

# SECURE DEPLOYMENT ENVIRONMENT

## 1. PRIPREMA

### PREPORUČENO ZA PROČITATI

1. The practical Linux hardening guide: <https://github.com/trimstray/the-practical-linux-hardening-guide>
2. 40 Linux Server Hardening Security Tips: <https://www.cyberciti.biz/tips/linux-security.html>

## 2. UVOD

Ova vežba je uvod u izvođenje pregleda (*review*) podešavanja/konfiguracije Linux sistema i njegovog očvršćavanja (*system hardening*). Predočeno je koji su to koraci koje bi trebalo načiniti kako bi se sistem pregledao, uz primere problema koji se obično mogu identifikovati na Linux serveru.

Međutim, sam postupak bi trebalo generalizovati, jer se predloženi bezbedosni problemi javljaju i drugačijim postavkama. Te probleme bi trebalo imati na umu i kada koristite druge operativne sisteme, pa čak i kada iznajmljujete tuđu infrastrukturu (od *cloud* provajdera, na primer).

Uz to, sve više sistema se stavlja u produkciju (*deployment*) pomoću (IaaS) alata za automatizaciju kao što su cfengine, Ansible, Puppet ili Chef. Međutim, ni to ne rešava bezbednosne probleme ako je automatski podignut sistem pogrešno podešen/iskonfigurisan.

### RAZUMEVANJE ULOGE SISTEMA (SYSTEM ROLES)

Pre početka bilo kakvog pregleda, važno je razumeti za šta se sistem koristi. Zahtevi za izloženost i bezbednost će se razlikovati u zavisnosti od namene sistema. Na primer, veb server na internetu i server datoteka u vašoj internoj mreži imaju potpuno različite bezbednosne modele.

Kao primer za ovu vežbu je prikazan veb server. U pitanju je Linux operativni sistem na kojem je pokrenuta PHP veb aplikacija (WordPress blog). Podešavanje operativnog sistema bi trebalo to da odslikava, te bi on trebalo da bude podešen tako da omogući rad ovoj aplikaciji, i ništa više. Aplikacija bi trebalo da bude dostupna na internetu, stoga je verovatno da će aplikacija, ali i sam sistem biti napadnuti. Gotovo je sigurno će biti meta makar automatizovanih *bruteforce* napada.

## 3. ZADATAK

1. Svaki student treba da podesi serversku mašinu na kojoj će fiktivni projekat (jednostavan HTTP server) sa svim odgovarajućim komponentama i alatima biti pušten u rad (*deployed*) (npr. baza podataka, Docker, itd.). Potom bi trebalo ispitati da li je sistem podešen po najboljim bezbednosnim praksama. Primer koji bi trebalo iskoristiti kao vodilju je dat pod tačkom 4. *System Review Example*, dole u tekstu. Sam postupak bi trebalo dokumentovati.

Opciono, zainteresovani studenti mogu isprobati i dokumentovati dodatne najbolje *defence-in-depth* (DID) prakse koje nisu pomenute u ovom materijalu. Preporuka je da se za ovo je da se iskoristi [Lynis](#) alat.

## Smernice za rad:

Zadatak možete odraditi koristeći bilo fizičku, bilo virtuelnu mašinu. Preporuka je da se za operativni sistem iskoristi **LTS** Ubuntu verzije  $\geq 22.04$ , ili neki od ekvivalenata. Prilikom kreiranja naloga potrebno je da za lozinku postavite "starcraftrules". Nakon podešavanja okruženja, potrebno je odraditi sledeće:

- Na podešenom okruženju pokrenuti podešen fiktivni projekat – aplikaciju.
- Dozvoliti pristup aplikaciji iz mreže (podesiti *Firewall* pravila, ali možda još po nešto u zavisnosti od podešavanja). Ako koristite virtuelnu mašinu, neka aplikacija bude dostupna makar fizičkoj *host* mašini.
- Dozvoliti SSH saobraćaj ka i od podešene serverske mašine. Napraviti takvu konfiguraciju da se za SSH pristup koriste ključevi.
- Ispratiti ovaj document do kraja i dati dokaze u vidu *screenshot*-ova da je pregled (review) vašeg sistema odrađen. Da biste izvršili pregled, biće vam potreban root ili ekvivalentan pristup sistemu. Izvođenje pregleda sistema bez ovog nivoa privilegija sprečiće vas da obavite ozbiljan i sveobuhvatan posao. Način za dobijanje root privilegija na linux operativnim sistemima je **sudo -s**
- Iz fajla */etc/shadow*, izvucite heš vaše lozinke, i probajte da ga krekujete upotrebom bilo kog *open-source* alata i *rockyou.txt wordlist*-a.

Radi simulacije što realnijeg scenarija, uputstvo za očvršćavanje (*hardening*) sistema je dato na engleskom jeziku. Ovaj pristup će pomoći i prilikom usvajanja velikog broja stručnih izraza koje je teško prevesti na srpski jezik. Tako da, let's switch to English now ☺

## TAKING NOTES

It is important to take notes during a host review, and a good way to keep notes during the review is to use screen logs. This can be done on the audited system directly or on your system.

First, you need to launch screen (if you decided to do this on the audited system, go to **/tmp** first):

```
$ screen
```

You can then start the logging by hitting **Ctrl-a** and then **H**, the following should appear:

```
Creating logfile "screenlog.0"
```

Then you can perform your review of the system. Once you are done, you can use the same key combination to stop the logging: **Ctrl-a** and then **H**. The following should now appear:

```
Logfile "screenlog.0" closed.
```

You can then rename the file to save it:

```
# mv screenlog.0 Audit-`hostname`-`date +%d-%b-%Y_%H:%M`.txt
```

For all commands ran during the review, it is a good thing to keep the output. A good way to do this is to create a directory in `/tmp` to put everything:

```
# mkdir /tmp/audit
```

And then run all commands one time to get the information and another time to save the results (standard output and standard error) to a file in this directory:

```
# uname
Linux
# uname &> /tmp/audit/uname.txt
```

Finally, if you need to read files, open them in read-only mode. You can do that using:

- `cat file` and `grep` the output
- vi: `vi -R file`
- nano: `nano -v file`

This will prevent you from modifying files you really shouldn't!

