System initialization and basic hardening

Marco Prandini
DISI
Università di Bologna

Introduction

- A few reminders about well-known concepts
  - Security is a process
    - We will examine several technical aspects, but don't forget that the appropriate countermeasures must be integrated in a wider perspective
    - Systems are complex and the (too much) easy task of bringing a new Linux server up should not lead to forget that security issues can arise from many logical and physical components
  - Default deny
    - One of the few widely acknowledged points in the security community is that selectively allowing what you need is better than selectively blocking what you fear
    - This principle can easily be reworded to a more comprehensive philosophy that can be applied to most of the system components, not only to a list of network services or access control rules.
  - Security = privacy, integrity, authenticity, availability
    - The right combination of properties must be enforced
      - for each system component, not only data (e.g. consider physical security)
      - for each copy of the data, not only within the system (e.g. consider backups)

Basic Hardening

- Secure the physical system
- Know where processes come from
- Handle account security
- Manage filesystem authorizations
- Audit system behavior

Unpacking the box

- Your server is, before anything, a piece of hardware located in some place and connected to a slew of peripherals
  - Most of the defence efforts are placed in guarding the system against attacks coming through software and/or from the network
  - The corresponding countermeasures can be easily rendered useless by an attacker having physical access to the server
  - The main threats an attacker poses are:
    - Stealing the disks or the whole box
    - Connecting data-gathering tools to the communication interfaces
    - Booting the system with a different operating system
  - The severity of these threats is strongly dependent from the specific environment
- Some of these issues are slightly different in the increasingly common scenario of a virtual machine, but many countermeasures still apply (maybe with minor adaptations)

Putting the pieces together

- Most of us acts as if any peripheral of the system was trusted
  - How may of you have a clear view of the back of your PC, where the keyboard and mouse are connected?
  - What about the inside, with its wealth of cables connecting the mainboard with disks and optical units?

Source: https://www.keelog.com/

Putting the pieces together

- If you don't trust the place where the server is housed enough for your security requirements, consider:
  - Selecting a case which can be locked close and tied to the rack/furniture/building
  - Installing tamper-detection devices
  - Adopting data protection measures to make the stolen hardware supports useless
  - Disabling hardware peripherals you don't use (but what if you later need them?)

Plan the installation

- What do you need?
  - Nowadays, most distributions come with a quite friendly install procedure
  - It's very easy to be led to install many packages whose existence (let alone function) is unknown to the owner... but not to the attackers!
  - Default deny starts with installing only the strictly needed software
    - If the installer at some point asks for a set of packages to install, it's better to deselect everything. Package managers will help you get what you need later on. (Hint: make an exception for the graphical interface, if you want it)
    - Choose a distribution that suits your needs (server, workstation, router...)
- Lay your disks out with availability and integrity in mind
  - Separate partitions that are easily filled up by stressing the system (`/var`, `/tmp`) or by users (`/home`) from those essential for the operating system (/)
  - An unpractical, yet additional layer of integrity defence would be mounting the `/usr` partition read-only

Reboot

- To reach its steady state, the system traverses the boot procedure, which can be split in the following phases:
  - (1) BIOS – Selects the boot device(s) and the order in which they are to be queried
    - Most BIOSes provide password protection either for booting the machine or for modifying their own configuration
  - (2) Boot Loader – Selects the operating system image and allows passing the OS additional informations
    - Some keywords can be specified in this step to let the OS start in maintenance mode
    - Same kind of password protection as described for the BIOS
  - (3) Operating system – Usually does nothing more than loading the correct set of device drivers (not to be underestimated!) and invoking the special *init* process
    - Tells *init* the initial runlevel (if overriding the default is needed)
  - (4) *init* – handles *runlevels* and System-V-style system initialization, i.e. starts the needed services in the right order
    - Usually configures the real and virtual terminals

Boot protection

- Password pros and cons:
  - If a password is needed for booting the system, unattended operation can be problematic: a simple power outage can make the system unavailable
    - For systems where privacy and integrity considerations override availability issues, this is a minor problem, since probably there will also be specific services refusing to start if a password is not manually entered (for example to decrypt private keys they use)
  - Password protection against system configuration alterations is always advisable
- NEVER rely on a single protection layer
  - BIOS passwords can often be overridden by manufacturer's defaults
  - Any password can be guessed
  - Risky defaults on some distributions (e.g.: if a means of requesting maintenance boot to the boot loader is found, *init* provides a root shell)

This is a very useful feature to legitimately gain control of a corrupted system which will not boot, the other being booting from an external media (→ BIOS)
Bottom line: lock-down = increased integrity, lower availability!

