Hardening the user accounts

Fortificación de S.O.
Master en Seguridad Informática. 2023/2024
Universidade da Coruña
Universidade de Vigo

Antonio Yáñez Izquierdo
José Rodríguez Pereira

Contents I

1 Introduction to users and groups
- users
- groups
- user and group definition files

2 Authentication: PAM modules
- Introduction to PAM
- Configuration of PAM
- `/etc/pam.conf`
- `/etc/pam.d` directory
- PAM modules

3 User accounts related vulnerabilities

4 Hardening authentification

5 Limiting privileges. Restricted shells

6 Becoming *root*
- login directly as root and the *su* command

7 sudo and *sudoers*

Introduction to users and groups

Introduction to users and groups
→users

what is a user?

- user accounts are the mean by which *real world users* present themselves to the system and are granted (or denied) access to it
- *authentification* is the process by which the system verifies that a user is who he/she claims to be
- a user in the system **may or may not** be a *real person*

users

- A *user* is an entity in the system that can
  - Own files (and/or directories, devices ...)
  - Create processes and execute programs
- A user is identified in the system by a number (*uid*)
- File permissions specify what a user can do to a spcific file
- There's a special user in the system (*uid=0*) that can access all files, signal all processes and execute all system calls
- Every file in the system is **owned** by **ONE** user
- Process credentials indicate which user is behind the execution of that process

privilege separation

- some users (sometimes known as *pseudousers*) exist only to execute specific services and own the files associated with those services.
  - **Example:** user *www-data* run the web server but is not associated with any individual person
  - This is done to increase system security: if the services were to be run by the *root* user and had some security issue that could be exploited, the root account-and thus the whole system, would be compromised. This way only the www-data account would be compromised in case such situation arised

username and UID

- each user has a name that identifies it, called *username*
- when adding a user, the system administrator has to provide both a username and a user identification number (UID)
- the system uses the UID (not the username) internally. The username is just *mapped* to the UID.
- when adding a user, the system administrator also assigns this user to one or more groups

Introduction to users and groups
→groups

what is a group?

- a group is a collection of users gathered together for *whatever reason*
- a group is identified by a groupname and internally by a **G**roup **ID**entification number, GID
- one group can have one or more users. Users are said to *belong* to that group
- one user can belong to one or more groups, although one of them is called the *primary group* of the user: the one defined in the `/etc/passwd` file
- the user and group behind the execution of a process determine which files in the system the process can access

Introduction to users and groups
→user and group definition files

userdefinition files

- the user information of the users **defined locally** in one system resides in the following ASCII text files
  - **/etc/passwd**
    This file defines the user acounts in the system. One line per user, constituted by fields separated by : On older systems the *crypted form of the password* (strictly speaking, the result of crypting a base text using the password as key) was stored here as well. Example:
    ```
    root:x:0:0:the_almighty_system_administrator:/root:/bin/bash
    daemon:x:1:1:daemon:/usr/sbin:/bin/sh
    ```
  - **/etc/shadow** (only on newer systems)
    Pasword definition file, one line per user. Example
    ```
    root:$6$pz0jXkuY$6M71ZfZk1ecQv...sxodXCCZh5CVeR.DxQ1bOHn37t5OL.:14578:0:99999:7:::
    daemon:*:14578:0:99999:7:::
    ```

group definition files

- **/etc/group**
  group definition file, one line per group. Example
  ```
  wheel:*:0:root,antonio
  daemon:*:1:daemon
  ```
- in linux we also have the file **/etc/gshadow** for group passwords. linux also has the *administrator(s) of a group*: a user(s) that can change de *password* of a group and/or modify its member list

Authentication: PAM modules

Authentication: PAM modules
→Introduction to PAM

What is PAM?

- PAM stands for **P**luggable **A**uthentication **M**odules
- Provides a way of changing the authentication machanisms without changing the applications
- is a generalized API for authentication-related services
    - allows a system administrator to add new authentication methods simply by installing new PAM modules
    - allows a system administrator to modify authentication policies by editing configuration files
- available in most linux distributions

What is PAM?