## 4. SYSTEM REVIEW EXAMPLE

### OPERATING SYSTEM

The system in the examples is a Debian 6 system; you can get this information by retrieving the content of `/etc/debian_version`. On other Linux based systems, you can:

- read the content of `/etc/redhat-release` for RedHat based systems like Redhat, CentOS and Fedora (`/etc/fedora-release` works as well)
- run `lsb_release -a` on Ubuntu based systems.

Debian 12 is the current stable version so you don't have to worry about end of life. But for older version, it is always important to check if the version is still supported to know if new vulnerabilities will be patched. For example, the previous version of Debian (Lenny) is not supported since February 2012 (<http://wiki.debian.org/DebianLenny>).

### KERNEL

The Kernel version can be retrieved by using the command `uname`:

```
# uname -a
Linux debian 2.6.32-5-amd64 #1 SMP Sun May 6 04:00:17 UTC 2012 x86_64 GNU/Linux
```

Based on this version information, you need to identify whether any vulnerabilities have been published for this specific version. Make sure that you are checking this version against the distribution vulnerabilities list, as most

Linux distributions back-port patches without changing the kernel major and minor version. Checking the uptime of a Linux system can also be a good indicator on when the last kernel upgrade was performed:

```
# uptime
21:14:58 up 70 days,  6:20, 21 users,  load average: 0.74, 0.51, 0.58
```

Here we can see that the system has been up 70 days, the kernel is really unlikely to have been patched during this period.

## TIME MANAGEMENT

It is really important that a system is correctly synchronized with an NTP server. It is important for all operations based on the time:

- Logs management;
- SSL verification of certificates;
- Authentication based on the time.

Furthermore, for sensitive production systems it is always better to use a time zone that doesn't present daylight savings to avoid time jump in the logs. Time jumps are bad as they prevent the accurate correlation between multiple log sources.

Here, we can see that the time zone is configured to UTC:

```
# cat /etc/timezone
Etc/UTC
```

Another thing is to use a NTP server instead of setting the time manually using `ntpdate`:

- the time will stay synchronized;
- there is no risk of a time jump occurring in the logs.

*If there is too much time to regain or loose, NTP servers are likely to give up. So it is good to check that the system is almost on time before starting a NTP server.*

Here, we can see that a NTP server is running:

```
# ps -edf | grep ntp
ntp      1397      1  0 02:21 ?          00:00:00 /usr/sbin/ntpd -p /var/run/ntpd.pid -
g -u 102:104
```

We however need to check that the NTP server can access the servers it is trying to connect to:

```
# ntpq -p -n
      remote           refid      st t when poll reach  delay  offset  jitter
```

```
=====
-121.0.0.41      204.152.184.72  2 u  250 1024  377  54.635  -2.797  0.877
+203.26.72.7     202.147.104.51  3 u  393 1024  377  58.965  -0.952  9.207
*130.102.128.23  130.102.132.164 2 u  495 1024  377  59.171  0.984  2.292
+208.87.107.28   206.246.122.250 2 u  461 1024  377  272.890  1.948  1.136
```

Here, it can correctly access its peers. NTP can also use keys for servers authentication in extremely sensitive environments.

## PACKAGES INSTALLED

It is always a good thing to check that the number of packages is limited to the strict minimum to limit the system exposure. Furthermore, since the audited system is used as a web server, some packages, such as those related to the graphical interface (X, Gnome, KDE) or games are not needed and should not be installed. On Debian, you can retrieve a list of packages installed using the following command:

```
# dpkg -l | less
Desired=Unknown/Install/Remove/Purge/Hold
| Status=Not/Inst/Conf-files/Unpacked/halF-conf/Half-inst/trig-aWait/Trig-pend
|/ Err?=(none)/Reinst-required (Status,Err: uppercase=bad)
||/ Name                               Version                               Description
+++-----
=====
ii  adduser                               3.112+nmu2                           add and remove u
sers and groups
ii  apache2                               2.2.16-6+squeeze7                    Apache HTTP Serv
er metapackage
ii  apache2-mpm-prefork                   2.2.16-6+squeeze7                    Apache HTTP Serv
er - traditional non-threaded model
ii  apache2-utils                         2.2.16-6+squeeze7                    utility programs
for webserver
ii  apache2.2-bin                         2.2.16-6+squeeze7                    Apache HTTP Serv
er common binary files
[ ... ]
```

You can then search this list for unneeded packages or packages presenting vulnerabilities.

## LOGGING

Another thing that needs to be checked is how the logging of events is performed. On this system, we can see that **rsyslog** is used:

```
# ps -edf | grep syslog
root      1305      1  0 Aug06 ?           00:00:00 /usr/sbin/rsyslogd -c4
```

**rsyslog** configuration is stored in `/etc/rsyslog.conf`. By reviewing the configuration file, we can see that **rsyslog** on this system is not configured to receive logs:

```
# provides UDP syslog reception
#$ModLoad imudp
#$UDPServerRun 514

# provides TCP syslog reception
#$ModLoad imtcp
#$InputTCPServerRun 514
```

The configuration use for files and directories permissions is restricted:

```
# Set the default permissions for all log files.
#
$FileOwner root
$FileGroup adm
$FileCreateMode 0640
$DirCreateMode 0755
$Umask 0022
```

However, none of the logging configuration is set to perform logging to a remote system. Unless logs are backed up by a remote system, it is always better to configure remote logging for servers, in rsyslog it is performed by `@servername`.

