Linux System Administration

PM-TRTNB319

Zsolt Schäffer, PTE-MIK

Legal note

These slides form an outline of the curriculum and contain multiple references to the official Red Hat training materials (codes RH124, RH134 and RH254), since students of this subject are also eligible for the named RH learning materials. Some diagrams and pictures originating from the RH courses will show up in this outline as there is an agreement in place which allows for our Faculty to teach this subject based on the Red Hat curriculum.

All other pictures and diagrams sourced from third parties are explicitly marked with a reference to the origin.

Legal note

All students of PTE-MIK who have officially taken the course „Linux System Administration" are eligible to download these slides from the Faculty's internal network, and are eligible for license keys for Red Hat System Administration online learning materials at rhlearn.gilmore.ca as part of their training program.

Do not share, redistribute, copy or otherwise offer these learning support materials, license keys or account information either for money or free of charge to anyone, as these materials contain **intellectual property.**

Unauthorized distribution is both against the law and university policy and will result in an in-campus inquiry, suing for damages and/or criminal prosecution by the University of Pécs, Red Hat Inc. and other parties.

Linux processes, jobs

Lecture 4

Chapter 1

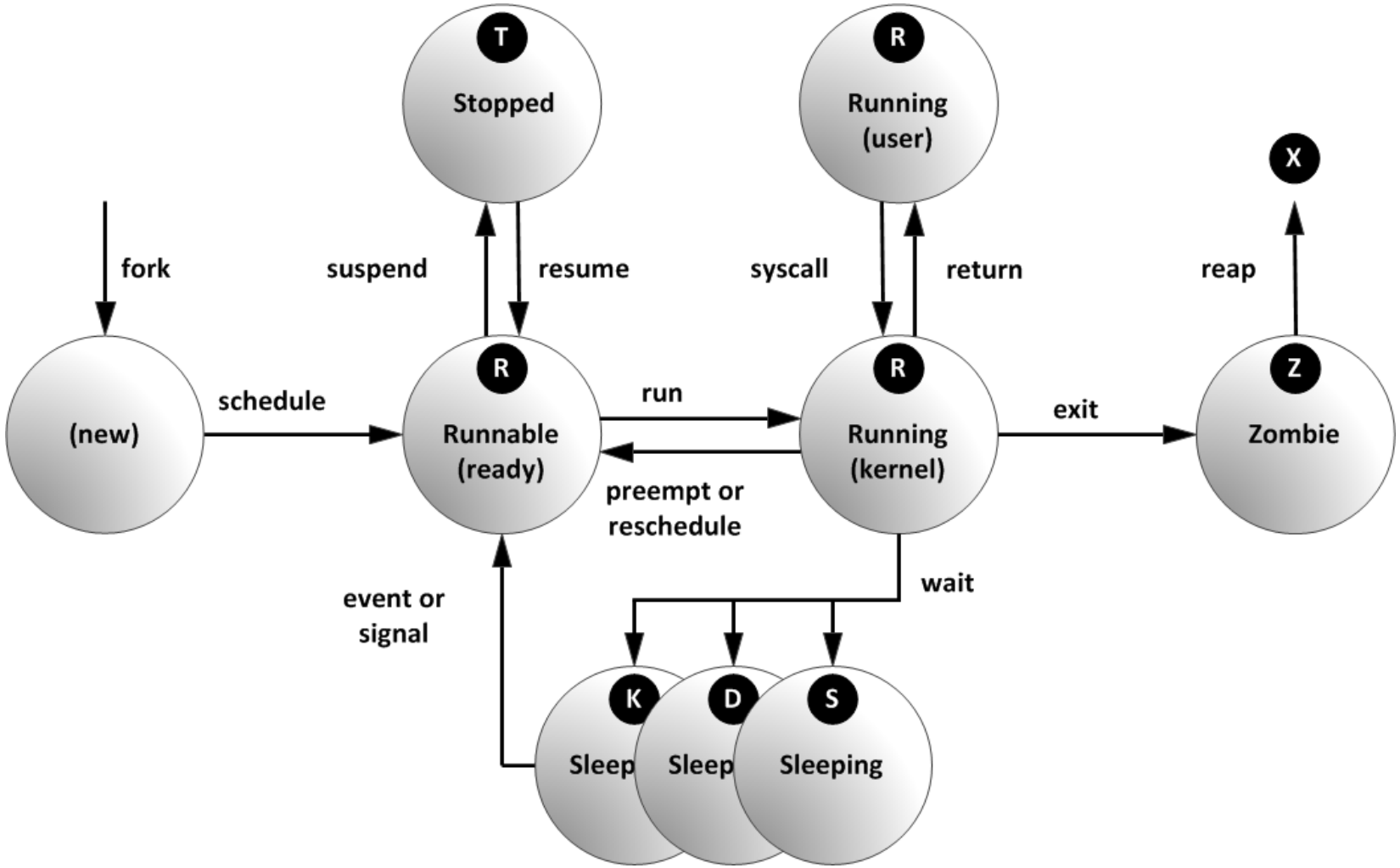
Processes

- Processes are running instances of an executable program. (Remember Lecture 1)
- Every process has a virtual, isolated address space configured in the MMU. It also has at least one thread of executable code.
- Threads are independently scheduled, just like processes, but threads of the same process share the same virtual address space.
- Processes also have some metadata and accounting information. For e.g. PID, PPID, security properties (SELinux context, capabilities), ownership (UID, EUID, GID, EGID), a current scheduling context, environment variables (refer to shell in Lecture 2), a list of allocated resources (mmaps, fds, open ports in the TCP/IP stack), etc. (Try **ls -la /proc/[PID]/fds**)

