LINUXVOICE TUTORIAL

# SSH, APACHE & TIGER: MAKE YOUR SERVERS SUPER SECURE

MIKE SAUNDERS

Lock down your Linux installations for maximum security and keep one step ahead of crackers.

**WHY DO THIS?**

- Stop bots and crackers getting easy access to your systems
- Understand the trade-offs between security and convenience
- Re-use the skills you learn here when you install distros in the future

Bruce Schneier, the well regarded American expert on cryptography and computer security, once said these wise words: "security is a process, not a product." Keeping your servers safe from malicious types isn't just achieved by chucking on a few extra pieces of software, but by having proper plans and procedures to deal with issues that come up. And security is a moving target – you might have your systems locked down and fully patched right now, but you never know what holes are going to be discovered in the future. Look at the OpenSSL Heartbleed mess, as an example...

Anyway, while most server-oriented Linux distros are pretty secure out of the box, they still make certain sacrifices for user-friendliness. In this tutorial we'll show you how to tighten key components in a server system, including OpenSSH and Apache, and demonstrate how you can mitigate potential problems in the future with scanning tools and an intrusion detection system.

In this case we'll be using a vanilla installation of Debian 7, as it's arguably the most popular GNU/Linux distribution used on servers, but the guides here will be applicable to other distros as well.

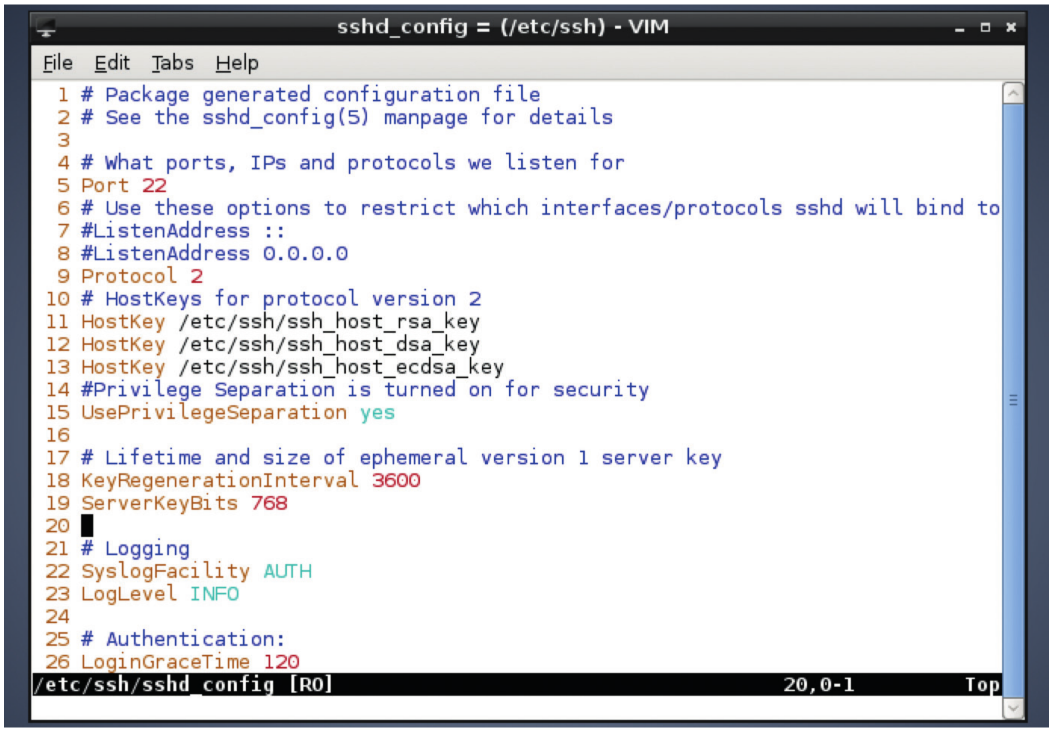
## 1 HARDENING OPENSSH

It's absolutely imperative that we start with OpenSSH. Why that's? Well, it's almost certainly the way you'll be interacting with your server, unless you have the luxury of logging into it directly via a physically connected keyboard and monitor. For headless servers, a good SSH setup is critical, because once you have that out of the way, you can focus on the other running programs.