- lets consider the *login* program
    - once it reads the password, it compares its crypted form with the one in the `/etc/passwd` (or `/etc/shadow`) file
    - a change in the way the *crypted* password is stored or the way it is crypted would make necessary to recompile the login program
- Solution: PAM
    - PAM provides a library of functions that an application may use to request that a user be authenticated
    - changing anything in the authentication process would mean to change the PAM library, no the aplications: in fact most of the changes in the authentication process can be made by just changing PAM configuration

Authentication: PAM modules
→Configuration of PAM

Configuration of PAM

- There are different implementation of PAM and their configuration can differ slightly in
  - the location and format of the configuration file(s)
  - the location of the PAM library
  - list of available modules
- there is however a thing in common: lack of configuration means no authentication
  - Deleting PAM configuration file(s) locks you out of the system

PAM facilities

- we designate as facilities each of the tasks that PAM can deal with. These are
  - **authentication management (auth):** to determine whether the user is who he/she claims to be
  - **account management:** to handle non-authentication-related issues of account availability (for example, login at only certain hours or from certaing machines)
  - **session management:** to perform tasks associated with session set-up and tear-down, such as login accounting, stablishing resource limits...
  - **password management:** to change the authentication token associated with an account

PAM modules

- A PAM module is a self-contained piece of program code that implements the primitives in one or more facilities for one particular mechanism
- For a particular facility a module can be considered
  - **sufficient:** if this module grants access, access is granted, no more modules are checked
  - **requisite:** if this module denies access, access is denied, no more modules are checked
  - **required:** this module must grant access, and the evalution continues with the following modules
  - **optional:** the result of this module will be used only if the result of no other modules is deterministic
  - **[new syntax]**: set of pairs of values

PAM modules: new syntax

- apart from *sufficient*, requisite, required and optional, the control field in a pam configuration file can have the form

```
[value1=action1 value2=action2 ...valueN=actionN]
```

- where vauleJ can be one of the following: *success*, *open_err*, *symbol_err*, *service_err*, *system_err*, *buf_err*, *perm_denied*, *auth_err*, *cred_insufficient*, *authinfo_unavail*, *user_unknown*, *maxtries*, *new_authtok_reqd*, *acct_expired*, *session_err*, *cred_unavail*, *cred_expired*, *cred_err*, *no_module_data*, *conv_err*, *authtok_err*, *authtok_recover_err*, *authtok_lock_busy*, *authtok_disable_aging*, *try_again*, *ignore*, *abort*, *authtok_expired*, *module_unknown*, *bad_item*, *conv_again*, *incomplete*, and *default*.
    - *default* stands for all values non explicitly listed.

PAM modules: new syntax (continuation)

- where actionJ can be one of the following

ignore	when used with a stack of modules, the module's return status will not contribute to the return code the application obtains.
bad	this action indicates that the return code should be thought of as indicative of the module failing. If this module is the first in the stack to fail, its status value will be used for that of the whole stack.
die	equivalent to bad with the side effect of terminating the module stack and PAM immediately returning to the application.

PAM modules: new syntax (continuation)

ok	this tells PAM that the administrator thinks this return code should contribute directly to the return code of the full stack of modules. In other words, if the former state of the stack would lead to a return of PAM_SUCCESS, the module's return code will override this value. Note, if the former state of the stack holds some value that is indicative of a modules failure, this 'ok' value will not be used to override that value.
done	equivalent to ok with the side effect of terminating the module stack and PAM immediately returning to the application.
reset	clear all memory of the state of the module stack and start again with the next stacked module.

PAM modules:new syntax (continuation)

- In fact, the control words *sufficient*, *requisite*, *required* and *optional* can be expressed in the new syntax, as follows
  - **[required]** [success=ok new_authtok_reqd=ok ignore=ignore default=bad]
  - **[requisite]** [success=ok new_authtok_reqd=ok ignore=ignore default=die]
  - **[sufficient]** [success=done new_authtok_reqd=done default=ignore]
  - **[optional]** [success=ok new_authtok_reqd=ok default=ignore]

A little example

- Consider the following example related to the *su* service

```
auth    sufficient    pam_rootok.so
auth    required      pam_wheel.so
auth    required      pam_unix.so
```

- inferring what the modules do from their name, this configuration of the *su* service states that
  - the access would be granted directly for the root user
  - other users would have to both belong to the wheel group and enter the correct password

PAM files

- in linux system the module files are usually located
  - `/lib/security`
  - `/lib64/security`
  - `/lib/x86_64-linux-gnu/security` (debian 12)
- Configuration files for different services are located in the `/etc/pam.d` directory
  - there is one file per service to be configured
  - the file is named after the service it configures
- If the aforementioned directory does not exist, configuration will be in the `/etc/pam.conf` file

Authentication: PAM modules

$\rightarrow$`/etc/pam.conf`

/etc/pam.conf file format

- plain text file.
- lines starting with # are comments
- each line has the format

```
service_name     facility  control_flag   module  options
```

`/etc/pam.conf` file format

- `service_name` is the name of the service to be configured, for example *sshd*, *telnetd*, *su* . . .
- `facility` is one of: *auth*, *session*, *account*, *password*
- `control_flag` states how the module affects the facility for that service, and can be: *sufficient*, *requisite*, *required*, *optional*
- `module` is the name of the modules (older versions of PAM used the complete path to the modules files)
- `options` are the parameters passed to the module in case the module accepts (or requires) options to be passed to it.