Scheduler

- Processes only get a series of limited time slices to execute code on the CPU. Each CPU runs only 1 process at a time. The time slices are assigned/distributed by the scheduler, which is a part of the kernel binary.
- The scheduler is run periodically by a hardware timer interrupt at regular intervals. The scheduler decides what process to put next in the CPU for active execution. This involves saving the instruction pointer, the stack pointer and other register states of the 'leaving' process, and loading the same registers for the newly scheduled one.
- There are different scheduling algorithms and different considerations to select the next process in each algorithm. For e.g.: how long has the process been starving, how 'nice' the process is, what state does the process currently have (waiting for disk data, waiting for a network packet.), etc.

Process states



Process states

- S,D,K states: When a process is waiting for user input, a disk read or a network packet, it would be inefficient for it to regularly check for the required condition (polling). Instead the scheduler itself is notified, that it should not give a time slice to the process in question, until a certain event occurs. (E.g. a packet arrives to the network port, which the process is listening on). In this case the process is in the **sleeping state**. Look up the select() syscall and synchronous I/O for more.
- T state: A process can also be manually set aside from the pool of schedulable tasks, by sending a stop signal. This is reversible with a resume signal. A debugger also 'pauses' processes if a breakpoint or certain condition is reached. This is called the **stopped state**.
- R state: All the processes that are not sleeping/stopped are eligible candidates for immediate activation by the scheduler. This is called the **runnable** or **ready** state.