OpenSSH's daemon (server) configuration file is stored in **/etc/ssh/sshd_config**, so you'll need to edit that (as root) to make changes to the setup. The first thing to do is find this line:

```
PermitRootLogin yes
```

Change **yes** to **no** here to disable direct root logins via SSH. This immediately adds an extra layer of security, as crackers will have to log in with a regular



```
# Package generated configuration file
# See the sshd_config(5) manpage for details

# What ports, IPs and protocols we listen for
Port 22
# Use these options to restrict which interfaces/protocols sshd will bind to
#ListenAddress ::
#ListenAddress 0.0.0.0
Protocol 2
# HostKeys for protocol version 2
HostKey /etc/ssh/ssh_host_rsa_key
HostKey /etc/ssh/ssh_host_dsa_key
HostKey /etc/ssh/ssh_host_ecdsa_key
#Privilege Separation is turned on for security
UsePrivilegeSeparation yes

# Lifetime and size of ephemeral version 1 server key
KeyRegenerationInterval 3600
ServerKeyBits 768

# Logging
SyslogFacility AUTH
LogLevel INFO

# Authentication:
LoginGraceTime 120
/etc/ssh/sshd_config [RO]                                    20,0-1        Top
```

A good Vim setup (see last month's cover feature) provides syntax highlighting for **sshd_config**, making it easier to read and edit.

user account and password first, and then know the root password as well. (Warning: make sure you have a regular user account on the system first, because if you only have a root account, you can lock yourself out by changing this!)

Next, add a line like this to the configuration file:

```
AllowUsers mike graham ben
```

This restricts which users can log in via SSH; if you have many accounts on the machine but only one or two will log in, this is worth doing.

Next, change this line:

```
Port 22
```

22 is the standard SSH port, so it's a good idea to change this to something else (and make sure that your router or firewall is also aware of the change if you'll be logging in from outside your network). A random number like 1234 is fine here – it adds a bit of "security through obscurity". When you log in with the **ssh** command now, you'll need to add **-p 1234** to the end of the command.

## Triple lock

Now, these three changes are useful enough on their own, but together they add a major layer of protection against automated cracking scripts and bots. These are programs that attempt to break into your machine by repeatedly trying username and password combinations, many times a second, until they get access. (If you have a net-facing machine with OpenSSH that has been online for a while, look in **/var/log/auth.log** and you'll probably see many login attempts from IP addresses around the world.)

The default OpenSSH configuration means that these bots don't have to do much work: they know that the root account is available, and they know to try on port 22. By disabling root access and switching to a different port, the bots have to do a lot more guesswork, trying random ports and usernames. If you have a strong password, this makes it very difficult for a bot to gain access.

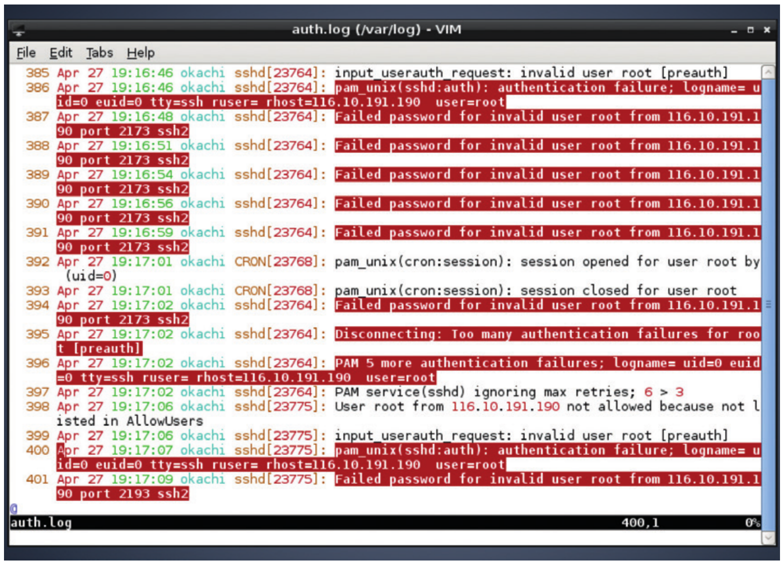
Once you've made your changes to **/etc/ssh/sshd_config**, you'll need to restart the OpenSSH daemon:

```
service ssh restart
```

### Passwordless authentication

While good passwords are hard to crack, you can make it almost impossible for nasty types to log in by disabling password authentication, and using public/private key pairs instead. On the machine(s) you use to log in, enter **ssh-keygen** to generate the keys, then accept the defaults for the file locations and the blank password. (If you suspect someone else might get access to the machine you're using, you can set a password for the key.)