## 5. NETWORK REVIEW

### GENERAL INFORMATION

It is always good to retrieve, examine, and store basic system configuration information before doing any other checks. I keep the output of the following commands:

- `ifconfig -a` to see what network interfaces are present (a *tip*: `ip` has replaced `ifconfig` in most places);
- `route -n` to get the system routes. If the system does not have the `route` command installed, `netstat -rn` can be a suitable substitute.
- `cat /etc/resolv.conf` and `cat /etc/hosts` to know more about the DNS configuration of the system.

*The system we are reviewing is a standard Debian installation. For more complex system using LDAP or NIS you will need to check the `/etc/nsswitch.conf` to get the full picture of how the DNS, users and groups are configured.*

Once you have this information, you will be able to review the firewall and services configuration.

## FIREWALL RULES

For firewall rules you need to check 2 things:

- the firewall rules themselves;
- if the rules will still be present in case of reboot.

Here we can retrieve the firewall rules by using `iptables`:

```
# iptables -L -v
Chain INPUT (policy DROP 2 packets, 1152 bytes)
  pkts bytes target     prot opt in     out     source         destination
    40  3696 ACCEPT     all  --  lo      any      anywhere       anywhere
    86  7401 ACCEPT     tcp  --  eth0    any      anywhere       anywhere
    tcp dpt:ssh
     0     0 ACCEPT     tcp  --  eth0    any      anywhere       anywhere
    tcp dpt:www
    29  4818 ACCEPT     all  --  any     any      anywhere       anywhere
    state RELATED,ESTABLISHED

Chain FORWARD (policy ACCEPT 0 packets, 0 bytes)
  pkts bytes target     prot opt in     out     source         destination

Chain OUTPUT (policy ACCEPT 16 packets, 1135 bytes)
  pkts bytes target     prot opt in     out     source         destination
```

We can see that these rules allow for in-going traffic:

- access from everywhere to the port 80 (HTTP);
- access from everywhere to the port 22 (SSH);
- access for already known traffic (response to connections initiated by the server itself).

The port 80 (HTTP) will need to be access by anyone on Internet to visit the hosted website. However the port 22 (SSH) should only be accessed by administrators of the server. A white-list of trusted IP addresses should be created to limit access to the SSH server.

We can also see that no rules are applied to the outgoing traffic. It is a best practice to limit outgoing traffic as much as possible by restricting the servers access to the Internet and other internal hosts. Limiting outgoing traffic is likely to slow down an intruder and provide an additional layer of defense-in-depth. For our system review, the server should only allow access on the way out to:

- already established connections: like packets going back to the client after he connects to a service;
- DNS servers for name resolution;
- Debian HTTP servers for software updates.
- NTP servers for time synchronization

We also need to check that the firewall rules will be applied automatically every time the server is started. The Debian way (<http://wiki.debian.org/iptables>) of doing this is to create a `/etc/network/if-pre-up.d/iptables` file containing the commands to load the rules:

```
# cat etc/network/if-pre-up.d/iptables
#!/bin/bash

/sbin/iptables-restore < /etc/iptables.up.rules
```

We need to check that the file used to load the rules and the current rules match:

```
# cat /etc/iptables.up.rules
*filter
:INPUT DROP [0:0]
:FORWARD ACCEPT [0:0]
:OUTPUT ACCEPT [27:3016]
-A INPUT -i lo -j ACCEPT
-A INPUT -i eth0 -p tcp -m tcp --dport 22 -j ACCEPT
-A INPUT -i eth0 -p tcp -m tcp --dport 80 -j ACCEPT
-A INPUT -m state --state RELATED,ESTABLISHED -j ACCEPT
COMMIT
```

Here we can see that the same rules are applied. It is however very common to get different rules on systems or even on devices (like routers or firewall). That's why it is really important to make sure they correctly match.

*After you modify the firewall rules, make sure that services like NTP and DNS can still get access to their respective servers.*

## IPV6

We can check that there is no firewall rule for IPv6 on this host:



```
# ip6tables -L -v
Chain INPUT (policy ACCEPT 0 packets, 0 bytes)
  pkts bytes target      prot opt in      out     source            destination

Chain FORWARD (policy ACCEPT 0 packets, 0 bytes)
  pkts bytes target      prot opt in      out     source            destination

Chain OUTPUT (policy ACCEPT 0 packets, 0 bytes)
  pkts bytes target      prot opt in      out     source            destination
```

At least the same firewall rules applied to IPv4 should be applied for IPv6. If IPv6 is not necessary on this server, it can be disabled, on Debian, by creating a file named `/etc/sysctl.d/disableipv6.conf` that contains the following `net.ipv6.conf.all.disable_ipv6=1`.

## 6. FILESYSTEM REVIEW

### MOUNTED PARTITIONS

This is how you can check that the configuration is secure. For each entries in the `/etc/fstab`, you need to review that:

- `noatime` is not used. "noatime" prevents update of inode access time. In case on intrusion, it is always a good idea to have this information;
- for file system like `/tmp` or `/home`, you can use `noexec` and `nosuid` to prevent users from executing binaries and respectively to prevent `setuid` to be interpreted. It is also recommended that `nosuid` is included for `/dev`, if `/dev` is using `devfs`.

### SENSITIVE FILES

It is important to check permission on sensitive files. As a general rule, you want to check:

- that files containing sensitive information (password, private keys) can't be read by any user;
- that file containing configuration can't be modified by any user.

For example, the following files shouldn't be readable by any user:

- `/etc/shadow` containing users' passwords (hashed);
- `/etc/mysql/my.cnf` containing `Debian-sys-maint`'s password;
- SSL private keys used by Apache.

The same obviously applies to any copy of these files, here we can see that a file named `/etc/shadow.backup` is present on the file system and can be read by any user. Even if the permissions are correctly set on the `/etc/shadow`, this "backup" file strongly reduces the security of the system.