/etc/pam.conf example

- The previous example would look like this in an /etc/pam.conf file

```
su      auth     sufficient     pam_rootok.so
su      auth     required       pam_wheel.so
su      auth     required       pam_unix.so
```

- services not explicitly defined use the modules defined in the *“other“* section

Authentication: PAM modules
→`/etc/pam.d` directory

/etc/pam.d

- should this directory exist, the `/etc/pam.conf` is not read
- there is a plain text file in the directory `/etc/pam.d` for each service to be configured
- each line in these files
  - is considered a comment if it starts with #
  - has the format

    ```
    facility  control_flag   module    options
    ```
  - the following sintax causes to include another configuration file in the present service (useful to have common policies for different services)

    ```
    @include other_file_in_the_pam.d_directory
    ```

Authentication: PAM modules

→PAM modules

PAM modules

- the list of PAM modules dependes on the PAM implementation
    - each pair OS/PAM implementation may have a different set of modules
- info on the modules can be obtained with the man page
- there are, however, modules that are common to almost every implementation
- some modules with the same name behave differently on different implementations

some common PAM modules

- `pam_deny` locks out PAM module
- `pam_getenv` returns the value for a PAM environment name
- `pam_rhosts` `pam_rhosts_auth` the rhosts PAM module
- `pam_unix` `pam_unix_auth` PAM authentication module for UNIX
- `pam_winbind` PAM module for winbind

basic linux PAM modules

- `pam_permit` always grants access
- `pam_deny` locks out PAM modules.
- `pam_access` delivers log-daemon-style login access control using login/domain names depending on pre-defined rules in /etc/security/access.conf.
- `pam_cracklib` checks the passwords against the password rules.
- `pam_env` sets/unsets environment variables from `/etc/security/pam_env_conf`.
- `pam_debug` debugs PAM.

basic linux PAM modules

- `pam_echo` prints messages.
- `pam_exec` executes an external command.
- `pam_ftp` is the module for anonymous access.
- `pam_localuser` requires the user to be listed in `/etc/passwd`.
- `pam_unix` provides traditional password authentication from `/etc/passwd`.

list of linux PAM modules I

```
pam_access (8)       - PAM module for logdaemon style login access control
pam_ck_connector (8) - Register session with ConsoleKit
pam_debug (8)        - PAM module to debug the PAM stack
pam_deny (8)         - The locking-out PAM module
pam_echo (8)         - PAM module for printing text messages
pam_env (8)          - PAM module to set/unset environment variables
pam_exec (8)         - PAM module which calls an external command
pam_filter (8)       - PAM filter module
pam_ftp (8)          - PAM module for anonymous access module
pam_getenv (8)       - get environment variables from /etc/environment
pam_group (8)        - PAM module for group access
pam_issue (8)        - PAM module to add issue file to user prompt
pam_keyinit (8)      - Kernel session keyring initialiser module
pam_lastlog (8)      - PAM module to display date of last login
pam_limits (8)       - PAM module to limit resources
pam_listfile (8)     - deny or allow services based on an arbitrary file
pam_localuser (8)    - require users to be listed in /etc/passwd
pam_loginuid (8)     - Record user's login uid to the process attribute
pam_mail (8)         - Inform about available mail
```

list of linux PAM modules II

```
pam_mkhomedir (8)    - PAM module to create users home directory
pam_motd (8)         - Display the motd file
pam_namespace (8)    - PAM module for configuring namespace for a session
pam_nologin (8)      - Prevent non-root users from login
pam_permit (8)       - The promiscuous module
pam_pwhistory (8)    - PAM module to remember last passwords
pam_rhosts (8)       - The rhosts PAM module
pam_rootok (8)       - Gain only root access
pam_securetty (8)    - Limit root login to special devices
pam_selinux (8)      - PAM module to set the default security context
pam_sepermit (8)     - PAM module to allow/deny login depending on SELinux en..
pam_shells (8)       - PAM module to check for valid login shell
pam_tally (8)        - The login counter (tallying) module
pam_time (8)         - PAM module for time control access
pam_timestamp (8)    - Authenticate using cached successful authentication at..
pam_umask (8)        - PAM module to set the file mode creation mask
pam_unix (8)         - Module for traditional password authentication
pam_userdb (8)       - PAM module to authenticate against a db database
pam_warn (8)         - PAM module which logs all PAM items if called
pam_wheel (8)        - Only permit root access to members of group wheel
```

