the CENTER for INTERNET SECURITY

Security Configuration Benchmark For

Apache Web Server 2.2.0

Version 2.2.0
November 2008

Leader:
Ryan Barnett

http://cisecurity.org
cis-feedback@cisecurity.org

Terms of Use Agreement

Background.

CIS provides benchmarks, scoring tools, software, data, information, suggestions, ideas, and other services and materials from the CIS website or elsewhere ("**Products**") as a public service to Internet users worldwide. Recommendations contained in the Products ("**Recommendations**") result from a consensus-building process that involves many security experts and are generally generic in nature. The Recommendations are intended to provide helpful information to organizations attempting to evaluate or improve the security of their networks, systems and devices. Proper use of the Recommendations requires careful analysis and adaptation to specific user requirements. The Recommendations are not in any way intended to be a "quick fix" for anyone's information security needs.

No representations, warranties and covenants.

CIS makes no representations, warranties or covenants whatsoever as to (i) the positive or negative effect of the Products or the Recommendations on the operation or the security of any particular network, computer system, network device, software, hardware, or any component of any of the foregoing or (ii) the accuracy, reliability, timeliness or completeness of any Product or Recommendation. CIS is providing the Products and the Recommendations "as is" and "as available" without representations, warranties or covenants of any kind.

User agreements.

By using the Products and/or the Recommendations, I and/or my organization ("**we**") agree and acknowledge that:

No network, system, device, hardware, software or component can be made fully secure;
We are using the Products and the Recommendations solely at our own risk;

We are not compensating CIS to assume any liabilities associated with our use of the Products or the Recommendations, even risks that result from CIS's negligence or failure to perform;

We have the sole responsibility to evaluate the risks and benefits of the Products and Recommendations to us and to adapt the Products and the Recommendations to our particular circumstances and requirements;

Neither CIS, nor any CIS Party (defined below) has any responsibility to make any corrections, updates, upgrades or bug fixes or to notify us if it chooses at it sole option to do so; and

Neither CIS nor any CIS Party has or will have any liability to us whatsoever (whether based in contract, tort, strict liability or otherwise) for any direct, indirect, incidental, consequential, or special damages (including without limitation loss of profits, loss of sales, loss of or damage to reputation, loss of customers, loss of software, data, information or

emails, loss of privacy, loss of use of any computer or other equipment, business interruption, wasted management or other staff resources or claims of any kind against us from third parties) arising out of or in any way connected with our use of or our inability to use any of the Products or Recommendations (even if CIS has been advised of the possibility of such damages), including without limitation any liability associated with infringement of intellectual property, defects, bugs, errors, omissions, viruses, worms, backdoors, Trojan horses or other harmful items.

Grant of limited rights.

CIS hereby grants each user the following rights, but only so long as the user complies with all of the terms of these Agreed Terms of Use:

Except to the extent that we may have received additional authorization pursuant to a written agreement with CIS, each user may download, install and use each of the Products on a single computer;

Each user may print one or more copies of any Product or any component of a Product that is in a .txt, .pdf, .doc, .mcw, or .rtf format, provided that all such copies are printed in full and are kept intact, including without limitation the text of this Agreed Terms of Use in its entirety.

Retention of intellectual property rights; limitations on distribution.

The Products are protected by copyright and other intellectual property laws and by international treaties. We acknowledge and agree that we are not acquiring title to any intellectual property rights in the Products and that full title and all ownership rights to the Products will remain the exclusive property of CIS or CIS Parties. CIS reserves all rights not expressly granted to users in the preceding section entitled "Grant of limited rights." Subject to the paragraph entitled "Special Rules" (which includes a waiver, granted to some classes of CIS Members, of certain limitations in this paragraph), and except as we may have otherwise agreed in a written agreement with CIS, we agree that we will not (i) decompile, disassemble, reverse engineer, or otherwise attempt to derive the source code for any software Product that is not already in the form of source code; (ii) distribute, redistribute, encumber, sell, rent, lease, lend, sublicense, or otherwise transfer or exploit rights to any Product or any component of a Product; (iii) post any Product or any component of a Product on any website, bulletin board, ftp server, newsgroup, or other similar mechanism or device, without regard to whether such mechanism or device is internal or external, (iv) remove or alter trademark, logo, copyright or other proprietary notices, legends, symbols or labels in any Product or any component of a Product; (v) remove these Agreed Terms of Use from, or alter these Agreed Terms of Use as they appear in, any Product or any component of a Product; (vi) use any Product or any component of a Product with any derivative works based directly on a Product or any component of a Product; (vii) use any Product or any component of a Product with other products or applications that are directly and specifically dependent on such Product or any component for any part of their functionality, or (viii) represent or claim a particular level of compliance with a CIS Benchmark, scoring tool or other Product. We will not facilitate or otherwise aid other individuals or entities in any of the activities listed in this paragraph.

We hereby agree to indemnify, defend and hold CIS and all of its officers, directors,

members, contributors, employees, authors, developers, agents, affiliates, licensors, information and service providers, software suppliers, hardware suppliers, and all other persons who aided CIS in the creation, development or maintenance of the Products or Recommendations ("**CIS Parties**") harmless from and against any and all liability, losses, costs and expenses (including attorneys' fees and court costs) incurred by CIS or any CIS Party in connection with any claim arising out of any violation by us of the preceding paragraph, including without limitation CIS's right, at our expense, to assume the exclusive defense and control of any matter subject to this indemnification, and in such case, we agree to cooperate with CIS in its defense of such claim. We further agree that all CIS Parties are third-party beneficiaries of our undertakings in these Agreed Terms of Use.

Special rules.

CIS has created and will from time to time create special rules for its members and for other persons and organizations with which CIS has a written contractual relationship. Those special rules will override and supersede these Agreed Terms of Use with respect to the users who are covered by the special rules. CIS hereby grants each CIS Security Consulting or Software Vendor Member and each CIS Organizational User Member, but only so long as such Member remains in good standing with CIS and complies with all of the terms of these Agreed Terms of Use, the right to distribute the Products and Recommendations within such Member's own organization, whether by manual or electronic means. Each such Member acknowledges and agrees that the foregoing grant is subject to the terms of such Member's membership arrangement with CIS and may, therefore, be modified or terminated by CIS at any time.

Choice of law; jurisdiction; venue.

We acknowledge and agree that these Agreed Terms of Use will be governed by and construed in accordance with the laws of the State of Maryland, that any action at law or in equity arising out of or relating to these Agreed Terms of Use shall be filed only in the courts located in the State of Maryland, that we hereby consent and submit to the personal jurisdiction of such courts for the purposes of litigating any such action. If any of these Agreed Terms of Use shall be determined to be unlawful, void, or for any reason unenforceable, then such terms shall be deemed severable and shall not affect the validity and enforceability of any remaining provisions. We acknowledge and agree that we have read these Agreed Terms of Use in their entirety, understand them and agree to be bound by them in all respects.

Pre-configuration Checklist

It is important to realize that "Web Security" extends beyond the Web Server itself. There are many different web security vulnerabilities, which do not directly involve the web server itself. In order to truly secure a web infrastructure, many different information technology divisions must work together. These include, but are not limited to Firewalls, Intrusion Detection Systems, DNS, Networks Branch, etc… Take the time to build relationships with these groups and discuss web security issues. Hopefully, you will be able to identify deficiencies in your environment and fix them prior to exploitation attempts.

The benchmark reader should review this sample checklist prior to applying the CIS Apache Benchmark.

- Reviewed and implement my company's security policies as they relate to web security.
- Implemented a secure network infrastructure by controlling access to/from your web server by using: Firewalls, Routers and Switches.
- Implemented a Network Intrusion Detection System to monitor attacks against the web server.
- Implemented load-balancing/failover capability in case of Denial of Service or server shutdown.
- Implemented a disk space monitoring process and log rotation mechanism.
- The WHOIS Domain information registered for our web presence does not reveal sensitive personnel information, which may be leveraged for Social Engineering (Individual POC Names), War Dialing (Phone Numbers) and Brute Force Attacks (Email addresses matching actual system usernames).
- Domain Name Service (DNS) servers have been properly secured to prevent domain hi-jacking via cache poisoning, etc…
- Harden the Underlying Operating System of the web server. This should include minimizing listening network services, applying proper patches and hardening the configurations.
- Educated developers about writing secure code.
    - OWASP Top Ten - http://www.owasp.org/index.php/OWASP_Top_Ten_Project
    - WASC Threat Classification - http://www.webappsec.org/projects/threat/

1 Level 1 Apache Benchmark Settings

The Prudent Level of Minimum Due Care

Level-I Benchmark settings/actions meet the following criteria.

1. *System administrators with any level of security knowledge and experience can understand and perform the specified actions.*
2. *The action is unlikely to cause an interruption of service to the operating system or the applications that run on the system.*
3. *The actions can be automatically monitored, and the configuration verified, by Scoring Tools that are available from the Center or by CIS-certified Scoring Tools.*

Many organizations running the CIS scoring tools report that compliance with a CIS "Level-1" benchmark produces substantial improvement in security for their systems connected to the Internet.

1.1 Installation

Question

Are you planning to use the precompiled Apache httpd binary that is available by default with many Unix Operating Systems or a commercial version supplied by a Vendor (such as WebLogic or Oracle's Application Server 9iAS/10G)?

If you answered yes, then continue with the section. If you answered no, or in the event vendor binaries are not available or suitable, recommended instructions for downloading, building from the source and installing are included this sample chapter from Apache Security[1] by Ivan Ristic - http://www.apachesecurity.net/download/apachesecurity-ch02.pdf.

Description

The CIS Apache Benchmark recommends using the Apache binary provided by your vendor for most situations. The benefits of using the vendor supplied binaries include[2]:

- Ease of installation as it will just work, straight out of the box.
- It is customized for your OS environment.
- It will be tested and have gone though QA procedures.
- Everything you need is likely to be included, probably including some third party modules. Many OS vendors ship Apache with `mod_ssl` and OpenSSL and PHP, `mod_perl` and ModSecurity for example.
- Your vendor will tell you about security issues in all those bits, you have to look in less places.
- Updates to fix security issues will be easy to apply. The vendor will have already verified the problem, checked the signature on the Apache download, worked out the impact and so on.
- You may be able to get the updates automatically, reducing the window of risk.

There are times when compilation from source code will be necessary or advantageous, however for most situations the vendor binaries will provide better security by ensuring that updates are applied according to the existing update process.

Which Apache Version to use?
There are currently three different Apache forks: 1.3.x, 2.0.x and 2.2.x. At the time of this writing, the current stable versions are 1.3.37, 2.0.59 and 2.2.4. There may be restrictions as to which version of Apache you must use, however if it is at all possible it is recommended that you use the 2.2.x fork. The main reasons are security related as this is the current branch where the majority of improvements are made. Additionally, in order to use the ModSecurity 2.x web application firewall package, you must be using either the 2.0.x or 2.2.x version.

Action
Install the Apache Software using vendor provided binaries if available

For Red Hat/Fedora:
```
# yum install httpd2
```

For Debian systems:
```
# apt-get install apache2-mpm-prefork
```

1.2 ModSecurity Overview

Important Note
ModSecurity has quickly matured over the years and has become the most widely deployed web application firewall. Due to the fact that is open source and free, coupled with its flexible rules language and extensive logging capabilities, the CIS Apache Benchmark highly recommends that all Apache deployments install it. We do however realize that not all deployments will be able to use this application. It is for this reason that many sections will be providing both an Apache and a ModSecurity setting that can be used to mitigate the issues.

There were many previous benchmark sections that attempted to leverage Apache modules and directives to achieve a specific security goal. Some of these settings worked to a certain degree however some were not flexible enough to fully handle the issue. ModSecurity, on the other hand, is able to better address these issues. It is for this reason that most of these benchmark settings will include an analogous ModSecurity feature or ruleset in addition to the standard Apache directive or configuration. We will still provide the Apache directive examples, however we will include information about its limitations and a rationale for using ModSecurity.