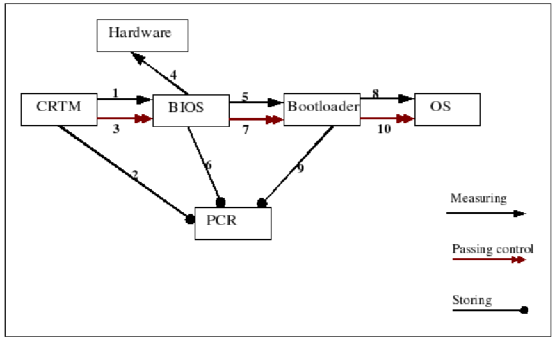
Secure / Trusted Boot

- Issue: how to be sure that every piece of software a computer executes is authentic/unmodified/benevolent?
  - Anti-malware check applications
  - Who checks anti-malware?? The OS (making AM useless…)
  - Who checks the OS? The boot loader
  - Who checks the boot loader? Special HW, which cannot be changed from within the OS, and thus is immune from infections

  → *root of (a chain of) trust*
- Two ways
  - *Trusted boot* makes use of the TPM (Trusted Platform Module)
    - Special hardware chip with crypto functionalities
  - *Secure boot* makes use of UEFI (Unified Extensible Firmware Interface)
    - Software implementation + firmware keys
    - Needs a standard BIOS for the most basic steps of POST
    - May use the TPM to speed-up/enhance the integrity checks

Trusted boot

- Starts from the TPM
  - Core Root of Trust for Measurement (CRTM)
  - Registers (PCR)
- Gathers evidence of integrity (violations)
- Delays checks until there is
  - Crypto keys availability
  - enough memory to perform the needed computations



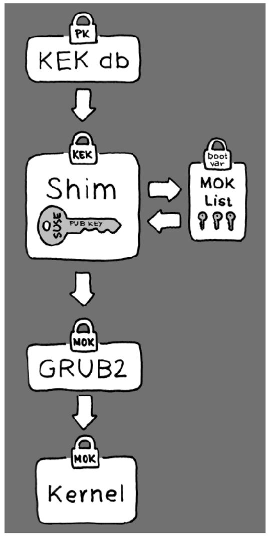
UEFI and the secure boot

- EFI (from Intel) was born as a more-flexible-than-BIOS interface between the OS and the firmware
- UEFI forum standardized and updated it
  - http://www.uefi.org/
- UEFI is a “mini OS”
  - BIOS boot via MBR:
    - 400 bytes of ASM in boot sector
    - 4 primary partitions or 3 primary parts + 11 logical units
  - EFI with GPT
    - its own filesystem (100-250MB) for boot loaders
    - nearly unlimited partitions of up to 9ZB
- UEFI verifies each piece of software before yielding control
  - It needs a key database to be always available
  - As soon as a verification fails, the boot process stops

UEFI and the secure boot of Linux

1) The official Platform Key verifies a small pre-boot-loader
   - The key used to sign shim must be provided by the HW vendor
   - It is a Microsoft key!
2) Shim can use / pass along MOKs (Machine Owner Keys)
   - To verify the bootloader
   - To verify custom-built kernel modules

- Additional kernel components must be signed to be loaded
  - User generates MOKs
  - User submits MOKs to shim
  - At next boot, shim finds the keys during the setup phase, and asks if they must be written in firmware → **explicit consensus always required!**



https://www.suse.com/communities/blog/uefi-secure-boot-details/

Interesting UEFI links

- https://www.linux.com/publications/making-uefi-secure-boot-work-open-platforms
- http://www.rodsbooks.com/linux-uefi/
- http://www.linux-magazine.com/Online/Features/Coping-with-the-UEFI-Boot-Process
- https://help.ubuntu.com/community/UEFI
- http://askubuntu.com/questions/760671/could-not-load-vboxdrv-after-upgrade-to-ubuntu-16-04-and-i-want-to-keep-secur
- https://knowledge.windriver.com/en-us/000_Products/000/010/040/060/020/000_Wind_River_Linux_Security_Profile_Developer's_Guide,_8.0/070/000/010
- https://www.suse.com/communities/blog/uefi-secure-boot-details/
- https://lwn.net/Articles/519618/

