

# Identity, Authentication, and Unix

## License



This work by [Z. Cliffe Schreuders](#) at Leeds Metropolitan University is licensed under a [Creative Commons Attribution-ShareAlike 3.0 Unported License](#).

## Contents

[License](#)

[Contents](#)

[General notes about the labs](#)

[Preparation](#)

[Introduction to authentication](#)

[Users and groups](#)


[Users and SSH](#)

[Hashes and salt](#)

[Conclusion](#)

## General notes about the labs

Often the lab instructions are intentionally open ended, and you will have to figure some things out for yourselves. This module is designed to be challenging, as well as fun!

However, we aim to provide a well planned and fluent experience. If you notice any mistakes in the lab instructions or you feel some important information is missing, please feel free to add a comment to the document by highlighting the text and click the comment icon (  ), and I (Cliffe) will try to address any issues. Note that your comments are public.

The labs are written to be informative and, in order to aid clarity, instructions that you should actually execute are generally **written in this colour**. Note that all lab content is

assessable for the module, but the colour coding may help you skip to the “next thing to *do*”, but make sure you dedicate time to read and understand everything. Coloured instructions in *italics* indicates you need to change the instructions based on your environment: for example, using your own IP address.

You should maintain a **lab logbook / document**, which should include your answers to the [questions posed throughout the labs \(in this colour\)](#).

## Preparation

As with all of the labs in this module, **start by loading the latest version of the LinuxZ** template from the IMS system. If you have access to this lab sheet, you can read ahead while you wait for the image to load.

To load the image: press F12 during startup (on the boot screen) to access the IMS system, then login to IMS using your university password. Load the template image: LinuxZ (load the latest version).

Once your LinuxZ image has loaded, **log in using the username and password allocated to you by your tutor**.

The root password -- **which should NOT be used to log in graphically** -- is “tiaspbique2r” (this is a **secure password but is quite easy 2 remember**). Again, never log in to the desktop environment using the root account -- that is bad practice, and should always be avoided.

These tasks can be completed on the LinuxZ system. Most of this lab could be completed on any openSUSE or using most other Linux distributions (although some details may change slightly).

## Introduction to authentication

Authentication plays the important role of verifying an identity. For example, when someone gets into an airplane, sits down at a computer, picks up a mobile device, or uses a website, authentication is what is used to confirm that the person is who they claim to be. Authentication is an important first step *before* deciding how the system should act and what to allow.

## Identity: users and groups

Most computer systems have the concept of a user account. Although some devices such as mobile phones typically only have one user account, most modern computers can support having multiple users, each with their own identity. For example, a

computer can have a separate account for each person that uses it, and if configured to do so may enable each user to have their own account preferences, and access to different resources.

On Unix/Linux systems every user account is identified by a user ID number (UID), which is a 32-bit integer (whole number), and can have one or more user names, which are human readable strings of text.

Open a terminal console.

One way to do this is to start Konsole from KDEMenu → Applications → System → Terminal → Konsole.

Assuming you have already logged in, you have already authenticated yourself on this system.

When and how did you authenticate yourself?

Use these commands to find out about your current identity (or more accurately the identity of the software you are interacting with):

```
whoami
```

```
groups
```

```
id
```

Make a note of your UID and username.

Note that your account is also a member of one or more groups. A primary group, and a list of other groups. Some Linux systems create a new primary group for each user, others such as openSUSE have a shared group (in this case named “users”) that all normal users are a member of. Similar to the relationship between user names and UIDs, each group has a group name, and a group ID (GID).

Information about user accounts is stored in the /etc/passwd file, which typically all users can read.

View the /etc/passwd file:

```
less /etc/passwd
```

Find the line that describes your user account.

This line defines the username, password (well it used to be stored here... we will come back to this), UID, primary group GID, full name, home directory, and shell for your account.

Confirm this matches the information you recorded earlier.

Find the line that describes the root user account.

Where is the root user's home directory?

Press 'q' to quit less.

View the /etc/group file:

```
less /etc/group
```

Groups are defined in this file, along with which users are members.

Which users are members of the "video" group?

Remember, primary groups do not appear in this file; for example, on openSUSE the "users" group, which all normal users are a member of, may not appear in the /etc/group file.

The "su" program can be used to run a program (usually a shell; that is, a command prompt) as another user, effectively enabling users to switch between user accounts at the command prompt.

Change your identity to root. Run:

```
su -
```

Enter the root password.

