Hardening Linux Web Servers

Comprehensive security spans several disciplines, learn how to secure a system, to host securely coded PHP and Java web services

Yousef Ourabi

Security is a process, not a result. It is a process which is difficult to adopt under normal conditions; the problem is compounded when it spans several job descriptions. All the system level security in the world is rendered useless by insecure web-applications. The converse is also true—programming best practices, such as always verifying user input, are useless when the code is running on a server which hasn't been properly hardened. Securing forward facing GNU/Linux web servers can seem like a daunting task, but it can be made much easier by breaking the process into manageable portions.

This article will cover installing, configuring and hardening free software web servers and associated software including *Apache 2.2.0*, *MySQL 5.0.18*, *PHP 5.1.2*, *Apache-Tomcat 5.5.16* and common Apache modules such as `mod_security`, `mod_ssl`, `mod_rewrite`, `mod_proxy` and `mod_jk`. Common security mistakes in web-applications and how to fix them will also be discussed, focusing on PHP and Java environments.

The most common and apt analogy for security is the onion. That is to say it is a layered approach—any one layer is inadequate, the onion is the sum of its layers. With that in mind, this article attempts to bridge the knowledge gap between system administrators and web developers, allowing individuals tasked with security to achieve a layered security solution.

Only a basic understanding of GNU/Linux and common command line tools is assumed.

Note: due to formatting constraints, long lines of code are often broken into several smaller lines using the \ character. This is not a return and when typing in the line you should not hit the enter key, it is just to prevent line wrapping. Output from commands will also be limited to relevant fields, so the output will look slightly different when you run the commands on your system.

Security at the system level

System level security is one of the most crucial layers in any defense. Hardening at the system level is roughly categorized into network security and file system security.

Network level security can be increased by securing common services such as `xinetd` (otherwise known as the super server) and `OpenSSH`, by correctly configuring or disabling them and enabling a firewall (in our case, `iptables`.

File-System security can be increased by: preventing common avenues of attack, such as root kits; enabling intrusion detections systems (IDS) to verify the integrity of key configuration files; by using tools to detect and remove root kits; and by configuring your logging system so that it will log to a remote host, thereby protecting the integrity of your system logs.

Network security

The first thing you need to do to secure a system from network attacks is find out which processes are listening for connections and on which ports. There are several time tested tools available for this: `nmap` and `netstat`.

netstat

The following command will show you which ports are being listened on, the IP address of the listening socket, and which program or PID is associated with the socket (note: running as the super-user or *root* is necessary for the program field to work properly).

```
$ netstat -l -n -p -t -u -w
```

(`-l` is for listening, `-n` is for IP information and `-p` is for program/PID information, `-t`, `-u`, `-w` are for `tcp`, `udp` and `raw` socket connections. By setting these flags, I disable displaying information about *unix* sockets which are not relevant to network security, as they are only used for interprocess communication on the current host.)

The output will look something like this:

Note: Certain columns have been omitted for space

```
proto Local Address      State    PID/Program name
tcp   127.0.0.1:8005    LISTEN   4079/java
tcp   0.0.0.0:8009      LISTEN   4079/java
tcp   0.0.0.0:3306      LISTEN   18542/mysqld
tcp   0.0.0.0:80        LISTEN   23736/httpd
tcp   0.0.0.0:8080      LISTEN   4079/java
tcp   0.0.0.0:22        LISTEN   11045/sshd
tcp   0.0.0.0:3128      LISTEN   23283/(squid)
tcp   127.0.0.1:25      LISTEN   24453/master
udp   0.0.0.0:3130               23283/(squid)
udp   0.0.0.0:32870              23283/(squid)
```

Understanding the output from netstat is pretty simple. The first field is the protocol, and you will notice that when the protocol is `udp`, there is no state (as obviously `udp` is stateless unlike `tcp`). The next interesting field is the Address field. `0.0.0.0:80` means that the server will respond to any IPs on port 80, while `127.0.0.1:80` means that the server is only listening to the loop back device.

nmap

Another tool in our arsenal is `nmap`, the network mapper. `nmap` is good for determining what ports and services are available on a server from other machines on the network.

(Note: The default option is `-sS`. However, when the system being scanned is running a firewall, such as `iptables`, it won't work, as firewalls that block `icmp` traffic will also block the subsequent scan and the results will be meaningless. The `-P0` option disables pinging the host before scanning it, The `-O` (as in "oh" rather than zero) is to enable `nmap`'s operating system detection via the network stack fingerprint.)

```
$nmap -P0 -O 10.0.2.10
```

The output will look something like this:

```
The 1661 ports scanned but not shown below are in
                                          state: filtered)
 PORT    STATE  SERVICE
 22/tcp  open   ssh
 443/tcp closed https

 Device type: general purpose
 Running: Linux 2.6.X
 OS details: Linux 2.6.7 - 2.6.8
 Uptime 40.462 days since Mon Dec 26 10:05:57 2005
```

Now that I know what services are listening on which ports, I can go about securing them. In some cases, the solution will be disabling the unwanted service via `inetd`; in others, I will use `iptables` rules to block external access to that port.