Bootloader (runtime) configuration

- LILO, the Linux Loader
  - used almost since the beginning of Linux
- GRUB, the Grand Unified Bootloader
  - GRUB is more powerful and flexible than LILO, providing an interactive shell that allows executing many commands and tailor-building the boot procedure: of course this feature is open many possible abuses.
- Both support parameter passing to the kernel, most notably (for security purposes)
  - single
  - init=...

Boot Loader passwords

- LILO

  `password=YourPasswordHere`

  Sets a password that will be asked for when booting the system, unless `restricted` is specified. In the latter case, the password will be asked for only when manually overriding LILO settings during boot.
- Global vs. Single-entry protection
  - `password` and `restricted` in the global section: ask for the password to allow any parameter addition – be careful with unsafe entries (floppy)
  - `password` and `restricted` in an image section: ask for the password to allow any parameter addition, only for the selected image
  - `password` in the global section and `restricted` in an image section: ask for the password to allow any parameter addition for the selected image, and ask the for the password for booting the other images

Boot Loader passwords

- GRUB

  `password [--md5] passwd [new-config-file]`

  If specified in the global section, sets a password that will be needed for entering the *interactive operation* of the bootloader. Will optionally cause the loading of a different configuration file.

  If put in a specific menu item, sets a password that will be needed for booting that configuration.

  `lock`

  Put in a specific menu item, right after `title`, marks that configuration password-protected.

  Effective only if preceded by a password definition in the global section

  `md5crypt`

  Type this command at the grub prompt to compute the password hash to use with `--md5`

Trusted terminals

- The *login* program usually handles user authentication on local and remote (serial) text consoles.
- If direct *root* access is undesirable on some of these, edit the file */etc/securetty* and remove them
  - Example of default settings:
    - `# /etc/securetty: list of terminals on which root is allowed to login.`
    - `# See securetty(5) and login(1).`
    - `console`
    - `# for people with serial port consoles`
    - `ttyS0`
    - `# for devfs`
    - `tts/0`
    - `# Standard consoles`
    - `tty1`
    - `tty2`
    - `...`

Process management

- So you think you installed almost nothing, and then...

```
milk:~# ps aux
USER        PID %CPU %MEM   VSZ   RSS TTY      STAT START   TIME COMMAND
root          1  0.0  0.0  1948   468 ?        Ss   May15   0:02 init [2]
[... kernel processes ...]
root       1753  0.0  0.0  2704   392 ?        S<s  May15   0:00 udevd --daemon
daemon     2953  0.0  0.0  1688   408 ?        Ss   May15   0:00 /sbin/portmap
root       3231  0.0  0.0  1624   568 ?        Ss   May15   0:26 /sbin/syslogd
root       3237  0.0  0.0  1576   340 ?        Ss   May15   0:00 /sbin/klogd -x
bind       3251  0.0  0.1 39732  1964 ?        Ssl  May15   0:00 /usr/sbin/named
root       3266  0.0  0.0 39500   944 ?        Ssl  May15   0:00 /usr/sbin/lwres
root       3339  0.0  0.0  1572   444 ?        Ss   May15   0:00 /usr/sbin/acpid
103        3344  0.0  0.0  2376   760 ?        Ss   May15   0:00 /usr/bin/dbus-d
106        3352  0.0  0.1  6116  1972 ?        Ss   May15   0:03 /usr/sbin/hald
root       3353  0.0  0.0  2896   716 ?        S    May15   0:00 hald-runner
106        3359  0.0  0.0  2016   472 ?        S    May15   0:00 hald-addon-acpi
106        3367  0.0  0.0  2020   480 ?        S    May15   0:00 hald-addon-keyb
root       3387  0.0  0.0  1808   360 ?        S    May15  14:15 hald-addon-stor
root       3414  0.0  0.0  1864   396 ?        Ss   May15   0:00 /usr/sbin/dhcdb
root       3421  0.0  0.1  3984  1164 ?        Ss   May15   0:00 /usr/sbin/Netwo
avahi      3433  0.0  0.1  2936  1424 ?        Ss   May15   4:14 avahi-daemon: r
avahi      3434  0.0  0.0  2552   180 ?        Ss   May15   0:00 avahi-daemon: c
root       3441  0.0  0.0  2908   536 ?        Ss   May15   0:00 /usr/sbin/Netwo
root       3457  0.0  0.0  1752   452 ?        Ss   May15   0:02 /usr/sbin/inetd
root       3477  0.0  0.0  4924   512 ?        Ss   May15   0:02 /usr/sbin/sshd
ntp        3507  0.0  0.0  4144   764 ?        Ss   May15   0:00 /usr/sbin/ntpd
root       3521  0.0  0.0  1976   724 ?        Ss   May15   0:02 /sbin/mdadm --m
daemon     3540  0.0  0.0  1828   280 ?        Ss   May15   0:00 /usr/sbin/atd
root       3547  0.0  0.0  2196   720 ?        Ss   May15   0:00 /usr/sbin/cron
root       3590  0.0  0.0  1572   372 tty2     Ss+  May15   0:00 /sbin/getty 384
root       3591  0.0  0.0  1576   372 tty3     Ss+  May15   0:00 /sbin/getty 384
root       3592  0.0  0.0  1572   372 tty4     Ss+  May15   0:00 /sbin/getty 384
root       3593  0.0  0.0  1572   372 tty5     Ss+  May15   0:00 /sbin/getty 384
root       3595  0.0  0.0  1576   372 tty6     Ss+  May15   0:00 /sbin/getty 384
```