Description
ModSecurity is an open-source, free web application firewall module that integrates into the Apache web server. The project website is www.modsecurity.org. It is available under the open source GPL license, with optional commercial support and licensing

(from Breach Security – www.breach.com). There are two versions of the module, one for each major Apache branch, and they are almost identical in functionality. In the Apache 2 version, *mod_security* uses the advanced filtering API available in that version, making interception of the response body possible. The Apache 2 version is also more efficient in terms of memory consumption. In short, ModSecurity does the following:

- Intercepts HTTP requests before they are fully processed by the web server
- Intercepts the request body (e.g., the `POST` payload)
- Intercepts, stores, and optionally validates uploaded files
- Performs optional anti-evasion actions
- Performs request analysis by processing a set of rules defined in the configuration
- Intercepts HTTP responses before they are sent back to the client (*Apache 2 only*)
- Performs response analysis by processing a set of rules defined in the configuration
- Takes one of the predefined actions or executes an external script when a request or a response fails analysis (a process called *detection*)
- Depending on the configuration, a failed request may be prevented from being processed, and a failed response may be prevented from being seen by the client (a process called *prevention*)
- Performs audit logging

In this section, we will cover the essentials for installation and configuration. For a detailed reference manual, visit the project documentation area at http://www.modsecurity.org/documentation/.

Action
In order to ensure that you are using the latest and greatest version, you should download the ModSecurity source code from the project website - http://www.modsecurity.org/download/index.html. As of this writing, the current version is 2.1.2. Follow the steps outlined in the Installation section of the Reference Manual - http://www.modsecurity.org/documentation/modsecurity-apache/2.1.2/modsecurity2-apache-reference.html#02-installation

If you run into any issues with installation, configuration or usage, you should sign up on the ModSecurity users mail-list here - https://lists.sourceforge.net/lists/listinfo/mod-security-users.

1.3 ModSecurity Core Rules Overview

Description
ModSecurity is a web application firewall engine. Being the Swiss army knife of application firewalls it can do everything but requires rules to tell it what to do. In order to enable users to take full advantage of ModSecurity out of the box, ModSecurity includes the Core Rule Set, an open source rule set licensed under GPLv2. ModSecurity Core Rule Set works with ModSecurity 2.1 and above.

Unlike intrusion detection and prevention systems, which rely on signature specific to known vulnerabilities, the Core Rules is based on generic rules in order to provide protection from zero day and unknown vulnerabilities often found in web applications, which are in most cases custom coded.

As a generic negative security rule set, the Core Rule Set is only one layer in your application protection. You can learn more about the pros and cons of a negative security model in the presentation "The Core Rule Set: Generic detection of application layer", presented at OWASP Europe 2007. In addition to the Core Rule Set you may want to harden your Apache installation, for which you can consult Ivan Ristic's book, Apache Security. You may also consider writing custom rules for providing a positive security envelope to your application or critical parts of it. Breach Security can provide you with training and professional services to assist you in doing that. On additional methodologies which complement the Core Rule Set including positive security and virtual patching you can read in Ryan Barnett's book, Preventing Web Attacks with Apache

Why The Core Rule Set?

The focus of the core rule set is to be a "rule set" rather than a set of rules. What makes a rule set different than a set of rules?

- *Performance* - The Core Rule Set is optimized for performance. The amount and content of the rules used predominantly determines the performance impact of ModSecurity, so the performance optimization of the rule set is very important.
- *Quality* - While there will always be false positives, a lot of effort is put into trying to make the Core Rule Set better. Some of the things done are:
    - *Regression tests* - a regression test is used to ensure that every new version shipped does not break anything. Actually every report of a false positive, once solved, gets into the regression test.
    - *Real traffic testing* – A large amount of real world capture files have been converted to tests and sent through ModSecurity to detect potential false positives.
- *Generic Detection* - The core rule set is tuned to detect generic attacks and does not include specific rules for known vulnerabilities. Due to this feature the core rule set has better performance, is more "plug and play" and requires less updates.
- *Event Information* - Each rule in the Core Rule Set has a unique ID and a textual message. In the future rules are also going to be classified using a new tag action in ModSecurity, as well as longer information regarding each rule using comments in the files themselves.
- *Plug and Play* – The Core Rule Set is designed to be as plug and play as possible. Since its performance is good and it employs generic detection, and since the number of false positives is getting lower all the time, the Core Rule Set can be installed as is with little twisting and tweaking.

Content

In order to provide generic web applications protection, the Core Rules use the following techniques:

Protocol compliance:

- HTTP request validation - This first line of protection ensures that all abnormal HTTP requests are detected. This line of defense eliminates a large number of automated and non targeted attacks as well as protects the web server itself.
- HTTP protocol anomalies - Common HTTP usage patterns are indicative of attacks.
- Global constraints - Limiting the size and length of different HTTP protocol attributes, such as the number and length of parameters and the overall length of the request. Ensuring that these attributed are constrained can prevent many attacks including buffer overflow and parameter manipulation.
- HTTP Usage policy – validate requests against a predefined policy, setting limitations request properties such as methods, content types and extensions.

Attack Detection:

- *Malicious client software detection* - Detect requests by malicious automated programs such as robots, crawlers and security scanners. Malicious automated programs collect information from a web site, consume bandwidth and might also search for vulnerabilities on the web site. Detecting malicious crawlers is especially useful against comments spam.
- *Generic Attack Detection* - Detect application level attacks such as described in the OWASP top 10. These rules employ context based patterns match over normalized fields. Detected attacks include:
    - SQL injection and Blind SQL injection.
    - Cross Site Scripting (XSS).
    - OS Command Injection and remote command access.
    - File name injection.
    - ColdFusion, PHP and ASP injection.
    - E-Mail Injection
    - HTTP Response Splitting.
    - Universal PDF XSS.
- *Trojans & Backdoors Detection* - Detection of attempts to access Trojans & backdoors already installed on the system. This feature is very important in a hosting environment when some of this backdoors may be uploaded in a legitimate way and used maliciously.

Other:

- *Error Detection* - Prevent application error messages and code snippets from being sent to the user. This makes attacking the server much harder and is also a last line of defense if an attack passes through.
- *XML Protection* – The Core Rule Set can be set to examine XML payload for most signatures.
- *Search Engine Monitoring* - Log access by search engines crawlers to the web site.

Action

The ModSecurity archive contains a version of the Core Rules that you can use right away. There may be situations, however, where updates are made to the Core Rules and they are released outside of the normal ModSecurity code updates. You can download the Core Rules themselves on the download page - http://www.modsecurity.org/download/index.html.

Quick Start

The Core Rule Set is a set of apache configuration files that should be added to your Apache configuration. To install the Core Rule Set:

- Extract the Core Rule Set content into a directory called `modsecurity` under your Apache configuration directory.
- Edit and customize `modsecurity_crs_10_config.conf`. This file contains comments which explain how to set up each directive. You may also want to edit `modsecurity_crs_30_http_policy.conf` which sets rules specific to your application.
- Add the directive `Include conf/modsecurity/*.conf` to your `httpd.conf` after the line where ModSecurity itself it loaded.
- Restart Apache.
- Check that the server works normally, and simulate an attack by browsing to the URL `http://yourhost/cmd.exe`. Instead of a "page not found" error, you should get a "Method Not Implemented" error.

We strongly recommend that you start using the Core Rule Set in monitoring only mode by changing the directive `SecRuleEngine` in file `modsecurity_crs_10_config.conf` from `On` to `DetectionOnly`. After operating in this mode for a period of time, analyze the alerts recorded in your Apache error log. If there are alerts that you dim as false positives refer to the section about handling false positives later in this document.
Only after such a process it is safe to move back to blocking by setting `SecRuleEngine` in file `modsecurity_crs_10_config.conf` to `On`.

If you run into any issues with installation, configuration or usage, you should sign up on the ModSecurity users mail-list here - https://lists.sourceforge.net/lists/listinfo/mod-security-users.

1.4 Minimized httpd.conf file

Description

In order to ensure that you have appropriately applied the "Principle of Least Privilege" concept to your Apache server, you should not use the default httpd.conf file. The default file is quite large as it includes directives for all modules and also includes extensive comments. All of this makes it difficult to read and understand what is enabled. It is for this reason that it is recommended that you start with an empty config file and then slowly enable features. You will most likely add in new directives and then use the `apachectl configtest` command to identify any errors.

Note

If you are using the 2.2 branch of Apache, the default configuration is suitable as it has removed much of the extra comment text and its default configuration values are more secure then previous versions.

Action
Create an empty configuration file (`/usr/local/apache2/conf/httpd.conf`) and add a couple of general-purpose directives:

```
# location of the web server files
ServerRoot /usr/local/apache2
# location of the web server tree
DocumentRoot /var/www/htdocs
# path to the process ID (PID) file, which
# stores the PID of the main Apache process
PidFile /var/www/logs/httpd.pid
# which port to listen at
Listen 80
# do not resolve client IP addresses to names
HostNameLookups Off
```

1.5 Minimize Apache Modules

Description
It is important to disable the Apache modules that are not needed, in order to reduce the risk to the web server, as well as increase the performance. This is similar to the OS security issue of running unnecessary network services; such as Telnet and FTP. By enabling these unused modules, you are potentially providing additional avenues of attack against your web server. You should only enable the modules that you absolutely need for the functionality of your web site. If you are not sure which modules you need, you may read about them at the Apache Documentation website - http://httpd.apache.org/docs/2.2/mod/, then disable the module and test the functionality without the module. Shown below are a few sample `LoadModules` directives which have been commented out on Fedora Code 5.

Action
Review the Apache documentation site for module information and comment out all `LoadModule` directives in the `httpd.conf` file for modules whose functionality you will not need. Here are some examples:

```
LoadModule access_module modules/mod_access.so
LoadModule auth_module modules/mod_auth.so
LoadModule auth_anon_module modules/mod_auth_anon.so
## LoadModule auth_dbm_module modules/mod_auth_dbm.so
## LoadModule auth_digest_module modules/mod_auth_digest.so
## LoadModule ldap_module modules/mod_ldap.so
## LoadModule auth_ldap_module modules/mod_auth_ldap.so
## LoadModule cern_meta_module modules/mod_cern_meta.so
## LoadModule dav_module modules/mod_dav.so
```

1.6 Creating the Apache User and Group Accounts

Description

One of the best ways to reduce your exposure to attack when running a web server is to create a unique, unprivileged userid and group for the server application. The "nobody" userid & group that comes default on Unix variants should NOT be used to run the web server, since the account is commonly used for other separate daemon services. Instead, an account used only by the apache software so as to not give unnecessary access to other services. Also the userid used for the apache user should be a unique value between 1 and 499 as these lower values are reserved for the special system accounts not used by regular users, such as discussed in User Accounts section of the CIS Red Hat benchmark.

Create an account with a name such as: apache, which runs the web server software. In the entry below, we designate the web document root as the apache user's home directory. Since this account will never be used to log into for shell access, we do not need to create the normal user account login files.

Action

To create a new account, execute the following two commands while running as root.

```
# groupadd apache
# useradd apache -g apache -d /dev/null -s /sbin/nologin
```

These commands create a group and a user account, assigning the account the home directory /dev/null and the shell `/sbin/nologin` (effectively disabling login for the account).

Add the following two lines to the Apache configuration file httpd.conf:

```
User apache
Group apache
```

1.7 Restricting Access

Description

The `Allow` and `Deny` directives are straightforward. The Allow directive grants access while the Deny denies access. The `Order` directive is trickier to understand. The order does matter! Don't confuse this with the order that items appear in the configuration file. Consider the following example:

```
Order allow,deny
allow from apache.org
deny from foo.apache.org
```

The `Order` directive states that the `Allow` directives should be evaluated first. The `Allow` directive states that everyone from `apache.org` can have access. Then the `Deny` directive states that everyone from `foo.apache.org` should be denied access. Taken together these directives instruct the server to "allow access to everyone from apache.org except for those from foo.apache.org".

If the `Order` directive in the above example were reversed to "`deny,allow`", then the server would grant access to everyone from apache.org, because the `Allow` directive is evaluated after the `Deny` directive. In other words the server was told to "deny access to anyone from foo.apache.org then grant access to everyone from apache.org". Since `foo.apache.org` is part of `apache.org`, access will be granted to its users. The order and allow/deny parameters should be applied to any portions of your web site, which you would like to protect. You should use IP addresses when possible, to prevent any DNS spoofing attacks. You can use the allow/deny options in conjunction with any password-protected methods as well.