Process states

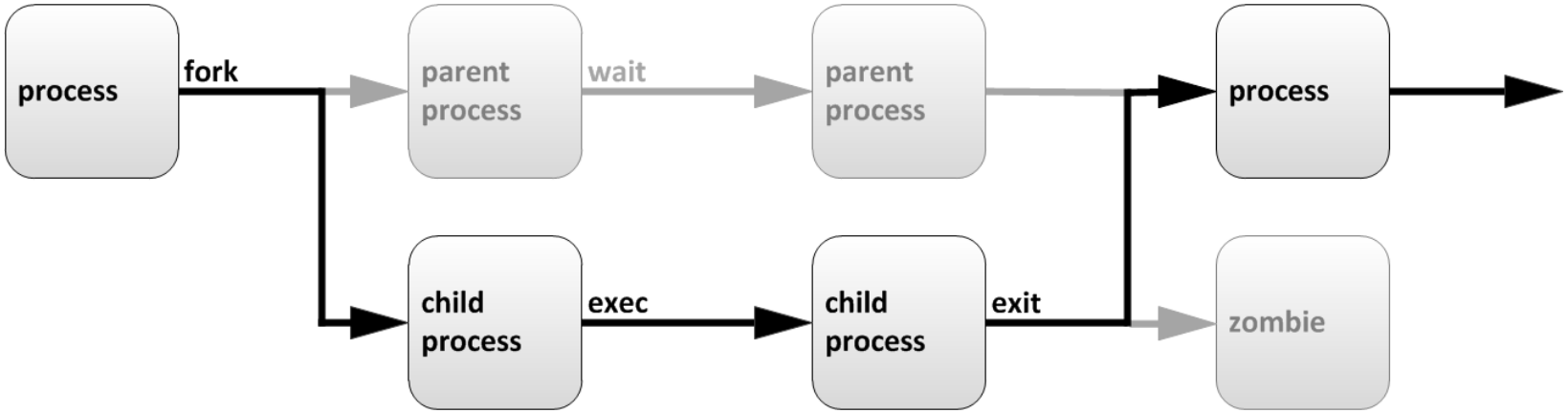
- A new process is also put in the pool of ready processes.
- The scheduler selects one process per CPU from the runnable pool and puts it in the **running** state. A running process is the one actively using the CPU core. The program flow can take excursions when executing syscalls, which are implemented in kernel space. The user space process is not progressing until the syscall function returns.
- When the time slice is over, the running process is preempted and put back to the runnable pool. Or if it requested an I/O operation, it is put in the waiting state.
- The selection process (the scheduling algorithm) usually runs 100 to 1000 times per second, depending on scheduler algorithm, the current dynamic of the load and some compile time kernel parameters.
- Refer to **RH124 CH 7.1.** for more details

Process creation

- In Linux a process can only create an other process by replicating his own self. There are multiple ways for this, like vfork (obsolete), fork, clone, etc.
- The fork() syscall creates an exact replica of the process. A new PID is assigned and PPID adjusted. Otherwise everything is an exact copy, including owner, security context, fds, environment, even memory page layout. Today's optimized kernels don’t copy the whole mem space of processes, instead the pages are initially shared and upon modification a dedicated copy is made on-the-fly to an available chunk of memory (COW).
- Note, that at the point of returning from fork() there are two exactly matching processes, with same binary code and same execution state: both in about to return from a system call. The call returns o inside the child process and returns the PID of the child in the parent process.

Process creation

- The exec() syscall replaces the current executable image with something loaded from a binary file. This is often called directly after fork(). Control is never returned to the original program from exec.
- The clone() syscall can create a child object with a more refined control over what is going to be shared, and what is going to be copied. Clone can create threads inside the same process space and also create full proc. replicas like fork(), it depends on arguments passed to clone().



Process creation

- When a child process finishes or terminates irregularly, the parent process is signaled and is expected to read the return value of the child.
- A process will be in **zombie state**, when it is terminated but not queried yet by the parent (this is called reaping, usually short term). A zombie does not hold resources but still reserves a PID.
- Zombie processes can't be killed, since they are already dead. They can be cleaned up by killing their parent.
- The zombie children of a killed process fall under the parenthood of init (PID 1), which regularly reaps them.
- Their accumulation can be dangerous, because there is only a finite number of PIDs available: 32767 in a 32 bit system.

Process ownership

- It comes from the nature of how processes are created, that the owner of a child process is the same as the owner of the parent (user and group).
- There is syscall for changing the UID to another one at runtime, named setuid(). This can only be done by root-owned processes. It is common, that daemons -after being started by init as root- give up their privileges for security reasons and fall back to a technical user like for e.g. apache. (note RUID, EUID, RGID, EGID)
- The other exception is when starting a process from a binary with the setuid flag. In this case the EUID is set to the owner of the file instead of the executor. (Lecture 2)
- Take the example of running passwd as regular user to change the hash in the /etc/shadow file. This file is not read/writable by regular users, but /bin/passwd binary has setuid flag, and is owned by root. (note RUID's role)