Now enter **ssh-copy-id** followed by the hostname or IP address of the server; your public key will be transferred over to that server. Try logging in and you should see that you don't need to specify a password any more. If it all works, edit **/etc/ssh/sshd_config**, change the **PasswordAuthentication** line to **no**, and restart OpenSSH. (And never give away your private key – it's **~/.ssh/id_rsa**!)



```
Apr 27 19:16:46 okachi sshd[23764]: input_userauth_request: invalid user root [preauth]
Apr 27 19:16:46 okachi sshd[23764]: pam_unix(sshd:auth): authentication failure; logname= u
id=0 euid=0 tty=ssh ruser= rhost=116.10.191.190  user=root
Apr 27 19:16:48 okachi sshd[23764]: Failed password for invalid user root from 116.10.191.1
90 port 2173 ssh2
Apr 27 19:16:51 okachi sshd[23764]: Failed password for invalid user root from 116.10.191.1
90 port 2173 ssh2
Apr 27 19:16:54 okachi sshd[23764]: Failed password for invalid user root from 116.10.191.1
90 port 2173 ssh2
Apr 27 19:16:56 okachi sshd[23764]: Failed password for invalid user root from 116.10.191.1
90 port 2173 ssh2
Apr 27 19:16:59 okachi sshd[23764]: Failed password for invalid user root from 116.10.191.1
90 port 2173 ssh2
Apr 27 19:17:01 okachi CRON[23768]: pam_unix(cron:session): session opened for user root by
 (uid=0)
Apr 27 19:17:01 okachi CRON[23768]: pam_unix(cron:session): session closed for user root
Apr 27 19:17:02 okachi sshd[23764]: Failed password for invalid user root from 116.10.191.1
90 port 2173 ssh2
Apr 27 19:17:02 okachi sshd[23764]: Disconnecting: Too many authentication failures for roo
t [preauth]
Apr 27 19:17:02 okachi sshd[23764]: PAM 5 more authentication failures; logname= uid=0 euid
=0 tty=ssh ruser= rhost=116.10.191.190  user=root
Apr 27 19:17:02 okachi sshd[23764]: PAM service(sshd) ignoring max retries; 6 > 3
Apr 27 19:17:06 okachi sshd[23775]: User root from 116.10.191.190 not allowed because not l
isted in AllowUsers
Apr 27 19:17:06 okachi sshd[23775]: input_userauth_request: invalid user root [preauth]
Apr 27 19:17:07 okachi sshd[23775]: pam_unix(sshd:auth): authentication failure; logname= u
id=0 euid=0 tty=ssh ruser= rhost=116.10.191.190  user=root
Apr 27 19:17:09 okachi sshd[23775]: Failed password for invalid user root from 116.10.191.1
90 port 2193 ssh2
@
auth.log                                                          400,1          0%
```

Here's **/var/log/auth.log** (again with lovely Vim syntax highlighting) on a sample server, with the red lines showing root login attempts by bots.

One enormously useful add-on for OpenSSH is Fail2ban. This is a program that monitors unsuccessful login attempts; if a certain IP address fails to log in too many times, that IP is automatically blacklisted. This again adds more work for crackers and bots, as they can't keep trying to log in from the same IP address and need to switch periodically.

**"The default OpenSSH configuration means that bots don't have to do much work."**

On Debian it's a simple **apt-get install fail2ban** away, and it starts up automatically. By default it automatically blocks IPs (using the system's **iptables** command) for 600 seconds if they have six failed login attempts. You may want to raise the duration to something much longer, and also allow IPs a few more attempts – you don't want to make a few typos when entering your password and accidentally ban yourself!

Fail2ban's main configuration file is **/etc/fail2ban/jail.conf**. However, it's a bad idea to edit that directly (as your changes could be overwritten by system updates), so copy it to **/etc/fail2ban/jail.local** and edit that file instead. The **bantime** and **maxretry** options towards the top control the default settings we mentioned before, and you can also exempt certain IPs from being banned in the **ignoreip** line.