Process management

- Even if these “things” are actually needed, its important to know
  - where they come from
  - how to get rid of them, possibly avoiding unwanted “resurrections”
  - **useless processes not only consume resources, but also offer attack paths!**
- Remember the basics
  - *man* is your best friend, and the Internet closely follows.
  - *ps*, *top*, *kill*, ... can quickly and effectively assist you in solving instant problems, but do not prevent them to reappear
- There are three main sources of processes (besides you)
  - Init-started procedures
  - Periodic and aperiodic schedulers
  - Daemons handling dynamic, event-based subsystems

Scheduled execution

- Periodic execution of programs is the task of *cron*
  - every user can have its *cron table* (*crontab*), look in */var/spool/cron* to spot them
  - system tasks are often placed into */etc/crontab*
    - commonly */etc/crontab* comes preconfigured so as to
      - include any configuration file placed into */etc/cron.d/*
      - run any program placed in the directories */etc/cron.hourly*, */etc/cron.daily*, */etc/cron.weekly*, */etc/cron.monthly*, with the obvious periodicity
  - edit */etc/crontab* freely, use

    `crontab -e [-u username]`

    for the user tables
- Delayed, one-shot execution of programs is the task of *at*
  - relevant commands for queuing up tasks, examining the queue of tasks waiting for their hour and cancelling tasks: *at, atq, atrm*

Event managers / IPC systems

- Dbus is an Inter-Process Communication architecture.
  - It starts some Dbus enabled subsystems, so they can exploit the advantages of the architecture.
  - Look around in */etc/dbus-1/* to see its configuration
  - Find in */etc/dbus-1/event.d* the startup scripts of managed subsystems.
- Udev replaced devfs as an **event manager** for the creation on-the-fly of device nodes when a new devices is hot-plugged.

  Look in */etc/udev/rules.d* to see files containing event-to-action mappings, like

```
# udev rules file for SynCE
BUS!="usb", ACTION!="add", KERNEL!="ttyUSB*",
  GOTO="synce_rules_end"
# Establish the connection
RUN+="/usr/bin/synce-serial-start"
LABEL="synce_rules_end"
```


Initialization and background activities

- *init* is the first process run by Linux in traditional distros
  - Handles different *runlevels*, that is working states defined by the set of running services
  - Orchestrates the proper sequence of events to reach a runlevel
  - Monitors some events that happen during the system's uptime
  - Gracefully shuts down the system
- Three main variants
  - (historical) SystemV-style initialization
  - Upstart (Canonical, 2006-2014)
  - Systemd (loosely RedHat, 2010-active)

