Matt Brock's Blog

Producer, songwriter and musician in Dicepeople · Freelance system administrator at CETRE Ltd

Security hardening on Ubuntu Server 14.04

Recently I've been involved with a project where I needed to perform some security hardening on Amazon Web Services EC2 instances running Ubuntu Server 12.04, so I used this excellent guide as a starting point, then I added, removed and modified things as needed.

I decided to take those procedures and modify them for Ubuntu Server 14.04 now that this new LTS version has been released. Some of the procedures from 12.04 no longer need to be performed, and some needed to be changed. The following guidelines are what I've ended up with. You might find these guidelines useful to varying extents on other Linux distributions, but there will be potentially very significant differences depending on which distro you're using.

Assume that all these operations need to be performed as root, which you can either do with `sudo` or by logging in as root first. (I've noticed that Ubuntu users seem particularly averse to logging in as root, apparently preferring instead to issue an endless series of commands starting with `sudo`, but I'm afraid that kind of extra hassle is not for me, so I just log in as root first.)

Harden SSH

I generally regard it as a very sensible idea to disable any kind of root login over SSH, so in */etc/ssh/sshd_config* change `PermitRootLogin` to `no`.

If SSH on your servers is open to the world then I also advise running SSH on a non-standard port in order to avoid incoming SSH hacking attempts. To do that, in */etc/ssh/sshd_config* change `Port` from `22` to another port of your choice, e.g. `1022`. Note that you'll need to update your firewall or EC2 security rules accordingly.

After making changes to SSH, reload the OpenSSH server:

```
service ssh reload
```

Limit `su` access to administrators only

It generally seems like a sensible idea to make sure that only users in the *sudo* group are able to run the `su` command in order to act as (or become) root:

```
dpkg-statoverride --update --add root sudo 4750 /bin/su
```

Improve IP security

Add the following lines to */etc/sysctl.d/10-network-security.conf* to improve IP security:

```
 # Ignore ICMP broadcast requests
net.ipv4.icmp_echo_ignore_broadcasts = 1

# Disable source packet routing
net.ipv4.conf.all.accept_source_route = 0
net.ipv6.conf.all.accept_source_route = 0
net.ipv4.conf.default.accept_source_route = 0
net.ipv6.conf.default.accept_source_route = 0
```

```
# Ignore send redirects
net.ipv4.conf.all.send_redirects = 0
net.ipv4.conf.default.send_redirects = 0

# Block SYN attacks
net.ipv4.tcp_max_syn_backlog = 2048
net.ipv4.tcp_synack_retries = 2
net.ipv4.tcp_syn_retries = 5

# Log Martians
net.ipv4.conf.all.log_martians = 1
net.ipv4.icmp_ignore_bogus_error_responses = 1

# Ignore ICMP redirects
net.ipv4.conf.all.accept_redirects = 0
net.ipv6.conf.all.accept_redirects = 0
net.ipv4.conf.default.accept_redirects = 0
net.ipv6.conf.default.accept_redirects = 0

# Ignore Directed pings
net.ipv4.icmp_echo_ignore_all = 1
```

Load the new rules:

```
service procps start
```

PHP hardening

If you're using PHP, these are changes worth making in */etc/php5/apache2/php.ini* in order to improve the security of PHP:

1. Add `exec`, `system`, `shell_exec`, and `passthru` to `disable_functions`.
2. Change `expose_php` to `Off`.
3. Ensure that `display_errors`, `track_errors` and `html_errors` are set to `Off`.

Apache hardening

If you're using Apache web server, it's worth making sure you have the following parameters set in */etc/apache2/conf-enabled/security.conf* to make sure Apache is suitably hardened:

```
ServerTokens Prod
ServerSignature Off
TraceEnable Off
Header unset ETag
FileETag None
```

For these to take effect you'll need to enable mod_headers:

```
ln -s /etc/apache2/mods-available/headers.load /etc/apache2/mods-enabled/headers.load
```

Then restart Apache:

```
service apache2 restart
```

Install and configure ModSecurity

If you're using Apache, the web application firewall ModSecurity is a great way to harden your web server so that it's much less vulnerable to probes and attacks. Firstly, install the necessary packages:

```
apt-get install libapache2-mod-security2
```

Prepare to enable the recommended configuration:

```
mv /etc/modsecurity/modsecurity.conf-recommended
/etc/modsecurity/modsecurity.conf
```

Then edit */etc/modsecurity/modsecurity.conf*:

1. Set `SecRuleEngine` to `On` to activate the rules.
2. Change `SecRequestBodyLimit` and `SecRequestBodyInMemoryLimit` to `16384000` (or higher as needed) to increase the file upload size limit to 16 MB.