But hang on – **maxretry** here at the top has a value of three, yet we mentioned earlier that there must be six failed login attempts for Fail2ban to take effect! This is because there's a special "[ssh]" section further down that overrides the default settings. You'll see that Fail2ban can be used with other services than SSH too. Once you've made you changes, restart the program like so:

```
service fail2ban restart
```

## 2 HARDENING APACHE

The standard Apache web server configuration in Debian is fairly secure and usable out of the box, but can be made even tighter by disabling a few features. For instance, try to access a non-existing URL in your Apache installation, and at the bottom of the "404 not found" screen that appears you'll see a line like this:

```
Apache 2.2.22 (Debian) Server...
```

It's best not to tell the world the exact version of Apache you're using. Vulnerabilities that affect specific versions occasionally appear, so it's best to leave crackers in the dark about your exact setup. Similarly, Apache includes version information in its HTTP headers: try **telnet <hostname> 80** and then **HEAD / HTTP/1.0** (hit Enter twice). You'll see various bits of information, as in the screenshot.

To disable these features, edit the Apache configuration file; in many distros this is **/etc/apache2/apache2.conf**, but in the case of Debian, its security-related settings are stored in **/etc/apache2/conf.d/security**, so edit that instead. Find the **ServerSignature** line and change **On** to **Off**, and then find the **ServerTokens** line and make sure it's just followed by **Prod** (ie the server will just say that it's the Apache "product", and not give out specific version information). After you've made the changes, restart Apache with:

```
service apache2 restart
```

Apache also tries to be helpful by providing directory listings for directories that don't contain an index**.html** file. This feature, provided by the Apache module **autoindex**, could be abused by hackers to poke around in your system, so you can disable it with:

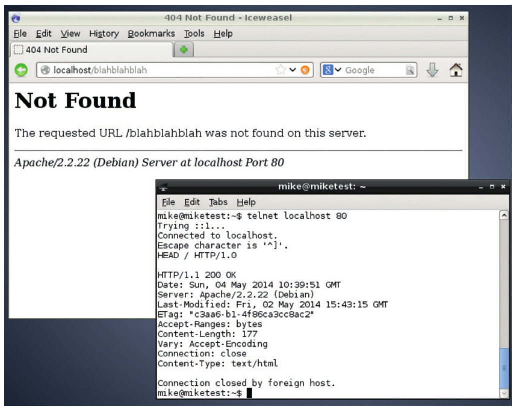
```
a2dismod autoindex
```

### Status report

Another initially helpful (but risky on production machines) module is **status**: this lets you go to **http://<hostname>/server-status** and get a bunch of information about the configuration and performance. In Debian it's only possible to access this page from the same machine on which Apache is running, but this may vary in other distros, so it's wise to turn. it off unless you really need it using **a2dismod status**.

There's a very useful module called ModSecurity, which you can grab with a quick:

```
apt-get install libapache2-modsecurity
```



Not Found

The requested URL /blahblahblah was not found on this server.

*Apache/2.2.22 (Debian) Server at localhost Port 80*

```
mike@miketest:~$ telnet localhost 80
Trying ::1...
Connected to localhost.
Escape character is '^]'.
HEAD / HTTP/1.0

HTTP/1.1 200 OK
Date: Sun, 04 May 2014 10:39:51 GMT
Server: Apache/2.2.22 (Debian)
Last-Modified: Fri, 02 May 2014 15:43:15 GMT
ETag: "c3aa6-b1-4f86ca3cc8ac2"
Accept-Ranges: bytes
Content-Length: 177
Vary: Accept-Encoding
Connection: close
Content-Type: text/html

Connection closed by foreign host.
mike@miketest:~$
```

Apache is telling the world its exact version details, both in 404 pages and HTTP headers – but we can fix that.

This is an exceptionally powerful module that can protect against SQL injection attacks, cross-site scripting, session hijacking, trojans and other risks. After installation a configuration file is placed in **/etc/modsecurity/modsecurity.conf-recommended**; rename this and remove the **-recommended** part to activate it. The rules for detecting attacks are provided in **/usr/share/modsecurity-crs/** – go there and have a look inside the **base_rules**, **optional_rules** and **experimental_rules** directory. Each **.conf** file inside has some comment text explaining what it does, so if you find something useful, copy (or symlink) it into the **/usr/share/modsecurity-crs/activated_rules** folder.