Useful to know because the current situation is an awful mix of legacy daemons and "modern" orchestrators

sysvinit

- /sbin/init daemon from the original SystemV Unix
  - configured by means of the file */etc/inittab*
  - *inittab* specifies the default runlevel
    - `id:2:initdefault:`
  - but if the special keyword *single* is passed as a parameter to the kernel during loading, this setting is overridden and *init* proceeds to *single user mode* (runlevel 1)
    - `~~:S:wait:/sbin/sulogin`
  - *init* also spawns the virtual terminals and serial line console handlers
    - `1:2345:respawn:/sbin/getty 38400 tty1`
    - `2:23:respawn:/sbin/getty 38400 tty2`
    - `...`
    - `T0:23:respawn:/sbin/getty -L ttyS0 9600 vt100`
    - `T3:23:respawn:/sbin/mgetty -x0 -s 57600 ttyS3`

sysvinit – started processes

- *init* is ultimately responsible for everything running on the system, but two activities can be directly traced to it
  - lines like

    `l0:0:wait:/etc/init.d/rc 0`

    start the *System-V-style startup process*
    - one line at a time is executed: when entering the corresponding runlevel 'N'
    - `rc` executes
      - every program with a name starting with 'S' in */etc/rcN.d/* with the parameter `start`
      - every program with a name starting with 'K' in */etc/rcN.d/* with the parameter `stop`
    - to avoid useless duplication of the scripts which start/stop daemons, they are all placed under `/etc/init.d/`, and linked from the 7 `/etc/rcN.d/` directories
      - Use **`chkconfig`** or **`update-rc.d`** to configure runlevels by updating the link sets
  - lines like

    `x:5:respawn:/usr/X11/bin/gdm`

    run the program specified as 4th field, and *init* monitors the process to restart it if it terminates

Upstart (mainly Ubuntu)

- An event-based replacement for *init*
  - Non-blocking, parallel initialization of subsystems
  - Consistent handling of all the asynchronous system events
    - Hardware addition/removal
    - Process started/stopped
  - Multi-stage initialization (e.g. hw detection -> firmware loading -> device activation -> device features scanning)
  - Prospective integration of planned events (cron jobs, at jobs)
- Known Users
  - Ubuntu 6.10 and later
  - Fedora 9 and later
  - Debian (as an option)
  - Nokia's Maemo platform
  - Palm's WebOS
  - Google's Chromium OS
  - Google's Chrome OS

A few concepts about upstart

- Philosophy (from the website):
  - Tasks and Services are started and stopped by events
  - Events are generated as tasks and services are started and stopped
  - Events may be received from any other process on the system
  - Services may be respawned if they die unexpectedly
  - Supervision and respawning of daemons which separate from their parent process
  - Communication with the init daemon over D-Bus
- Working
  - The `/etc/init` directory contains a file for each job definition
  - The `init` demon remains the system's director
    - any modification to conf files is noticed via inotify and immediately applied
  - The `initctl` command interact with the jobs by sending appropriate signals (see event.h in the sources) to `init` (via sub-commands):
    - `start / stop / status`
    - `list / emit / reload-configuration`

Systemd (mainly RedHat – now widespread)

- What problems does systemd solve?
  - Service dependencies
  - Starting services on-demand
  - Early syslog
  - Output of daemons is preserved
  - Tracks cgroups (for fine-grained HW resources control)
  - Tracks and manages mount points
  - System snapshots and restores
  - Manages hostname, locale, and other system-wide settings
  - Predictable service environment
  - Offline system updates
  - Faster boot process
  - Shell-free boot

"Systemd 101" - Steven Pritchard
https://docs.google.com/presentation/d/10YwWZdBa3ffl7kVa2p21L9VqET2CRmVoWJpVBW6ujgg/edit#slide=id.g34f773849_010

Systemd

- What does systemd aim to replace?
  - init (etc.)
  - udev
  - pm-utils
  - inetd
  - acpid
  - crond/atd
  - ConsoleKit
  - automount
  - watchdog
  - syslog