## SETUID

Setuid files are files ran with owner's privileges, as opposed to normal binaries that are run with current user's privileges. These files are obviously potential security risks, especially if their owner is `root`. A common example of a setuid file is the utility `passwd`. In order for users to change their password, they need access to the `/etc/shadow` file, but they cannot be allowed to read the file directly, since this contains the hashed passwords for all users. The setuid bit being set means that this executable will execute as the owner of the file (in this case `root`), allowing `/etc/shadow` to be read and modified without allowing the user to directly access the contents of this file.

You can retrieve a list of the setuid files on the system by running the following command:

```
# find / -perm -4000 -ls
[...]
```

4948	15	-rwsr-sr-x	1	libuuid	libuuid	15208	Jan 25	2011	/usr/sbin/uidd
------	----	------------	---	---------	---------	-------	--------	------	----------------

```
[...]
```

41584	34	-rwsr-xr-x	1	root	root	34248	Oct 14	2010	/bin/ping
41585	36	-rwsr-xr-x	1	root	root	36640	Oct 14	2010	/bin/ping6
41590	34	-rwsr-xr-x	1	root	root	34024	Feb 15	2011	/bin/su
41595	53	-rwsr-xr-x	1	root	root	53648	Jan 25	2011	/bin/umount

*To avoid all the `No such file or directory` messages, you can redirect stderr (standard error) to `/dev/null` by adding `2>/dev/null` after the command.*

For each file found, you need to check if this file is legitimate and if its permissions are set correctly.

## NORMAL FILES

Using `find`, you can retrieve a list of files that are readable and write-able by any users using the following command:

```
# find / -type f -perm -006 2>/dev/null
```

and a list of files write-able by any users using:

```
find / -type f -perm -002 2>/dev/null
```

You probably want to filter out everything in `/proc` using `| grep -v /proc` after the commands above. Once you have done that, you can see that the file in `/var/www/` can be read from and written to by any users. This is likely to ease the progress of an attacker gaining illegitimate access to the system.

## BACKUP

A common mistake performed by system administrators comes from backups. Too often, the backup files can be read by unprivileged users.

On this system, we can see that a directory named `/backup` is present, if we check the permissions of its content we can see that the files can be read by any user. An attacker can use this to extract `etc.tgz` and read the `shadow` file. It is important to change permissions on these files and on the `/backup` directory and to make sure that the script used for backup correctly set the permissions every time it gets called.

## 7. USERS REVIEW

### REVIEWING THE PASSWD FILE

A common misconfiguration/backdoor of the `/etc/passwd` is to have a user with the uid 0. On traditional systems, only `root` is supposed to have the uid 0. If another user has the uid 0, he's basically `root` on the system.

*On FreeBSD, for maintenance and historical reason, `root` and `toor` have the uid 0.*

When reviewing `/etc/passwd` it is always good to check which users have a shell (`/bin/bash`, `/bin/sh` ...) and which don't (`/bin/false`, `/usr/sbin/nologin` ...), restricting shell access will ensure the user can't connect and run commands on the system.

### REVIEWING THE SHADOW FILE

In the `/etc/shadow`, the most important is to review algorithms used for password encryption and passwords' quality.

You can check what algorithm is used by checking the hash format, if the hash:

- does not have a `$` sign, DES is used;
- starts by `$1$`, MD5 is used;
- starts by `$2$` or `$2a$`, Blowfish is used;
- starts by `$5$`, SHA-256 is used;
- starts by `$6$`, SHA-512 is used.

On modern Linux systems, you want to avoid DES and MD5. DES algorithm for password encryption will limit the size of the password to 8 characters and is really easy to crack. MD5 doesn't present the same length weakness but is pretty quick to brute force compared to other algorithms.

You can review what is the default algorithm used to encrypted password by checking the content of `/etc/pam.d/common-password`. Here we can see that `sha512` is used:

```
$ cat /etc/pam.d/common-password
[...]  
password [success=1 default=ignore] pam_unix.so obscure sha512  
[...]
```

To improve the system security, you can add some requirements on the password by installing `libpam-cracklib`:

```
# apt-get install libpam-cracklib
```

To enforce the complexity, you can play with the parameter `minlen` to enforce the complexity and `lcredit`, `ucredit`, `dcredit` and `ocredit` to force the number of lowercase, uppercase, digit and special characters in the password. After installing `libcrack`, the following line has been added to the `/etc/pam.d/common-password` to enforce the complexity of passwords:

```
$ cat /etc/pam.d/common-password
[...]  
password      requisite                                pam_cracklib.so retry=3 minlen=8 difo  
k=4  
[...]
```

We can see here another important parameter: `difok`, that tells `pam` how many characters in the new password must be different from the previous password.

To force the number of special characters and digits in the password, you can change the line above to:

```
$ cat /etc/pam.d/common-password
[...]  
password      requisite                                pam_cracklib.so retry=3 minlen=8 difo  
k=4 ocredit=-2 dcredit=-1  
[...]
```

It will enforce one digit and two special characters in the password. You can read more about `pam_cracklib` configuration by running: `man pam_cracklib` on the system.

John-The-Ripper can be used to crack this password, most modern Linux distribution include a version of John, in order to crack this password you need to tell John what algorithm has been used to encrypted it.

