Users and Groups

Marek Kozłowski

Faculty of Mathematics and Information Sciences
Warsaw University of Technology

Outline of the Lecture

Outline of the lecture

Local Users Database

- For most systems user identity database consists of the following three files:
  - `/etc/passwd` – provides user account information,
  - `/etc/shadow` – stores encrypted passwords and aging limits (some BSD systems use `/etc/master.passwd` for this purpose),
  - `/etc/group` – specifies system groups.
- All those are ASCII files in which each line defines a separate entry; fields are colon-separated.

Account Information – */etc/passwd*

/etc/passwd line format

```
username:password:uid:gid:gecos:directory:shell
```

- The field descriptions are:
  - *username* – account name,
  - *password* – a password placeholder (usually 'x') for backward compatibility,
  - *uid* – user's ID – a unique numerical identifier,
  - *gid* – numerical ID of the user's primary group,
  - *gecos* – comment / real name,
  - *directory* – user's home directory,
  - *shell* – the program to run at login – the default shell (for Linux it is usually `/bin/bash`).

Sample */etc/passwd* entry

```
smithj:x:1001:100:John Smith:/home/smithj:/bin/bash
```

Username vs. UID

- Records stored in `/etc/passwd`, `/etc/shadow` and `/etc/group` are identified and linked to each other by names.
- During login users are prompted for names.
- For all other purposes, that is for storing file and process ownership information and for resource access control IDs are used, although most utilities display names assigned to them.
- Note that IDs stored in memory or file systems don't require corresponding entries in user identity databases. In such case numerical values are displayed by utilities.
- Two users with the same UID are undistinguishable for the system.

Account Types

- Three kinds of accounts can be distinguished:
  - `root` (UID: 0, GID: 0) – a superuser with unlimited privileges,
  - regular users,
  - *system accounts*.
- Each process runs on behalf of some user and with their privileges. It is extremely unsafe to run services like: web server, SQL server, mail server, print server etc. which require limited access to resources with root privileges. For each such service a separate account (system account) is created. Fake login shells and invalid passwords prevent normal login as such users.
- It's a safe practice to leave UIDs up to 1000 for system accounts and GIDs up to 100 for pre-defined groups added automagically during software installation.

Login Shells – */etc/shells*

- `/etc/shells` stores all valid login shells.
- Users that have their login shells (set in `/etc/passwd`) not listed here are unable to login (start a tty session).
- Typically shells are automagically added here during their installation.
- For shells that have symlinks or hardlinks all names must be listed here.

Accounts – Example

Sample */etc/passwd*

```
root:x:0:0:root:/root:/bin/bash
bin:x:1:1:bin:/bin:/bin/false
daemon:x:2:2:daemon:/sbin:/bin/false
adm:x:3:4:adm:/var/adm:/bin/false
lp:x:4:7:lp:/var/spool/lpd:/bin/false
sync:x:5:0:sync:/sbin:/bin/sync
shutdown:x:6:0:shutdown:/sbin:/sbin/shutdown
halt:x:7:0:halt:/sbin:/sbin/halt
mail:x:8:12:mail:/var/spool/mail:/bin/false
news:x:9:13:news:/usr/lib/news:/bin/false
uucp:x:10:14:uucp:/var/spool/uucppublic:/bin/false
operator:x:11:0:operator:/root:/bin/bash
man:x:13:15:man:/usr/share/man:/bin/false
postmaster:x:14:12:postmaster:/var/spool/mail:/bin/false
nobody:x:65534:65534:nobody:/:/bin/false
smithj:x:1001:100:John Smith:/home/smithj:bin/bash
```

Encrypted Passwords – */etc/shadow*

- All users are able to read the files `/etc/passwd` and `/etc/group` .
- For security purposes encrypted passwords have been moved to a separate file which is not readable by regular users. For most Unix systems this file name is: `/etc/shadow` .
- First three fields contain a username, encrypted password and the date of last password change (0 forces the user to change it upon the next login). Next fields contain aging information and are often empty.
- All dates are expressed as the number of days since Jan 1, 1970 (*Unix epoch*).