list of linux PAM modules III

```
pam_winbind (8)       - PAM module for Winbind
pam_xauth (8)         - PAM module to forward xauth keys between users
```

User accounts related vulnerabilities

User accounts related vulnerabilities

- When we thing of the vulnerabilities associated with user accounts we find
  a) Some user is not who he/she claims to be
  b) Some user is missusing (or overusing) system resources causing the system to not perform correctly
  c) Some user has more privileges than she/he strictly needed to perform his/her task
  d) Use of the administrator (*root*) account

Hardening authentication

- In order to prevent **a)** we can harden the autentication procedures
- This is usually done through the PAM modules
  - We can impose rules on password generations
  - we can limit the terminals from which the user can login (the times at which login happens . . .)
  - We use logging to dectect strange behaviours
- We also have utilities (i.e. *john*) to check for password weaknesses

Limiting resource usages

- To prevent **b)** we can impose limits on the resources used by a user account in several ways
  - through some pam modules (for example **pam_limits**) we can limite the maximum simultaneous login sessions, the maximun file size, the maximun memory usage . . . for a user
  - use of quotas on the filesystems (to be seen in the filesystem lesson)
  - through some applications (i.e. *setcpulimits*) or the control groups (*cgroups*)

Limiting user privileges

- We can further limit what a user can do in the system using a restricted shell
- In additiong to hardening authentification or limiting the resources a user account can use, a restricted shell goes further by
  - defining exactly the set of executables the user can execute in the machine

The administrator *root* account

- What has been said concerning the normal user accounts is valid also for the root account
  - we use pam modules to restrict access to the root account directly from terminal, by so doing we force to login first as a normal user and then use the *su* command
  - we use pam modules to restrict access to the root account to members of certain groups
  - as the *root* account has **ALL** the privileges we usually allow some users limited administrator access (to perfrom only certain tasks) via the *sudo* command and the *sudoers* definition

Hardening authentification

Hardening authentification

- most of the hardening is used through the pam modules called in `/etc/pam.d/login` `/etc/pam.d/common-auth` y `/etc/pam.d/common-password`
- the graphical login is handled by `/etc/pam.d/lightdm`, `/etc/pam.d/slim`,`/etc/pam.d/gdm` `/etc/pam.d/xdm`. . . depending po the graphical login program used
- IMPORTANT: should we change the graphical login program, the corresponding authentication configuration might be different
- We can also enable two factor authentication with the appropiate PAM modules

Hardening authentification

- typically used pam modules
    - **pam_unix** password hash definition and some password characteristics
    - **pam_pwquality** additional password characteristics
    - **pam_pwhistory** to disallow recycling of passwords
    - **pam_securetty** to limit root logins from certain devices
    - **pam_faildelay** to establish delay after failed login attempts
    - **pam_google_authenticator** for two step verification

Limiting privileges. Restricted shells

Limiting privileges. Restricted shells

- if we really want to narrow down the number of things a user can do in a system we can make use of restricted shells
- there are restricted versions of several shells
  - *'rksh'* or *'ksh -r'* for the restricted version of ksh
  - *'rbash'* or *'bash -r'* for the restricted version of bash
- we define a restricted shell as the user's login shell (either directly in `/etc/passwd` or using the *chsh* command
- configuring it adecuately we can in fact reduce what the user can do in the system

What is a restricted shell?

- a restricted shell is a shell with the following characterstics
  - it does not allow the use of *'cd'* command, thus restricting the user to stay confined in a single directory
  - it does not allow to execute anything with a slash ('/') in its name, thus restricting the user to execute only what it is in his/her PATH
  - it does now allow to modify environment variables such as the **PATH**
  - it does not allow to redirect standard input, output or error using any of the redirection operators
  - it does not allow to exit the restricted mode nor directly (with **set +r**) neither within the use of scripts

How can what the user can do be limited with a restricted shell?