In most Linux distributions, the version of John-The-Ripper provided only supports a small number of formats. You can run `john` without any arguments to get a list of the supported formats from the usage information. For example on Fedora, the following formats are supported:

```
$ john  
# ...usage information...  
--format=NAME          force hash type NAME: DES/BSDI/MD5/BF/AFS/LM/crypt  
# ...usage information...
```

Here, we saw that we have two formats used. Using the `john` installed on this system, you can crack the user password since DES is supported by this version of John:

```
# john password
```

Unfortunately (for the attacker), the root password is using SHA-512. In order to crack this password, we will need a version of John supporting crypt. The community-enhanced version supports crypt and can be used. The following command line can be used to crack the password previously retrieved:

```
$ ./john password --format=crypt --wordlist=dico --rules
```

The following options are used:

- `shadow` tells john what file contains the password hash
- `--format=crypt` tells john that the password hash can be cracked using crypt;
- `--wordlist=dico` tells john to use the file `dico` as a dictionary
- `--rules` tells john to try variations for each word provided

John-The-Ripper has an opportunistic behavior regarding algorithm selection, it will take the first it recognizes. You need to make sure that you run John with each of the format available in the shadow file.

## REVIEWING THE SUDO CONFIGURATION

The `sudo` configuration is stored in `/etc/sudoers`:

```
# egrep -v '^#|^$' /etc/sudoers
Defaults    env_reset
root    ALL=(ALL) ALL
%sudo    ALL=(ALL) ALL
user    ALL=(ALL) NOPASSWD: ALL
```

We can see there that the user named `user` can be any users (including `root`) without any passwords and without restriction.

A common mistake with sudo is to provide a user with a limited set of commands that will still allow him to get a `root` shell on the system. For example, a user with access to `/bin/chown` (change owner) and `/bin/chmod` (change mode) will be able to copy a shell in his home and change the shell owner to root and add the setuid bit on the file. This way, this user will be able to have a `root` shell on the system.

## 8. SERVICES REVIEW

### IDENTIFYING RUNNING SERVICES

Using `ps`, you can quickly identified running services:

```
# ps -edf
UID          PID  PPID  C  STIME TTY          TIME CMD
```

root	1	0	0	02:21	?	00:00:00	init [2]
root	2	0	0	02:21	?	00:00:00	[kthreadd]
root	3	2	0	02:21	?	00:00:00	[migration/0]
root	4	2	0	02:21	?	00:00:00	[ksoftirqd/0]
root	5	2	0	02:21	?	00:00:00	[watchdog/0]
root	6	2	0	02:21	?	00:00:00	[events/0]
root	7	2	0	02:21	?	00:00:00	[cpuset]
root	8	2	0	02:21	?	00:00:00	[khelper]
root	9	2	0	02:21	?	00:00:00	[netns]
root	10	2	0	02:21	?	00:00:00	[async/mgr]
root	11	2	0	02:21	?	00:00:00	[pm]
root	12	2	0	02:21	?	00:00:00	[sync_supers]
root	13	2	0	02:21	?	00:00:00	[bdi-default]
root	14	2	0	02:21	?	00:00:00	[kintegrityd/0]
root	15	2	0	02:21	?	00:00:00	[kblockd/0]
root	16	2	0	02:21	?	00:00:00	[kacpid]
root	17	2	0	02:21	?	00:00:00	[kacpi_notify]
root	18	2	0	02:21	?	00:00:00	[kacpi_hotplug]
root	19	2	0	02:21	?	00:00:00	[kseriod]
root	21	2	0	02:21	?	00:00:00	[kondemand/0]
root	22	2	0	02:21	?	00:00:00	[khungtaskd]
root	23	2	0	02:21	?	00:00:00	[kswapd0]
root	24	2	0	02:21	?	00:00:00	[ksmd]
root	25	2	0	02:21	?	00:00:00	[aio/0]
root	26	2	0	02:21	?	00:00:00	[crypto/0]
root	219	2	0	02:21	?	00:00:00	[ata/0]
root	220	2	0	02:21	?	00:00:00	[ata_aux]
root	221	2	0	02:21	?	00:00:00	[scsi_eh_0]
root	222	2	0	02:21	?	00:00:00	[scsi_eh_1]
root	294	2	0	02:21	?	00:00:00	[aufs0]
root	295	2	0	02:21	?	00:00:00	[aufs_pre/0]
root	323	2	0	02:21	?	00:00:00	[loop0]
root	738	1	0	02:21	?	00:00:00	udev --daemon
root	957	2	0	02:21	?	00:00:00	[kpsmouse]

root	991	738	0	02:21	?	00:00:00	udevd --daemon
root	992	738	0	02:21	?	00:00:00	udevd --daemon
root	1307	1	0	02:21	?	00:00:00	/usr/sbin/rsyslogd -c4
root	1341	1	0	02:21	?	00:00:00	/usr/sbin/apache2 -k start
www-data	1344	1341	0	02:21	?	00:00:00	/usr/sbin/apache2 -k start
root	1395	1	0	02:21	?	00:00:00	/usr/sbin/cron
ntp	1397	1	0	02:21	?	00:00:00	/usr/sbin/ntpd -p /var/run/nt
www-data	1423	1341	0	02:21	?	00:00:00	/usr/sbin/apache2 -k start
www-data	1424	1341	0	02:21	?	00:00:00	/usr/sbin/apache2 -k start
www-data	1425	1341	0	02:21	?	00:00:00	/usr/sbin/apache2 -k start
www-data	1426	1341	0	02:21	?	00:00:00	/usr/sbin/apache2 -k start
www-data	1427	1341	0	02:21	?	00:00:00	/usr/sbin/apache2 -k start
root	1447	1	0	02:21	?	00:00:00	/bin/sh /usr/bin/mysqld_safe
mysql	1555	1447	0	02:21	?	00:00:02	/usr/sbin/mysqld --basedir=/u
root	1556	1447	0	02:21	?	00:00:00	logger -t mysqld -p daemon.er
root	1618	1	0	02:21	tty1	00:00:00	/bin/login -f
root	1619	1	0	02:21	tty2	00:00:00	/bin/login -f
root	1620	1	0	02:21	tty3	00:00:00	/bin/login -f
root	1621	1	0	02:21	tty4	00:00:00	/bin/login -f
root	1622	1	0	02:21	tty5	00:00:00	/bin/login -f
root	1623	1	0	02:21	tty6	00:00:00	/bin/login -f
user	1626	1623	0	02:21	tty6	00:00:00	-bash
user	1628	1619	0	02:21	tty2	00:00:00	-bash
user	1629	1621	0	02:21	tty4	00:00:00	-bash
user	1630	1620	0	02:21	tty3	00:00:00	-bash
user	1631	1622	0	02:21	tty5	00:00:00	-bash
user	1633	1618	0	02:21	tty1	00:00:00	-bash
root	1727	1	0	02:21	?	00:00:00	dhclient -v -pf /var/run/dhcl
root	1758	1	0	02:21	?	00:00:00	/usr/sbin/sshd
root	1760	1633	0	02:21	tty1	00:00:00	sudo -s
root	1761	1760	0	02:21	tty1	00:00:00	/bin/bash
root	1845	1758	0	03:46	?	00:00:00	sshd: user [priv]
user	1848	1845	0	03:46	?	00:00:00	sshd: user@pts/0
user	1849	1848	0	03:46	pts/0	00:00:00	-bash

```

root      1910  1849   0 06:01 pts/0    00:00:00 sudo -s
root      1911  1910   0 06:01 pts/0    00:00:00 /bin/bash
root      1912  1911   0 06:01 pts/0    00:00:00 ps -edf

