2. Boot process debugging, security and beyond - Linuxdz courses derja - Language local Algerian dialect

Linux Kernel Boot process debugging, systemd, timers (cron), security and beyond

Linuxdz courses derja

Figure 1: Boot-process-derja header

We will discuss:

- Quickly Linux boot process in general
- systemd init boot process and targets
- Start system or programs at boot and timer services (cron jobs)
- Techniques how to save your data or backup your system on emergency
- Linux boot process debugging techniques
- Discuss some security points across the whole document

Information

Videos are in local Algerian dialect.

Video Link Part 1: Introduction

Video Link Part 2: systemd, services, timers cron jobs and logins

Video Link Part 3: Debug Linux kernel and systemd boot with some security points

Pdf Link: Boot-process-derja.pdf

Markdown Link: Boot-process-derja.md

Djalal Harouni - github - twitter

Email for corrections here: tixxdz+linuxdz@gmail.com - (sorry if I do not reply to all emails.)

Date: 10-04-2020

Last Modified: Mon 13 Apr 2020

Note to convert to pdf:

```
pandoc --variable urlcolor=blue linuxdz/Boot-process-derja.md -o linuxdz/Boot-process-der
```

What is about ?

References: Linux Foundation fundamentals of Linux

Adapted to be easy with video in derja language, Algeria local dialect.

- Ghir important things!
  - Dirou research alone, no excuses!
- 3leh ?
  - Tal3ou level
- Goal - Hadef ?
  - Debug Linux boot process and beyond
  - Services, security, logins and timers (cron jobs)
- Teacher ?
  - Djalal Harouni - Open Source Software maintainer - systemd, linux kernel developer... wrote code used in millions of machines and devices.

Plan

1) Linux Boot Process - Bootloader
   - Linux Kernel and initramfs
   - Init systemd and Services
2) Systemd run program during boot and timers (cron jobs)
3) Logins and session
4) Debug boot and Security
5) Conclusion

1. Linux boot process

BIOS - BIOS POST

Boot loader (grub2 - uboot for embedded, etc)

Linux kernel and initramfs (initrd)

Systemd - only distributions with systemd

1.1 BIOS - BIOS POST

Basic I/O System - is hardware working ?

Bios_Interrupts

Find boot record load into ram and transfert execution to 2) Boot loader

1.2 Boot loader (Grub2 - uboot for embedded, etc)

Grub2 (GRand Unified Bootloader, version 2)

No more multiple stages: Grub Doc

Load Linux kernel and initrd (initramfs)

Files and configs: /boot/grub2/

Find kernel and initramfs (initial kernel ramdisk) load in ram and transfert execution to 3) Linux kernel

1.3 Linux kernel and initramfs

Kernel can be compressed "vmlinuz" (Z) self extracting or uncompressed "vmlinux" image. Located in /boot/

Initramfs (initrd) image - basic root file system and modules need by kernel

Make sure to backup your old initrd first.

```
mkinitrd -o /boot/initrd.img-$(uname -r) $(uname -r)
mkinitrd -o /boot/initrd.img-4.19 4.19
```

Kernel detects and initializes hardware

Kernel reads disks detects the root file system and replaces initramfs

Kernel starts its threads and transfert execution to 4) INIT Program /sbin/init

```
$ ls -lha /sbin/init
lrwxrwxrwx 1 root root 20 Oct 16 14:24 /sbin/init -> /lib/systemd/systemd
```

1.4 Systemd init - only distributions with systemd

Parent of all processes.

Sets up the machine

Bootup targets order

Sysinit.target, basic.target, multi-user.target and graphical.target

systemd - journald tools , systemd-cgls and others.

2 Systemd run program during boot and timers (cron jobs)

2.1 Example service or program running during each boot:

File hello-world.service

```
[Unit]
Description=Hello World Service

[Service]
ExecStart=/bin/bash -c 'for i in {1..30}; do echo "Hello at $(date)"; sleep 3; done'

[Install]
WantedBy=multi-user.target
```

Installation commands:

```
sudo cp hello-world.service /etc/systemd/system/
sudo systemctl daemon-reload
sudo systemctl enable hello-world.service
sudo systemctl start hello-world.service
```

Stop:

```
sudo systemctl stop hello-world.service
```

Disable:

```
sudo systemctl disable hello-world.service
```

NetworkManager File NetworkManager.Service example.

```
cat /lib/systemd/system/NetworkManager.service
```

2.2 Example timer service (cron job)

systemd timer services instead of cron jobs

Example timer-hello-world timer service - execute each 30 seconds

File timer-hello-world.service

```
[Unit]
Description=Timer hello world - Service unit

[Service]
Type=oneshot
ExecStart=/bin/bash -c 'echo Timer hello world at $(date)'

[Install]
WantedBy=multi-user.target
```