- we put the restricted shell as his/her login shell
- we create a `/bin` directory under his/her home directory
- we create symbolic links to the programs we allow him/her to execute in his $HOME/bin
- we create his/her shell's configuration files (for example `.bash_profile` and `.bashrc`) and make them root owned and non writable

How can what the user can do be limited with a restricted shell?

- in these files we define the PATH to be $HOME/bin
- we give his/her $HOME the adecuate ownership and premissions so that he/she may write to it but cannot delete shell configuration files
- should the user be allowed to execute a program that allows shell escapes or things like that, we revise its configuration and, if necessary, we create an adecuate (ownership and permissions) non writable configuration file

Becoming root

Becoming *root*

- in order to do system configuration and maintenance we have to use the administrator account, and in linux system that is become *root*
- we **MUST** use the *root* account **ONLY** to do administrative tasks, and **NEVER** for other use of the system
- there are three ways to become *root*
  - login directly as *root*
  - use the `su`
  - use the `sudo` command

Becoming root

→login directly as root and the su command

login directly as *root*

- login directly as *root* **SHOULD BE DISABLED**
  - anyone knowing (or guessing) the *root* password would become root
  - no trace is left on the system of who has becomed root
- the usual thing to do is to become *root* through the su command and allowing that only to certain users (maybe just one) in the system (see module `pam_wheel`). So to become root with *su* someone has to
  - have a valid account in the system
  - be a member of the group of allowed users
  - know the *root* passwd

How do I disable login directly as *root*

- we configure the file `/etc/securetty` adecuately
- we list the module *pam_securetty* as required in
    - `/etc/pam.d/login`
    - `/etc/pam.d/`*whatever-graphical-login-program*
- (other modules con be used for the same purpose, `pam_shells`, `pam_succeed_if` ...
- if the machine is running a *ssh* server we should also disable direct root login via ssh in the *sshd configuration* (typically `/etc/ssh/sshd_config`)

sudo and sudoers

sudo and *sudoers*

- one problem with the *su* command is that it gives you access to the root account in an **all or nothing** fashion
- if you become *root*, you have **ALL** the privileges of the root account.
- maybe we'd like to just allow some users to perform certain administration task. The *sudo* command allows a user, after authenticating as his/herself, execute some command with administrator privileges, provided the *sudores* file allows him to. Example

```
user@somemachine $ sudo shutdown -h now
```

sudo

- the general syntax of the sudo command is

  ```
  sudo targetuser command
  ```

  so, provided that the user issuing the command is authorized to run *command* as *targetuser* in the *sudoers* file
  - user will be prompted for HIS/HER password (not targetuser's)
  - *command* will be executed with targetuser's credentials

sudoers file

- usually located at `/etc/sudoers`. Configuration can be appended at `/etc/sudoers.d`
- should not be edited directly but with the command *visudo*
  - *visudo* checks the syntax is correct before saving the file. In case there's an error in the syntax the sudo command will be disabled, so *visudo* prevents us from accidentally disabling *sudo*
- this file is formed by a series of lines in the form

```
user-spec   host-spec = (runasuser-spec) command-spec
```

- this sample line allows user antonio to run the command shutdown as root in host abyecto

```
antonio  abyecto=(root) shutdown
```

- as the *sudoers* file is checked locally, the host-spec only makes sense when we have a common *sudoers* file for several machines

sudoers file

- the `user-spec` can be an username, an #userid, a %groupname or a %#groupid, an *alias* or a list of those elements separated by comma (**,**)
- the `host-spec` can be a hostname, a qualified hostname, a host address, a network address, an *alias* or a list of those elements separated by comma (**,**)
- the `runasuser-spec` can be an username, an #userid, a %groupname or a %#groupid, an *alias* or a list of those elements separated by comma (**,**)
- the `command-spec` can be a command name, an alias or a list of those elements separated by comma (**,**)
- any of those `*-spec` can be '**ALL**', specifying any user, host, or command

sudoers file

- aliases can be defined with
  TypeOfAlias ALIASNAME = list of members in that alias
  where TypeOfAlias can be *User_Alias*, *Runas_Alias*, *Host_Alias* and *Cmnd_Alias*
- The following example shows how to allow users *pepe, pepa and user2* to execute any of the commands that can power down the machine *rutercillo*

```
User_Alias  DOWNDOERS =  pepe, pepa, user2
Cmnd_Alias  POWERDOWN = /sbin/shutdown, /sbin/halt, /sbin/reboot, /sbin/resta

DOWNDOERS  rutercillo=(root) POWERDOWN
```