Gotchas

If you’re relying on PHP to generate random numbers and you’re using a PHP version before 5.2.6 you should be aware that PHP has some random number generation issues. PHP has had some issues where the random numbers generated could be easily guessable based on the seed values. This is why we should prefer to use `/dev/urandom` (or `/dev/random`) for helping with our random number generation in PHP.

Note that there is another advisory in the way that PHP uses random numbers to generate session IDs. The article is available at: http://seclists.org/fulldisclosure/2010/Mar/519

To avoid the above issue, upgrade your PHP installation to version 5.3.3 or later or use the Suhosin patch (0.9.31 or later).

Salting your Passwords

Protecting your passwords is one of the first things good web site administrators do. Do you store your usernames and passwords for your web sites in the database in clear text? Do you encrypt the passwords?
Did you know that running the PHP `md5()` function on the same string will always produce the same results? This may lead to some big-time security vulnerabilities when dealing with rainbow tables and the like (see Troy Jessup’s past presentations at UtahSAINT on rainbow tables). It basically allows anybody with a big enough database of MD5 hashes to decrypt your password based on the MD5 result. This is where password salting comes into play.

Using a unique random salt each time you encrypt a password will make it practically impossible to determine every possible password hash for every possible password. This adds another layer of complexity to your passwords and allows you the assuredness that your passwords are extra “secure”.

Here are some example functions that we use at my place of employment for generating random salt and using that salt to encrypt a password with:

```php
# Read 1-65536 (inclusive) bytes from /dev/urandom and return it as a raw binary string:
function randbytes($bytes) {
    $rand = '';
    if ($bytes < 0 || $bytes > 65535) {
        return $rand;
    }
    $rfp = fopen("/dev/urandom", "r");
    $len = $bytes;
    do {
        if ($rfp > 0) {
            $rand .= fread($rfp, $len);
        } else {
            $rand .= substr(md5(md5(mt_rand(), 1) . md5(microtime(), 1), 1), 0, $len);
        }
        $len = $bytes - strlen($rand);
    } while ($len > 0);
    if ($rfp > 0) {
        fclose($rfp);
    }
    return substr($rand, 0, $bytes);
}

# Do one-way password encryption using FreeBSD's crypt() function and using the MD5 password encryption format:
function encrypt_pass($pass) {
    # MD5-encrypted passwords always begin with '$1$' followed by a 48-bit base-64-encoded salt (encoded with chars './0-9A-Za-z')
    $salt = '$1$';
    $rand = preg_split('/\//', randbytes(8));
    for ($i = 0; $i < 8; $i++) {
        # Discard the 2 high bits of each byte - keeps it simple and fast:
```
By calling the `encrypt_pass()` function above, you'll get a random generated salted password which you can store in the database. To validate that the password is correct on the next attempt just use this pre-configured `check_pass()` script:

```php
# Compare a plain text password to an encrypted password to see if they match.
# Handles all crypt() supported password formats.
function check_pass($plainpass, $cryptpass) {
    if (preg_match('/^[a-zA-Z0-9\./]{13}$/', $cryptpass)) {
        # This is a simple DES encrypted password:
        $salt = substr($cryptpass, 0, 2);
    } else if (preg_match('/^[a-zA-Z0-9\./]{19}$/', $cryptpass)) {
        # Extended DES encrypted password:
        $salt = substr($cryptpass, 0, 9);
    } else if (preg_match('/^[a-zA-Z0-9\./]{8}$/iU', $cryptpass)) {
        # MD5 modular encrypted password:
        $salt = substr($cryptpass, 0, 12);
    } else if (preg_match('/^[a-zA-Z0-9\./]{22}$/iU', $cryptpass)) {
        # Blowfish modular encrypted password:
        $salt = substr($cryptpass, 0, 12);
    } else {
        # Unsupported encrypted password format!
        return false;
    }
    return (strcmp($cryptpass, crypt($plainpass, $salt)) == 0) ? true : false;
}
```

If the function returns “true” then you have a valid password. If the function returns “false” then the encrypted password does not match the plain-text version.

