SECURING A WEB SERVER

By: Chris Huey (hueyise.com)

Publication Date: 20 July 2013

Updated: 12 March 2017

This article provides <u>general</u> guidelines for securing a web server. The guidelines pertain to web servers running Linux and Apache, but the principles can be applied to any web server.

https://www.hueyise.com/index.php/web-server-security-advice

ModSecurity

ModSecurity is an extremely important component for securing your web server. *ModSecurity* is a module integrated within the HTTP service that acts as a web application layer firewall to filter common attacks such as those found in the OWASP Top Ten list. I used advice from Tecmint.com for implementing *ModSecurity*. The OWASP ModSecurity Core Rule Set should be installed as the filtering rules for *ModSecurity*, and then tailored as necessary. The core rule set can be downloaded using this link: OWASP.org. Below are a couple of tips I compiled from my experience using *ModSecurity*:

Tip: SecRuleEngine. Within the *conf.d/modsecurity.conf* file, the default setting for *SecRuleEngine* is *DetectionOnly*. The *DetectionOnly* setting instructs *ModSecurity* to process the rules but not deny transactions. This causes transactions that match a *ModSecurity* rule to pass (even malicious transactions) and a log message for the action is recorded in the web server error log. I recommend initially using the default setting while actively monitoring the *Apache error_log* for "false-positive" traffic filtering. The modsecurity.org site provides good information for handling "false positives". Once you are satisfied the *ModSecurity* rules are filtering appropriately, you should set *SecRuleEngine* to *On*. This enables *ModSecurity* to process transactions according to the installed rules (i.e., deny potentially malicious transactions).

Tip: Whitelisting. Web-based administrative functions can trigger rules to be fired resulting in the transaction being blocked. For example, a Web Content Management System (CMS) may interface with a database for administrative back-end functionality. These actions will likely trigger *ModSecurity* SQL Injection rules. I strongly caution you about disabling these types of rules since your web server needs to protect against legitimate SQL Injection attacks and the multitude of other web-based attacks. An effective option to consider is using ModSecurity Actions to whitelist specific IP addresses (preferably within your administrative subnet) to allow administrative functions to be performed without being filtered by *ModSecurity*. This link contains a very good example of how to whitelist IPs: blog.modsecurity.org.

Host-Based Firewall

Even if a perimeter firewall is deployed, you should enable a host-based firewall to implement custom, fine-grained network filtering to enforce your web server's unique security policy. I strongly recommend using *iptables*, a Linux host-based firewall that can be tailored to ensure only authorized interconnections are established with your web server. The cyberciti.biz site provides a good overview on *iptables* and includes general instructions on how to correctly implement *iptables*.

If your network perimeter protection does not include an IDS/IPS, you may want to augment *iptables* with *fwsnort*. *Fwsnort* is a Freeware product that implements an application layer IDS/IPS by integrating rules into *iptables*. *Fwsnort* uses rules from the *SNORT®* IDS/IPS and builds an equivalent set of *iptables* rules for as many rules as technically possible. Implementing *fwsnort* involves running a script that places *SNORT* rules into the system's *iptables* configuration. *Fwsnort* spliced 10,097 rules into my *iptables* policy.

Filter Visitor Access

Some countries have a dubious reputation for performing malicious activities on the Internet. I implemented a custom script to download sets of IP addresses associated with particular countries from the ipdeny.com web site and developed a script to dynamically block IP addresses originating from certain countries. I implemented *ipset* within *iptables* to load and block source IP addresses originating from certain countries using the technique described in hueyise.com. You may also consider a more traditional approach to blocking countries by implementing ConfigServer Security & Firewall (CSF). Note however that none of these approaches are full-proof since attackers can use compromised systems or proxy services from "friendly" sources.