In the context of a web server, I would recommended disabling all services managed by `inetd` (if they aren't already).

`/etc/xinetd.conf` (Red Hat): this file usually has some minimalistic configuration of the logging software and then an include statement for all the files under `/etc/xinetd.d`, which are configuration files for each service run through the super server.

`/etc/inetd.conf` (Debian): Debian has a much simpler configuration layout—one simple file `/etc/inetd.conf` containing one line for each service managed by `inetd`.

iptables

The venerable `iptables` has been the standard Linux firewall since the 2.4 kernel. The kernels that come with Red Hat and Debian have the proper modules enabled; however, on Debian systems you may need to install the `iptables` user land tools. Configuring `iptables` is fairly simple: `iptables` has chains, rules and targets. `iptables` has three built in chains: `FORWARD`, `INPUT`, and `OUTPUT`. To create an effective firewall I will append rules to chains that will be matched by connection type, source or destination address or state. In more advanced configurations, it is favorable to create custom chains and then reference them in the default chains; but, to demonstrate the basic principles, I am just going to append rules to the three default chains. When a connection is being matched against the configured rules, each rule is checked. If it matches, it is executed, if not, the next rule is tested. As such, the rules allowing traffic should be appended first, and the very last line in any chain should be a deny rule. This is the most secure firewall configuration, where everything is dropped except the explicitly allowed connections.

If you use Debian, run:

```
  $apt-get install iptables ( to install iptables )
  $apt-cache search iptables ( to search for packages
                               related to iptables)
```

To get started with iptables I will list the current rule set using the following command:

```
  $iptables --list
```

(Note: Output has been modified due to formatting constraints.)

```
   Chain INPUT (policy ACCEPT)
   target     prot   opt     source   destination
   ACCEPT     all       anywhere  anywhere \
    state RELATED,ESTABLISHED

   Chain FORWARD (policy ACCEPT)
   target     prot   opt     source   destination
   ACCEPT     all       anywhere anywhere   \
                        state RELATED,ESTABLISHED

   Chain OUTPUT (policy ACCEPT)
   target     prot   opt     source   destination
   DROP       tcp       anywhere anywhere  \
                                       tcp dpt:ssh
```

The partial listing above shows rules that allow incoming traffic that isn't new; that is to say: the connection has been established from inside the network. IP forwarding follows the same rule, and using `ssh` to connect out to other hosts is blocked.

The flush command with no options will flush all rules; if a chain is passed, all rules in that chain will be flushed. I'll flush all rules and begin configuring the firewall.

```
  $iptables -F
    or
  $iptables -F INPUT
  $iptables -F FORWARD
  $iptables -F OUTPUT
```

Next, I am going to append the rules to the appropriate chain. A high level overview of the firewall will be the following:

1. Allow outgoing connections initiated from the host
2. Allow inbound ssh connections on port 2
3. Allow inbound http connections on port 80
4. Allow inbound https connections on port 443
5. Block outbound ssh connections
6. Block everything else

```
  # Enable stateful filtering allowing connections
  # initiated on host be allowed.
  $iptables -A INPUT -m state --state \
RELATED,ESTABLISHED -j ACCEPT

  $iptables -A OUTPUT -m state --state \
NEW,RELATED,ESTABLISHED -j ACCEPT

  # Allow Incoming SSH, HTTP, HTTPS
  $iptables -A INPUT -p tcp -m tcp \
--dport 22 -j ACCEPT
  $iptables -A INPUT -p tcp -m tcp \
--dport 80 -j ACCEPT
  $iptables -A INPUT -p tcp -m tcp \
--dport 443 -j ACCEPT

  # Allow Everything from the local host
  $iptables -A INPUT -s 127.0.0.1 -j ACCEPT

  # Block Outgoing SSH connections
  $iptables -A OUTPUT -p tcp -m tcp \
--dport 22 -j DROP

  # Block Everything else
  $iptables -A INPUT -j DROP
  $iptables -A FORWARD -j DROP
```

To save the changes I have made to the firewall rules I use the `iptables-save` command:

```
  $iptables-save > /root/firewall
```

Later if I wanted to restore my saved rules I would run the `iptables-restore` command:

```
  $iptables-restore -c /root/firewall
```

It's a very good idea to have these rules applied at boot time; check your distribution's documentation for this. In general, on Debian systems the network configuration scripts can be used for this, and on Red-Hat systems a startup script in `/etc/init.d` is appropriate.

Hardening SSH

The OpenSSH package comes installed by default on most distributions. The default configuration on most distributions is pretty lax and favors functionality over security. Allowing *root* logins, listening on all IPs on port 22, and allowing all system accounts to ssh-in are all potential security holes.