```

Another good way to list potentially interesting services is to use **lsof** to get the list of network connection and especially services listening on the network.

We can first work on the service listening on UDP:

```

# lsof -i UDP -n -P
COMMAND  PID  USER  FD  TYPE DEVICE SIZE/OFF NODE NAME
ntpd     1397  ntp    16u  IPv4  3861      0t0  UDP *:123
ntpd     1397  ntp    17u  IPv6  3869      0t0  UDP *:123
ntpd     1397  ntp    18u  IPv4  3875      0t0  UDP 127.0.0.1:123
ntpd     1397  ntp    19u  IPv6  3876      0t0  UDP [::1]:123
ntpd     1397  ntp    20u  IPv4  5418      0t0  UDP 10.0.2.15:123
ntpd     1397  ntp    21u  IPv6  5419      0t0  UDP [fe80::5054:ff:fe12:3456]:123
dhclient 1727  root    6u  IPv4  3937      0t0  UDP *:68

```

Here we can see:

- ntpd is running on port UDP/123 on IPv6 and IPv4;
- dhclient for dynamic network configuration is running on port UDP/68.

Then get information about the TCP services:

```

# lsof -i TCP -n -P
COMMAND  PID  USER  FD  TYPE DEVICE SIZE/OFF NODE NAME
mysqld   1555  mysql  10u  IPv4  4073      0t0  TCP *:3306 (LISTEN)
sshd     1758  root    3u  IPv4  4966      0t0  TCP *:22 (LISTEN)
sshd     1758  root    4u  IPv6  4968      0t0  TCP *:22 (LISTEN)
sshd     1845  root    3r  IPv4  5615      0t0  TCP 10.0.2.15:22->10.0.2.2:52439 (
ESTABLISHED)
sshd     1848  user    3u  IPv4  5615      0t0  TCP 10.0.2.15:22->10.0.2.2:52439 (
ESTABLISHED)
apache2  1966  root    3u  IPv4  6502      0t0  TCP *:80 (LISTEN)
apache2  1969  www-data  3u  IPv4  6502      0t0  TCP *:80 (LISTEN)
apache2  1970  www-data  3u  IPv4  6502      0t0  TCP *:80 (LISTEN)
apache2  1971  www-data  3u  IPv4  6502      0t0  TCP *:80 (LISTEN)

```



```
apache2 1972 www-data      3u  IPv4    6502      0t0  TCP *:80 (LISTEN)
```

Here we can see:

- Apache2 is running and listening on the port TCP/80 on IPv4;
- mysqld is listening on port TCP/3306 on IPv4;
- sshd is running on port TCP/22 on IPv6 and IPv4;
- someone (user) is connected to the SSH server.

## OPENSSSH

OpenSSH is running on this system; its configuration file can be found in `/etc/ssh/` and is named `sshd_config`. The main thing to check for is the `PermitRootLogin` option that will prevent `root` to connect directly to the system. The default value of this option is `yes`, so you should be sure that it is set to `no` and that the line is not commented in the configuration file.

Additionally, it is wise to check that SSH version 1 is disabled, if it is not required. This line can be disabled by adding a line specifying only version 2 is to be used:

```
protocol 2
```

*Using sudo instead of providing `root` access to the system will increase the traceability of administrative action. Furthermore, you won't have to change the `root` password every time someone leaves your company and you can keep the `root` password in the safe in case of emergency.*

Since the system is available to the Internet, it is a good idea to change the default port (don't forget to put this new value in the firewall configuration) to make it less likely that automated brute-force/SSH scanners will locate this service. This can be done by changing the `Port` value in the configuration file (`/etc/ssh/sshd_config`).

Since this server is not a bounce box, `AllowTcpForwarding` should be set to `no` to prevent users from using this system to access other system or to bypass network-layer security restrictions. The default value is `yes`, so you need to make sure that `no` is used and that the line is not commented.

To update its configuration, SSH will need to be restarted.

## MYSQL

Using `lsuf`, we saw that MySQL was listening on all interfaces, given the current use of the system, this is unnecessary, and to limit the system's attack surface, it is better to bind the socket on localhost to prevent external access to this service. This can be done by adding the following option to the `[mysqld]` section of the `/etc/mysql/my.cnf` file:

```
[mysqld]
bind-address = 127.0.0.1
```

To update its configuration, MySQL will need to be restarted.