Action: Edit/Verify the following bolded directives in the httpd.conf file
Apply Access Control for OS Root and DocumentRoot Directories

OS Root Directory Access Control
In order to prevent any form of directory traversal trying to get outside of the document root, we will specify the directive listed below. One aspect of Apache, which is occasionally misunderstood, is the feature of default access. That is, unless you take steps to change it, if the server can find its way to a file through normal URL mapping rules, it can serve it to clients.

```
<Directory />
Options None
AllowOverride None
deny from all
</Directory>
```

DocumentRoot Directory Access Control
The example listed below essentially allows anyone to access the web site:

```
<Directory "/var/www/html/">
Order allow,deny
allow from all
</Directory>
```

You can use more restrictive options to the allow/deny options such as specific hostnames, IP addresses, domain names and IP ranges.

```
<Directory "/var/www/html/">
Order allow,deny
deny from all
allow from 202.54.
</Directory>
```

This would restrict access to only hosts who reside in the appropriate IP range (202.54.X.X).

1.8 Directory Functionality Control with the Options Directive

Description

For increased security, only those features that are absolutely necessary should be enabled. All other features should be disabled. As a side note, since we disabled numerous Apache modules when we compiled the new httpd binary, many of these `Options` directives would not work anyways. This adheres to security-in-layers and prevents the accidental enabling of an unauthorized service or feature. The `Options` directive controls what extended web server functions are applied to directories and/or files.

- The ***All*** setting states that all features are available except for Multiviews.
- The ***ExecCGI*** setting permits the execution of CGI scripts within the directory. This feature should only be applied to the designated cgi-bin directory. See next section.
- The ***FollowSymLinks*** setting allows the server to follow symbolic links found in the directory. The `FollowSymlinks` directive will instruct the Apache web server on how to handle the processing of files, which are symbolic links. If you must use symbolic links for the functionality of your web site, consider the security risks that follow. It is possible for an attacker to gain access to areas outside the specified document root if the web server is configured to follow symbolic links. We will configure this parameter setting to NOT follow symbolic links. This option is preferred over the `SymLinksIfOwnerMatch` due to the performance hit when Apache verifies a symlink and its ownership.
- The ***SymLinksIfOwnerMatch*** setting instructs the server to only follow symbolic links if the file has the same owner as the symbolic link. This directive should be used only if the use of symbolic links is absolutely necessary for proper functioning of you web server. This will apply an additional security check to verify that the file the symbolic link points to is owned by the same UID that the web server is running under. If proper ownerships and permissions are set for the `DocumentRoot`, `ServerRoot` and the OS directories (addressed in a later section), then the chances of exploiting a symbolic link is significantly reduced.
- The ***Includes*** setting permits the execution of server side includes. This directive in the terminal screen below will prevent the web server from processing Server Side Includes (SSI). These are OS commands located within the html code of a web page. SSIs are executed by the web server before the page is sent to the client. If you must use SSI, then it is recommended that you use the `IncludesNoExec` option, which will allow the server to parse SSI enabled web pages but it will not execute any system commands.
- The ***IncludesNOEXEC*** refines the Includes setting by disabling the exec command.
- The ***Indexes*** setting tells the server to automatically create a page that lists all the files within the directory if no default page exists (in other words - no index.html). Directory listings should not be allowed, since they reveal too much information to an attacker (such as naming conventions and directory structures). This directive below will prevent the web server from producing any dynamic directory listings when a default index page is not present. This `httpd.conf` file

directive is actually redundant, since we have already disabled the `mod_autoindex` module when we compiled our httpd binary file.

- The ***AllowOverrride*** setting tells the server how to handle access control from .htaccess files. When the server finds a `.htaccess` file (as specified by `AccessFileName`) it needs to know which directives declared in that file can override earlier access information. When this directive is set to `None`, then `.htaccess` files are completely ignored. In this case, the server will not even attempt to read `.htaccess` files in the filesystem. When this directive is set to `All`, then any directive which has the `.htaccess` Context is allowed in `.htaccess` files. While the functionality of htaccess files is sometimes relevant, from a security perspective, this decentralizes the access controls out from the `httpd.conf` file. This could make it much more difficult to manage the security of your web site if rogue `.htaccess` files are created.
- The ***Multiviews*** setting allows for multiple files to refer to the same request. It can be used to have the same page in different languages with each language having a different final file suffix. This setting could also be leveraged as part of an IDS Evasion attack.

Action

Edit/Verify the following bolded directives for the `DocumentRoot` directory in the `httpd.conf` file. The `Options` directives listed above will implement the security related features.

```
<Directory "/var/www/html">
 Order allow,deny
 Allow from all
 Options None
 AllowOverride None
</Directory>
```

1.9 Enabling CGI Scripts

Description

Only enable CGI scripts if and when you need them. When you do, a good practice is to have all scripts grouped in a single folder (typically named `cgi-bin`). That way you will know what is executed on the server. It is possible to enable CGI script execution based on file extension or permission settings however this makes script control and management almost impossible as developers may install scripts without your knowledge. This may become a factor in a hosting environment.

Action

To allow execution of scripts in the `/var/www/cgi-bin directory`, include the following `<Directory>` directive in the configuration file:

```
<Directory /var/www/cgi-bin>
    Options ExecCGI
    SetHandler cgi-script
</Directory>
```

1.10 Limiting HTTP Request Methods

Description

We want to restrict the functionality of our web server to only accept and process certain HTTP Methods. For normal web server operation, you will typically need to allow the both the GET and POST request methods (and in some cases HEAD requests). This will allow for downloading of web pages and uploading any type of basic form submission information.

Action

Apache Directives

If you can not use ModSecurity, then you could use the LimitExcept directive to specify the allowed Request methods either globally or within a certain directory location.

```
<LimitExcept GET POST OPTIONS>
Deny from all
</LimitExcept>
```

There are two limitations with LimitExcept vs. using ModSecurity:

1. The HEAD method is tied to the GET method so there is no way to disallow only HEAD requests.
2. You can not deny TRACE requests with this directive.

If you want to disable the TRACE method, you can use the TraceEnable directive and set it to Off.

ModSecurity Alternative

Use the ModSecurity Core Rules as the modsecurity_crs_30_http_policy.conf file has the following rule to restrict which HTTP Request Methods clients may use –

```
# allow request methods
#
# TODO Most applications only use GET, HEAD, and POST request
# methods, if so uncomment the line below. Otherwise you are advised
# to edit the line before uncommenting it.
#
SecRule REQUEST_METHOD "!^((?:(?:POS|GE)T|OPTIONS|HEAD))$" \
    "phase:1,log,auditlog,status:501,msg:'Method is not allowed by
policy', severity:'2',,id:'960032',"
```

You may update the Regular Expression to allow for other Request Methods as necessary as some applications such as WebDAV will need others not listed.

1.11 Restrict HTTP Protocol Version

Description

Many malicious automated programs, vulnerability scanners and fingerprinting tools will send abnormal HTTP protocol versions to see how the web server responds. These requests are usually part of the attacker's enumeration process and there it is important that we respond by denying these requests.

Action

Apache Directives – Mod_Rewrite or SetEnvIf

If you can not use ModSecurity, then you could use the Mod_Rewrite directives to specify the allowed HTTP version information.

```
RewriteEngine On
RewriteCond %{THE_REQUEST} !HTTP/(0\.9|1\.[01])$
RewriteRule .* - [F]
```

Another alternative would be use `SetEnvIf` to evaluate the protocol version and then set an environmental variable that could then be controlled with the "allow" –

```
SetEnvIf REQUEST_PROTOCOL ^HTTP/(0\.9|1\.[01])$ valid_protocol=1
<Directory "/var/www/html">
 Order allow,deny
 Allow from env=valid_protocol
 Options None
 AllowOverride None
</Directory>
```

ModSecurity Alternative

Use the ModSecurity Core Rules as the `modsecurity_crs_30_http_policy.conf` file has the following rule to restrict which HTTP Protocol version clients may use –

```
# Restrict protocol versions.
#
# TODO All modern browsers use HTTP version 1.1. For tight security,
# allow only this version.
#
# NOTE Automation programs, both malicious and non malicious many times
# use other HTTP versions. If you want to allow a specific automated
# program to use your site, try to create a narrower exception and not
# allow any client to send HTTP requests in a version lower than 1.1
#
SecRule REQUEST_PROTOCOL "!^HTTP/(0\.9|1\.[01])$" \
    "t:none, deny,log,auditlog,status:505,msg:'HTTP protocol version is
not allowed by policy', severity:'2',,id:'960034',"
```

1.12 Restrict File Extensions

Description

There are many files that are left within the web server `DocumentRoot` that could provide an attacker with sensitive information. Most often these files are left behind by mistake after installation, trouble-shooting for backing up files before editing. Regardless of the reason for their creation, these file can still be served by Apache even when there is no

hyperlink pointing to them. It is for this reason that files with sensitive file extensions, such as `.bak`, `.config`, `.old`, etc should be restricted from client access.

Action

Apache Directive

If you can not use ModSecurity, then you could use the `FilesMatch` Apache directive to specify the file extension information to block.

```
<FilesMatch
"\.(?:c(?:o(?:nf(?:ig)?|m)|s(?:proj|r)?|dx|er|fg|md)|p(?:rinter|ass|db|
ol|wd)|v(?:b(?:proj|s)?|sdisco)|a(?:s(?:ax?|cx)|xd)|d(?:bf?|at|ll|os)|i
(?:d[acq]|n[ci])|ba(?:[kt]|ckup)|res(?:ources|x)|s(?:h?tm|ql|ys)|l(?:ic
x|nk|og)|\w{,5}~|webinfo|ht[rw]|xs[dx]|key|mdb|old)$">
Deny from all
</FilesMatch>
```

ModSecurity Alternative

Use the ModSecurity Core Rules as the `modsecurity_crs_30_http_policy.conf` file has the following rule to restrict common unsafe file extensions –

```
# Restrict file extension
#
# TODO the list of file extensions below are virtually always
# considered unsafe and not in use in any valid program. If your
# application uses one of these extensions, please remove it from the
# list of blocked extensions.
# You may need to use ModSecurity Core Rule Set Templates to do so,
# otherwise comment the whole rule.
#
SecRule REQUEST_BASENAME
"\.(?:c(?:o(?:nf(?:ig)?|m)|s(?:proj|r)?|dx|er|fg|md)|p(?:rinter|ass|db|
ol|wd)|v(?:b(?:proj|s)?|sdisco)|a(?:s(?:ax?|cx)|xd)|d(?:bf?|at|ll|os)|i
(?:d[acq]|n[ci])|ba(?:[kt]|ckup)|res(?:ources|x)|s(?:h?tm|ql|ys)|l(?:ic
x|nk|og)|\w{,5}~|webinfo|ht[rw]|xs[dx]|key|mdb|old)$" \
    "t:urlDecodeUni, t:lowercase, deny,log,auditlog,status:500,msg:'URL
file extension is restricted by policy', severity:'2',,id:'960035',"
```

1.13 Denial of Service Prevention Tuning

Description

Denial of Service (DoS) is an attack technique with the intent of preventing a web site from serving normal user activity. DoS attacks, which are easily normally applied to the network layer, are also possible at the application layer. These malicious attacks can succeed by starving a system of critical resources, vulnerability exploit, or abuse of functionality.

Many times DoS attacks will attempt to consume all of a web site's available system resources such as: CPU, memory, disk space etc. When any one of these critical resources reach full utilization, the web site will normally be inaccessible.

As today's web application environments include a web server, database server and an authentication server, DoS at the application layer may target each of these independent components. Unlike DoS at the network layer, where a large number of connection attempts are required, DoS at the application layer is a much simpler task to perform. We will be verifying/updating the following Apache directives:

- ***Timeout***
  One way of attacking systems on the Internet is to try to prevent the target system from operating correctly by overloading it. This is called a 'denial of service' attack. One method of doing this is to open multiple connections to a server and never close them. The more connections the server has open at once, the more resources are tied up holding details of those connections, which can lead to increased load and eventually to the server running out of resources. The Timeout directive tells the server how long to wait to receive a GET request, the amount of time between receipt of TCP packets on a POST or PUT request, or the amount of time between ACKs on transmissions of TCP packets in responses. In order to prevent a denial of service attack from shutting down our web server, we need to change the default setting of 300 (which is 5 minutes) to 10 (which is 10 seconds).