Edit /etc/ssh/sshd_config in your favorite editor and change the following lines.

```
# ListenAddress defines the IP address ssh will
# listen on

#ListenAddress 0.0.0.0 -> ListenAddress 10.0.2.10

#Only accept SSH protocol 2 connections
#Protocol 2,1 -> Protocol 2

#Disable root login
PermitRootLogin yes -> PermitRootLogin no

#Disable allowing all system accounts to ssh in,
# only allow certain users (space delimited)
AllowUsers userName1 userName2 userName3

# Change Default port
Port 22 -> Port 2200
```

After making the changes, restart the SSH server for the changes to take affect:

```
$ /etc/init.d/ssh restart
```

File system security

The UNIX file system has several standard directories: /, /tmp, /var, /usr and /home. The two that present the weakest links for a variety of attacks are /tmp and /var. The two most common attacks are: "Denial of Service", by causing the root partition to fill up with logs or other junk (assuming all these directories are mounted on one partition); and running rootkits from the /tmp directory.

One solution to file system Denial of Service attacks is to have these directories mounted on their own partitions, this will prevent the / file system from filling up and stop that avenue of attack.

Rootkits typically write to the /tmp directory and then attempt to run from /tmp. A crafty way to prevent this is to mount the /tmp directory on a separate partition with the noexec, nodev, and nosuid options enabled. This prevents binaries from being executed under /tmp, disables any binary to be *suid* root, and disables any block or character devices from being created under /tmp.

Edit /etc/fstab with your favorite editor, find the line corresponding to /tmp and change it to look like this one.

```
/dev/hda2  /tmp  ext3  nodev,nosuid, noexec  0  0
```

Wikipedia [6] defines rootkits as a set of software tools frequently used by a third party (usually an intruder) after gaining access to a computer system. This translates to custom versions of ps that won't list the irc server the attacker installed, or a custom version of ls that doesn't show certain files. Tools like chkrootkit must be run in combination with IDS systems like fcheck to prevent the successful deployment of rootkits.

chkrootkit is very simple to run, and doesn't require any installation or configuration.

It's a good idea to run chkrootkit at regular intervals, see the script below used by fcheck for inspiration.

```
# Use the wget utility to download the latest
# version of chkrootkit

wget ftp://ftp.pangeia.com.br/pub/seg/pac/ \
  chkrootkit.tar.gz
tar -xzvf chkrootkit.tar.gz
cd chkrootkit-version (whatever version is)
./chkrootkit
```

The next layer of file system security is maintaining and verifying the integrity of configuration files that are typically located under /etc. Intrusion Detection Systems (IDS) allow us to create cryptographic identifiers of important configuration files and store them in a database. They are then periodically re-created and verified against those stored in the database. If there is a mis-match, the file has been changed, you know your system integrity has been violated and which aspects of it are affected. Two well known IDS packages are tripwire and fcheck, which work equally well. However, fcheck has a much simpler configuration and installation process, which is why I favored it for this article.

fcheck

Download fcheck (see resources) and unpack it. fcheck is a cross-platform *Perl* script which runs on UNIX and Windows systems (as long as they have Perl installed).

```
$mkdir /usr/local/fcheck
$cp fcheck /usr/local/fcheck
$cp fcheck.cfg /usr/local/fcheck
```

Edit `/usr/local/fcheck/fcheck.cfg` with your favorite editor and change the following values: `Directory`, `FileTyper`, `Database`, `Logger`, `TimeZone`, and `Signature`.

```
# Directories that will be monitored
# if there is a trailing / it will be recursive


Directory       = /etc/
Directory       = /bin/
Directory       = /sbin/
Directory       = /lib/
Directory       = /usr/bin/
Directory       = /usr/sbin/
Directory       = /usr/lib/
TimeZone        = PST8PDT # For Pacific Standard


# Database of file signatures

DataBase        = /usr/local/fcheck/sol.dbf
Logger          = /usr/bin/logger -t fcheck


# Utility to determin file type
FileTyper       = /bin/file

# What to use to create signatures Database of
# file signatures

$Signature      = /usr/bin/md5sum#
DataBase        = /usr/local/fcheck/sol.dbf
Logger          = /usr/bin/logger -tfcheck

# Utility to determin file type

FileTyper       = /bin/file
```

Also edit the `fcheck` script and change the path of the configuration file to `/usr/local/fcheck/fcheck.cfg` Then run `fcheck` for the first time to create the baseline database.

```
# Options explained:
# c create the database
# a is for all
# d is to monitor directory creation
# s is to create signatures for all files
# x is for extended permissions monitoring

$ ./fcheck -cadsx
```

To test that everything has been setup correctly run the following commands and `fcheck` should alert you to the difference.

```
$ touch /etc/FOO
$ ./fcheck -adsx
```

`fcheck` should display some information about `/etc/FOO`. `$rm /etc/FOO` will prevent future messages.