We can now review the configuration of MySQL. First, we need to connect to MySQL, here we are lucky and no password is set for the root user:

```
$ mysql -u root
```

MySQL has its own `root` user, even if you are not `root` on the server, you can connect to the database using the root user. The system `root` user and the MySQL `root` user are two different things. When you connect to the database, by default, the `mysql` client use your current user name to connect.

If you don't have the password to connect as root to the database but you are root on the server, you can use the command **strings** on the Mysql users table stored in `/var/lib/mysql/mysql/user.MYD` to get the hashes:

```
# strings /var/lib/mysql/mysql/user.MYD
localhost
root
127.0.0.1
root
localhost
debian-sys-maint*70B6D2E96920B5E2BEE4AE1C719752A021342EF9
localhost wordpress*C260A4F79FA905AF65142FFE0B9A14FE0E1519CC
```

On Debian, a maintenance user named **debian-sys-maint** has access to the database and can be used to dump information from the database. You can find the password for this account in `/etc/mysql/debian.cnf`.

Once connected, we should first retrieve the version of the database:

```
mysql> select @@version;
+-----+
| @@version          |
+-----+
| 5.1.63-0+squeeze1   |
+-----+
1 row in set (0.00 sec)
```

Based on this version, we can then search for any vulnerabilities in this release.

Next, we should get the list of users and their passwords:

```
mysql> SELECT Host, User, Password FROM mysql.user;
+-----+-----+-----+
| Host      | User                | Password                                |
+-----+-----+-----+
```

```

| localhost | root          |          |
| 127.0.0.1 | root          |          |
| localhost | debian-sys-maint | *70B6D2E96920B5E2BEE4AE1C719752A021342EF9 |
| localhost | wordpress     | *C260A4F79FA905AF65142FFE0B9A14FE0E1519CC |
+-----+-----+-----+
4 rows in set (0.00 sec)

```

Here we can see the following users have access to the database:

- root from localhost;
- debian-sys-maint;
- wordpress;

On MySQL, two algorithms are used to encrypt passwords:

- the old one, with this algorithm, encrypted passwords look like 43e9a4ab75570f5b.
- the new one, with this algorithm passwords will look like \*4ACFE3202A5FF5CF467898FC58AAB1D61502944.

You can easily test this once you are connected to the database:

```

mysql> select password('admin') ;
+-----+
| password('admin') |
+-----+
| *4ACFE3202A5FF5CF467898FC58AAB1D615029441 |
+-----+

mysql> select old_password('admin') ;
+-----+
| old_password('admin') |
+-----+
| 43e9a4ab75570f5b      |
+-----+

```

The new version of the algorithm is obviously stronger than the old version and should be used.

Using John-The-Ripper, you will be able to test the strength of the MySQL passwords, you just need to copy them in a file with the correct format, as shown below:

```
$ cat mysql-password  
wordpress:*C260A4F79FA905AF65142FFE0B9A14FE0E1519CC
```

And crack them using the command `mysql-sha1` (available in the community version).

*Make sure you keep the `` in the password when you save it to a file.*

Another important check is to review what users have the `FILE` privilege, as this privilege can be used by a user to access or create files on the system using MySQL. The list of users having the `FILE` privilege can be reviewed using the following SQL query:

```
mysql> SELECT user,file_priv FROM mysql.user WHERE FILE_PRIV='Y';  
  
+-----+-----+  
| user          | file_priv |  
+-----+-----+  
| root          | Y         |  
| root          | Y         |  
| root          | Y         |  
| debian-sys-maint | Y         |  
| wordpress     | Y         |  
+-----+-----+  
5 rows in set (0.00 sec)
```

Here we can see that the user `wordpress` has `FILE` privilege, this user is just used by the blog to store information, the FILE privilege is not needed here and should be removed.

*An attacker accessing or creating files using MySQL will be restricted by the privileges provided to the `mysql` user and won't be able to access `/etc/shadow` for example.*

## APACHE CONFIGURATION

Since the server is used as a web server, it is important to review the configuration of the web server: Apache.

First, it is always good to check which user the web server is running as, you can find out by running `ps` or by reviewing Apache's configuration and search for the values `User` and `Group`. Here they are available in `/etc/apache2/apache2.conf`:

```
# These need to be set in /etc/apache2/envvars  
User ${APACHE_RUN_USER}  
Group ${APACHE_RUN_GROUP}
```

We can see that the values used are extracted from `/etc/apache2/envvars`:

```
export APACHE_RUN_USER=www-data
export APACHE_RUN_GROUP=www-data
```

All good, we have here a traditional Debian setup and the service is not running as **root**. Running a server as **root** will increase the consequence of a bug's exploitation in the service itself or in any web applications hosted.

To ensure that no local users can modify the hosted website and to harden exploitation of possible vulnerability, it's always a good idea to limit access to the web root. Here, we can see that the **DocumentRoot** is **/var/www/wordpress**, using **ls**, we can check that the permission are not correctly restricted:

```
# ls -lR /var/www/wordpress
/var/www/wordpress/:
total 125
-rwxrwxrwx 1 www-data www-data 395 Jul 11 07:02 index.php
-rwxrwxrwx 1 www-data www-data 19929 Jul 11 07:02 license.txt
-rwxrwxrwx 1 www-data www-data 9177 Jul 11 07:02 readme.html
-rwxrwxrwx 1 www-data www-data 4264 Jul 11 07:02 wp-activate.php
drwxrwxrwx 9 www-data www-data 1913 Aug 6 03:26 wp-admin
-rwxrwxrwx 1 www-data www-data 1354 Jul 11 07:02 wp-app.php
-rwxrwxrwx 1 www-data www-data 271 Jul 11 07:02 wp-blog-header.php
-rwxrwxrwx 1 www-data www-data 3522 Jul 11 07:02 wp-comments-post.php
-rwxrwxrwx 1 www-data www-data 3456 Jul 11 21:51 wp-config.php
[ ... ]
```

The files should not be available to any users. In order to restrict the risk of overwrite by the web server, the complete application ownership can be given to a third user. This way, the web server will not be able to modify the web application. This is however likely to break some applications.