Systemd - terms

- Different kinds of [control] *units* named with the convention **`name.type`**
- Types:
  - **`Service`**: control and monitoring of deamons
  - **`Socket`**: set-up of IPC channels of any kind (file, net socket, Unix socket)
  - **`Target`**: set of units that replaces a runlevel
  - **`Device`**: created by the kernel via hw interaction
  - (filesystem-related)
    - **`Mounts`**
    - **`Automounts`**
    - **`Swap`**
  - **`Snapshots`**: save state, for testing
  - **`Timers`**: timer-based tasks (→ cron, at)
  - **`Paths`**: path monitoring via inotify
  - **`Slices`**: resource management via cgroup
  - **`Scopes`**: group processes for clearer organization

Systemd – unit definition locations

- “library” of reference unit definitions
  - **`/lib/systemd/system`**
- Location of mantainer files
  - Mainly links to the reference files
  - **`/usr/lib/systemd/system`**
- Location of user customizations
  - They take priority over system defaults
  - **`/etc/systemd/system`**

Systemd – basic operations

- Runtime control of services
  - **`systemctl {start|stop|status|restart|reload} servicename`**
    - ...intuitive
    - richer output for status
      - current status and past steps
      - process tree
      - relevant log entries
    - “-H [hostname]” connects to remote host via ssh
- Persistent configuration of services boot
  - **`systemctl {enable|disable|mask|unmask} servicename`**
    - `disable` leave the possibility of manual start intact
    - `mask` makes the unit definition void, blocking also manual control

Systemd configuration display

- Just a few examples
  - **systemctl list-units**
    - shows all managed units (all the aforementioned kinds!)
  - **systemctl -t *type***
    - e.g.: `systemctl -t timers`
    - shows all loaded units of the given type
  - **systemctl list-unit-files [-t *type*]**
    - e.g.: `systemctl list-unit-files -t services`
    - shows all installed units
  - **systemctl --state *state***
    - e.g.: `systemctl --state failed`
    - shows all units in the given state

Systemd startup

- Runlevel are replaced by targets
- /etc/inittab no longer used
  - default is queried/set with
    systemctl get-default
    systemctl set-default [target]
    - e.g.: `systemctl set-default graphical.target`
- Equivalences
  - Look inside `/lib/systemd/system`

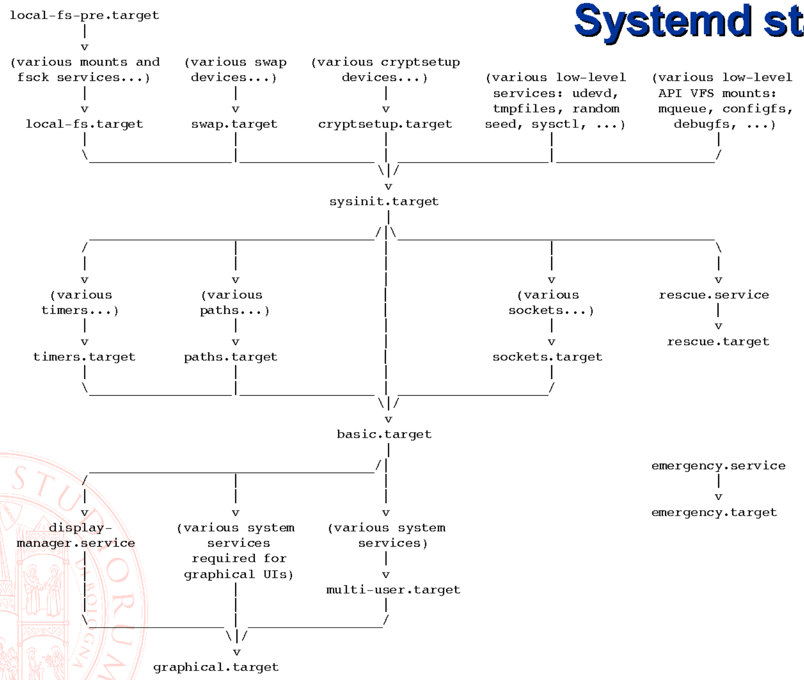
Runlevel	Systemd Target	Description
0	poweroff.target, runlevel0.target	System halt
1	rescue.target, runlevel1.target	Single user mode
3 (2,4)	multi-user.target, runlevel3.target	Multi-user, non graphical
5	graphical.target, runlevel5.target	Multi-user, graphical
6	reboot.target, runlevel6.target	System reboot