* The above functions were written by Aaron D. Gifford
Validating E-Mail Addresses

Honestly, there are a lot of PHP email address validators out there. Plenty of RegEx to get your head around that use. Some will work, some will probably not work fully against the RFC2822 which defines which characters can make up an email address.

In doing research for this subject I stumbled upon an article by the Linux Journal entitled “Validate an E-Mail Address with PHP, the Right Way”. Check it out for a lot of the nitty gritty of what works, and what doesn’t work.

Some functions wouldn’t accept valid characters from the RFC, some wouldn’t validate the domain for existence. In the end, they came up with this function that should validate an email address and make sure the domain exists via DNS and if so, it’ll return True.

```php
/** Validate an email address. Provide email address (raw input) 
Returns true if the email address has the email address format and the domain exists. */
function validEmail($email){
    $isValid = true;
    $atIndex = strrpos($email, "@");
    if (is_bool($atIndex) &amp; !$atIndex) {
        $isValid = false;
    } else {
        $domain = substr($email, $atIndex+1);
        $local = substr($email, 0, $atIndex);
        $domainLen = strlen($domain);
        $localLen = strlen($local);
        if ($localLen < 1 || $localLen > 64) {
            // local part length exceeded
            $isValid = false;
        } else if ($domainLen < 1 || $domainLen > 255) {
            // domain part length exceeded
            $isValid = false;
        } else if ($local[0] == '.' || $local[$localLen-1] == '.') {
            // local part starts or ends with '.'
            $isValid = false;
        } else if (preg_match('/\.\./', $local)) {
            // local part has two consecutive dots
            $isValid = false;
        } else if (!preg_match('/^[A-Za-z0-9-\-\.]+$/', $domain)) {
            // character not valid in domain part
            $isValid = false;
        } else if (preg_match('/\.[.]\.', $domain)) {
            // domain part has two consecutive dots
            $isValid = false;
        } else if (!preg_match('/^[A-Za-z0-9-\-\!\#\%\&\_\=\$\^\*\(\)\+\-,\;\.\[\]\{\}\|\~\-\s]+$/', str_replace("\", "\", $local))) {
            // character not valid in local part unless
            // local part is quoted
            if (!preg_match('/'^([^\^])+$/', str_replace("\", ",", $local))) {
                $isValid = false;
            }
        }
        if ($isValid &amp; !(checkdnsrr($domain,"MX") || checkdnsrr($domain,"A"))) {
            // domain not found in DNS
            $isValid = false;
        }
    }
    return $isValid;
}
```

The article is available here:  [http://www.linuxjournal.com/article/9585](http://www.linuxjournal.com/article/9585)
Simply paste the function above into your PHP application and call it like so:

```php
validEmail("is.tHls@Valid@no-where.boo");
```

## Using Suhosin

Back in 2004 a couple of German guys got together and formed the Hardened PHP Project. Their initial goal was helping us secure our apps and web pages. They manage to release the Hardening-Patch initially and most recently, the Suhosin project.

Suhosin comes in two parts. The first being a small patch against the PHP core while implementing a few low-level protections against bufferoverflows or format string vulnerabilities. The second part is a PHP extension that implements all the other protections.

There are numerous features that Suhosin provides, too many to document here. These features are listed at: [http://www.hardened-php.net/suhosin/a_feature_list.html](http://www.hardened-php.net/suhosin/a_feature_list.html). A couple of the ones worth mentioning include:

- Adds protection against newline attacks to mail()  
- Transparent Cookie Encryption  
- Protects the PHP core and extensions against format string vulnerabilities  
- Transparent session hijacking protection

Some of your applications may have an issue when enabling Suhosin. I recommend if you start using it, you do so on a test installation first to see what issues arise (I've seen issues with vbulletin and Suhosin not playing nice together).

Either way, it'll warrant a look for anybody that's serious about security in PHP.