Next, create a short shell script that will be run periodically by `cron` and check for changes. Open your favorite editor and create `/usr/local/bin/fcheck_script`.

```
#!/bin/bash

# Use mktemp instead of $$ to prevent sym-link
# attacks
FCHECK_LOG='mktemp'

# Grep for any changes
/usr/local/fcheck/fcheck -adsx  \
| grep -Ev ^PROGRESS: |^STATUS:^$ > $FCHECK_LOG

# If there were any changes email the sys-admin
if [-s $FCHECK_LOG ] then
    /usr/bin/mail -s fcheck \
    'hostname' youremail@yourprovider.com  < \
     $FCHECK_LOG
    /bin/rm $FCHECK_LOG
fi
```

The `cron` utility will be used to run periodic checks of the file-system and will compare it to the baseline database. The following command will edit root's *crontab*:

```
$ crontab -e

# Add this line to run the script every 15 minutes
# using nice lower priority when the system load
# is high.
*/15 * * * * nice /usr/local/bin/fcheck_script > \
          /dev/null
```

Symlink Attacks

Side Note: Symlink Attacks running an IDS package usually involve running a script at a pre-configured time using the cron utility. This opens up systems to symlink attacks. Symlink Attacks rely on the attacker knowing that a certain file is going to be created at a certain time with a certain name. A common shell scripting technique that generates some randomness is the use of $$, which is the *PID* of the running script. However, this is vulnerable to Symlink Attacks because most PIDs are below 35K and most file systems can have 35K files. The correct technique is the use of `mktemp`, which is a truly random file name.

Install and configure common services

At this stage, you should have a solid base to build upon. The next step is to compile and install the software servers you will use to serve up your web applications. Installing software from source can be tedious; and there is a great temptation to use the packaged binaries that come with your distribution of choice. I would recommend against this. In a world of zero day exploits, the time it takes the package maintainer to compile and distribute the binaries may be unacceptable. By compiling from the source, you will be in full control of your security situation.

Apache 2.2.0

Now to start compiling and install the services you will be using. *Apache* is a good place to start, since other packages have compile time dependencies on it.

```
md5sum httpd-2.2.0.tar.gz
tar -xzvf httpd-2.2.0.tar.gz
./configure --prefix=/usr/local/apache \
   --enable-ssl --enable-speling \
   --enable-rewrite --enable-proxy
make
make install
```

Apache can be used to configure other web consoles such as the Tomcat Manager application

MySQL 5.0.18

Download the *MySQL* binaries from the mysql site (see resources). This is an exception to my mantra of compiling from source—since the binaries come directly from *MySQL*, as soon as there is an update you can download the latest version.

Note: due to space constraints {.} is shorthand for the version of the tarball.

```
md5sum mysql-{.}-linux-i686-glibc23.tar.gz
cp mysql-{.}-linux-i686-glibc23.tar.gz /usr/local
tar -xzvf mysql-{.}-linux-i686-glibc23.tar.gz
ln -s /usr/local/mysql-{.}-linux-i686-glibc23 \
                                 /usr/local/mysql

groupadd mysql
useradd -g mysql mysql
cd mysql
scripts/mysql_install_db --user=mysql
chown -R root  .
chown -R mysql data
chgrp -R mysql .
bin/mysqld_safe --user=mysql &
cp  support-files/mysql.server /etc/init.d/mysql

# Make sure that mysql is started if
# there is a reboot

cd /etc/rc3.d/
ln -s /etc/init.d/mysql S90mysql
ln -s /etc/init.d/mysql K90mysql

# Copy the configuration file
cp support-files/my-medium.cnf /etc/my.cnf
```

PHP 5.1.2

Now download the *php* package from the php.net site (see resources).

```
md5sum php-5.1.2.tar.gz
tar -xzvf php-5.1.2.tar.gz
./configure --with-prefix=/usr/local/php \
--with-apxs2=/usr/local/apache/bin/apxs \
--with-mysql=/usr/local/mysql
make
make install
cp php.ini-dist /usr/local/php/php.ini
```

Tomcat 5.5.16

Apache-Tomcat is also an exception to my always compile rule.

```
md5sum apache-tomcat-5.5.16.tar.gz
tar -xzvf apache-tomcat-5.5.16.tar.gz
```

mod_jk

Download `mod_jk` from the the tomcat project page (see resources).

```
tar -xzvf jakarta-tomcat-connectors.tar.gz
cd jakarta/jk/native
./configure --with-apxs=/usr/local/apache/bin/apxs
make
make install
```

mod_security

`mod_security` is the most excellent Apache module written by Ivan Ristic.

```
md5sum modsecurity-apache-1.9.2.tar.gz
tar -xzvf modsecurity-apache-1.9.2.tar.gz
cd modsecurity-apache-1.9.2/apache2
/usr/local/apache/bin/apxs -cia mod_security.c
```

Configuring Apache 2.2.0

By this point, you should have installed all of the services and apache modules needed to host and secure PHP and Java environments. Now it's time to take a look at properly configuring everything for security.

Apache 2.2.0 introduces a new configuration layout and new default options in the `httpd.conf` file. In previous versions of Apache, there was only one configuration file by default, which was `conf/httpd.conf`; in the current version, the Apache project has taken another step towards its goal of making configuration more managable and modular. The `conf/httpd.conf` is the main configuration file with include statements for the various configuration files under `conf/extra` such as `httpd-ssl.conf` and `httpd-vhosts.conf`.