File timer-hello-world.timer

```
[Unit]
Description=Timer hello world - Timer unit

[Timer]
OnBootSec=1min
OnUnitActiveSec=30sec

[Install]
WantedBy=timers.target
```

Install timer service commands:

```
sudo cp timer-hello-world.service /etc/systemd/system/
sudo cp timer-hello-world.timer /etc/systemd/system/
sudo systemctl daemon-reload
sudo systemctl enable timer-hello-world.service
sudo systemctl enable timer-hello-world.timer
sudo systemctl start timer-hello-world.timer
```

3. Logins and session

Display logins with loginctl - systemd-logind

Display seats with loginctl - multiseats

Figure 2: Multi-seats source wikipedia

Lock and unlock sessions with loginctl:

```
loginctl lock-session $id
loginctl unlock-session $id
```

loginctl help (press `q` to exit):

```
loginctl --help
```

4. Debug boot and Security

First lets see virtual consoles

4.1 Virtual terminals or virtual consoles

Comes from early days where single machines with multiple terminals teletypes - the TTY demystified

Figure 3: oldschool source the TTY demystified

Separate logins - these days due to personal computers it is called virtual termnials (simulates physical terminal)

On Linux normally: **6 virtual consoles + default physical one**

Figure 4: Virtual consoles source www.cv.nrao.eduo/~pmurphy/

Virtual Consoles pmurphy

Linux switch between virtual consoles

```
Combo keys Ctrl+Alt+f(1,2,3,4,5,6,7)  or  Ctrl+Alt+fn+f(1,2,3,4,5,6,7)
```

List and get current TTY:

```
loginctl -a
```

Example switch to virtual console 3:

```
Press combo keys Ctrl+Alt+f3 or Ctrl+Alt+Fn+f3
```

Example switch back easy:

```
Press combo keys Ctrl+Alt+f2 or Ctrl+Alt+Fn+f2
```

Or example switch back with `chvt`

```
sudo chvt 2
```

On Qemu emulator:

- Switch to Qemu console first

  ```
  Press combo keys Ctrl+Alt+2
  ```

- Use the command: `sendkey ctrl-alt-f3` and press Enter on Qemu console

  ```
  sendkey ctrl-alt-f3
  ```

- Switch back to Qemu VGA output where you will be in virtual console 3

  ```
  Press combo keys Ctrl+Alt+1
  ```

Why I need this ? bech never mata7sel :-D ! (except of kernel panic, keyen 7al apres. . .)

4.2 Debug boot kernel - early boot

4.2.1 Kernel Boot logs

Logs are stored in files: `/var/log/dmesg` `/var/log/syslog` or `/var/log/kern.log`

Commands to read logs:

```
sudo dmesg
sudo journalctl -k
```

4.2.2 Kernel cmdline debug options

Remove cmdline: `quiet splash vt.handoff=7`

Add cmdline kernel:

```
console=ttyS0,115200 console=tty1
debug                          # or debug=vc
systemd.log_level=debug systemd.log_target=console
```

Change kernel ring buffer size at cmdline kernel:

```
log_buf_len=16M
```

4.2.3 Kernel debug options at runtime:

printk() print to kernel log - printf() C language

```
cat /proc/sys/kernel/printk
4       4       1       7
current default minimum boot-time-default
```

Get all debug messages must be root:

```
# echo 8 > /proc/sys/kernel/printk
```

dmesg - print control kernel ring buffer

```
sudo dmesg -n 5
```

Kernel developers to inspect if messages are getting there (it uses printk internally)

```
# echo "insert from userspace by user $(whoami)" > /dev/kmsg
```

4.2.4 Boot fails

Try `Ctrl+Alt+Del` to reboot or hard reset

- Debug with hardware Serial Console - Kernel boot command line options:

```
systemd.log_level=debug systemd.log_target=console
```

- Forward to console:

```
systemd.journald.forward_to_console=1
```

- Boot into rescue mode if problem is happens later - Kernel boot command line

```
systemd.unit=rescue.target
```

- Boot into emergency shell: Kernel boot command line options:

```
systemd.unit=emergency.target
```

- Remount root filesystem r/w to be able to edit files and change configuration

```
mount -o remount,rw /
```

- Boot directly into root shell - Kernel boot command line options:

```
init=/bin/sh
```

If you have an early shell, you can restore your system, change passwords etc. . .