Listing processes

- You can use th **ps** command to list processes. The -e switch enables listing of all processes, otherwise own processes are printed, only the ones with a comtrolling terminal. Switch -f turns on additional columns like user, command, arguments and PPID. -H turns on process tree view.
- Note the BSD style equivalent is **ps auxf**
- Use the **top** command to display a repeatedly updated, sorted list of processes. Note the f command key for field selection, H for displaying threads, 1 for per-CPU load, u for user filter, k for kill, r for renice, and q for quit.
- Use **pstree** to see parent-child relationship formatted as a tree with ASCII graphics.
- For more on ps refer to **RH124 CH.7.1.** For more on top refer to **RH124 CH7.6.**

Load average

- The load average triplet is the load numbers calculated exponential moving average for the past 1, 5 and 15 minutes. Try the **top** and **uptime** commands.
- The load number show the total number of processes currently in the wait queue for being scheduled to the 'run' state plus the number of processes waiting for I/O.
- The load average can only be interpreted in conjunction with the number of logical CPU cores (hyperthreads).

Shell command line

```
# cat /proc/uptime
1573985.02 154800441
# cat /proc/loadavg
0.00 0.01 0.05 1/103 10130
# grep "model name" /proc/cpuinfo | wc -l
4
```

Signals

- Signals are software delivered interrupts to a process.
- A signal source can be an external event (like I/O request), a hardware originated critical event (SIGSEGV, SIGFPE), or an explicit user command. (e.g. a key combination in terminal or the kill command)
- Signals are meant for the scheduler, and always target a specific process. Some of the signals can be handled or overridden by the targeted process. In this case a handler function (implemented in the user space application itself) is triggered. Otherwise the scheduler itself takes direct action to the process (e.g. unschedules, resumes, or forcefully terminates the process).
- Some known signals: SIGINT (ctrl+c), SIGTERM, SIGTSTP (ctrl+z), SIGCONT, SIGKILL, SIGHUP
- Refer to **RH124 CH 7.4.** for more details.

Sending signals

- You can use the **kill [-signal] PID** command to send a specific signal to a process defined by its ID number. If the signal parameter is omitted the default SIGTERM is sent. Use the -l switch to list the available signals. Signal numbers are architecture dependent, but names are standardized.
- You can use the **killall [-signal] pattern** command to kill processes matching 'pattern' by name. Use -u to filter for user name.
- The **pkill** command can also send signals to multiple processes by filter criteria. To forcefully log out a user: **pkill -SIGKILL -u bob** This kills all his processes. To terminate a single problematic process: send SIGTERM to the bash process of the controlling terminal. Bash will survive, but it's children won't.
- Complete practice **RH124 CH 7.5.**

Jobs

- Processes and schedulers are operating system concepts. On the other hand jobs are shell related entities. Every pipeline of commands entered in the shell is a job. The processes in a pipe chain are the members of the same job.
- A shell can handle multiple jobs, but only one job can be in foreground. The fg. job is the only one that can get characters and control signals from the terminal input.
- Background jobs can be either be in executing state or can be suspended and resumed later. To start a job right in the background add an **&** (ampersand) to the end of the command. To suspend and send an already executing fg. command to background press **ctrl+z**.

Shell command line

```
# example_command | sort | mail -s "Sort output" &
```

Jobs

- Jobs sent to background with ctrl+z are also immediately paused (suspended).
- To list jobs, use the **jobs** command.
- To bring a job back to foreground use **fg %jobnum**. This will also make it resume.
- To resume a suspended job while still keeping it in the background use **bg %jobnum**.
- Use **ctrl+c** to ask a process for terminating itself cleanly.
- Refer to **RH124 chapter 7.2.** for more.
- Complete practice **RH124 CH 7.3.**

Process priority

- Processes managed by the SCHED_NORMAL scheduler policy have a numeric representation of their priority.
- This is called niceness. A process can have a nice value of any integer ranging from -20 to +19.
- A nice process politely lets others take the CPU first. The higher the nice value the lower the priority.
- Regular users can only increase niceness and can only assign 0 or positive initial nice values. Root can also define negative values and decrease the niceness of a running process.
- To start a process with a specific nice value use the following example.