Another new and exciting security feature in *Apache 2.2.0* is that the root `httpd.conf` access is denied to all directories. This can be confusing to users coming from previous versions such as 2.0.55 but it is a great step forward in terms of security.

When using the `cron` utility lookout for *symlink attacks*

Here is the configuration directive mentioned above; I would suggest leaving it the way it is. We will allow access for specific virtual-hosts and directories later on.

```
# Default access control  \
# Highly restrictive and applies \
# to everything below it.


<Directory />
  Options FollowSymLinks
  AllowOverride None
  Order deny,allow
  Deny from all
</Directory>
```

Every host, which Apache will be responsible for, will be a virtual host or "vhost". So, start by uncommenting the `Include` directives at the bottom of `conf/httpd.conf`. This is done so that the `httpd-vhosts.conf`, `httpd-ssl.conf` and `httpd-dedfault.conf` are included in the main `httpd.conf` configuration file.

Yet another cool new feature in *Apache 2.2.0* is the ability to see all modules both statically compiled and shared with the `-M` option to `apachectl`.

```
# Edit /usr/local/apache/conf/httpd.conf

# Move the LoadModule Statement for mod_security
# to the top of all module
# statements. This is needed to use SecChrootDir

# Add the following line to load mod_jk
 LoadModule jk_module   modules/mod_jk.so

# The default User and Group is set to daemon,
# create and apache user and group and then
# configure apache to run as such.
User apache
Group apache

# List all modules to verify that mod_jk,
# mod_security, mod_php, and mod_ssl
# are correctly installed and loaded.
 /usr/local/apache/bin/apachectl -M

# Virtual hosts

Include conf/extra/httpd-vhosts.conf \
-> Include conf/extra/httpd-vhosts.conf

# Various default settings
Include conf/extra/httpd-default.conf \
-> Include conf/extra/httpd-default.conf

# Secure (SSL/TLS) connections
 Include conf/extra/httpd-ssl.conf \
-> Include conf/extra/httpd-ssl.conf
```

Now it's time to descend into the extra directory to configure virtual hosts. Open `httpd-vhosts.conf` in your favorite editor. Next, configure the document root to be `/srv/www/vhost1`. Also, add the `AddType` directive for *php*. *PHP* files don't have to have the `.php` file extension. In fact, it's probably a good idea if they are `.html` files. By using the `.php` file extention you are advertizing more information about your setup than you need to.

`mod_security` makes `chrooting` Apache easy!

Here is the full vhost configuration stanza. It is annotated with inline comments: please take the time to read through it.

```
  # Enable mod_security engine
  SecFilterEngine On

  # Chroot Apache the easy way just make sure
  # your web content is under the chrooted directory

  # Note: The log directives must also be valid
  # directories releative to the chroot dir.

  # Note: THIS CANNOT GO INSIDE VHOST STANZA
  # as it applies to the entire apache configuration
  SecChrootDir /chroot/apache

  # Delete the 2nd Vhost stanza as you will only be
  # using one for now
  <VirtualHost *:80>
    ServerAdmin webmaster@mydomain.com
    DocumentRoot /srv/www/vhost1
    ServerName vhost.mydomain.com
    ServerAlias www.mydomain.com
    ErrorLog logs/vhost.mydomain.com-error_log
    CustomLog logs/vhost.mydomain.com-access_log \
     common

    # The PHP engine will interpret all
    # .php and .html files for php code
    AddType application/x-httpd-php .php .phtml \
      .html

    AddType application/x-httpd-php-source .phps

    # Add a local Directory directive to override
    # the global Deny From all in conf/httpd.conf
    <Directory />
      Options FollowSymLinks
      AllowOverride None
      Order deny,allow
      Allow from all
    </Directory>

    # Restrict Access to sensitive files
    <FilesMatch "\.(inc|txt|tar|gz|zip)$">
      Deny from all
    </FilesMatch>



    # Configure MOD_JK
    JkWorkersFile \
             "/usr/local/apache/conf/workers.properties"
    JkLogFile \
               "/usr/local/apache2/logs/mod_jk_www.log"
    JkLogLevel info
    JkLogStampFormat "[%a %b %d %H:%M:%S %Y] "
    JkOptions +ForwardKeySize +ForwardURICompat \
                                -ForwardDirectories
    JkRequestLogFormat "%w %V %T"

    # Any URL that begins with /java/ will be
    # forwarded to the java webapp in tomcat
    JkMount /java/* ajp13

    # Enable and configure MOD_SECURITY

    # See the mod_security documentation
    # link in resources [3]


    # POST Scanning is disabled by default
    SecFilterScanPOST On

    # Make sure only content with standard
    # encoding types is accepted
    SecFilterSelective HTTP_Content-Type \

    "!(^$|^application/x-www-form-urlencoded$| \
     ^multipart/form-data;)"


    # Default Action is to reject request, log, and
    # then return HTTP Response 404 which is
    # File Not Found.
    # Another option would be 403 which is access
    # denied.
    SecFilterDefaultAction "deny,log,status:404"

    # Enable URLEncoding validation. This can
    # prevent Cross-Site Scripting attacks
    SecFilterCheckURLEncoding On

    # Catch and Prevent PHP Fatal Errors from
    # being displayed to the USER
    SecFilterSelective OUTPUT "Fatal error:" \
     deny,status:500

    # Obviously you have to code up this custom
    # Error Page
    ErrorDocument 500 /php-fatal-error.html

    # This can be useful to avoid stack overflow
    # attacks default is 0 255 ie All bytes allowed
    SecFilterForceByteRange 32 126

    </VirtualHost>
```