  Note:
  For apache 2.0 it has been noted, that this Timeout affects the connection timeout to a backend server in a Reverse Proxy setup as well. This is done regardless of the ProxyTimeout directive. This behavior limits the use of very low Timeout values. This problem is solved for apache 2.2.
- ***KeepAlive***
  In order to make our web server more efficient, and therefore more resistant to DoS attacks, we want to allow TCP `KeepAlives` generally. This setting will allow for multiple HTTP requests to be serviced across one connection. Without this setting, a new Apache server would have to be assigned or even spawned for every subsequent HTTP request for gifs, etc. Recommended value is thus `On`. However, `KeepAlive` trades RAM for CPU. The Apache servers have to do less handshakes, which is very important in a SSL-Setup. But the servers are assigned to a particular client for a longer period of time. The effect will be, that you will need more servers and thus more Ram, but that the servers have less work to do, thus freeing your CPU. While "`KeepAlive On`" is generally a good setting, it can mean a DoS problem in itself, when an attacker tries to consume the system's RAM. It helps to know your particular traffic pattern and your hardware limits very well, when setting the `KeepAlive` option.
- ***KeepAliveTimeout***
  This directive will limit the time the Apache web server will wait in seconds for a `KeepAlive` request to be completed. Setting `KeepAliveTimeout` to a high value may cause performance problems as pointed out above. Recommend value is 15 which is also the default value.

  Note:

mod_qos has an option to stop serving KeepAlive requests when the load becomes too high.

- ***AcceptFilter***
  This directive helps to prevent DoS types of attacks where a malicious client initiates a connection that opens a socket, however they do not send any data. This directive will limit the impacts of this type of attack by not sending the socket to the Apache process unless data is received. This directive, however, is not supported on all OS platforms.
- ***Other Performance and DoS Related Modules and Directives***
  Taking together the directives `StartServers`, `MinSpareServers`, `MaxSpareServers` and `MaxClients`, along with the MPM (`MultiProcessing Module`) work to establish a reasonable yet dynamic number of child processes and/or threads. Starting with Apache 1.3 the process creation process was dramatically improved so that tweaking the `StartServers`, `StartServers`, `MaxSpareServers` and `MaxClients` settings is unnecessary for most situations. Therefore the default settings are fine for most usages. For very high traffic servers, and optimal performance consult the Apache performance recommendations at http://httpd.apache.org/docs/2.2/misc/perf-tuning.html.

Action
Edit/Verify the following bolded directives in the httpd.conf file.

```
Timeout 10
KeepAlive On
KeepAliveTimeout 15
AcceptFilter http data
AcceptFilter https data
```

1.14 Buffer Overflow Protection Tuning

Description
Buffer Overflow exploits are attacks that alter the flow of an application by overwriting parts of memory. Buffer Overflow is a common software flaw that results in an error condition. This error condition occurs when data written to memory exceed the allocated size of the buffer. As the buffer is overflowed, adjacent memory addresses are overwritten causing the software to fault or crash. When unrestricted, properly-crafted input can be used to overflow the buffer resulting in a number of security issues.

A Buffer Overflow can be used as a Denial of Service attack when memory is corrupted, resulting in software failure. Even more critical is the ability of a Buffer Overflow attack to alter application flow and force unintended actions. This scenario can occur in several ways. Buffer Overflow vulnerabilities have been used to overwrite stack pointers and redirect the program to execute malicious instructions. Buffer Overflows have also been used to change program variables.

Buffer Overflow vulnerabilities have become quite common in the information security industry and have often plagued web servers. However, they have not been commonly seen or exploited at the web application layer itself. The primary reason is that an attacker needs to analyze the application source code or the software binaries. Since the attacker must exploit custom code on a remote system, they would have to perform the attack blind, making success very difficult.

Buffer Overflows vulnerabilities most commonly occur in programming languages such as C and C++. A Buffer Overflow can occur in a CGI program or when a web page accesses a C program.[3]

In addition to OS network stack tuning, the Apache directives listed below limit the size of the various HTTP header "strings" being copied. Implementing these directives greatly reduces the chance of a successful buffer overflow.

- ***LimitRequestBody***
  Limits the total size of the HTTP request body that is sent to the Apache web server. These parameters usually come into effect during HTTP PUT and POST requests where the client is sending data back the web server from a form, or sending data into a CGI script. The setting below will restrict the request body size to be no more than **10K**. You will need to increase this size if you have any forms that require larger input from clients.

- ***LimitRequestFields***
  Limits the number of additional headers that can be sent by a client in an HTTP request, and defaults to 100. In real life, the number of headers a client might reasonably be expected to send is around 20, although this value can creep up if content negotiation is being used. A large number of headers may be an indication of a client making abnormal or hostile requests of the server. A lower limit of **40** headers can be set with the setting below.
- ***LimitRequestFieldsize***
  Limits the maximum length of an individual HTTP header sent by the client, including the initial header name. The default (and maximum) value is 8190 characters. We can set this to limit headers to a maximum length of **100** characters with the setting below.
- ***LimitRequestlLine***
  Limits the maximum length of the HTTP request itself, including the HTTP method, URL, and protocol. The default limit is 8190 characters; we can reduce this to **500** characters with the line below. The effect of this directive is to effectively limit the size of the URL that a client can request, so it must be set large enough for clients to access all the valid URLs on the server, including the query string sent by GET requests. Setting this value too low can prevent clients from sending the results of HTML forms to the server when the form method is set to GET.

Caution
You should test these setting extensively prior to deploying into production. You should verify that all CGI scripts and uploading scripts work appropriately with these settings, otherwise, you can effectively cause a Denial of Service attack against yourself!

Action
Apache Directive
You can use the following Apache `Limit` directives to help restrict some general request characteristics.

```
# Maximum size of the request body.
LimitRequestBody 10000

# Maximum number of request headers in a request.
LimitRequestFields 40

# Maximum size of request header lines.
LimitRequestFieldSize 100

# Maximum size of the request line.
LimitRequestLine 500
```

ModSecurity Alternative
The ModSecurity rule file `modsecurity_crs_23_request_limits.conf` has additional example limit directives that can be enabled to restrict sizes on other aspects of a request. These settings may need to be tweaked for your environment.

```
## -- Arguments limits --
```

```
# Limit argument name length
#SecRule ARGS_NAMES "^.{100}"
"phase:2,t:none,deny,log,auditlog,status:403,msg:'Argument name too
long',id:'960209',severity:'4'"

# Limit value name length
#SecRule ARGS "^.{400}"
"phase:2,t:none,deny,log,auditlog,status:403,msg:'Argument value too
long',id:'960208',severity:'4'"

# Maximum number of arguments in request limited
SecRule &ARGS "@gt 255"
"phase:2,t:none,deny,log,auditlog,status:403,msg:'Too many arguments in
request',id:'960335',severity:'4'"

# Limit arguments total length
#SecRule ARGS_COMBINED_SIZE "@gt 64000"
"phase:2,t:none,deny,log,auditlog,status:403,msg:'Total arguments size
exceeded',id:'960341',severity:'4'"
```

1.15 Implementing Mod_SSL

Description

Digital certificates encrypt data using Secure Sockets Layer (SSL) technology, the industry-standard method for protecting web communications developed by Netscape Communications Corporation. The SSL security protocol provides data encryption, server authentication, message integrity, and optional client authentication for a TCP/IP connection. Because SSL is built into all major browsers and web servers, simply installing a digital certificate turns on their SSL capabilities. Server certificates are designed to protect you and visitors to your site. Installing a digital certificate on your server lets you:

- **Authenticate your site.**
  A digital certificate on your server automatically communicates your site's authenticity to visitors' web browsers. If a trusted authority signs your certificate, it confirms for the visitor they are actually communicating with you, and not with a fraudulent site stealing credit card numbers or personal information.
- **Keep private communications private.**
  Digital certificates encrypt the data visitors that exchange with your site to keep it safe from interception or tampering using SSL (Secure Sockets Layer) technology.

Important

Implementing SSL does NOT directly make your web server more secure! SSL is used to encrypt traffic and therefore does provide confidentiality of the users credentials when access your web server. Just because you have encrypted the data in transit does not mean that the data provided by the client is secure while it is on your server. Also, attackers will target SSL-Enabled web servers, since the encrypted channel will hide their activities

from Network Based Intrusion Detection Systems. (See the Level II mod_security section for IDS functionality over SSL).

Action

For most systems it should be a simple matter of getting the mod_ssl and openssl rpms. For Fedora Core the simple command below will install mod_ssl if it is not already installed.

```
# rpm -q mod_ssl || yum install mod_ssl
```

If you have built from source code, starting with Apache 2 the `mod_ssl` is bundled in and you simply need to openssl and openssl-devel rpm's installed and add the `--enable-ssl` option to the configure script, before compiling, then configure mod_ssl as explained below. If you have built Apache 1.3 from source code, follow the instructions found at the mod_ssl website http://www.modssl.org/example/

The mod_ssl will automatically generate a dummy certificate which will not allow visitors to authenticate your server, but will provide encrypted communications. To get a trusted SSL certificate, follow the instructions below substituting example.com etc. with your organizations information. If the web server is only for lab use, a trusted certificate is not required, and you can skip this step. There are 3 requirements for the certificate to be trusted:

1. The common name (CN) on the certificate must match the URL host name which the user typed into the browser or received in an html link.
2. The certificate must be signed by a certificate authority trusted by the users browser (or by the web service / application).
3. The current date must not be beyond the certificate expiration date.

```
# cd /etc/pki/tls/certs
# make www.example.com.key
umask 77 ; \
/usr/bin/openssl genrsa -des3 1024 > example.com.key
Generating RSA private key, 1024 bit long modulus
.........++++++
.................................++++++
e is 65537 (0x10001)
Enter pass phrase:
Verifying - Enter pass phrase: # chown root www.example.com.key
# chmod 600 www.example.com.key
# mv www.example.com.key /etc/pki/tls/private/
# make www.example.com.csr
umask 77 ; \
/usr/bin/openssl req -utf8 -new -key example.com.key -out
example.com.csr
Enter pass phrase for example.com.key:
You are about to be asked to enter information that will be
incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a
DN.
There are quite a few fields but you can leave some blank
```

```
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [GB]:US
State or Province Name (full name) [Berkshire]:New York
Locality Name (eg, city) [Newbury]:Lima
Organization Name (eg, company) [My Company Ltd]:Durkee Consulting
Organizational Unit Name (eg, section) []:
Common Name (eg, your name or your server's hostname)
[]:www.example.com
Email Address []:webadmin@example.com
Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:
An optional company name []:
# more example.com.csr
-----BEGIN CERTIFICATE REQUEST-----
MIIByzCCATQCAQAwgYoxCzAJBgNVBAYTAlVTMREwDwYDVQQIEwhOZXcgWW9yazEN
MAsGA1UEBxMETGltYTEaMBgGA1UEChMRRHVya2VlIENvbnN1bHRpbmcxGDAWBgNV
. . CUT . . .
```

Next send the text content of `example.com.csr` to your certificate authority to be signed. It should be sent via a means that ensure that it arrives at the proper destination and is protected it from being modified in transit. Typically the certificate signing requests are submitted, not surprisingly, to a web site with an SSL connection. The resulting signed certificate we will name `example.com.crt` and placed in `/etc/pki/tls/certs/`. Please note that the certificate authority does not need the private key (`example.com.key`) and this file must be carefully protected. With a decrypted copy of the private key, it would be possible to decrypt all conversations with the server.