Next, you'll need to tell ModSecurity to use these rules. Edit **/etc/apache2/mods-enabled/mod-security.conf** and beneath the **Include "/etc/modsecurity/*.conf"** line, add these lines:

```
Include "/usr/share/modsecurity-crs/*.conf"
Include "/usr/share/modsecurity-crs/activated_rules/*.conf"
```

Now restart Apache to activate the configuration. By default, ModSecurity only detects problems and doesn't act on them, logging its work to **/var/log/apache2/modsec_audit.log**. This gives you time to see how the rules will affect your site (and if they could break anything). When you're confident with everything, make ModSecurity actively prevent exploits by opening **/etc/modsecurity/modsecurity.conf** and changing the **SecRuleEngine** option from **DetectionOnly** to **On**. Finally, restart Apache.

**LV PRO TIP**

ModSecurity is loaded with advanced features, so visit **www.modsecurity.org/documentation** for all the details.

## 3 HARDENING YOUR SYSTEM

So that's two of the most commonly used server programs hardened: OpenSSH and Apache. What you do from here depends on your particular setup, eg whether your server will primarily be used for email or databases. Still, there are many other things you can do to enhance the general security of your Linux installation. It's a good idea to use an IDS, for instance – an Intrusion Detection System, which keeps an eye on critical system files and alerts you if they change. This is a good way to see if someone has gained remote access to one of your machines and is tampering with configuration files.

Another useful program is an auditing tool. There's a good one in Debian's package repositories, called

Tiger, and although it hasn't been updated for a while, it's still useful for finding holes in your setup. Run:

```
apt-get install tiger
```

Doing this will also install Tripwire, the IDS we'll be using. Once the packages have been downloaded you'll be prompted for two passwords; these are used to protect two keys that will be used to protect configuration files (after all, auditing and file checking tools aren't much use if they can also be easily exploited). Enter something memorable, and once the configuration has finished, enter:

```
tiger -H
```

This will start an extensive security scan of the system, and might take a few minutes depending on the speed of your machine. (Don't be alarmed if your hard drive thrashes a lot during this procedure!) At the end, Tiger will generate a HTML file and show you exactly where it is stored in **/var/log/tiger/**. Open it up (you could use the brilliant text-mode Elinks browser if you're logged in via SSH) to get a comprehensive report that lists potential risks in your system.

These include: file permission problems; processes listening on network sockets; poor configuration file settings; accounts without valid shells; and more. Tiger uses checksums to see if system files have changed after their initial installation, so if an intruder puts a trojan in a binary in **/sbin**, for instance, Tiger will tell you in the report that it differs from the original packaged version.

Every warning is accompanied by a code such as **acc022w**. To get a detailed description of the warning, enter this as root:

```
tigexp acc022w
```

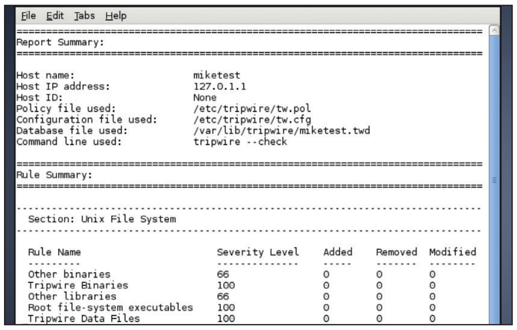
It's very helpful, as it often suggests fixes as well. See the manual page for Tiger (**man tiger**) for other report formats and extra options.