By default, website CMS administration logon pages are accessible to the general public. If left unprotected, hackers will frequently seek administrator-level access to your website by performing brute-force attacks using your administration logon

Configure your web server to only allow access to website administration pages from domains you control (such as an out-of-band management subnetwork). Below provides an example of how you can control website administration access using Apache Rewrite directives.

httpd.conf

```
Listen 80
<VirtualHost *:80>
            :
      RewriteEngine on
      RewriteCond %{REMOTE_ADDR} "!^192.168.*"
      RewriteCond %{REQUEST_URI} "/wp-login.php [OR]
      RewriteCond %{REQUEST_URI} "/private.php"
      RewriteRule "(.*)" "-" [R=404,L]
</Virtualhost>
```

Security Best Practices

There are a multitude of industry-standard security best practice guidelines available to assist you with securing your servers. These guidelines are developed primarily by teams of individuals who are experts in implementing and securing a particular technical component, like a web server application. It is highly advised to use available security guidelines to help safeguard your web server against attacks. I prefer using the Center for Internet Security (CIS) benchmarks for two important reasons:

(1) The CIS benchmarks are typically up-to-date with the latest web server application versions; and

(2) The CIS benchmarks provide very good rationale explaining the benefit of each configuration setting, steps to verify compliance, and instructions on how to remediate a discrepancy.

Other security configuration guidelines are provided by DISA, SANS and NIST, to name a few others. You should choose one guideline for each server component (e.g., one for the OS and one for the web server application) and try to comply with as many configuration settings as possible. But realize 100% compliance with the guidelines is not practical since it will likely result in your web server not operating as intended. You will need to pick-and-choose the security configuration settings that make the most sense while considering the risk if you opt to disregard a particular security setting.

As you systematically secure your web server application using a particular security guideline, be sure to make frequent backups and conduct tests after you apply a couple of settings to ensure your web server functions properly; this will allow you to narrow down the culprit if your web server fails.

Enable SELinux

Security-Enhanced Linux (SELinux) is now integrated by default into the kernel of most Linux operating systems. *SELinux* enforces access control policies by confining programs and services to the minimum privileges needed to operate securely. Traditional Unix systems relied on file permissions and access control lists to restrict access to files and directories by users and processes. *SELinux* provides a more rigorous access control enforcement by "denying" access to "all" unless a specific policy exists to "allow" access by a particular "subject" (e.g., user, process). Refer to the CentOS SELinux How-To page or the Fedora SELinux User Guide for some additional details about *SELinux*.

At a minimum, I suggest you implement *SELinux* on your web server as follows:

(1) The default *SELinux* policy installed on Linux is the *targeted* policy. This policy *confines* many of the popular system processes, including *httpd*. All of the other system processes and programs execute in an *unconfined* domain, which means *SELinux* is not protecting those resources. You must ensure *SELinux* is configured to be *Enforcing* after bootup. The *SELinux targeted* policy should provide you with a high-degree of assurance since it includes enforcement controls relative to protecting a standard *httpd* environment. Employing a *strict* policy is far superior to the *targeted* policy in that *unconfined* domains do not exist (so all system resources are controlled by *SELinux*). However, the effort to correctly implement a pure *strict* policy is enormous for individuals trying to secure their personal web server.

(2) You will need to tune the *SELinux* policies based on your unique web server implementation. For example, by default *SELinux* will not allow a web cam to periodically upload snapshots into a directory that is accessible by the web server application. If you desire this functionality, then you must add a policy to *SELinux* to allow the access. Whenever *SELinux* denies access to a resource, an Access Vector Cache (AVC) message is logged to the */var/log/audit/audit.log* file. You should review AVC log messages regularly to determine whether policy violation attempts are resulting from an attack on your system or the *SELinux* policy needs to be tuned to allow accesses.

The Fedora Searching For and Viewing Denials page is a good starting-point for understanding how to view AVC log messages. The RedHat Allowing Access: audit2allow page provides some important details about how to tune your *SELinux* policies to meet your needs. I used the *settroubleshoot* package addressed in the *Fedora Searching For and Viewing Denials* page to monitor *SELinux* access "denials". The nice thing about *settroubleshoot* is that it allows you to assess the access attempt, and if you decide the access should be allowed you can use the commands contained within the AVC details message to update *SELinux* with the required "grant" policy.

Monitor the Audit Logs

It is extremely important to frequently monitor the log files for potential malicious activity. Below is my **Top-5 Audit Review Criteria** tailored specifically for monitoring the security of a web server. I also included tips for using automation to assist with effectively complying with the criteria.

Criterion #1: Review logs for successful exploits.

If a hacker gains a shell on your server by exploiting the *Apache* web server application, shell commands will be executed as user *apache*. Once a shell is gained, the next step for the hacker will likely involve attempts to escalate privileges. The g0tmi1k.com site provides techniques hackers use to discover ways to escalate privileges. You can use the *g0tmi1k.com* link to identify commands and files that indicate a hacker gained shell access and is working to escalate privileges. You should tailor your audit configuration to log *apache* accesses to these commands and files. Your audit review should focus on identifying suspicious *apache* system activity.