Use these commands to find out about your new identity:

```
whoami
```

```
groups
```

```
id
```

What is the UID of root? What does this mean about this user?

What gives this user special privileges: the name of the account, or the UID?

Use the `useradd` command to create a new user "fred"

Hint: refer to the man page for `useradd`, by running "`man useradd`".

Change identity to fred.

Hint: "`su - fred`"

Run:

```
id
```

Compare the result to the previous output.

How does this compare to your other normal user account?

What is different, and what about it is the same?

Run the single command "`id`" as root:

```
sudo id
```

What is the difference between `sudo` and `su`?

Which is most likely protect against accidental damage and also log the commands used?

## Users and SSH

Remotely login to the computer at the front of the room using SSH, or alternatively to a classmate's system, or as a last resort, your own system.

If you need to allow someone to ssh to your system, enable the `sshd` service:

```
sudo /sbin/service sshd start
```

To ssh to a remote system use the command:

```
ssh username@server_IP
```

Where *username* is the username allocated to you.

Display details of all users logged on to the system:

```
who
```

List all the processes run by all users:

```
ps -eo user,comm
```

List all the processes running as root:

```
ps -o user,comm -u root
```

Run a command to list all the processes running as your normal user.

## Passwords, hashes and salt

Given that important security decisions are made based on the user accounts, it is important to authenticate users, to ensure that the subjects are associated with the correct identity.

What are the kinds of factors that can be used to verify a user's identity? Hint: for example, "something they have".

Which category of authentication factors is a password considered to be?

Originally passwords were stored "in the clear" (not enciphered). For example, Multics stored passwords in a file, and once at MIT a software bug caused the password file to be copied to the motd file (message of the day), which was printed every time anyone logged into the system. A solution is not to store the password in the clear. Instead a hash can be computed, using a one way hash function, and stored. When the user enters a password, a new hash is computed and compared to the original.

On Linux, the command "shasum" can be used to check the integrity of files (hash functions have many uses), and works on the same principle. We can use it to generate a hash for any given string, for example a password:

```
shasum
```

Type "hello" without the quotes. Press Ctrl-D (which indicates "EOF"; that is, end of input).

Repeat the above, with the same password ("hello"), and with a slight difference ("hello.").

Are the outputs the same?

Are the different hashes similar?

Is this good? Why?

Which one-way hash function does the `shasum` program use? Would this be a good option for hashing passwords?

For password authentication, the hash still needs to be stored. On Unix, password hashes were once stored in the world-readable file `/etc/passwd`, now they are typically stored in `/etc/shadow`, which only root (the superuser) can access.

Open a new local shell (in Konsole, you can open a new tab using `Ctrl-Shift-T`).

View the shadow file:

```
sudo less /etc/shadow
```

The format of the shadow file is:

```
username:password:last-changed(since 1-1-1970):days-until-may-change:days-until-must-change:days-warning-notice:days-since-expired-account-disabled:date-disable:reserved-field
```

What is the hash of your user account's password?

Exit `less` ("q").

Use the `passwd` command to change your password:

```
passwd
```

When prompted, enter a new password of your choosing.

View the shadow file, and confirm that the stored password has changed.

With reference to the shadow file, and the man page for `crypt` (Hint: "man crypt"), answer these questions:

- On Linux, the password hash stored in `/etc/shadow` has a prefix that specifies the hash function used.  
What hash function is used for your password?
- When was the root password last changed?
- Do any accounts have a setting that will force a password change at a specific date?

A salt is a random string, used as further input into a one-way hash function (concatenated to the password). The salt is typically stored along with the hash. As a result the same password will have different hashes, so long as the salt is different.

Why is that a good thing?

What kind of attack does a salt defend against?

What is the current salt for your account? Hint: it is stored after the second "\$".

## Password weaknesses

The strength of a password depends on its entropy: its degree of randomness. If a user chooses a word from a dictionary, it would not take long to attempt every dictionary word until finding one that results in the same hash.

### If you have not previously cracked passwords using John the ripper:

Try your hand at cracking the passwords on the accounts on the image for this class

Add some new users with these passwords:

Hello

hellothere

H3ll0

hello1

12hell012356

Use john the ripper to crack the passwords.

Hint: "man john", on the LinuxZ image or a Kali Linux system.

- Which passwords are cracked the fastest?
- How long did they take?
- How realistic would it be to try to crack the LinuxZ passwords that are already configured? How long would it take? What could you do to make it as fast as possible? Are there some that have high enough entropy to not be cracked?