Systemd startup

- What does a target do? From the `systemd.target` man page:

  "Target units [...] exist merely to group units via **dependencies** (useful as boot targets), and to establish **standardized names** for synchronization points used in dependencies between units."
- Dependencies = proper automation
  - Sysvinit = sequential = slow, no error handling
  - Systemd = parallel, units start if they have all they need
    - `Requires` directive: other units to start when this unit is started/stopped; failing their start, this unit is stopped; configurable timing (after, before, same time)
    - `Wants` directive: softer version of start (failed deps do not block this unit)
    - `Conflicts` directive: negative requirement → mutually exclusive units
    - `OnFailure` directive: units to start when this unit fails
    - `RequiredBy`/`WantedBy`: create Require/Want in other units when this is installed
- Standardized names = special, fixed names!
  - Some units are pre-defined, with fixed names and a fundamental function
  - Mainly targets, and a few slices (see `systemd.special(7)` and `bootup(7)`)
  - e.g. boot sequence sync. points – boot sequence aims at <u>`default.target`</u>

Systemd startup



Service managers cheat sheet

	SysVinit (Debian) (RedHat)	Upstart	Systemd
Start service	/etc/init.d/name start	service name start	systemctl start name
Stop service	/etc/init.d/name stop	service name stop	systemctl stop name
Status check	/etc/init.d/name status	service name status	systemctl status name
Enable service start at boot	update-rc.d name enable chkconfig name on	rm /etc/init/name.override	systemctl enable name
Inhibit service start at boot	update-rc.d name disable chkconfig name off	echo manual > /etc/init/name.override	systemctl disable name
List installed services	ls /etc/init.d chkconfig --list	service --status-all && initclt list	systemctl list-unit-files -t services
List services starting at boot	ls /etc/rcX.d/S* chkconfig --list | grep X:on	Give up and upgrade to systemd.	systemctl list-unit-files -t services --state=enabled

Put the default runlevel number in **X** place

Common assumption: installed services start at boot

User management

- Linux users can be created using different command-line (e.g. *useradd*) or graphical tools
- Each user belongs to at least one group (typically created together with the user and containing only that user)
- Each user can belong to a variable number of other groups
- User accounts can be in a locked state, that prevents them to log in, but allows processes running in their names (useful for daemons started by root that after startup "demote" themselves)
- The *passwd* command is used
  - to change the user password (root only can add a username as a parameter to change anyone's password)
  - to lock (`-l`) and unlock (`-u`) accounts (root only, obviously)

Manage users, groups, ownership

- adduser and addgroup … quite self-explanatory
- See effects on
  - /etc/passwd
  - /etc/shadow
  - /etc/group
  - /etc/gshadow
- chown <new_owner:new_group> <file> changes the file's owner and/or group

User authentication - basics

- Typically, user credentials for local authentication are kept in
  - */etc/passwd*, world-readable, one line per user, like:

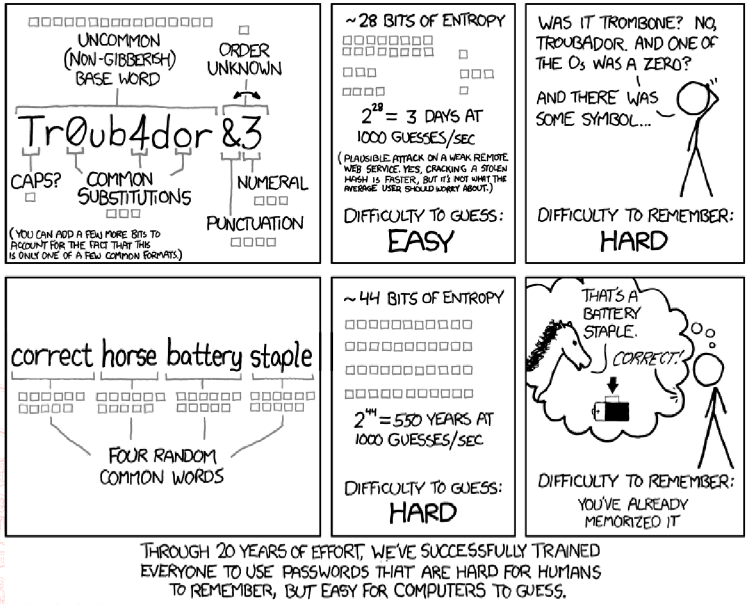
    `prandini:x:500:500:Marco Prandini:/fat/home:/bin/bash`
  - */etc/shadow*, accessible only to root, with lines corresponding to passwd

    `prandini:$1$/PBy29Md$kjC1F8dvHxKhnvMTWelnX/:12156:0:99999:7:::`
  - Note: Do not remove the 'x' placeholder in the password field, or the system will not lookup the shadow file and will not prompt the user for a password at the login prompt.