Security is a process, not a result

Here is the `mod_jk` configuration file `apache/conf/workers.properties` (referenced above):

```
  worker.list=ajp13
  workers.tomcat_home= \
   /usr/local/java/apache-tomcat-5.5.12

  workers.java_home=/usr/local/java/jdk1.5.0_06
  worker.ajp13.type=ajp13
  worker.ajp13.host=localhost
  worker.ajp13.port=8009
```

The configuration stanza below is a variation on an article:

```
<VirtualHost *:80>
  ServerAdmin webmaster@zero-analog.com
  DocumentRoot /srv/www/hercules.zero-analog.com
  ServerName hercules.zero-analog.com
  ErrorLog logs/hercules.zero-analog-error_log
  CustomLog logs/hercules.zero-analog-access_log \
                                             common

  <Directory />
      Options FollowSymLinks
      AllowOverride None
      Order deny,allow
  </Directory>

  <FilesMatch "\.(inc|txt|tar|gz|zip)$">
      Deny from all
  </FilesMatch>

  # This whole stanza is really to forward http
  # requests to https:// tomcat manager
  # so appended /html to the rewrite target.
  # since the full path is manager/html

  <IfModule mod_rewrite.c>
   <IfModule mod_ssl.c>
       <Location /manager>
         RewriteEngine on
         RewriteCond %{HTTPS} !^on$ [NC]
         RewriteRule . \
          https://%{HTTP_HOST}%{REQUEST_URI}/html \
              [L]
      </Location>
   </IfModule>
  </IfModule>

</VirtualHost>

# This is the ssl-vhost under extra/httpd-ssl.conf

<VirtualHost _default_:443>
  # General setup for the virtual host
  DocumentRoot "/srv/www/outpost.zero-analog.com"
  ServerName hercules.zero-analog.com:443
  ServerAdmin webmaster@zero-analog.com

  ErrorLog /usr/local/apache2/logs/error_log
  TransferLog /usr/local/apache2/logs/access_log

  <Directory />
    Options FollowSymLinks
    AllowOverride None
    Order deny,allow
  </Directory>

  RewriteEngine on
  RewriteRule "^/manager$" \
    "https://outpost.zero-analog.com/manager/html" \
                                               [R,L]

   JkWorkersFile \
       "/usr/local/apache2/conf/workers.properties"
   JkLogFile \
       "/usr/local/apache2/logs/mod_jk_hercules.log"
   JkLogLevel info
   JkLogStampFormat "[%a %b %d %H:%M:%S %Y] "
   JkOptions +ForwardKeySize +ForwardURICompat \
                             -ForwardDirectories
   JkRequestLogFormat "%w %V %T"
   JkMount /manager/* ajp13

   #   Enable/Disable SSL for this virtual host.
   SSLEngine on

   # List the ciphers that the client is permitted
   # to negotiate. See the mod_ssl documentation
   # for a complete list.
   SSLCipherSuite  ALL:!ADH:!EXPORT56:RC4+RSA: \
         +HIGH:+MEDIUM:+LOW:+SSLv2:+EXP:+eNULL

   # Server Certificate: Point SSLCertificateFile
   # at a PEM encoded certificate.

   SSLCertificateFile \
                /usr/local/apache/conf/server.crt

   # Server Private Key:
   SSLCertificateKeyFile \
                /usr/local/apache/conf/server.key

   SSLOptions +FakeBasicAuth \
              +ExportCertData \
              +StrictRequire

   <FilesMatch "\.(cgi|shtml|phtml|php)$">
           SSLOptions +StdEnvVars
   </FilesMatch>

   <Directory "/usr/local/apache2/cgi-bin">
           SSLOptions +StdEnvVars
   </Directory>

   BrowserMatch ".*MSIE.*" \
           nokeepalive ssl-unclean-shutdown \
            downgrade-1.0 force-response-1.0

   # Per-Server Logging:
   # The home of a custom SSL log file.
   # Use this when you want a compact non-error
   # SSL logfile on a virtual host basis.

   # *** Note there are \ characters in this
   # string. These are not my artificial line-
   # breaks, please include them ***

   CustomLog /usr/local/apache2/logs/ssl_request_log
     %t %h %{SSL_PROTOCOL}x %{SSL_CIPHER}x \"%r\" %b"

</VirtualHost>
```