- Kernel module fails during boot - Kernel boot command line if module fails during kernel:

```
module_blacklist=modulename
```

- Kernel boot command line options if module fails during initramfs:

```
rd.blacklist=modulename
```

4.2.5 Boot or system blocked

Ctrl+Alt+Del combo - the following file controls the handling of the combo:

```
cat /proc/sys/kernel/ctrl-alt-del
0   (means graceful restart)
```

Magic SysRq Documentation link Magical Linux kernel keys.

Trigger SysRq with method (1) combo keys `Alt+SysRq+$command` usually `SysRq` is `PrintScreen` or `ImpÉc` in azerty keyboards on x86

SysRq Command keys link

Control the combo `Alt+SysRq+$command` and which operation is allowed with:

```
cat /proc/sys/kernel/sysrq
```

Important: dont use these commands unless you read the full documentation

```
# echo "number" >/proc/sys/kernel/sysrq
```

Otherwise trigger SysRq directly by method (2) write the `$command` to `/proc/sysrq-trigger`

```
# echo $command > /proc/sysrq-trigger
```

Useful examples to save our system:

- Only if in GUI environment, and if X server, wayland or for some reasons the keyboard is taken by another resource, then lets put in raw mode then `Ctrl+Alt+fX` switch to another virtual console:

  ```
  Press combo key 'Alt+SysRq+r'
  ```

- Will attempt to sync all mounted filesystems, flushes cache to disk and perform an Emergency Sync `Emergency Sync complete` - combo keyboard `Alt+SysRq+s` or:

  ```
  # echo s > /proc/sysrq-trigger
  ```

- Send a SIGTERM to all processes, except for init - combo keyboard `Alt+SysRq+e` or:

  ```
  # echo e > /proc/sysrq-trigger
  ```

- Will call the oom killer to kill a memory hog proces - combo keyboard `Alt+SysRq+f` or:

  ```
  # echo f > /proc/sysrq-trigger
  ```

- Reboot the system **danger: without syncing or unmounting your disks** - combo keyboard `Alt+SysRq+b` or:

  ```
  # echo b > /proc/sysrq-trigger
  ```

- Save files and restart your system if **kernel is still in safe state** use the following combos and **wait some seconds between each combo to give it time**:

  ```
  Combo key  Alt+SysRq+s
  Combo key  Alt+SysRq+e
  Combo key  Alt+SysRq+s
  Combo key  Alt+SysRq+i
  Combo key  Alt+SysRq+u
  Combo key  Alt+SysRq+b
  ```

- **If kernel is in bad state do not sync as you may cause corruption, (but modern filesystems are robust)**:

  ```
  Combo key  Alt+SysRq+e
  Combo key  Alt+SysRq+i
  Combo key  Alt+SysRq+u
  Combo key  Alt+SysRq+b
  ```

- Emergency or debug shells use the above to sync data to disk, and to reboot the system.
- Send a SIGKILL to all processes, except for init - combo keyboard `Alt+SysRq+i` or:

```
# echo i > /proc/sysrq-trigger
```

- Secure Access Key (SAK) Kills all programs on the current virtual console. (Kill trojan programs running at console that could grab password on logins)

```
# echo k > /proc/sysrq-trigger
```

Debug system for **system or kernel developers only**:

show-memory-usage

```
# echo m > /proc/sysrq-trigger
```

show-backtrace-all-active-cpus

```
# echo l > /proc/sysrq-trigger
```

show-task-states

```
# echo t > /proc/sysrq-trigger
```

show-blocked-tasks (uninterruptable)

```
# echo w > /proc/sysrq-trigger
```

crash kernel

```
# echo c > /proc/sysrq-trigger
```

You can also debug locks, etc...

4.3 Debug boot systemd - later boot stage

- Debug systemd boot problems if it fails - Kernel boot command line options:

```
systemd.debug-shell=1
```

- systemd boot log inspection - Get logs of boot

```
sudo journalctl -b
sudo journalctl -b -1
sudo cat /var/log/syslog
sudo cat /var/log/messages
```

- Failed services

```
sudo systemctl --failed
```

- Status of service

```
sudo systemctl status sshd.service
```

- Debug slow boot

```
sudo systemd-analyze
sudo systemd-analyze blame
sudo systemd-analyze critical-chain
```

- Config file:

```
/etc/systemd/system.conf
```

- Do your own research

5. Conclusion

- These debug techniques made a root shell available, make sure to undo anything for security reasons.
- We did see quickly Linux boot process in general
- We did see systemd boot process targets
- We did see Linux boot process debugging techniques
- We did see some techniques on how to save your data or backup your system
- We did see example of a systemd timer job (cron like tool)
- We did see discuss some security points across the whole document

Sahitou!

Figure 5: Constantine