## XSRF Preventions

More than likely, you've already heard about Cross Site Request Forgeries. It seems now, more than ever, the number of patches for software are relating to fixing potential CSS and XSRF vulnerabilities.

One of the most popular methods now to facilitate a workable solution is by using a unique token based approach. The basics of this are as follows: “The synchronizer token pattern requires the generating of random "challenge" tokens that are associated with the user’s current session. These challenge tokens are the inserted within the HTML forms and links associated with sensitive server-side operations. When the user wishes to invoke these sensitive operations, the HTTP request should include this challenge token. It is then the responsibility of the server application to verify the existence and correctness of this token. By including a challenge token with each request, the developer has a strong control to verify that the user actually intended to submit the desired requests. Inclusion of a required security token in HTTP requests associated with sensitive business functions helps mitigate CSRF attacks as successful exploitation assumes the attacker knows the randomly generated token for the target victim's session. This is analogous to the attacker being able to guess the target victim's session identifier.” Check out [http://www.owasp.org/index.php/Cross-Site_Request_Forgery_%28CSRF%29_Prevention_Cheat_Sheet](http://www.owasp.org/index.php/Cross-Site_Request_Forgery_%28CSRF%29_Prevention_Cheat_Sheet) for more details on this topic.
Implementing the token based challenge is fairly easy.

First generate a token in your script with:

```php
$token = md5(uniqid(rand(), true));
```

Next, save the token in the users SESSION:

```php
$_SESSION['token'] = $token;
```

Inject the token now into the form as a hidden field:

```html
<input type="hidden" name="token" value="<?php echo $token; ?>">
```

Valdiating the token is as easy as running this:

```php
if (!empty($_SESSION['token']) && !empty($_POST['token'])) {
    if ($_SESSION['token'] == $_POST['token']) {
        //process the form
    }
}
```

Using PHP’s default $_SESSION is not very secure. However, if you decide to store your session data in a database, it’ll prevent attackers from gaining access to your unique tokens as they could easily with data stored in the browsers cookies.

There is plenty of reading on this topic on the Internet. Be extra vigilant when you’re coding your application and forms to prevent XSS and XSRF attacks against your application and your users.

### Using Database Abstraction Layers

If you’re the kind of coder that doesn’t trust your database calls will be SQL injection safe, try using a PHP database abstraction layer and following its methods. This will prevent you from talking to MySQL directly from `mysql_query()` or `mysql_result()` style functions. It might simplify things by using someone elses tried methods rather than making your own.

Coders are pretty religious when it comes to using a Database Abstraction Layer or not. It will never be as easy as “just change the type from MySQL to Postgres and you’re done”, but it should help secure things up to avoid shooting yourself in the foot.

For suggestions of layers to try check out either [ADOdb](http://www.adodb.org/) or the [PHP PEAR](https://pear.php.net) classes.

### Using Captchas

Everybody should know now about Captchas. If you haven’t used a captcha then you’ve been hiding under a rock for years now. Implementing captchas (or some type of real-user validation) is something most people need to worry about when allowing for user-generated information. This helps prevents comment spam, bots abusing your code and the likes.

Google makes available some nice easy libraries for integrating captchas into your site. Check it out at: [http://code.google.com/apis/recaptcha/docs/php.html](http://code.google.com/apis/recaptcha/docs/php.html)

All it takes it getting an API key and putting this somewhere in your form code:
require_once('recaptchalib.php');
$publickey = "your_public_key"; // you got this from the signup page
echo recaptcha_get_html($publickey);

After that you'll need to add some code to the page that you submit your form to:

require_once('recaptchalib.php');
$privatekey = "your_private_key";
$resp = recaptcha_check_answer ($privatekey,
$_SERVER["REMOTE_ADDR"],
$_POST["recaptcha_challenge_field"],
$_POST["recaptcha_response_field"]);

if (!$resp->is_valid) {
    // What happens when the CAPTCHA was entered incorrectly
    die ("The reCAPTCHA wasn't entered correctly. Go back and try it again." .
    "(reCAPTCHA said: " . $resp->error . ")");
} else {
    // Your code here to handle a successful verification
}

There are plenty of other Captchas on the market. This one seemed to me to be one of the easiest
to implement. Plus..Google makes it, so it can't be that bad, can it?

phpMyAdmin Gotchas

phpMyAdmin is one of the most popular web applications used on the planet. As such, it's one of
the most sought-after applications to exploit by hackers to gain access to your servers and
database installations.