Password strength

- Still many users choose easy-to-guess passwords
- User education is important, but not always effective
  - see this essay by Bruce Schneier on how to choose a good password for some ideas http://www.wired.com/politics/security/commentary/securitymatters/2007/01/72458?currentPage=all
- Two countermeasures:
  - proactive (don't allow weak passwords... but post-it as a side effect)
    - see next slides for examples of PAM configuration
  - reactive (check for weak passwords and talk to the user)
    - use tools for password-cracking: John the Ripper http://www.openwall.com/john/

Password strength - http://xkcd.com/936/



Password ageing

- The shadow file format holds temporal information that can be examined and changed with *chage*:

`<name>:<pw>:<date>:PASS_MIN_DAYS:PASS_MAX_DAYS:PASS_WARN_AGE:INACTIVE:EXPIRE:`

- Meaning of the fields and file where default values (assigned at user creation) are stored:

/etc/login.defs	PASS_MAX_DAYS	Maximum number of days a password is valid.
/etc/login.defs	PASS_MIN_DAYS	Minimum number of days before a user can change the password since the last change.
/etc/login.defs	PASS_WARN_AGE	Number of days when the password change reminder starts.
/etc/default/useradd	INACTIVE	Number of days after password expiration that account is disabled.
/etc/default/useradd	EXPIRE	Account expiration date in the format YYYY-MM-DD.

Enforcing password strength

- pam_cracklib.so is the component that checks password features when a new one is chosen.
- In */etc/pam.d/system-auth* or */etc/pam.d/common-password* find the line starting with `password requisite` and append any combination you like of the following parameters after `pam_cracklib.so`:
  - minlen = (Minimum length of password)
  - lcredit = (Length credit for lower case letters)
  - ucredit = (Length credit for upper case letters)
  - dcredit = (Length credit for digits)
  - ocredit = (Length credit for other characters)
- Example of the credit mechanism:
  - minlen=8 dcredit=1
    - any 8-char password is accepted
    - any (8-*n*) char password is accepted if *n* chars are digits

Limiting password reuse

- The same PAM files allow specifying limits for password reuse. For example, by placing the underlined parameters in the configuration lines:

```
password    required     pam_cracklib.so ... difok=3
password    sufficient   pam_unix.so ... remember=26
```

1) a new password must have at least 3 different characters from the old one
2) the last 26 passwords are remembered and cannot be reused

User lockout on failed login attempts

- BE CAREFUL since this countermeasure is often more effective for an attacker (that easily prevents legitimate users from accessing the system) than useful
- This said, in the same PAM files it is possible to configure the use of the *tally* module

```
auth       required   pam_tally.so onerr=fail no_magic_root
account    required   pam_tally.so deny=5 no_magic_root reset
```

  - the first line enables counting the failed login attempts
  - the second line locks the account when the number of failed attempts reaches the threshold specified with deny
  - a successful login resets the counter
- the *faillog* command allows inspecting an account's condition and to reset the access locked after too many failed attempts

The superuser

- A best practice for safety and security is avoiding the use of the *root* account for common work
  - use a non-privileged account 99% of the time
  - disable direct *root* access on the GUI and consoles if necessary
  - gain *root* rights to perform administrative tasks
- Two ways to gain *root* rights
  - su (switch user) is easy but not suitable for shared administration
    - requires to know the root password
    - gives unrestricted control of the system
  - sudo (do as super-user)
    - requires the password of the invoking user (to prevent coffee break attacks)
    - configurable (limits which programs can be run by each user)
    - see the man page *sudoers* for the syntax of the configuration file
    - use *visudo* for editing */etc/sudoers* (checks syntax and installs the file preventing errors that could lock users out)