## Pluggable Authentication Modules (PAM)

In the past all Linux/Unix programs that required the user to enter a password for authentication (such as su, sudo, and login) would access and interpret /etc/passwd using its own code. However, it was hard to maintain all this code, since any change



in the way the passwords were stored (such as using a shadow file, or using new hash functions) would mean all the software that provides authentication needed to be changed. The solution to the problem was PAM.

*Pluggable Authentication Modules (PAM)* enables applications that make use of authentication to be independent of the specific authentication schemes in use. For example, a program such as a login screen that uses PAM can be configured to authenticate using a password, smartcard, and/or biometrics, simply by changing PAM configuration files.

PAM is supported in most distributions of Linux, Mac OS X, FreeBSD, and many other Unix-like systems.

View which PAM modules are available on the system:

```
/usr/sbin/pam-config --list-modules
```

The .so files are typically in /lib/security or /lib64/security. List them:

```
ls /lib*/security/
```

As you can see, there are lots of different features and authentication schemes, and these can be used with *any* PAM compatible program. This includes not only typical authentication schemes, such as pam\_unix2.so, which does the usual password comparison with /etc/passwd and /etc/shadow, but also can impose time limits (pam\_time.so) or simply display messages to the user (pam\_motd.so).

It is possible to determine whether a specific program is compiled to use PAM, by checking what dynamic libraries it uses. (On Linux .so shared objects are similar to DLL files on Windows, they contain library code that programs can reuse). Check what shared objects the passwd program uses:

```
which passwd
```

```
ldd path_to_passwd
```

Where path\_to\_password is the output from the “which passwd” command above, which identifies the absolute path to a program. For example, “ldd /usr/bin/passwd”

Note that the output will include a line starting with “libpam.so”, such as:

```
libpam.so.0 => /lib64/libpam.so.0 (0x00007fcc4afe6000)
```

This would indicate that the program loads code from “/lib64/libpam.so.0”, and does indeed make use of PAM.

PAM configuration is located in /etc/pam.d. Take a look at which programs currently have pam configuration files:

```
ls /etc/pam.d
```

Depending on what is installed on the system, there will be a few configuration files. Each file contains a PAM configuration for the program it is named after. If a PAM-aware program does not have a configuration file the “other” file is used, which should deny access by default.

View the “other” file:

```
less /etc/pam.d/other
```

```
#!/PAM-1.0
auth    required      pam_warn.so
auth    required      pam_deny.so
account required      pam_warn.so
account required      pam_deny.so
password required     pam_warn.so
password required     pam_deny.so
session required      pam_warn.so
session required      pam_deny.so

/etc/pam.d/other lines 1-10/10 (END)
```

The output contains instructions to log the attempt to Syslog (which is what pam\_warn.so does) and deny access (using pam\_deny.so) any attempt to authenticate (the lines starting with “auth”), request access to anything (account), change passwords (password), or starting a session (session).

The syntax of the configuration file is that each line starts with is typically:

*type control module-path module-arguments*

The type is auth, password, account or session. The control (such as “required” or “optional”) defines whether the module needs to pass or not before moving on to the next module, then the module name is defined. It is not shown in this example, but the module-path can be followed with some settings for the module.

All the modules for a type (such as auth) are called a module stack. When the program requests PAM perform authentication each of the auth modules in the module stack are run in the order they appear. If a “required” module fails, the authentication process tries the next module to see if it passes, if no required modules pass, then the authentication fails. Note that in the example above, pam\_warn.so returns “PAM\_IGNORE”. Therefore the next module pam\_deny.so is started, which returns an error, and the authentication fails.

Possible control values include:

**required**

failure of such a PAM will ultimately lead to the PAM-API returning failure but only after the remaining stacked modules (for this service and type) have been invoked.

**requisite**

like required, however, in the case that such a module returns a failure, control is directly returned to the application.

**sufficient**

success of such a module is enough to satisfy the authentication requirements of the stack of modules (if a prior required module has failed the success of this one is ignored). A failure of this module is not deemed as fatal to satisfying the application that this type has succeeded. If the module succeeds the PAM framework returns success to the application immediately without trying any other modules.

**optional**

the success or failure of this module is only important if it is the only module in the stack associated with this service+type.

**include**

include all lines of given type from the configuration file specified as an argument to this control.

-- from the man page for pam.conf

There is also a more complex rule syntax available, described in the man page.