## Advanced file checking

While Tiger is useful for checking executables against their original packaged versions, Tripwire goes a lot further and lets you spot changes all over the filesystem. To set it up, enter:

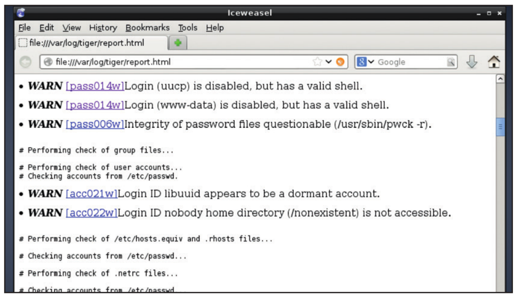
```
tripwire --init
```

This creates a database of file information that will be used when you perform a check. (You may be prompted for one of the passwords you specified



```
===============================================================================
Report Summary:
===============================================================================

Host name:                    miketest
Host IP address:              127.0.1.1
Host ID:                      None
Policy file used:             /etc/tripwire/tw.pol
Configuration file used:      /etc/tripwire/tw.cfg
Database file used:           /var/lib/tripwire/miketest.twd
Command line used:            tripwire --check

===============================================================================
Rule Summary:
===============================================================================

-------------------------------------------------------------------------------
  Section: Unix File System
-------------------------------------------------------------------------------

  Rule Name                       Severity Level    Added    Removed  Modified
  ---------                       --------------    -----    -------  --------
  Other binaries                  66                0        0        0
  Tripwire Binaries               100               0        0        0
  Other libraries                 66                0        0        0
  Root file-system executables    100               0        0        0
  Tripwire Data Files             100               0        0        0
```

Tripwire can monitor any directory on your system, and give you an instant report listing any files that have changed.


Iceweasel
File Edit View History Bookmarks Tools Help
file:///var/log/tiger/report.html
file:///var/log/tiger/report.html
Google

- *WARN* [pass014w]Login (uucp) is disabled, but has a valid shell.
- *WARN* [pass014w]Login (www-data) is disabled, but has a valid shell.
- *WARN* [pass006w]Integrity of password files questionable (/usr/sbin/pwck -r).

```
# Performing check of group files...

# Performing check of user accounts...
# Checking accounts from /etc/passwd.
```

- *WARN* [acc021w]Login ID libuuid appears to be a dormant account.
- *WARN* [acc022w]Login ID nobody home directory (/nonexistent) is not accessible.

```
# Performing check of /etc/hosts.equiv and .rhosts files...

# Checking accounts from /etc/passwd...

# Performing check of .netrc files...

# Checking accounts from /etc/passwd
```

Tiger gives a good overview of potential security flaws in your setup, and the **tigexp** tool provides more detailed descriptions.

when you installed Tiger earlier.) To see that the database works, edit a file in **/etc** – you could add a comment to **/etc/rc.local** for instance. Then run:

```
tripwire --check > report.txt
```

Now look in **report.txt** and do a search for "**rc.local**" (or the file you changed). You'll see something like this:

```
Modified:
"/etc/rc.local"
```

Nice and simple – it tells you exactly which files have changed. At the start of the report you'll see a useful summary as well. There's one problem in the default setup, though: Tripwire monitors **/proc**, and as that's constantly changing (because it contains information about running processes), it clogs up the report with unimportant text. To fix this, we need to change the Tripwire policy that defines which directories it should monitor. Edit **/etc/tripwire/twpol.txt** and find this line:

```
/proc -> $(Device) ;
```

Delete this line and enter the following to update the policy database:

```
twadmin --create-polfile /etc/tripwire/twpol.txt
```

Now we need to rebuild the filesystem database, so go into the **/var/lib/tripwire** directory and remove the **.twd** file contained therein. Run **tripwire --init** and generate a report, and you'll see that **/proc** is no longer included in the report.

Have a more detailed look inside **/etc/tripwire/twpol.txt** to see what Tripwire can do, including different types of warnings for different directories. If you make a change to a system file and don't want Tripwire complaining in every report, you'll need to update the database. In **/var/lib/tripwire/report**, find the most recent report (eg with **ls -l**). Then run:

```
tripwire --update --twrfile <report>
```

Replace **<report>** here with the most recent version. The report will open in a text editor, and as you scroll down, you'll see changed files listed like so:

```
[x] "/etc/rc.local"
```

This means that the file is selected for updating in the database, so you won't be warned about it next time. (If you still want to be warned about that file, remove the **x**.) Save the file and exit the editor, and after the next **--check** command you'll see that the complaint is gone.

**Mike Saunders is the author of The Haynes Linux Manual, writer of the MikeOS assembly language operating system and has been messing with Linux since 1998.**

**LV PRO TIP**
If you'd like us to run a separate tutorial on hardening another piece of server software, drop us a line at **letters@linuxvoice.com**.