The easiest way to address Criterion #1 is: (1) tailor your *auditd* rules to log specific commands and accesses performed by the *apache* user; and (2) write a simple *bash* script to identify audits for *apache* restricted command execution and file accesses. Below is a snippet of my */etc/audit/audit.rules* file that configures *auditd* to generate a log message when the *apache* user (uid=48) performs suspicious activities.

```
<snippet>
# Commands indicating apache shell
-w /usr/bin/ps -F uid=48
-w /usr/bin/cd -F uid=48
-w /usr/bin/ls -F uid=48
-w /etc/passwd -F uid=48
-w /usr/bin/uname -F uid=48
-w /usr/bin/netstat -F uid=48
<more>
```

Next, you can use *"grep auid=48 /var/log/audit/audit.log"* OR "*ausearch --uid 48"* to locate log messages that indicate hacker activity on your server. I scheduled a *cron* job to execute a bash script which simply runs the *grep* command and emails an alert to me when any log messages are returned.

Another recommendation I have is to implement a product called syslog-ng Open Source Edition™ to dynamically monitor and manage your log data. I customized *syslog-ng* on my web server to collect and filter log messages from a variety of different audit files and funnel security-critical log messages into a single file. I then manually analyze and review the logs contained within the centralized audit file and I use an automated custom process to report various metrics (such as *htaccess* denials, failed logon attempts, etc.).

Criterion #2: Review logs to determine if changes were made to static web content and critical OS configuration files. Hackers frequently modify web content and OS configuration files to create backdoors into your system.

I recommend implementing Advanced Intrusion Detection Environment (AIDE), a Freeware product for Linux (and other UNIX operating systems) which verifies the integrity of the file system. You should configure *AIDE* to monitor all static web content files, important *httpd* files (e.g., *httpd.conf*, *modsecurity.conf*) and critical OS configuration files (e.g., config files in */etc*, executables in */usr/sbin*). You should schedule *AIDE* to verify the integrity of your file system regularly and monitor the *AIDE* log file for integrity discrepancies (using *syslog-ng*).

Criterion #3: Review the error_log for *ModSecurity* "access denied" messages.

When *ModSecurity* is properly tuned, "access denied" messages signify an attack was attempted. The *syslog-ng* utility can be used to monitor the *Apache* error_log for *ModSecurity* access denial messages. You should consider blocking subnets or ip addresses that frequently attempt attacks on your server. I developed a custom script to identify *ModSecurity* access denials with a severity above NOTICE, parse out the source IP address of attacking systems, and then automatically add the IP addresses to the *.htaccess* file so they cannot used to attack my web server in the future.

Criterion #4: Monitor the *iptables* logs, which are normally contained in */var/log/messages*, for INPUT and OUTPUT filter messages. This is a tricky review process since normal network activity frequently triggers *iptables* "deny" rules (e.g., blocking noisy Windows traffic on your subnet). You will need to recognize obvious suspicious network activity such as filtered outbound port 4444 connection attempts which could indicate a hacker is using

Metasploit to establish a reverse shell. I suggest tailoring *syslog-ng* to direct important *iptables* log messages into a centralized audit file while filtering out the benign *iptables* messages.

Criterion #5: Review the *Apache* access log for tell-tail signs that a hacker has been performing enumeration activities on your web server. For example, one day I noticed a large volume of *HEAD requests* in my Apache access log. After some investigation, I discovered a hacker was using a tool like *BlindElephant.py* (within Kali Linux) to determine if my web server was running a particular Content Management System based on the existence of certain themes and plug-in pages. I decided to develop a custom script which monitors the access log for 404 return messages (Not Found) and block IP addresses using a custom utility described in hueyise.com.