As best practices, it is always good to avoid extra-information in the server headers. To prevent Apache from leaking too much information, the following parameters should be reviewed:

- **ServerTokens**: the recommended value is **Prod**;
- **ServerSignature** should be turned **Off**.

On Debian, these two parameters are configured in the **/etc/apache2/conf.d/security**.

To limit the server exposure to information leak, it is always good to disable directory listing. Directory listing is configured by the parameter **Indexes**. By reviewing the configuration of the website available in **/etc/apache2/sites-enabled/000-default**, we can see that directory listing is enabled:

```
<Directory /var/www/wordpress/>
```

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

You can access the following page to confirm this setting: [<http://vulnerable/wp-admin/images/>](<http://vulnerable/wp-admin/images/>). The configuration should be reviewed to replace `Indexes` by `Indexes`, to prevent potential information leak.

*On this server, SSL is not enabled. If you have to review a SSL configuration on an Apache server, you should make sure that weak ciphers and SSLv2 are disabled and that permissions on the private keys are correctly set.*

We can see that PHP is enabled on this server:

```
# ls /etc/apache2/mods-enabled/php*
/etc/apache2/mods-enabled/php5.conf
/etc/apache2/mods-enabled/php5.load
```

We can also see that PHP is disabled for user public directories:

```
# cat /etc/apache2/mods-enabled/php5.conf
<IfModule mod_php5.c>
    <FilesMatch "\.ph(p3?|tml)$">
        SetHandler application/x-httpd-php
    </FilesMatch>
    <FilesMatch "\.phps$">
        SetHandler application/x-httpd-php-source
    </FilesMatch>
    # To re-enable php in user directories comment the following lines
    # (from <IfModule ...> to </IfModule>.) Do NOT set it to On as it
    # prevents .htaccess files from disabling it.
    <IfModule mod_userdir.c>
        <Directory /home/*/public_html>
            php_admin_value engine Off
        </Directory>
    </IfModule>
```

```
</IfModule>
```

This is a good practice since it will prevent users from running PHP code in their own public directory (`~/username/`), which allows them to jump from their user ID to that of the web server.

## PHP CONFIGURATION

Here, we are going to focus on the PHP configuration used by Apache2. We can see in Apache configuration that the following file is loaded:

```
# cat /etc/apache2/mods-enabled/php5.load  
LoadModule php5_module /usr/lib/apache2/modules/libphp5.so
```

If we run `strings` on that binary and search for apache2, we can see some interesting information:

```
# strings /usr/lib/apache2/modules/libphp5.so | grep apache2  
/etc/php5/apache2  
/etc/php5/apache2/conf.d  
/tmp/buildd/php5-5.3.3/sapi/apache2handler/mod_php5.c  
apache2hook_post_config  
apache2handler  
/tmp/buildd/php5-5.3.3/sapi/apache2handler/sapi_apache2.c
```

We can see two paths that are actually where PHP will look for its configuration. By reading the Debian documentation, we can confirm that the PHP configuration used by Apache2 is stored in the `/etc/php5/apache2` directory.

First, as we saw for Apache, it is always a good idea to hide the value of PHP in use. The parameter `expose_php` tells PHP to add an extra HTTP header `X-Powered-By` in all responses. It is better to hide this information by turning `expose_php` off.

Furthermore, the following options need to be reviewed:

- `display_errors` should be turned `Off` in production;
- `error_reporting` should be set to `E_ALL`;
- `log_errors` should be turned `On`;
- `safe_mode` can be bypassed but it will definitely slow down an attacker; it should be turned `On`;
- `disable_functions` can be used to block access to sensitive functions like: `eval`, `exec`, `passthru`, `shell_exec`, `system`, `proc_open`, `popen`...
- `allow_url_include` should be turned `Off`.

As often on recent version of Debian, Suhosin (<https://suhosin.org/stories/index.html>) is installed on this server. Suhosin adds an extra-layer of protection for the PHP engine and all hosted PHP applications. We will first remove all commented and empty lines, to get a simple export of the Suhosin configuration:

```
# egrep -v -e '^;|^$' /etc/php5/apache2/conf.d/suhosin.ini
extension=suhosin.so

[suhosin]
```

Here, we can see that no options have been set, Suhosin is in its default configuration.

The following options provided by Suhosin can be enabled to increase the security of the PHP application:

- `suhosin.log.syslog` to log to syslog using the value `S_ALL`;
- `suhosin.executor.include.max_traversal` to prevent directory traversal, a value of 3 should prevent limit attacks impact while keeping a working environment;
- `suhosin.executor.disable_eval` and `suhosin.executor.disable_emodifier` to disable the function `eval` and possible code execution using `preg_replace /e` modifier.

More options can be found in Suhosin documentation.

## CRONTAB

Crontab is used on Linux/UNIX to run tasks at a given time. You can find the list of crontabs in `/var/spool/cron/crontabs` or using the command `crontab -u <user> -l` to list the tasks for a given user.

One of the important thing to look for is the permission on scripts called by a task. For example, here the script `/root/backup.sh` is automatically ran in `root`'s task (`/var/spool/cron/crontabs/root`):

```
0 2 * * * /root/backup.sh
```

We can check the permissions on this file and see that anyone can modify this file:

```
# ls -l /root/backup.sh
-rwxrwxrwx 1 root root 84 Aug  6 03:38 /root/backup.sh
```

The permissions on this script should be modified to prevent any user from modifying the script to gain root access the next time the task is ran.

It's also important to avoid sensitive tasks between 2am and 3am or they will be ran 0 or 2 times during daylight savings (expect if the system time zone does not have daylight savings).