Configuring PHP

I have already copied the `php.ini` to `/usr/local/php/php.ini` so that it will be read by the *PHP* engine at startup. By default it's fairly secure, but there are one or two things you can do to improve security.

```
# Edit /usr/local/php/php.ini with your
# favorite editor

# Since you're are going through the trouble of
# hiding PHP files you might as well disable
# this as well
 expose_php = On -> expose_php = Off

# You really don't want users, or worse yet
# an attacker to see error messages
 display_errors = On -> \
    display_erros = Off

# But you do want them logged \
 log_errors = Off -> log_errors = On

# Log to a file
;error_log = filename -> \
     error_log = /var/log/php-err
```

Another option to consider is the *Hardened-PHP project* (see resources section). This is the brain child of three German developers, who continously perform code audits of popular PHP applications. They also release a patch for the standard PHP code, which fixes bugs and security holes in fringe configuration cases, where the main project developers have not had the time or the desire to find a fix.

Configuring Tomcat

The main configuration files from Tomcat are under `$CATALINA_HOME/conf`. The following files are of interest:

- tomcat-users.xml
- server.xml
- web.xml

As you might have guessed, the `tomcat-users.xml` file contains user access information. It is important to create a custom user with a hard-to-guess password for the manager application.

```
# Edit tomcat-users.xml with your favorite
# editor and append the following line.

<user username="bob" password="subGen1us" \
      roles="admin,manager,tomcat,role1"/>
```

Another important tenet of security is preventing information leakage. That's why you enable PHP pages to masquerade as html files. It's also why you want to disable directory listings in Tomcat. This is achieved by editing the `tomcat/conf/web.xml` file.

```
# Open web.xml in your favorite editor and
# look for the following lines:

<init-param>
  <param-name>listings</param-name>
  <param-value>false</param-value>
</init-param>

# The default value is true, which enables
# directory listings, simply change this to false
# to prevent tomcat directory listings.
```

You can also hide JSP files by using a `servlet-mapping` directive in the webapps `web.xml` configuration file. The following lines will map all html files to the JSP servlet, which is internal to Tomcat.

Always verify the `checksums` of packages you download, if there is a mismatch start over and download it again

Note: The listing above goes in the main `tomcat/conf/web.xml`, while this listing should go in the `web.xml` of each application. You can put it in the main `web.xml`. However, there may be cases where this is undesirable.

```
<servlet-mapping>
  <servlet-name>jsp</servlet-name>
    <url-pattern>*.html</url-pattern>
</servlet-mapping>
```

I want to restrict access to `apache-tomcat` to control the flow through Apache. `iptables` is already blocking port 8080 by default, but following the onion principle I'm going to bound Tomcat to loop-back device. This will prevent direct traffic to Tomcat in case of any unforeseen circumstances such as the `iptables` rules being flushed.

```
# Open sever.xml in your favorite editor
# and look for the following lines:

<!-- Define a non-SSL HTTP/1.1 Connector
```

```
       on port 8080 -->
 <Connector port="8080" maxHttpHeaderSize="8192"
   maxThreads="150" minSpareThreads="25"
   maxSpareThreads="75" enableLookups="false"
   redirectPort="8443" acceptCount="100"
   connectionTimeout="20000"
   disableUploadTimeout="true" />

 # And change to this:

  <!-- Define a non-SSL HTTP/1.1 Connector

<c>   on port 8080 -->
  <Connector port="8080" maxHttpHeaderSize="8192"
     address="127.0.0.1" maxThreads="150"
     minSpareThreads="25" maxSpareThreads="75"
     enableLookups="false" redirectPort="8443"
     acceptCount="100"  connectionTimeout="20000"
     disableUploadTimeout="true" />
```

Configuring MySQL

The default MySQL installation is not very secure. This is the case when it is installed manually and also when you use your distribution's precompiled binaries. The default root password is blank, which means anyone can login is as the DBA. This is understandable, as, obviously, a DBA has to login to set the password. However, it's too easy to forget that anyone can login as the MySQL root user and anonymous users are also enabled by default.

```
# Set the root password
mysqladmin -u root -h localhost password subGen1us

# Once this is done, log in as the root user and
# disable anonymous accounts
mysql -u root -p

# Drop the test database which comes installed
# by default
mysql> drop database test;

# Disable anonymous accounts
mysql> use mysql;

mysql> delete from db where User='';
mysql> delete from user where User='';

# Change DBA NAME
mysql> update user set user="mydbadmin" \
      where user="root";

mysql> flush privileges;

# Make sure to login again to make sure
# all the changes work

mysql -u mydbadmin -p
password: subGen1us
```

```
# Configure /etc/my.cnf for security Uncomment
# the following line to disable TCP connections
# to mysql.  As with tomcat this prevents remote
# connections event in the even of the firewall
# even in the even of the firewall rules being
# flushed.

skip-networking
```

Security mistakes in web applications

Now that you are done with configuration, it's time to put your web developer hat on. You now have a very solid base upon which to build your web applications. This brings me to the Achilles heal: the web applications themselves.

What is Cross Site Scripting (XSS)?