Here's a brief snapshot from one of my Apache access_logs that shows the lengths that people
are using to try and exploit old versions of phpMyAdmin:

"GET //phpmyadmin/config/config.inc.php?p=phpinfo(); HTTP/1.1" 404 230
"GET //pma/config/config.inc.php?p=phpinfo(); HTTP/1.1" 404 223
"GET //admin/config/config.inc.php?p=phpinfo(); HTTP/1.1" 404 225
"GET //phpmyadmin/config/config.inc.php?p=phpinfo(); HTTP/1.1" 403 234
"GET //dbadmin/config/config.inc.php?p=phpinfo(); HTTP/1.1" 404 227
"GET //pma/config/config.inc.php?p=phpinfo(); HTTP/1.1" 403 227
"GET //mysql/config/config.inc.php?p=phpinfo(); HTTP/1.1" 404 225
"GET //admin/config/config.inc.php?p=phpinfo(); HTTP/1.1" 403 229
"GET //php-my-admin/config/config.inc.php?p=phpinfo(); HTTP/1.1" 404 232
"GET //dbadmin/config/config.inc.php?p=phpinfo(); HTTP/1.1" 403 231
"GET //myadmin/config/config.inc.php?p=phpinfo(); HTTP/1.1" 404 227
"GET //mysql/config/config.inc.php?p=phpinfo(); HTTP/1.1" 403 229
"GET //PHPMYADMIN/config/config.inc.php?p=phpinfo(); HTTP/1.1" 404 230
"GET //php-my-admin/config/config.inc.php?p=phpinfo(); HTTP/1.1" 403 236

All of this is only by one user in just a matter of seconds. The bot was looking for an exploitable
version of the "config.inc.php" file that was able to run commands by passing them in the
URL with the “p” variable. If it got the phpinfo() in return, it’d know that it found a vulnerable
installation and could start executing commands as it saw fit. When this file was removed from the
phpMyAdmin installations, some package maintainers did not remove it from old installations on upgrade causing some newer phpMyAdmin installs to be compromised because of an out-of-date file. Because of this, I typically extract phpMyAdmin by hand into a new directory and use a symlink to point to the new source and delete the old after I verify the new is working as expected.

The number one thing you need to worry about with phpMyAdmin is keeping it up-to-date. It seems like there is always at least one security vulnerability with each sub-release.

**Locking phpMyAdmin Down**

The first thing you should do when securing phpMyAdmin is to install it into a unique directory. All of the bots and hackers will look for it in /phpMyAdmin. Move it to something unique that only you’ll know.

The second thing you should do is change the authentication type in the phpMyAdmin configuration file to 'cookie' auth. This way you will be required to login at the index.php page.

```
$c['Servers'][$i]['auth_type'] = 'cookie';
```

Third. Use the lessons from the Securing Apache section of this manual to lock down the directory to only the IP addresses you wish to access it from. This will prevent others from being able to access the phpMyAdmin installation and keeping it extra protected from being brute-forced password guessed.