Mask Server Details

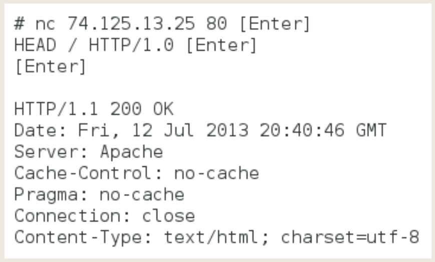
Hackers perform scanning and enumeration techniques to discover information about potential target systems. They use information gathered during these activities to establish possible attack vectors for gaining access to vulnerable systems. If a hacker learns that your web server uses a particular version of ExpressionEngine, then the hacker can identify known vulnerabilities with *ExpressionEngine* to mount an attack. Therefore, the less a hacker learns about your web server the lower the risk of a successful attack. Below are three things you should do to hide details about your web server. This will not fully protect your web server from exploitation of existing vulnerabilities, but it will make the hackers job much more difficult.

Tip #1: Ensure the following directives exist within the Apache configuration file:

- *ServerTokens ProductOnly*
- *ServerSignature Off*

The *ServerTokens ProductOnly* returns *Server:Apache* within the response header without additional details. In comparison, *ServerTokens Full* (which is default) returns something like: *Server: apache/2.0.41 (Unix) PHP/4.2.2 MyMod/1.2.*

The *ServerSignature Off* directive suppresses the trailing footer line (which contains information like the server version number) within generated documents. Use *netcat* to verify minimal server details are returned in the HTTP header (see example below):



```
# nc 74.125.13.25 80 [Enter]
HEAD / HTTP/1.0 [Enter]
[Enter]

HTTP/1.1 200 OK
Date: Fri, 12 Jul 2013 20:40:46 GMT
Server: Apache
Cache-Control: no-cache
Pragma: no-cache
Connection: close
Content-Type: text/html; charset=utf-8
```

Tip #2: Suppress the *PHP banner X-Powered-By: PHP/X.X.X-X...* by ensuring the *php.ini* file contains: *expose-php = off*. Certain PHP versions have known vulnerabilities which could lead to a successful attack on your web server. Therefore, suppressing the PHP banner within the HTTP header is a good idea. You can verify the banner is suppressed using the same *netcat* command described in #1 above.

Tip #3: Remove details from HTML page sources. Use a web browser to view the page source for each HTML page on your web server. If you are using *Firefox*, visit a particular page and then press *[Ctrl]+u* to view the page source. Inspect the page for information that should be masked such as applications and version numbers. Simply find the file containing the information to mask and replace it with bogus information. Remember to repeat this process whenever you upgrade the application since your changes will likely be overwritten with the new web content files associated with the upgrade.

NOTE: If you are using a Content Management System (CMS), it may be pointless to attempt to mask the identity of your CMS, but you should still mask the version. A hacker performing manual enumeration will likely be able to determine the CMS by simply inspecting Page Sources to identify tell-tale information within the page (such as a *template.css* file signifying a particular CMS). Also, there are automated tools such as *cms-explorer.pl* (in BackTrack) that can be used by hackers to quickly discover the CMS product (but not necessarily the version) used as well as the themes and plugins installed. Plugins are a favorite target of hackers since they can contain exploitable vulnerabilities. Therefore, keeping up-to-date on patches (the next guideline) becomes extremely important.

Patch Regularly

One of the most important aspects of maintaining the security of your web server is to install available patches as soon as possible. The patch management process involves more than just installing patches when your OS notifies you that updates are available. You also need to be vigilant in patching the applications, add-ons, plug-ins, and components that are not covered by the OS-controlled patch process. When I was preparing for the Offensive Security Certified Professional (OSCP) exam, I used the Penetration Testing with BackTrack (PWB) virtual lab to gain hands-on experience with hacking techniques. The virtual lab was full of exploitable servers caused by misconfigured settings and unpatched applications. As an example, there were web servers running unpatched versions of the *WordPress* Content Management System. I was able to exploit these servers by discovering vulnerabilities from sites like securityfocus.com, exploit-db.com, and packetstormsecurity.com. Frequently, these sites provide Proof of Concept (PoC) code and/or detailed exploit information hackers use as attack vectors. I suggest you frequently visit these sites and monitor vulnerabilities that apply to components of your web server, especially extensions installed within Content Management Systems. Third-party add-ons, plug-ins, and components frequently contain exploitable vulnerabilities due to poor programming practices (e.g., failure to sanitize user inputs, etc.). Use the information from reputable vulnerability management services (like *securityfocus.com*) to discover needed patches and techniques for mitigating known attack vectors. You will probably be surprised to learn the volume of new web-based vulnerabilities discovered on a daily basis.