Wikipedia [4] defines the term Cross Site Scripting as inaccurate as it really refers to an entire class of vulnerabilities. In general XSS vulnerabilities come down to an age old security problem: not verifying user input. The most common vector of attack is when data is passed to a processing program such as a PHP or JSP script, and then printed back out to the page without being URLEncoded.

The following (highly contrived) PHP code is vulnerable to XSS. If the database to this PHP script contains javascript, it will be executed.

```
# Vulnerable Code
<?php
        $userInput = $_GET['input'];
        print $userInput;
?>

# Secure Code
<?php
    $userInput = urlencode($_GET['input']);
    print $userInput;
?>
```

JSP's or Java Servlets are no less vulnerable. First of all, it is important to understand that all JSP's are compiled to servlets the first time a JSP is called. So, the two are basically the same thing, with different source code representations.

Here is the same vulnerability in the Java world. Note: JSP's have access to the same `HttpServletRequest` object as the servlets they are compiled to. So, in a JSP page, this would manifest itself as `request.getParameter()`.

```
# Vulnerable Code
public class myServlet extends HttpServlet {
  public static void doGet
            (HttpServletRequest req,
             HttpServletResponse res) {

   // Get User Input
   String userInput = req.getParameter("input");

   // Print User Input to page
   PrintWriter out = response.getWriter();
   out.write("<html>");
   out.write(userInput);
   out.write("</html>");

 }
}

# Secure Code
import java.net.URLEncoder;
public class myServlet extends HttpServlet {
  public static void doGet
            (HttpServletRequest req,
             HttpServletResponse res) {

     // Get User Input
     String userInput = req.getParameter("input");
     // URLEncode Input
     userInput =
          URLEncoder.encode(userInput, "UTF-8");

     // Print User Input to page
     PrintWriter out = response.getWriter();
     out.write("<html>");
     out.write(userInput);
     out.write("</html>");

 }
}
```

What is SQL Injection?

SQL Injection is the ability to insert and execute arbitrary SQL code through a web-application. Like XSS attacks, it involves mishandling user input. In this case, properly escaping the input that is to become part of the SQL query. The PHP solution is to use the `mysql_real_escape_string` statement, and the java solution is to use `PreparedStatements`, with the user input as bind variables.

The following code snippets are from Wikipedia [5]:

```
# Partial PHP
$query_result = mysql_query
 ( "select * from users where name = \""
     .
    mysql_real_escape_string($user_name)
     .
    "\"" );
```

```
# Partial Java, ? is the bind variable
Connection con = (acquire Connection)

PreparedStatement pstmt =
  con.prepareStatement
     ("SELECT * FROM users WHERE name = ?");

pstmt.setString(1, userInput);
ResultSet rset = pstmt.executeQuery()
```

Conclusion

There is no magic bullet. "Conclusion" is a misleading heading: there is no conclusion when it comes to security. Security holes are constantly found and patched. Attackers change their methods, and security professionals respond; it is a process, rather than a result. When it's implemented correctly, this process can mitigate or prevent the damage done by attacks.

Changing the default port that OpenSSH listens on is a good way to avoid brute force attacks

There is a silver lining: there is an entire community of security professionals sharing their experience, tips and tricks on the web. Sites like Securityfocus.com (`http://www.securityfocus.com/`) provide heaps of useful information, and the latest trends in security. It's also the home of bugtraq, a mailing list of security holes that I highly recommend you subscribe to. Sites like freshmeat.net (`http://freshmeat.net/`) are the yellow pages of free software projects, listing new releases and updates. All of these sites have RSS feeds, a resource I would also recommend you take advantage of to stay abreast of the latest news.

Resources

- FCheck Main Site (`http://www.geocities.com/fcheck2000/fcheck.html`)
- Useful tools for configuring iptables (`http://software.newsforge.com/software/05/05/09/1846213.shtml`)
- ModSecurity.org Main Site (`http://www.modsecurity.org/`)

- ModSecurity Documentation (`http://www.modsecurity.org/documentation/modsecurity-apache-manual-1.9.2.html#03-configuration`)
- MySQL Download (`http://dev.mysql.com/downloads/mysql/5.0.html`)

Bibliography

[1] Kofler, Michael The Definitive Guide to MySQL, 2nd Edition, Apress:2004

[2] Heironimus, Michael Securing Web Consoles with Apache, Sys Admin Magazine, Feburary 2006, Volume 15, Number 2

[3] Maj, Artur Securing MySQL: step-by-step, Security Focus, http://www.securityfocus.com/infocus/1726

[4] Unknown Autor/Authors Cross site scripting, http://en.wikipedia.org/wiki/Cross_Site_Scripting

[5] Unknown Autor/Authors SQL injection, http://en.wikipedia.org/wiki/SQL_Injection

[6] Unknown Autor/Authors Rootkits, http://en.wikipedia.org/wiki/Rootkit

Copyright information

About the author

By day, Yousef Ourabi is a developer for a major US bank in the San Francisco bay area. And, by night, he works on his company Zero-Analog, which is still pre-launch.