Fourth. Set the blowfish_secret in the phpMyAdmin config.inc.php file so that it'll encrypt the password.

```
$c['blowfish_secret'] = 'someRandomBlowfishSecret';
```

Fifth. If you are allowing multiple users into your database via phpMyAdmin, try using the AllowDeny rules to restrict access from what IPs each user is allowed to access phpMyAdmin from.

```
$c['Servers'][$i]['AllowDeny']['order'] = 'allow,deny';
$c['Servers'][$i]['AllowDeny']['rules'] = array(
    'allow user1 from 134.133.1.2',
    'allow user2 from all',
    'allow user3 from 144.38.0.0/16');
```

Finally, make sure to not allow root login via phpMyAdmin:

```
$c['Servers'][$i]['AllowRoot'] = FALSE;
```

There are plenty of things to watch out for when dealing with phpMyAdmin. As phpMyAdmin provides access to your database server(s) directly from the web, it makes it an easy point for exploiting your “accessible only from private IP addresses” database servers. Be vigilant and watch the log files for potential hackers. Keep it secure and you benefit from a great PHP project for manipulating your MySQL databases.
LAB EXERCISES

Put Some Salt In Your Passwords
Implement the functions from the above section in order to create a random salt for a password encryption and validate that the password was encrypted correctly when comparing it against a plain-text version. The files you’ll want to use are here:

/var/www/section5-salty/adduser.php - Adds a username and password to the database

/var/www/section5-salty/checkpass.php - Checks a username and password in the database

Add the appropriate code to the adduser.php file to salt and encrypt the password before storing it in the database. Then make sure that the same username and password you enter match on the checkpass.php file by using the check_pass() function.

Validate User-Submitted E-Mail Addresses
For this lab exercise we’ll want to hook up the e-mail address validation function in order to see what addresses will validate and which will not. This way we can make sure the implementation is correct and behaves like it should before we go home and change all of our coding to use it. The files you’ll want to use are here:

/var/www/section5-emails/validate.php - Allows you to submit a single email for validation

/var/www/section5-emails/many.php - Allows you to see a long list of emails in the database and which emails are valid and which are not.

You’ll want to hook up the email validation function from the section above in each of the files so we can verify if an email submitted is valid or not. Test it by entering valid and non-valid email syntax (and domains) in the validate.php script. Check a list of valid and invalid emails on the many.php page.

Catchya..Captcha
Using the example in the section above, implement a Captcha using the Google reCAPTCHA libraries. The files you’ll want to modify are in the following places:

/var/www/section5-captcha/form.php (Add the Captcha code here)
/var/www/section5-captcha/results.php (Add code to validate the Captcha here)

Google API Private Key: 6LfRm70SAAAAAP1T07aMK_Y4he4Vwhtt69Ra9Lz-
Google API Public Key: 6LfRm70SAAAAABwCPHjB6RxpufKRN-vbP-vX8tU1

The recaptchalib.php is in the /var/www/section5-captcha folder.
Securing MySQL

Adjusting The Bind Address
MySQL, by default, will listen on all interfaces for incoming connections. This may be a security risk depending on your organization and network setup. I typically setup MySQL to listen on a private IP network that is only used for MySQL access. This prevents public access from the Internet to access my databases and provides me an extra level of security in my database environment.

To adjust the IP address that MySQL listens to, pop open the my.cnf file on your server (typically found in /etc/my.cnf, /etc/mysql/my.cnf, or /var/db/mysql/my.cnf depending on your package install) and find the section that starts with [mysqld]. You'll want to add the following line that appropriately matches the IP network you want MySQL to listen on:

```plaintext
bind-address     = AA.BB.CC.DD
```

If, perhaps, you do not need MySQL to be accessible from any other machine than the local applications, you can use the `skip-networking` option to disable MySQL from listening on any TCP port. This would require access via a Unix socket or named pipe. It all depends on your installation and application needs.

Note that keeping a public accessible MySQL install will attract password guessers. If MySQL absolutely needs to be accessible from the public facing interface do yourself a favor and lock it down with the firewall.

Who’s Running MySQL?
Normally, MySQL will be run as a separate user. Typically, your package or port will auto-create a `mysql` user account for the use of running the MySQL process as. This way if there’s a vulnerability with MySQL, the potential damage is limited to what the user running the server process has access to. If MySQL is running as the `root` user, database users with the `FILE` privilege may be using files with root privileges. This could potentially allow them access to files that are usually only allowed by the `root` user.
Check your MySQL data directory. Who owns it? I recommend keeping your data directory owned by the MySQL user and group with the directory permissions set to 750. This will keep out pesky abusers from being able to figure out what database names you might have.