Next, install the Open Web Application Security Project Core Rule Set:

```
cd /tmp
wget https://github.com/SpiderLabs/owasp-modsecurity-crs/archive/master.zip
apt-get install zip
unzip master.zip
cp -r owasp-modsecurity-crs-master/* /etc/modsecurity/
mv /etc/modsecurity/modsecurity_crs_10_setup.conf.example
/etc/modsecurity/modsecurity_crs_10_setup.conf
ls /etc/modsecurity/base_rules | xargs -I {} ln -s
/etc/modsecurity/base_rules/{} /etc/modsecurity/activated_rules/{}
ls /etc/modsecurity/optional_rules | xargs -I {} ln -s
/etc/modsecurity/optional_rules/{} /etc/modsecurity/activated_rules/{}
```

To add the rules to Apache, edit */etc/apache2/mods-available/security2.conf* and add the following line near the end, just before `</IfModule>`:

```
Include "/etc/modsecurity/activated_rules/*.conf"
```

Restart Apache to active the new security rules:

```
service apache2 restart
```

Install and configure mod_evasive

If you're using Apache then it's a good idea to install mod_evasive to help protect against denial of service attacks. Firstly install the package:

```
apt-get install libapache2-mod-evasive
```

Next, set up the log directory:

```
mkdir /var/log/mod_evasive
chown www-data:www-data /var/log/mod_evasive
```

Configure it by editing */etc/apache2/mods-available/evasive.conf*:

1. Uncomment all the lines except `DOSSystemCommand`.
2. Change `DOSEmailNotify` to your email address.

Link the configuration to make it active in Apache:

```
ln -s /etc/apache2/mods-available/evasive.conf /etc/apache2/mods-
enabled/evasive.conf
```

Then activate it by restarting Apache:

```
service apache2 restart
```

Install and configure rootkit checkers

It's highly desirable to get alerted if any rootkits are found on your server, so let's install a couple of rootkit checkers:

```
apt-get install rkhunter chkrootkit
```

Next, let's make them do something useful:

1. In */etc/chkrootkit.conf*, change `RUN_DAILY` to `"true"` so that it runs regularly, and change `"-q"` to `""` otherwise the output doesn't make much sense.
2. In */etc/default/rkhunter*, change `CRON_DAILY_RUN` and `CRON_DB_UPDATE` to `"true"` so it runs regularly.

Finally, let's run these checkers weekly instead of daily, because daily is too annoying:

```
 mv /etc/cron.weekly/rkhunter /etc/cron.weekly/rkhunter_update
mv /etc/cron.daily/rkhunter /etc/cron.weekly/rkhunter_run
mv /etc/cron.daily/chkrootkit /etc/cron.weekly/
```

Install Logwatch

Logwatch is a great tool which provides regular reports nicely summarising what's been going on in the server logs. Install it like this:

```
apt-get install logwatch
```

Make it run weekly instead of daily, otherwise it gets too annoying:

```
mv /etc/cron.daily/00logwatch /etc/cron.weekly/
```

Make it show output from the last week by editing */etc/cron.weekly/oologwatch* and adding `--range 'between -7 days and -1 days'` to the end of the `/usr/sbin/logwatch` command.

Enable automatic security updates

N.B. Be warned that enabling automatic updates can be potentially dangerous for a production server in a live environment. Only enable this for a server in such an environment if you really know what you are doing.

Run this command:

```
dpkg-reconfigure -plow unattended-upgrades
```

Then choose *Yes*.

Enable process accounting

Linux process accounting keeps track of all sorts of details about which commands have been run on the server, who ran them, when, etc. It's a very sensible thing to enable on a server where security is a priority, so let's install it:

```
apt-get install acct
touch /var/log/wtmp
```

To show users' connect times, run `ac`. To show information about commands previously run by users, run `sa`. To see the last commands run, run `lastcomm`. Those are a few commands to give you an idea of what's possible; just read the manpages to get more details if you need to.

Edit: I recently threw together a quick Bash script to send a weekly email with a summary of user activity, login information and commands run. To get the same report yourself, create a file called */etc/cron.weekly/pacct-report* containing the following (don't forget to make this file executable) (you can grab this from GitHub if you prefer):

```
#!/bin/bash

echo "USERS' CONNECT TIMES"
echo ""

ac -d -p

echo ""
echo "COMMANDS BY USER"
echo ""

users=$(cat /etc/passwd | awk -F ':' '{print $1}' | sort)

for user in $users ; do
  comm=$(lastcomm --user $user | awk '{print $1}' | sort | uniq -c | sort -
nr)
  if [ "$comm" ] ; then
    echo "$user:"
    echo "$comm"
  fi
done

echo ""
echo "COMMANDS BY FREQUENCY OF EXECUTION"
```

```
echo ""

sa | awk '{print $1, $6}' | sort -n | head -n -1 | sort -nr
```

Things I haven't covered

There are some additional issues you might want to consider which I haven't covered here for various reasons:

1. This guide assumes your Ubuntu server is on a network behind a firewall of some kind, whether that's a hardware firewall of your own, EC2 security rules on Amazon Web Services, or whatever; and that the firewall is properly configured to only allow through the necessary traffic. However, if that's not the case then you'll need to install and configure a firewall on the Ubuntu server itself. The recommended software for this on Ubuntu is ufw.
2. If you're running an SSH server then you're often told that you must install a tool such as fail2ban immediately if you don't want your server to be hacked to death within seconds. However, I've maintained servers with publicly-accessible SSH servers for many years, and I've found that simply moving SSH to a different port solves this problem far more elegantly. I monitor logs in order to identify incoming hacking attempts, and I haven't seen a single one in the many years I've been doing this. However, using this "security by obscurity" method doesn't mean that such an attack can't happen, and if you don't watch your logs regularly and respond quickly to them as I do, then you would be well advised to install fail2ban or similar as a precaution, in addition to moving your SSH server to another port as described above.
3. Once you've hardened your server, you're advised to run some vulnerability scans and penetration tests against it in order to check that it's actually as invincible as you're now hoping it is. This is a topic which requires a post all of its own so I won't be covering it in any detail here, but a good starting point if you're not already familiar with it is the excellent Nmap security scanner.

brock / June 23, 2014