Unix Passwords

- Algorithms for password encryption can differ (`man 3 crypt`).
- Encryption is unidirectional (encrypted passwords cannot be decrypted).
- If some users have the same plain passwords then their encrypted passwords usually differ.
- Valid encrypted passwords cannot start with some characters (for example: `!`, `*`); this allows temporary locking some accounts.
- Publicly readable encrypted passwords allow *brute force attacks* (guessing).
- Alerts can be activated in case of too many unsuccessful authorization attempts for a single tty or IP address (see: `https://www.fail2ban.org`).

Changing Password

- `root` doesn't know user passwords but can change them.
- Regular users can change their own passwords.
- Password related commands include:
  - `passwd [username]` – interactive password change,
  - `pwgen` – random, secure password generator,
  - `chpasswd` – batch-mode utility that processes a list of *user:pass* pairs (intended for `root`) from the standard input stream.

Groups – */etc/group*

- Each user is assigned a primary group (GID in `/etc/passwd`) .
- Users can belong to other (secondary) groups: privileges or restrictions set for a group apply to all group members.
- Group definitions and membership information are stored in the file `/etc/group` .

/etc/group line format

```
groupname:password:gid:users
```

- The field descriptions are:
  - *groupname* – group name, for example: *users*,
  - *password* – a password placeholder (`x`) for backward compatibility,
  - *gid* – group's ID – a unique numerical identifier,
  - *users* – comma-separated list of group members.

Unix Groups

- File permissions can be set for group members.
- Additional privileges to hardware management can be granted to group members (`audio`, `camera`, `disk`, `optical`, `power`, `storage`, `tty`, `video`, etc).

IMPORTANT! Modern desktop-oriented systems and Linux distros may grant extended hardware privileges to all users logged in via display managers to desktop environments (via so-called *consolekits*, *policykits*, etc).

- Some actions may be restricted to members of the `wheel` group.
- Members of the `wireshark` group can set network cards to so-called *promiscuous mode* (raw frames analysis).
- Using some features or software (scheduler, databases, etc) may require membership in respective groups.

Pluggable Authentication (PAM)

- Unix authentication is performed by invoking a sequence of procedures (PAM modules).
- Authentication against `/etc/passwd` and `/etc/shadow` is just one of available PAM modules.
- Other modules may allow using other identity sources (databases, LDAP-compliant Directories, etc.) in addition to or instead of standard ones.
- Additional actions (creating directories, attaching file systems, setting limits, dynamic group membership granting, etc) may be invoked upon successful authentication.

Configuring PAM

- Configuration file `/etc/pam.conf` may be split into several files in `/etc/pam.d` directory (they usually include each other).
- Modules can be added for 4 tasks (see: `man 8 pam`):
  - `auth`entication management,
  - `account` management,
  - `password` management,
  - `session` management.
- The following specifiers define actions on module exit status:
  - `sufficient` – on success don't execute subsequent modules for this task and return task success,
  - `optional` – exit status is irrelevant, execute next modules,
  - `required` – on failure don't execute subsequent modules for this task and return task failure.

Querying Other Databases – NSS

- NSS *Name Service Switch* allows querying additional sources defined in `/etc/nsswitch.conf` for users and groups.

Sample *nsswitch.conf* entries

```
passwd: files ldap
shadow: files ldap
group:  files ldap
```

- A command line interface for NSS is `getent`:

NSS queries (for local and LDAP entries)

```
$ getent passwd
$ getent group
```

PAM vs. NSS

- PAM modules are intended only for authentication tasks.
- File/task ownerships and permissions are stored in the form of UIDs and GIDs.
- Obtaining names for UIDs/GIDs is performed by NSS libraries.
- Displaying additional information via `id`, `finger`, etc. involves NSS.

Running Commands as Other User (*su*)

```
$ su [-] [username]
```

- `su` starts a shell of other user.
- If no username is specified it defaults to `root`.
- The dash symbol applies new user's login shell environment.
- Regular users are prompted for passwords. `root` can use it with no password.
- For security reasons numerous administrator disable `ssh` for `root`. For remote access one has to authenticate as a `wheel` group member then invoke `su -` .

User Management Utilities

- User and group management can be performed manually by editing respective files.
- Most systems provide tools for user and group management. Under Linux the following commands may be available: `useradd`, `userdel`, `usermod`, `newgrp`, `groupadd`, `groupdel` and `groupmod`. Under FreeBSD run `adduser`, etc.