Do not forget the pass phrase used to encrypt the private key. It will be required every time the server is started in https mode. To avoid requiring an administrator having to type the passphrase every time the httpd service is started, the private key may be stored in clear text. Storing the private key in clear text increases the convenience while increasing the risk of disclosure of the key, but may be appropriate if the risks are well managed. To decrypt the private the key and store it in clear text file the following `openssl` command may be used. You can tell by the private key headers whether it is encrypted or clear text.

```
# openssl rsa -in example.com.key -out example.com.key.clear
# chmod 600 example.com.key.clear
# chown root example.com.key.clear
# more example.com.key
-----BEGIN RSA PRIVATE KEY-----
Proc-Type: 4,ENCRYPTED
DEK-Info: DES-EDE3-CBC,2329D7D488CCA032
fdvTGcfVDJR9wteGVkqUgAe5lHYUmKcaW20IupDRcVxjWH7ieKs1ETgIVmrpJ9T3
5nJEp4d9Sulcv6NSNGptmEPpEiWuoLEzl5wTGKzGxdF+l2Nw/2Wl6AXtGAnlTrN4
. . CUT . . . .

# more example.com.key.clear
-----BEGIN RSA PRIVATE KEY-----
```

```
MIICWwIBAAKBgQDw7TZBR83WEoG40Podbe4ruvaCAMuGVtXpvC+8NSK8JOrREMA7
lDoJNEkxKhgGqrc9TqNHIcQiuM0ZZv4hiNDw9I2w46r3iRPnAJUwhlwHeU4SVvxW
. . . CUT . . .
```

To view the information details in a certificate, use:

```
# openssl x509 -in www.example.com.crt -text | more
```

Now we need to update the SSL entries in the the `ssl.conf` located in `/etc/httpd/conf.d` or in the `httpd.conf file`, or possibly the `httpd-ssl.conf` file located in the conf/extra directory.

```
ErrorLog /var/log/httpd/ssl_error_log
TransferLog /var/log/httpd/ssl_access_log
LogLevel info
SSLEngine on
# Allow SSLv3 and TLSv1 but not SSLv2
SSLProtocol all -SSLv2
# Disallow ciphers that are weak (obsolete or
# known to be flawed in some way). The use of
# an exclamation mark in front of a cipher code
# tells Apache never to use it. EXP refers to 40-bit
# and 56-bit ciphers, NULL ciphers offer no encryption.
# ADH refers to Anonymous Diffie-Hellman key exchange
# which effectively disables server certificate validation,
# and LOW refers to other low strength ciphers.
SSLCipherSuite ALL:!EXP:!NULL:!ADH:!LOW:!SSLv2
SSLCertificateFile /etc/pki/tls/certs/www.example.com.crt
SSLCertificateKeyFile /etc/pki/tls/certs/www.example.com.key

# Default CA file, can be replaced with your CA's certificate.
SSLCACertificateFile /etc/pki/tls/certs/ca-bundle.crt
SSLVerifyClient none
```

The `SSLProtocol` line disables the weak SSLv2 protocol, which is important. In addition, the `SSLCipherSuite` selects which ciphers are allowed. The `openssl` command can be very useful in debugging and testing the SSL configurations. See http://www.openssl.org/docs/apps/ciphers.html as well as OWASP testing tips http://www.owasp.org/index.php/SSL/TLS_Testing:_support_of_weak_ciphers.
The CA file can be replaced by your own CA, or the self signed certificate if you did not setup a trusted certificate. If you did use an accepted Certificate Authority, you can use the certificate of the CA instead of the CA bundle, to speed up the initial SSL connection.
Lastly, start or restart the `httpd` service and verify correct functioning with your favorite browser:

```
# service httpd start
```

1.16 Software Information Leakage Protection

Description

Information is power, and identifying web server details greatly increases the efficiency of any attack, as security vulnerabilities are extremely dependent upon specific software

versions and configurations. Excessive probing and requests may cause too much "noise" being generated and may tip off an administrator. If an attacker can accurately target their exploits, the chances of successful compromise prior to detection increase dramatically. Script Kiddies are constantly scanning the Internet and documenting the version information openly provided by web servers. The purpose of this scanning is to accumulate a database of software installed on those hosts, which can then be used when new vulnerabilities are released. There are a few Apache configuration directives, which will aid in protecting the discloser of some of this information.

One way to protect the Apache server is to make it limit the information provided to a potential attacker about the web server version. There are several ways that the server can leak identifying information. We will be verifying/updating the following Apache directives:

- ***ServerTokens***
  This configuration setting aids in hiding the web server software version and module versions. We do not want to give away any more information about our web server than is absolutely necessary. We will only give out the minimum server token information, therefore the recommended setting is `Prod` or `ProductOnly` which does not provide any information on versions or modules loaded, but only provides the "Apache" in the server HTTP response header.
  - ServerTokens Prod[uctOnly]
    Server sends (e.g.): Server: Apache
  - ServerTokens Major
    Server sends (e.g.): Server: Apache /2
  - ServerTokens Minor
    Server sends (e.g.): Server: Apache /2.0
  - ServerTokens Min[imal]
    Server sends (e.g.): Server: Apache/2.0.41
  - ServerTokens OS
    Server sends (e.g.): Server: Apache/2.0.41 (Unix)
  - ServerTokens Full (or not specified)
    Server sends (e.g.): Server: Apache/2.0.41 (Unix) PHP/4.0 MyMod/1.2

  Note: Section 2.7 provides instruction for configuring Apache, along with ModSecurity, to advertise false server information.

- ***ServerSignature***
  In order to protect which web server software we are using, we should not disclose this information in any of our system generated Error pages. The `ServerSignature` directive instructs Apache to append certain footer information to the bottom of error pages. Here is an example response with a signature:

  ```
  The requested URL /nosuch was not found on this server.
  Apache Server at 10.1.1.12 Port 80
  ```

If we do not disable the `ServerSignature` setting, we may be reducing the security benefit gained by changing the `ServerTokens`. While it is true that the `ServerSignature` would show only the "Apache" server token setting we specified in the `ServerTokens` directive, this signature feature is still in the Apache style and may be an additional identifier. By removing the `ServerSignature`, we can take another step towards protecting our web server software information.

Action
Apache Directive
Edit/Verify the following directives in the `httpd.conf` file:

```
# Reveal full identity (standard Apache directive)
ServerTokens Prod
ServerSignature Off
```

1.17 Logging

Description
The server logs are invaluable for a variety of reasons. They can be used to determine what resources are being used most. They can also be used to spot any potential problems before they become serious. Most importantly, they can be used to watch for anomalous behavior that may be an indication that an attack is pending or has occurred. If there are multiple web sites, or with large websites with multiple people responsible for portions of the web site, each responsible individual or organization needs access to their own web logs, and needs the skills/training/tools for monitor the logs. We will be covering the following logging directives:

- ***LogLevel***
  This directive controls the verbosity of information that is logged in the error log file. This style of level warning is similar to that of the `syslog` facility (`emerg`, `alert`, `crit`, `error`, `warn`, `notice`, `info` and `debug`). The parameter in the recommendation below specifies that all information which is at a level of `notice` or higher will be logged.

- ***ErrorLog***
  This directive sets the name of the file to which the server will log any errors it encounters. Make sure that there is adequate disk space on the partition that will hold the all log files, and that log rotation is configured. Do not hold any Apache log files on the Root partition of the OS. This could result in a denial of service against your web server host by filling up the root partition and causing the system to crash.

- ***LogFormat***
  The `LogFormat` directive allows for the definition of log entries that have exactly the information desired. The basic structure of the directive is `LogFormat format_specification format_name`. The format specification is compromised of a series of variable replacements. A named `LogFormat` directive does nothing, but is used in other directives such as the `CustomLog` entry.

- ***CustomLog***
  This directive specifies both which file to log access attempts to and what type of format the file should use. The directive above says that the `access_log` file will be combined and will contain both `referrer` and `user_agent` information. This information will be use in a later section when we receive alerts for malicious http requests. The `CustomLog` directive is used to log requests to the server. Its structure is `CustomLog logfile_name format_specification`. The entry below uses the format specification name "combined". The `CustomLog` directive is also valid in a virtual host context which means that different user requests can be logged by virtual host.

  The `CustomLog` directive can also pipe its output to a command, but this can represent a severe security risk. This is because the command will be executed as the user that started the Apache server. Remember - in order to listen on port 80, Apache must be started with root. That means that when the log output is piped to a command the command will be running with root privileges. If that command can be compromised, then the attacker has gained root access to the server.

  Make sure that there is adequate disk space on the partition that will hold the all log files. Do not hold any Apache log files on the Root partition of the OS. This could result in a denial of service against your web server host by filling up the root partition and causing the system to crash.

Action

Apache Directives

Edit/Verify the following bolded directives in the `httpd.conf` file

```
LogLevel notice
ErrorLog /var/log/httpd/error_log
LogFormat "%h %l %u %t \"%r\" %>s %b \"%{Accept}i\" \"%{Referer}i\"
\"%{User-Agent}i\"" combined
CustomLog /var/log/httpd/access_log combined
```

Edit/Verify the log rotation configuration, such as `/etc/logrotate.d/httpd` is in place to rotate the logs.

```
/var/log/httpd/*log {
missingok
notifempty
sharedscripts
postrotate
/bin/kill -USR1 `cat /var/run/httpd.pid 2>/dev/null` 2> /dev/null ||
true
endscript
}
```

ModSecurity Alternative

ModSecurity has outstanding logging capabilities. The directive `SecAuditEngine` works independently of the `SecRuleEngine`, so you can log data without applying any filters. This is sometimes useful when gathering logging data prior to creating filters. There are four main directives that work together for auditing:

```
SecAuditEngine    RelevantOnly
SecAuditLogRelevantStatus "^[45]"
SecAuditLogType Serial
SecAuditLog logs/modsec_audit.log
```

After these directives are set, ModSecurity will log all requests to the `modsec_audit.log` file in the `logs` directory. An example audit log entry looks like this –

```
--7b781024-A--
[15/Apr/2007:03:53:57 --0400] Cfjlm38AAAEAAHkzZRIAAAFS 211.39.116.16
39116 192.168.100.133 80
--7b781024-B--
GET /index.php?pagina=http://www.lifesciencessociety.org/surveyor/
lang/xpl/pro18.txt?&cmd=id;ls%20/;w HTTP/1.1
TE: deflate,gzip;q=0.3
Connection: TE, close
Host: www.example.com
User-Agent: libwww-perl/5.805

--7b781024-F--
HTTP/1.1 404 Not Found
Content-Length: 1635
Content-Type: text/html
Connection: close

--7b781024-E--
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<HTML><HEAD><TITLE>The page cannot be found</TITLE>
--CUT--
</TD></TR></TABLE></BODY></HTML>

--7b781024-H--
Message: Warning. Pattern match
"(?:\\b(?:(?:n(?:et(?:\\b\\W+?\\blocalgroup|\\.exe)|(?:map|c)\\.exe)|t(
?:racer(?:oute|t)|elnet\\.exe|clsh8?|ftp)|(?:w(?:guest|sh)|rcmd|ftp)\\.
exe|echo\\b\\W*?\\by+)\\b|c(?:md(?:(?:32)?\\.exe\\b|\\b\\W*?\\/c)|d(?:\
\b\\W*?[\\\\/]|\\W*?\\.\\.)|hmod.{0,40}? ..." at ARGS:cmd. [id
"950006"] [msg "System Command Injection. Matched signature <;ls>"]
[severity "CRITICAL"]
Apache-Handler: proxy-server
Stopwatch: 1176623637915035 7832 (446 1921 7507)
Response-Body-Transformed: Dechunked
Producer: ModSecurity v2.1.0-4 (Apache 2.x)
Server: ModSecurity HTTP Proxy/1.1.0 (Unix) mod_ssl/1.1.0
OpenSSL/0.9.7f

--7b781024-Z--
```

1.18 Remove Default Content

Description

Most web server applications come with sample applications or features that can be remotely exploited and which can provide different levels of access to the server. In the Microsoft arena, Code Red exploited a problem with the index service provided by the Internet Information Service. Usually these routines are not written for production use and consequently little thought was given to security in their development. The primary function for these sample routines is to demonstrate the capabilities of the web server. Sometimes they only exist to prove that the system is capable of executing a CGI script.

We will be removing the following files:

- ***Default HTML Files***
  By default, Apache will install a number of files within the document root directory. These files are meant to be of assistance to the Web Admin after successfully installing Apache. Included in these files is the famous "Seeing this instead of the web site you expected?" page. This is the page that will be displayed if you have not created any new html index pages. Also included in these files is the entire Apache documentation library. While all of these files are helpful, they do not conform to our security goal of hiding which type of web server software we are running. It would be foolish to go through all of our previous steps to protect our web server software version, only to loudly announce with these web pages that we are running Apache. By the way, all of the Apache documentation is available at the Apache web site.