**Protecting the Socket File**

Make sure your `mysql.sock` file, typically stored in the `/tmp` directory (`/tmp/mysql.sock`) and used for database interactions on the local machine, is protected. On some operating systems, any user can remove files from the `/tmp` directory. Removing `mysql.sock` will prevent any interaction with the database.

To protect against this, you can either set permissions on /tmp files or move `mysql.sock` to another location using the `socket=/path/to/socket` option in your `my.cnf` and client configuration files. Refer to [http://dev.mysql.com/doc/mysql/en/problems-with-mysql-sock.html](http://dev.mysql.com/doc/mysql/en/problems-with-mysql-sock.html) for more information.

**Setting the Root Password**

By default, the MySQL root account password may be left blank during install. You must add a password for the root (administrator) user immediately.

Don’t use the root account to manage your databases. Set up other admin accounts for that purpose.

In order to set the MySQL root password you’ll need to use the following example from a command line:

```
shell> mysql -u root
mysql> SET PASSWORD FOR 'root'@'localhost' = PASSWORD('newpwd');
mysql> SET PASSWORD FOR 'root'@'127.0.0.1' = PASSWORD('newpwd');
mysql> SET PASSWORD FOR 'root'@'host_name' = PASSWORD('newpwd');
```

The above lines will set the root passwords accordingly.

**Cleaning out Anonymous Users and Test Database**

MySQL, on most installations, will create a test database as well as some anonymous users. These can lead to some security risks and should be cleaned up immediately after installation.

In order to remove the Anonymous users follow these instructions:

```
shell> mysql -u root -p
Enter password: (enter root password here)
mysql> DROP USER ''@'localhost';
mysql> DROP USER ''@'127.0.0.1';
mysql> DROP USER ''@'host_name';
```

You will also want to remove any access that was pre-configured for the test database. To do so run the following commands:
Finally, you’ll want to remove the test database from the server itself. Do so by running the following while still in the MySQL CLI:

```sql
mysql> DROP DATABASE test;
```

## Using MySQL Prepared Statements

Server-side prepared statements have been a feature since the MySQL 4.1 days. Since then, they have become common place in most web applications seeking to implement greater security.

Server-side prepared statements allow you to setup the statement once, and then execute it multiple times with different parameters. They allow you to replace the casual QUERY string and do so more securely and efficiently. A typical prepared statement will look like:

```sql
SELECT * FROM `County` WHERE code = ?
```

The `?` in the above statement is a placeholder. It allows you to supply a value for it when you execute the prepared-statement.

Using prepared statements allows you to separate the SQL logic from the data being supplied. This allows you to overcome the popular SQL injection attacks that plague custom websites. Typically, when you deal with a QUERY string it may look something like this:

```sql
$QUERY = "SELECT * FROM `Country` WHERE code = "$code";"
```

The above query string will be vulnerable to SQL injections unless you escape single and double quotes and backslashes. Otherwise, someone can insert their own code to DROP a table, or SELECT or INSERT more rows than you wanted.

Using prepared statements avoids the whole escaping of characters. The separation of data from the actual MySQL query automatically takes into account the escaping of characters and they do not need to be escaped using any special function.

Increase in performance in prepared statements is another useful feature. There is only a need to process the query a single time, and ever time after that there is less overhead involved. Pre-parsing of the query can lead to speed increase if you re-run it multiple times, such as doing many INSERT statements.

If you code your own web applications, I highly suggest you start using MySQL prepared statements (if you have not already done so).
Securing MySQL

1. Set a password for the root MySQL user using the instructions above.

2. Verify that you have to use a root password now to login to MySQL.

3. Follow the instruction above to remove the Anonymous users and test databases.

4. Adjust the bind-address to only listen on localhost (127.0.0.1) and restart MySQL.

5. Verify that you can not access MySQL from outside of the VM by telnetting to the VM IP on port 3306.