Linux *useradd* – Example

Linux *useradd*

```
# useradd -u 1001 -g users -c "John Smith" -m -s /bin/bash smithj
# passwd smithj
```

- The options are:
  - `-u` – UID (by default: the smallest available one greater than 999),
  - `-g` – GID (default behaviour can vary),
  - `-c` – GECOS (a real name),
  - `-m` – create a home directory with the same name as the username in default location (usually `/home`) and copy `/etc/skel/*` to it,
  - `-s` – default shell.
- `/etc/default/useradd` stores defaults for `useradd` .
- New accounts have invalid passwords by default.
- `/etc/skel/` directory contains files that are supposed to be copied to home directories of regular users on automated account creation (for example: `.bash_logout`, `.bash_profile` and `.bashrc`).

Outline of the lecture

File Owners and Groups

- Each file or directory is assigned a pair: an *owner* and a *group* (see `ls -l` output). Those are normally UID and GID of its creator.
- File owners:
  - can manage file permissions,
  - can change file group (see the next slide),
  - may have defined limits on total size of owned files (*quota*).
- File group members other than the file owner have dedicated file access permissions defined.

File Owners and Groups – *chown*, *chgrp*

- Only `root` is able to change file ownership.
- File owners are able to change file group to other group they belong to.
- The commands `chown` and `chgrp` allow ownership/group change. The option `-R` forces recursion on directories.

chown and _chgrp_ syntax

```
# chown [-R] user[:group] file
$ chgrp [-R] group file
```

File Permissions

- Unix systems define three file access rights (denoted as `rwx`):
  - `r` (*Read*) allows viewing the content and properties,
  - `w` (*Write*) allows modifying the content,
  - `x` (*eXecute*) for files allows executing as a program.
- There are separate file permissions for the file owner (`u`), file group members (`g`) and other users (`o`):
  - `u` (*User*) rights apply to the file owner,
  - `g` (*Group*) rights apply to file group members other than the file owner,
  - `o` (*Others*) rights apply to users other than the owner which don't belong to the file group.
- `ugo` rights are denoted in sequence in the form: `rwxrwxrwx` (see `ls -l` output).

File Permissions – Examples

`rwxr-xr-x` (recommended for a program) the owner is granted all rights, other users can read and execute it,

`rw-r--r--` (recommended for public data) all users can read it, the owner can modify it,

`rw-------` (recommended for private data) only the owner can read and modify it,

`rw----r--` all users except file group members can read the content.

Directory Permissions

- Permissions for directories work the same but their meaning is slightly different:
  - `r` allows listing names in a directory,
  - `w` is necessary for changing content names (stored in directories),
  - `x` allows finding files/directories if their names are known.
- Full read access requires `r-x` permissions.
- Setting directory permissions to `--x` disables directory listing. However it allows access to files/directories in it providing their names are known and permissions to them are granted.

Modifying File Permissions – *chmod*

- The command `chmod` modifies file permissions. The syntax is similar to the one of `chown` :

chmod syntax

```
$ chmod [-R] permissions file
```

- Permissions can be defined in:
  - symbolic mode,
  - numeric mode.

chmod (Symbolic Mode)

- Symbolic string is a list of comma-separated triplets: a *class*:
  - `u` – file owner,
  - `g` – file group
  - `o` – others,
  - `a` or none – all users (`ugo`),

  an *operator*:
  - `=` – set,
  - `+` – add
  - `-` – revoke,

  and *permissions* specified using the `rwx` notation.
- Permissions for unreferenced classes are left untouched.

chmod (Symbolic Mode) – examples

chmod examples

```
$ chmod u=rw,g=r,o=r somefile  # sets rw-r--r--
$ chmod u=rw,go=r somefile     # the same as above
$ chmod go-rwx somesecretfile  # sets ???------
$ chmod +x someprogram         # sets ??x??x??x
```

- Note that no blanks are allowed in symbolic strings:

chmod examples

```
$ chmod u=rw,g=r,o=r somefile    # sets rw-r--r--
$ chmod u=rw, g=r, o=r somefile  # an error!
```

chmod (Numeric Mode)