- ***Sample CGI Scripts***
  Attackers will often try to exploit CGI programs on the web server. They will either use these programs for reconnaissance purposes or to try and exploit the web server/OS directly. CGI programs have a long history of security bugs and problems associated with improperly accepting user-input. Since these programs are often targets of attackers, we need to make sure that there are no stray CGI programs that could potentially be used for malicious purposes. By default, Apache 1.3.27 comes with two stock CGI scripts. These are called - printenv and test-cgi. Both of these programs should be either renamed or removed entirely from the web server. This is due to the sensitive information, which an attacker could gain if they are able to successfully access these files. In addition to removing the stock CGIs, care should be taken to review the functionality and code of any new CGIs which are created. The topic of safe CGI scripting is beyond the scope if this document, however, it should not be overlooked. The Web Developers should follow safe coding practices, such as, those outlined in the WWW Security FAQ - http://www.w3.org/Security/Faq/wwwsf4.html.

- ***Apache Manual Files***
  Make sure to remove the Apache manual directory and files that are present in the `ServerRoot`. This virtual directory will not be accessible by clients if you removed the default httpd.conf file directives during the minimization steps.

- ***Apache Source Code***

In order to keep our compiled installations of Apache secure, we will not keep the Apache source code on the production server. This will prevent any unauthorized users from re-compiling/compiling a new version of Apache.

Action
Execute the following commands. Remove Any Default Apache HTML Files, if any:

```
# rm -rf /usr/local/apache2/htdocs/*
```

Remove Default Apache CGI Scripts, if any:

```
# rm –rf /usr/local/apache2/cgi-bin/*
```

Remove the Apache Manual directory and files:

```
# rm –rf /usr/local/apache2/manual
```

Remove Apache Source Code if built from source:

```
# rm -rf /path/to/httpd-2.2.4*
```

1.19 Updating Ownership and Permissions

Description
Setting the appropriate ownership and permissions of the Apache files and directories can help to prevent/mitigate exploitation severity and information disclosure. These changes should be made just before deployment into production to correct any insecure settings during your testing phase. It is also advisable to check/update these settings on a continued basis through a `cron` job.

Action
Update the permissions to make sure only *root* has write access:

```
# chown -R root:root /usr/local/apache2
# find /usr/local/apache2 -type d | xargs chmod 755
# find /usr/local/apache2 -type f | xargs chmod 644
```

It is generally a best practice to limit administrator access to your web server and restrict the read permissions on the configuration and log files to the root account. This may not be applicable in enterprise setups. You should put those users who need to read these files into a special group to conform with the least privilege policy.

```
# chmod -R go-r /usr/local/apache2/conf
# chmod -R go-r /usr/local/apache2/logs
```

1.20 Monitor Vulnerability Lists

Description

One of the most frustrating aspects of web attacks is that most can be prevented if the appropriate patches are applied. Both OS and web server vendors are constantly issuing patches in response to flaws found within their application's code. Keeping abreast of new patches can be a daunting task to say the least. To keep abreast of Issues specific to Apache software and the operating system platform the individuals responsible for security and/or administration of the server should subscribe to a notification service such as those listed below that will alert them to newly discovered security issues.

Action
Subscribe to the appropriate Security Advisory List[2]

- ***Apache httpd mailing list***
  http://httpd.apache.org/lists.html
  The main announcement mailing list is going to tell you whenever a new release of Apache comes out and about security fixes but doesn't usually contain much information about the actual issues. Serious vulnerabilities tend to get their own advisories written up which also get posted to the announce list.

  Other lists such as the httpd developer list are also available but are generally high volume. The httpd developer list rarely contains any details or analysis of security issues anyway.

- ***Apache Web site***
  http://httpd.apache.org/
  The web site doesn't contain any more information than the mailing list. It's hard to keep track of a site anyway, it's not like you'd check it on a daily basis.

- ***CERT CC***
  http://www.cert.org/
  The Computer Emergency Response Team Co-ordination Centre monitors security incidents – mostly focused on those that have a significant impact. CERT advisories are well researched and a good source of information, especially when CERT was notified of an issue in advance. Not all issues are notified to CERT so it cannot be relied upon as a sole source of information, and since CERT deal with issues across all products and operating systems they are not always able to give immediate updates. Even so, it is well worth subscribing to their alert lists.

- ***Bugtraq***
  http://online.securityfocus.com/archive/1
  Bugtraq is a moderated security list that covers vulnerabilities in everything from Windows through Apache to hardware routers. Hence there is quite a bit of traffic on the list, expect 10+ posts a day. The information on Bugtraq isn't always accurate or first-hand information and since it's a moderated list there is often a delay.

1.21 Apply Applicable Patches

Description

Obviously knowing about newly discovered vulnerabilities is only part of the solution; there needs to be a process in place where patches are tested and installed on the systems. These patches fix diverse problems, including security issues, and are created from both in-house testing and user-community feedback.

Possible update and patch steps listed below for selected platforms will ensure that your Apache software is up to date with the most current fixes available. If kernel updates are included, it will be necessary reboot the server, and when the apache software or other services are updated you should verify that services have been properly restarted after being updated.

Action
Follow the patch/update process for your organization to update your OS and Apache software at least once per month. Use the update mechanism that comes with your OS or compile from source as outlines in Section 1.1.

Fedora Core Manual Update:

```
# yum update
```

Fedora Core Automated Nightly Update:

```
# chkconfig yum on
# chkconfig --list yum
yum 0:off 1:off 2:on 3:on 4:on 5:on 6:off
```

Patch for Source Code:
If you built from Apache source code:

- Check the following web site for the latest patches for the version of Apache you downloaded: http://www.apache.org/dist/httpd/patches/.
- Look for a directory called “apply_to_2.2.XX/” which matches your source version.
- If this directory exists, read the description to verify if the patch is applicable to your OS and then download the file.
- Also verify the integrity of the patch by importing the Apache PGP keys and checking the PGP signature. The apache PGP key's are available on the same web site http://www.apache.org/dist/httpd/KEYS however it's best to retrieve them from a different source, such as the MIT PGP key servers, or any of the PGP key mirrors.

Place this file in the new Apache source build directory, which is created once you have unzipped and untarred the Apache archive in the section below. Once this patch file is in this directory, issue the following commands (this is an example and shows how to apply the patch called "no_zombies.patch". You may not need to apply any patches):

```
# pwd
/apache/build/directory
# gpg --verify httpd-2.2.XX.tar.gz.asc
# tar xzf httpd-2.2.XX.tar.gz
# cd httpd-2.2.XX
# patch -s < /apache/build/directory/name_of_path.patch
```

2 Level 2 Apache Benchmark Settings

Prudent Security Beyond the Minimum Level
Level 2 security configurations vary depending on network architecture and server function. These are of greatest value to system administrators who have sufficient security knowledge to apply them with consideration to the operating systems and applications running in their particular environments.

***Important** – Level II Benchmark settings also differ from Level I settings in that you do NOT have to apply all of the Level II settings to address each security issue. There are settings, which are an either/or option. The reader should review all Level 2 settings before determining which sections to implement. Level 2 settings expects the reader to have sufficient knowledge of the security issues involved to determine the best option for their environment.*

2.1 Chroot

Description
This part focuses on preventing Apache from being used as a point of break-in to the system hosting it. Apache by default runs as a non-root user, which will limit any damage to what can be done as a normal user with a local shell. Of course, allowing what amounts to an anonymous guest account falls rather short of the security requirements for most Apache servers, so an additional step can be taken - that is, running Apache in a chroot jail.

The main benefit of a chroot jail is that it will limit the portion of the file system the daemon can see to the root directory of the jail. Additionally, since the jail only needs to support Apache, the programs available in the jail can be extremely limited. Most importantly, there is no need for setuid-root programs, which can be used to gain root access and break out of the jail.

Pros and Cons of Chroot

- If apache is ever compromised, the attacker will not have access to the entire file system.
- Poorly written CGI scripts that may allow someone to access your server will not work.
- There are extra libraries you'll need to have in the chroot jail for Apache to work.
- If you use any Perl/CGI features with Apache, you will need to copy the needed binaries, Perl libraries and files to the appropriate spot within the chroot space. This includes `/bin/mail`, `/bin/ls`, etc…
- The same applies for SSL, PHP, LDAP, PostgresSQL and other third-party programs.
- Chroot usually requires extensive testing to ensure proper functionality/security are working properly.

ModSecurity's Chroot Approach[4]

Properly implementing and testing a traditional chroot setup for a dynamic web application can be quite challenging. Fortunately, ModSecurity includes support for Apache file system isolation, or chrooting. Chrooting is a process of confining an application into a special part of the file system, sometimes called a "jail". Once the chroot (short for “change root”) operation is performed, the application can no longer access what lies outside the jail. Only the root user can escape the jail (in most cases, there are some circumstances when even non-root users can escape too, but only on an improperly configured jail). A vital part of the chrooting process is not allowing anything root related (root processes or root SUID binaries) inside the jail. The idea is that if an attacker manages to break in through the web server he won't have much to do because he, too, will be in jail, with no means to escape.

Notes

What follows is a list of facts about the ModSecurity internal chroot functionality for you to consider:

1. Unlike external chrooting (mentioned previously) ModSecurity chrooting requires no additional files to exist in jail. The chroot call is made after web server initialization but before forking. Because of this, all shared libraries are already loaded, all web server modules are initialized, and log files are opened. You only need your data files in the jail.
2. To create new processes from within jail your either need to use statically-compiled binaries or place shared libraries in the jail too.
3. With Apache 2.x, the default value for the `AcceptMutex` directive is `pthread`. Sometimes this setting prevents Apache from working when the chroot functionality is used. Set `AcceptMutex` to any other setting to overcome this problem (e.g. `posixsem`). If you configure chroot to leave log files outside the jail, Apache will have file descriptors pointing to files outside the jail. The chroot mechanism was not initially designed for security and some people feel uneasy about this.
4. If your Apache installation uses `mod_ssl` you will find that it is not possible to leave the logs directory outside the jail when a file-based SSL mutex is used. This is because `mod_ssl` creates a lock file in the logs directory immediately upon startup, but fails when it cannot find it later. This problem can be avoided by using some other mutex type, for example `SSLMutex sem`, or by telling `mod_ssl` to place its file-based mutex in a directory that is inside the jail (using `SSLMutex file://path/to/file`).
5. If you are trying to use the chroot feature with a multithreaded Apache installation you may get the following message "libgcc_s.so.1 must be installed for pthread_cancel to work". Add `LoadFile /lib/libgcc_s.so.1` to your Apache configuration to fix this problem.
6. The files used by Apache for authentication must be inside the jail since these files are opened on every request.
7. Certain modules (e.g. `mod_fastcgi`, `mod_fcgi`, `mod_cgid`) fork in the module initialization phase. If they fork before chroot takes place they create a process that lives outside jail. In this case ModSecurity must be configured to initialize after most modules but before the modules that fork. This is a manual process

with Apache 1.3.x. It is an automated process with Apache 2.x since ModSecurity 1.9.3.