Information about each module is available in The Linux-PAM System Administrators' Guide:

[http://linux-pam.org/Linux-PAM-html/Linux-PAM\\_SAG.html](http://linux-pam.org/Linux-PAM-html/Linux-PAM_SAG.html)

Look at which authentication methods are used by passwd:

```
less /etc/pam.d/passwd
```

Note that this indicates that PAM will apply the rules in “/etc/pam.d/common-auth”, “common-account”, “common-password”, and “common-session” for the passwd program.

Edit the rules in common-password:

```
sudo vi /etc/pam.d/common-password
```

Edit the line (adding “minlen=10” to the end of the line):

```
password      requisite      pam_pwcheck.so  cracklib
minlen=10
```

**Reminder:** Vi is ‘modal’: it has an insert mode, where you can type text into the file, and normal mode, where what you type is interpreted as commands. Press the “i” key to enter “insert mode”. Type your changes to the file, then exit back to “normal mode” by pressing the Esc key. Now to exit and save the file press the “:” key, followed by “wq” (write quit), and press Enter.

Change a password using `passwd`.

Confirm that normal users can no longer use a password that is less than 10 characters long.

View the man page for this PAM module:

```
man pam_pwcheck
```

Based on the options described in the man page, configure the `pam_pwcheck` module to remember the last three passwords used, to prevent them from being reused.

Confirm that normal users can no longer reuse a recent password.

Note that there is likely no lockout for failed password attempts when using `su` to change user. Confirm this by running this command a few times:

```
su - student; su - student; su - student; su - student;
```

When prompted each time, enter an incorrect password.

Now, let's assume our aim is to add a 5 minute lockout time for when a user enters the wrong password 3 times in a row when using "su". Log-tally lockouts can be achieved using pam\_tally2.

View the man page for this PAM module:

```
man pam_tally2
```

Edit /etc/pam.d/su-l, and insert this line as the first module (after the comment at the start of the file):

```
auth required pam_tally2.so deny=3 unlock_time=300
```

Figure out how I configured the LinuxZ system to automatically create a user home directory the first time a user logs in.

Hint: look through the files in /etc/pam.d

Finally, apply what you have learned to **configure PAM to only allow:**

- **The user "dropbear" to login between 9am and 5pm**
- **And only on a Tuesday**

For testing purposes add another rule that allows a user to log in only 10 minutes from the current time

Hint: use pam\_time.so and edit /etc/security/time.conf (and read the documentation in the configuration file to figure out how to set the correct limits)

Another hint: try the "account" type.

Configure a cron job to force the user to logout at the end of their allowed time

Hint: as root, "crontab -e" and add a job to run at 5pm on Tuesday killing all of their processes

For example, to disconnect dropbear on Wednesday at 4pm, add this line to cron (run "crontab -e", then press "i" to add):

```
00 16 * root * wed skill-KILL-u dropbear
```

Try to figure out how to also send a warning to the user 10 minutes before they are kicked off (Hint: add another cron job, you could send a message via "wall", etc)

## securetty and security

Backup the "securetty" file, which specifies which virtual terminals root is allowed to login to:

```
cp /etc/securetty /etc/securetty.backup
```

Edit /etc/securetty and place a "#" in front of "tty3"

Press Ctrl-Alt-F3 and try to login as root

Press Ctrl-Alt-F4 and try to login as root

Press Ctrl-Alt-F7 to return to the desktop

## The (optional) extra mile: SEED Lab (Computer Security Education)

This lab is a look at PAM from a programmer's point of view. **Recommended** if you are happy to do some programming in C. The lab involves examining some C code that uses PAM, and understanding the detail of what it is doing.

"In many programs, authentication are hard-coded. To use another authentication scheme, the programs need to be rewritten. PAM provides a way to develop programs that are independent of authentication scheme. These programs need "authentication modules" to be attached to them at run-time in order to work. Which authentication module is to be attached is dependent upon the local system setup and is at the discretion of the local system administrator."

Lab available under the GNU Free Documentation License

<http://www.cis.syr.edu/~wedu/seed/Labs/PAM/>

These SEED labs should run in most Linux systems, if you want or need their exact setup you can follow the instructions here: [http://www.cis.syr.edu/~wedu/seed/lab\\_env.html](http://www.cis.syr.edu/~wedu/seed/lab_env.html)

## Conclusion

At this point you have:

- Applied authentication concepts to Unix/Linux
- Experimented with user accounts and identity
- Experimented with one-way hash functions, salts, and password storage
- Cracked passwords with low entropy using dictionary attacks
- Configured PAM to apply various authentication techniques to programs such as passwd

Well done!