- The numbers: `4`, `2` and `1` denote respectively `r`, `w` and `x` permissions.
- Permissions for each class (`u`, `g` and `o`) are expressed by a single number – a sum of file permissions for that class, that is: `rwx` is represented by `7`, `rw` – `6`, `rx` – `5` and `wx` – `3`.
- On the contrary to the symbolic mode `chmod` in the numerical mode modifies all permissions (none are left untouched).

chmod examples

```
$ chmod 755 someprogram    # sets rwxr-xr-x
$ chmod 700 somesecretfile # sets rwx------
$ chmod 644 somefile       # sets rw-r--r--
```

Special File Permissions/Flags

- In addition to `rwx` rights for `ugo` file additional three permissions/flags are defined:
  - *setuid*,
  - *setgid*,
  - *sticky bit*.
- Processes – analogously to files – have their owners and groups. Normally those are UID and GID of the user who started them. Setuid (`s`) and setgid (`s`) flags on an executable file make the program started with *effective UID* and *effective GID* equal to the file owner and/or file group respectively.
  - Exemplary use: `/usr/bin/passwd` program used for password change has the setuid flag set in order to be run by regular users.
- Sticky bit (`t`) on directories prevents unprivileged users from removing or renaming a file in the directory unless they own the file or the directory.
  - Exemplary use: `/tmp` directory has the sticky bit flag set.

Special File Permissions/Flags (2)

- Special flags are represented as file permissions, for example:
  - *setuid* – `rwsr-xr-x`,
  - *setgid* – `rwxr-sr-x`,
  - *sticky bit* – `rwxrwxrwt`.
- In symbolic mode those are set as any other rights.

Setting special permissions by symbols

```
$ chmod +x,u+s someprogram
$ chmod +x,g+s someprogram
$ chmod o+t somedirectory
```

- In numerical mode an extra (preceding) number is used.

Setting special permissions by numbers

```
$ chmod 4755 someprogram
$ chmod 2755 someprogram
$ chmod 1777 somedirectory
```

Notes on *setuid* and *setgid*

- Effective for binary executable files.
- Ignored for scripts due to security reasons.
- Revoked on file contents modification.
- May introduce critical system vulnerabilities! Use if really necessary and only for small, extremely safe programs.

Default File Permissions

- Each program is free to set some default permissions on newly created files/directories. For example compilers add the `x` while text editors do not.
- Right specified by *umask* are then extracted from those permissions. Typically the umask value is equal to `0022` which revokes the `w` permissions for users other than the file owner.
- Users are able to check/change their current umask with the command `umask` .

✍ *`touch` sets `0666` for newly created files:*

Umask and file permissions

```
$ umask 0000; touch a
$ umask 0777; touch b
$ ls -l
  -rw-rw-rw-  1 aki  users    0 Apr  1  8:00 a
  ----------  1 aki  users    0 Apr  1  8:00 b
```

Outline of the lecture

Access Control Lists

- Serious limitation of standard file permissions model is that rights to a file can be granted to only one, pre-defined (defined by `root`) group.
- Most modern Unix systems are able to implement *Access Control Lists* (*ACLs*) which overcome this limitation. Beside standard permissions any of `rwx` rights for other users and groups can be granted as well.
- A plus symbol following file rights (for example: `rwxr-xr-x+`) indicates that additional permissions are granted using ACLs.
- There are two commands that allow setting and viewing ACLs: `setfacl` and `getfacl`.

Access Control Lists (2)

- Some file systems (for example: NFS4) don't support ACLs. Some require enabling it explicitly.
- In addition to *normal* ACLs directories may have so-called *default ACLs* defined. They apply to directory content that will be created in the future.
- Setting normal and default ACLs for a directory are two separate/independent tasks.

setfacl by Examples

✍ *Set (modify: -m) ACL on a directory and its contents (recursively: -R)*

Modifying ACL entries

```
$ setfacl -Rm u:smithj:rwx,u:jonesj:rx somedir
```

✍ *Set (modify: -m) default (-d) ACL on a directory and its contents (recursively: -R)*

Modifying ACL entries

```
$ setfacl -Rdm u:smithj:rwx,u:jonesj:rx somedir
```

✍ *Remove (-x) ACL entry for a file*

Modifying ACL entries

```
$ setfacl -x u:smithj,g:users somefile
```