Action
Assuming Apache was installed into `/usr/local/apache2` and you want the jail will be placed at `/chroot/apache`, execute these commands:

```
# mkdir -p /chroot/apache/usr/local
# cd /usr/local
# mv apache2 /chroot/apache/usr/local
# ln -s /chroot/apache/usr/local/apache2
```

Now instruct ModSecurity to perform chroot upon startup by adding the following line to your ModSecurity configuration file:

```
SecChrootDir /chroot/apache
```

And start Apache:

```
/usr/local/apache2/bin/apachectl startssl
```

2.2 Mod_Log_Forensic

Description
What Apache child process segmentation faulted? This is a common question asked when reviewing error_log entries and seeing a message similar to this:

```
[Sun Apr 24 09:11:02 2005] [notice] child pid 5500 exit signal
Segmentation fault (11)
```

Generally speaking, a segmentation fault is not good; either there is a problem with the application code and it is exiting abnormally, or worse, someone is attempting to exploit your web server and causing it to crash. Either way, these types of messages need to be looked into. The biggest problem in tracking down these types of messages is associating the segfault error message with the actual client request that generated it. This is what `mod_log_forensic` aims to fix.[5]

Action
First, we must check that the `mod_log_forensic` and mod_unique_id DSO modules are enabled in the `httpd.conf` file:

```
# egrep 'log_forensic|unique_id' /usr/local/apache2/conf/httpd.conf
LoadModule log_forensic_module modules/mod_log_forensic.so
LoadModule unique_id_module modules/mod_unique_id.so
```

ForensicLog Directive
There is only one directive for this module due to its focused goal. The directive is the output log file. This directive tells Apache where to log the output of the

mod_log_forensic data. The file can either be a regular file or the output can be sent to a program. Here is the basic directive entry:

```
ForensicLog /usr/local/apache2/logs/forensic.log
```

The concept of this module is pretty simple; mod_log_forensic will generate a log file containing two entries for all requests. The first entry is the client request, which is prepended with a unique id number and a plus (+) sign. The second entry is the corresponding server response entry after successfully servicing the request, which is identified by a negative (-) sign and also has the same unique id number. Here is an example log entry of a successful request/response transaction:

```
# tail -2 /usr/local/apache2/logs/forensic.log
+cDkrlsCoAWUAAC4xChEAAAAA|GET / HTTP/1.1|Accept:image/gif, image/x-
xbitmap,
image/jpeg, image/pjpeg,application/vnd.ms-powerpoint,
application/vnd.ms-excel,
application/msword, application/x-shockwave-flash, */*|Accept-
Language:en-us|Accept-
Encoding:gzip, deflate|User-Agent:Mozilla/4.0 (compatible; MSIE 5.5;
Windows NT
5.0)|Host:192.168.1.101|Connection:Keep-Alive
-cDkrlsCoAWUAAC4xChEAAAAA
```

If a request has segfaulted, then the corresponding negative entry from the server response will be missing. The Apache source code actually comes with a shell script called, appropriately enough, check_forensic, which will help to automate the process of parsing the error_log file and identifying any processes that segfaulted. Here is an example of running the tool:

```
# /tools/httpd-2.0.52/support/check_forensic
/usr/local/apache2/logs/forensic.log
+Ll@PbH8AAAEAAFYcFXkAAAAE|GET / HTTP/1.1|Accept:*\*|Accept-Language:en-
us|Accept-
Encoding:gzip, deflate|If-Modified-Since:Sat, 19 Feb 2005 16%3a06%3a07
GMT;
length=1833|User-Agent:Mozilla/4.0 (compatible; MSIE 6.0; Windows NT
5.1; SV1; .NET
CLR 1.1.4322)|Host:192.168.26.134|Connection:Keep-Alive|Transfer-
Encoding:Chunked
+NKqZ6X8AAAEAAFYhFuwAAAAF|GET / HTTP/1.1|Accept:*/*|Accept-Language:en-
us|Accept-
Encoding:gzip, deflate|If-Modified-Since:Sat, 19 Feb 2005 16%3a06%3a07
GMT;
length=1833|User-Agent:Mozilla/4.0 (compatible; MSIE 6.0; Windows NT
5.1; SV1; .NET
CLR 1.1.4322)|Host:192.168.26.134|Connection:Keep-Alive|Transfer-
Encoding:Chunked
```

This output shows that two requests exited abnormally. By looking at the client headers, it seems that this client may be attempting to exploit the Chunked-Encoding vulnerability

identified in earlier versions of Apache (www.cert.org/advisories/CA-2002-17.html). This is indicated by the use of the Transfer-Encoding: Chunked client request header.

2.3 Denial of Service and Brute Force Identification and Response

Description

In the Level 1 section, we highlighted some basic tuning that could be performed to help Apache child process performance. In this section, we will present a method to specifically help identify Denial of Service and Brute Force attacks. There are two main issues to deal with when considering these types of attacks at the HTTP layer:

- Any mechanism used must be able to be stateful so that it can track and correlate from request to request, and
- It must be able to implement protections at lower network levels to truly help mitigate the effects.

The multi-process nature of the Apache prefork execution model is usually an advantage (one process can go down while leaving the remaining processes unaffected; the controlling parent process simply creates another child). But there is a drawback: it is very difficult to share information between processes. Without information sharing, it is impossible to detect several classes of attack (for example, denial-of-service attacks).[6]

Note
The previous versions of the Apache benchmark recommended the use of the mod_evasive tool (http://www.zdziarski.com/projects/mod_evasive/) and it works relatively well for small to medium sized brute force or HTTP level DoS attacks. There is, however, an important limitation. The mod_evasive module is not as good as it could be because it does not use shared memory in Apache to keep information about previous requests persistent. Instead, the information is kept with each child process or thread. Other Apache children that are then spawned know nothing about abuse against one of them. When a child serves the maximum number of requests and dies, the DoS information goes with it. So, what does this mean? This means that if an attacker sends their HTTP DoS requests and they do not use HTTP Keep-Alives, then Apache will spawn a new child process for every request and it will never trigger the mod_evasive thresholds. This is not good…

Fortunately, you can use the piped logging mechanism, already well-supported in Apache, to export the relevant information out of ModSecurity to an external process that shared among all Apache processes. The ModSecurity facility is named the Guardian log. The Guardian log mechanism is coupled together with a perl script tool called httpd-guardian, whose goal is to defend against denial-of-service attacks.

Action

Download the Apache Security tools archive from the Apachesecurity.net website - http://www.apachesecurity.net/tools/index.html. Edit the httpd-guardian file to point to the proper perl binary on your system. Also make sure that the script has execute permissions. Then add the following directive to your ModSecurity configuration file:

```
SecGuardianLog "|/path/to/httpd-guardian"
```

The next time you start (or restart) Apache you will find the httpd-guardian process running alongside it. This process will monitor the requests coming, watching for DoS attacks. The script keeps two counters for every IP address it sees: one counter for the most recent one-minute interval and another for the most recent five-minute interval. If an IP address sends too many requests in a measured period, httpd-guardian will take an action against it. By default, the action is a mere warning, but you can configure httpd-guardian to talk to your firewall to add the offending IP address to the blacklist (using iptables, pf, or SnortSam).

If you have to restart Apache on regular basis, you will be happy to know that httpd-guardian saves the counter values to a file on regular basis and reloads them the next time Apache starts.

Note
mod_qos is another module that be very helpful for identifying and responding to these types of attacks - http://mod-qos.sourceforge.net/.

2.4 Buffer Overflow Protection Tuning

Description
In the Level 1 section, we highlighted some basic Apache directives that may help to prevent some Buffer Overflow attacks. This will certainly help with placing adequate restrictions on the size of these portions of the client's request; however, these LimitRequest directives listed previously are a bit too broad to handle individual buffer overflow vulnerabilities in application parameters. We can, however, leverage ModSecurity's granularity capabilities to place proper restrictions on specific application parameters.

In Level 2, we will now present a few more protection mechanisms that ModSecurity can provide. Often, a Buffer Overflow attack will include random binary data in order to fill up the buffer and then to execute the desired shell code. ModSecurity has a few different operators that will help to identify and prevent this data from executing.

It is important to note, however, that while these settings will provide protection, they will need to be tested extensively and may also need to be re-evaluated when the application it is protecting is changed.

Action
The ModSecurity Core Rules file modsecurity_crs_20_protocol_violations.conf file has numerous rules that handle possible buffer overflow attacks.

The following two rules use the `@validateUrlEncoding` and `@validateUtf8Encoding` operators to validate the encodings used in requests:

```
# Check decodings
```

```
SecRule
REQUEST_FILENAME|ARGS|ARGS_NAMES|REQUEST_HEADERS|!REQUEST_HEADERS:Refer
er "@validateUrlEncoding" \
      "chain, deny,log,auditlog,status:400,msg:'URL Encoding Abuse
Attack Attempt',,id:'950107',severity:'4'"
SecRule
REQUEST_FILENAME|ARGS|ARGS_NAMES|REQUEST_HEADERS|!REQUEST_HEADERS:Refer
er "\%(?![0-9a-fA-F]{2}|u[0-9a-fA-F]{4})"

SecRule
REQUEST_FILENAME|ARGS|ARGS_NAMES|REQUEST_HEADERS|!REQUEST_HEADERS:Refer
er "@validateUtf8Encoding" "deny,log,auditlog,status:400,msg:'UTF8
Encoding Abuse Attack Attempt',,id:'950801',severity:'4'"
```

The following two rules use the @validateByteRange operator to restrict the characters accepted. By default, it only disallows the null byte character as it is often used in evasions. You can optionally restrict this down to 32-126 to only allow printable ASCII characters.

```
#
# Restrict type of characters sent
#
# NOTE In order to be broad and support localized applications this
# rule only validates that NULL Is not used.
#
#        The strict policy version also validates that protocol and
#        application generated fields are limited to printable ASCII.
#
# TODO If your application use the range 32-126 for parameters.
#
SecRule
REQUEST_FILENAME|REQUEST_HEADERS_NAMES|REQUEST_HEADERS|!REQUEST_HEADERS
:Referer \
      "@validateByteRange 1-255" \
      "deny,log,auditlog,status:400,msg:'Invalid character in
request',,id:'960018',severity:'4',t:urlDecodeUni,phase:1"

SecRule ARGS|ARGS_NAMES|REQUEST_HEADERS:Referer "@validateByteRange 1-
255" \
      "deny,log,auditlog,status:400,msg:'Invalid character in
request',,id:'960901',severity:'4',t:urlDecodeUni,phase:2"
```

Note
These examples use Regular Expressions to create rules for size restrictions. In the ModSecurity 2.5.0 development release, there is a new transformation function called "t:length" that provides better performance over using RegExs.

2.5 Syslog Logging

Description
Apache has the capability to send its error log output to the local Syslog daemon process by setting the `ErrorLog` directive to "`ErrorLog syslog`". This directive specifies that all of the error messages get sent through the syslog facility. We want to ensure the integrity of your error logs in case the web server ever gets compromised. After a compromise, the

log files on the host cannot be trusted, since the attacker could have easily altered them. In the steps below, we have configured the syslog facility to log all of the errors locally to the normal `/usr/local/apache2/logs/error_log` file, as well as, log remotely to the syslog facility on a secure, remote host. This ensures that the error logs have not been altered if we are investigating a break-in.

Note
Using syslog to centrally log Apache messages is also useful when a SIM application is used to correlate messages from across the enterprise. This data can be critical for 24x7 SOC Incident Response teams.

Action
Edit the httpd.conf file and update the ErrorLog directive:

```
#
# ErrorLog: The location of the error log file. If you do
# not specify an ErrorLog directive within a <VirtualHost>
# container, error messages relating to that virtual host
# will be logged here. If you *do* define an error
# logfile for a <VirtualHost> container, that host's
# errors will be logged there and not here.
#
ErrorLog syslog:local7
```

This configuration will send all error log entries to syslog and be logged with the "local7" facility. You can specify other syslog facility levels if you wish to segregate your Apache syslog data from your other system messages. Keep these issues in mind when using syslog for Apache logging:

- The syslog daemon will not create files. If you are logging to a file (as specified in the `syslog.conf` configuration file) that file must already exist and have permissions that allow the syslog daemon to write to it.
- You must restart the syslog daemon for it to recognize changes to its `syslog.conf` configuration file.
- The syslog daemon must be active prior to starting Apache.
- Apache will default the facility to "`local7`" if you omit the facility name from the `ErrorLog` directive (that is "`ErrorLog syslog`").
- The syslog facility name must be one that is recognized by both Apache and the `syslog.h` header file. The facility names "`local0`" through "`local7`" are explicitly set aside for your use.
- Although "`local0`" through "`local7`" are recommended user facility names, here is the complete list of names recognized by both Apache and TPF's `syslog.h`: `auth`, `cron`, `daemon`, `kern`, `local0`, `local1`, `local2`, `local3`, `local4`, `local5`, `local6`, `local7`, `lpr`, `mail`, `news`, `syslog`, `user`, and `uucp`.
- You won't see the normal Apache startup/shutdown messages when you use syslog with your Apache error log.

Verify that the Apache error messages are being sent to syslog.

```
# apachectl start
```

```
# tail -2 /var/log/messages
Aug 8 15:39:18 netwk8 httpd[3345]: [notice] Digest: done
Aug 8 15:39:19 netwk8 httpd[3345]: [notice] Apache configured --
resuming normal operations
```

Edit the `/etc/syslog.conf` file to log locally and to add in the ability to send the Apache error data off to a remote syslog host.

```
# vi /etc/syslog.conf
# grep local7 /etc/syslog.conf
Local7.* /usr/local/apache2/logs/error_log
Local7.* @IP_of_Remote_Host
# killall -v -HUP syslogd
```

Verify that the syslog data is being captured correctly on the remote syslog host.

```
# hostname
remote_sysloghost
# echo "SYSLOGD_OPTIONS="-m 0 -r" >> /etc/sysconfig/syslog
# service syslog restart
#
# tail -f /var/log/messages
Aug 8 15:44:19 netwk8 httpd[3417]: [error] [client 10.1.2.16] File does
not exist: /var/www/html/foo
```

2.6 Virtual Patching with ModSecurity

Description

Fixing identified vulnerabilities in Apache or web applications always requires time. Organizations often do not have access to a commercial application's source code and are at the vendor's mercy while waiting for a patch. Even if they have access to the code, implementing a patch in development takes time. This leaves a window of opportunity for the attacker to exploit. Virtual patching is one of the biggest advantages of using ModSecurity a custom ruleset can be created to quickly fix this problem externally. A fix for a specific vulnerability is usually very easy to design and in most cases it can be done in less than 15 minutes.

Imagine that you have received the following vulnerability alert email:

```
Name: Oracle iSQL*Plus buffer overflow
Systems: Oracle Database 9i R1,2 on all operating systems
Severity: High Risk
Vendor URL: http://www.oracle.com/
Author: David Litchfield (david@ngssoftware.com)
Advisory URL: http://www.ngssoftware.com/advisories/ora-isqlplus.txt

Details
*******
The iSQL*Plus web application requires users to log in. After accessing
the default url, "/isqlplus," a user is presented with a login screen.
By sending the web server an overly long user ID parameter, an internal
buffer is overflowed on the stack and the saved return address is
```

overwritten. This can allow an attacker to run arbitrary code in the security context of the web server. On most systems, this will be the "oracle" user and on Windows the "SYSTEM" user. Once the web server has been compromised, attackers may then use it as a staging platform to launch attacks against the database server itself.

Action

We will now outline the steps needed to conduct analysis on the vulnerability and create a custom virtual patch with ModSecurity.

- *Verify Software Version*
  The first step would be to determine if the version of software that you are running is vulnerable. You can verify your Apache (in this case, it would be the Oracle Application Server) version by executing the httpd binary with the appropriate flag.

```
# ./httpd -V
Server version: Oracle-Application-Server-10g/9.0.4.1.0 Oracle-
HTTP-Server
Server built:   Dec  2 2004 18:39:07
Server's Module Magic Number: 19990320:15
Server compiled with....
--CUT--
```

  Let's assume that the version you are running matches one listed within the vulnerability advisory. Now that you have confirmed that your web server is vulnerable, you must now figure out if the patch is available.

- *Check Patch Availability*
  Normally, the vulnerability alert notice will provide a patch status. If a patch is available, then it will have downloading instructions. More often than not, the alert emails will refer you to the specific vendor's web site. Even if a patch is available, you still have a problem. What is your organization's patch management process? How quickly are you able to follow your policy and procedure for the patch engineering and test phases? Remember, you are racing against the clock to apply these patches on your systems. Sometimes, patches are not yet available from the vendor. This is most often the case when a new 0-day vulnerability is made public. In this case, you are in a similar predicament as the time period before you successfully apply a patch. What can you do in the meantime? Luckily, we have the capability to create some custom security filters with ModSecurity to protect our Oracle application from this attack.

- *Identify Vulnerability Details*
  The following two websites usually provide vulnerability and exploit details:
  - ***SecurityFocus Web site***
    SecurityFocus' vulnerability database (www.securityfocus.com/vulnerabilities) is an excellent resource for identifying vulnerability details such as exploit information and mitigation

strategies. If exploit code has been released, odds are that it will be listed on the SecurityFocus web site.

- ***Snort Web site***
  The combination of Snort user contribution and the folks at Sourcefire (the commercial face of Snort development) are incredibly responsive to generating attack signatures for newly released vulnerabilities. It is not uncommon to have some new Snort signatures available for download within hours of a vulnerability announcement. This allows us to let the experts, who get paid to analyze these vulnerabilities, develop signatures. Keep in mind that these signatures were created for a Network IDS and may not be directly translatable to ModSecurity. Some amount of analysis will be required to use the signature details.

  The actual Snort signature is shown in the following:

```
alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS $HTTP_PORTS
(msg:"WEB-MISC Oracle
iSQLPlus login.uix username overflow attempt";
flow:to_server,established;
uricontent:"/login.uix"; nocase;
pcre:"/username=[^&\x3b\r\n]{250}/smi";
reference:bugtraq,10871;
reference:url,www.nextgenss.com/advisories/ora-
isqlplus.txt;
classtype:web-application-attack; sid:2703; rev:1;)
```

  The bolded information is relevant to create a ModSecurity virtual patch. This data will help us to create a comparable ModSecurity filter to block this vulnerability. We are interested in both the `uricontent` and `pcre` keyword values. The `uricontent` keyword specifies that this rule should match if the text string `/login.uix` exists in the URI field of the request. This prevents Snort from attempting to match the text string in other locations of the request, such as in one of the other request headers. The `pcre` keyword stands for PERL Compatible Regular Expression. If we interpret the PCRE value specified in the signature into plain English, it says to look for a text string that starts with `username=`. If this is found, then look at the data that follows. Only match if the following character is not either an `&` (ampersand), `\x3b` (semicolon represented in byte form), `\r` (return), or `\n` (newline) and there 250 or more characters.

- *Creating a Negative Security Virtual Patch*
  With the PCRE data taken from the Snort signature, we can now create a custom filter to identify and block a request that is attempting to exploit this vulnerability. The first part that we need to define is the URI location.

  We can use the Apache Location directive to define a context in which to place our ModSecurity filters. Here is an example:

```
<Location /isqlplus/login.uix>
</Location>
```

The next step is to create a ModSecurity filter to identify the username variable overflow. We already have the regular expression from the Snort signature, so we only need to choose which ModSecurity filter location to use. The example login data showed that the username data is sent within the POST payload. We should therefore use the SecRule directive and specify the username argument as the location. Here is the updated signature inside the location directive:

```
<Location /isqlplus/login.uix>
SecRule ARGS:username "^[^&\x3b\r\n]{250}$" phase:2,msg:'Oracle
iSQLPlus login.uix username overflow attempt'"
</Location>
```

- *Creating a Positive Security Virtual Patch*
  While the previous example would be able to block attempts to exploit that specific vulnerability, a more comprehensive security approach would be to implement a virtual patch that enforces a positive security policy for this particular portion of the application. The following ruleset also enforces rules to only allow the POST request method, ensures that there is only one parameter named "username" (as to avoid possible evasions) and finally it enforces a more strict character set for the username parameter value only allowing alpha-numeric characters between zero and thirty-two in length.

```
<Location /isqlplus/login.uix>
SecRule REQUEST_METHOD "!^post$"
SecRule &ARGS:username "!@eq 1"
SecRule ARGS:username "!^\w{0,32}$"
</Location>
```

For more information on Virtual Patching with ModSecurity, please refer to the archived webcast on this topic on the Breach website - http://www.breach.com/resources/webinars.html.

2.7 Additional Software Information Leakage Protection

Description

The following steps will further reduce an attacker's ability to identify the target web server as Apache. If you want to change the Server token data entirely, you can do this with the ModSecurity `SecServerSignature` directive. With this directive, you can alter the data returned by the "Server:" HTTP response header. This option allows you to change this data from within the httpd.conf file, without the need to edit source code and recompile. Apache modules are not allowed to change the name of the server completely, but ModSecurity works by finding where the name is kept in memory and overwriting the text directly. The `ServerTokens` directive must therefore be set to `Full` to ensure the web server allocates a large enough space for the name, giving ModSecurity enough space to make its changes later.

We will be verifying/updating the following Apache directives:

- ***SecServerSignature***
  ModSecurity introduces the `SecServerSignature` directive which allows an administrator to alter the `ServerSignature` to an arbitrary value. This functionality can be used to mislead an attacker by advertising a non-Apache service banner, such as `Microsoft-IIS/6.0`.

- ***ErrorDocument (Custom Error Pages)***
  Each type of web server has its own distinct style of error pages. The server sends these pages when an error, such as "404 - Not found," has occurred. By issuing a request for a file that is not present on a web server, an attacker may determine the web server software by simply identifying the 404 – Not Found error pages displayed. To avoid this software disclosure, the default error pages presented by the web server must be changed. There are two possible choices:

  - Edit the default error pages to a style that is consistent with the website's template. This may include changing color schemes and altering the text message displayed.
  - For deception purposes, edit the error pages to the exact style of a different web server. For example, if a web server is currently running Apache, change the error pages to resemble a different web server version.

Action
Apache Directive
Edit/Verify the following directives in the `httpd.conf` file:

```
ServerTokens Full
SecServerSignature "Microsoft-IIS/6.0”
ErrorDocument 400 /custom400.html
ErrorDocument 401 /custom401.html
ErrorDocument 403 /custom403.html
ErrorDocument 404 /custom404.html
ErrorDocument 405 /custom405.html
ErrorDocument 500 /custom500.html
-- CUT --
```

Note – the ModSecurity Core Rules `modsecurity_crs_10_config.conf` file already has a `SecServerSignature` directive that can be used.

3 References

1. Ristic, Ivan. "Apache Security" Oreilly, ISBN: 0596007248, Published: March 2005 - http://www.apachesecurity.net/index.html
2. Cox, Mark J. "Apache Security Secrets: Revealed" ApacheCon 2002, Las Vegas - http://www.cgisecurity.com/webservers/apache/tu04-handout.pdf
3. Web Application Security Consortium (WASC) Web Security Threat Classification - http://www.webappsec.org/projects/threat/
4. ModSecurity Reference Manual v 1.9.3 - http://www.modsecurity.org/documentation/modsecurity-apache/1.9.3/modsecurity-manual.html
5. Barnett, Ryan C. "Preventing Web Attacks with Apache" Addison-Wesley Publishing, ISBN: 0321321286, Published January 2006 - http://www.amazon.com/Preventing-Attacks-Apache-Ryan-Barnett/dp/0321321286
6. Ristic, Ivan. "What's new in ModSecurity?" SecurityFocus Article, January 12, 2005 - http://www.onlamp.com/lpt/a/6350

4 Acknowledgements

The following people were instrumental in the development of this guide:

- Ryan Barnett
- George Jones
- Art Stricek
- Jim Jagielski
- Hal Pomeranz
- Brian Eppinger
- Jeremiah Grossman
- Christian Folini
- Chris Calabrese
- Ralf Durkee
- Kevin Binsfield
- Ivan Ristic
- Glenn Brunette
- Jack Simons
- David A. Kennel
- Blake Frantz
- John Banghart

5 Change History

Date	Version	Changes for this version
October, 2008	2.2.0	• Updated 1.19 to refer to /usr/local/apache2 instead of /usr/local/apache. • Updated 2.1 to refer to /usr/local/apache2 instead of /usr/local/apache. • Updated 2.2 to refer to /usr/local/apache2 instead of /usr/local/apache. • Updated 2.5 to refer to /usr/local/apache2 instead of /usr/local/apache. • Formatted inline 'verbatim' text for readability. • Added new cover page, Change History, and

	TOU. • Added footer with page numbers • Updated 1.14 example to match explanation text. • Updated 1.17 example to reference /var/log/httpd for logging. This aligns with the recommendation to log off the server root. • Updated 1.21 to state users should use the update mechanism provided by their OS to update Apache. • Updated 1.15 ssl config to reference /var/log/httpd for logging location vie logs/. • Updated 1.15 to disallow SSLv2 Ciphers • Added Acknowledgement section • Moved the SecServerSignature and ErrorDocument recommendations within 1.16 to new level 2 recommendation – 2.7.