Shell command line

```
# nice -n 15 bitcoin_miner &
```

Process priority

- To print the ninceness of processes:

Shell command line

```
# ps axo pid,comm,nice -sort=-nice
  PID COMMAND          NI
   15 netns           -20
   16 perf            -20
   17 writeback       -20
[...]
10624 sshd             0
10626 bash             0
10653 ps               0
   18 ksmd             5
  554 alsactl         19
```

- To re-nice an already running process use the following command: **renice -n <LEVEL> <PID>**.
- To change niceness in top, press r then type the PID, then the new nice level.

Practice

- For listing, sorting, signaling, and re-niceing processes with the **top** utility refer to **RH124 CH 7.6.** and **RH134 CH 5.3.**
- Complete **RH124 CH 7.7. and 7.8.**
- Complete **RH134 CH 5.3. and 5.4.**
- Also note other useful tools like: iftop, iotop, iostat

su, sudo and setuid()

Lecture 4

Chapter 2

Switching users with su

- It is a security based recommendation even for administrators to use the system as a regular unprivileged user and gain administrative privileges only temporarily, when a specific command requires it.
- To switch user account without logging out, use the **su** command. The password of the to-be-acquired user will be prompted for. Ending the elevated session with **exit** will return to the original user session. (It has not been logged out, just been passive during the su session)
- Privilege escalation is achieved to following way: the su binary has the setuid bit set and owned by root. It will start as a root owned process. It will then check for password validity and call setuid() syscall to set its own EUID according to the requested username, than spawn a shell process with the relevant user. (Note the importance of trusted binaries owned by root with setuid bit.)

Switching users with su

- Take the the following example: **su apache -c 'cat /var/www/html/index.html' -s /bin/bash**
- If a username is given as parameter, it switches to that one, otherwise if unspecified, root user is assumed.
- The su binary can be instructed to override the users shell with the -s switch.
- It can spawn a login shell with -l (remember Lecture2).
- You can instruct su to run a single command as another user, then end the session immediately. This is achieved with the -c switch. The command should be enclosed in quotes if it contains any spaces or special characters.
- To simply switch to root with root environment (login shell) use this: **su -**
- **Target** users **password** is **always** required, except when already root, and want to switch to a regular user.

Temporary elevation with sudo

- The **sudo** command works in a similar way, but has a fine grained control over who can change to what user and what subset of commands he/she can execute with the elevated/changed user credentials.
- Similar to su you can use -s switch to specify shell, but use -i to start a login shell and -u to specify a user. If username is omitted root is implied.
- Without -i the sudo command is useful for executing only one single command in elevated mode.

Shell command line

```
# sudo usermod -aG wheel elvis
```

- A ruleset of who can do what is specified in the file /etc/sudoers file. Sudo will ask for the current users password as the user has prove who he/she is. The elevated command will spawn if rules allow. **Target** users **password** is **never** required when using sudo.

/etc/sudoers file format

- The ruleset is specified in this file. One rule in each line. The /etc/sudoers.d drop-in directory is also parsed for files with lines of the same format.

/etc/sudoers

```
root      ALL=  (ALL)  ALL
%chem     CHEM= SHUTDOWN, MOUNT
harvey    ALL = NOPASSWD: SHUTDOWN
<user l> <host l>=<operator l> <tag l> <command l>
```

- All lists can be specified as a comma separated list. Aliases are also possible to use. (E.g SHUTDOWN is an alias for /sbin/shutdown, /sbin/halt, /sbin/poweroff) ALL is a universal alias for everything/everyone.
- User list is a list of users who are allowed to run the command specified by the rest of the line (% for groups).
- Host list is a list of hosts where the given rule applies. (Mass distribution of rules, rules not relevant to the current host are ignored.)

/etc/sudoers file format

- Operator list contains the possible users to be 'impersonated' by the sudo command. (Refer to -u switch) This section can be omitted, then ALL is assumed.
- Tag list is a : (colon) separated list of tags, like NOPASSWD:NOEXEC: Can be omitted if no tags are needed.
- Command list is a list of allowed commands for the given user to run.
- Users can be given restricted rights to do limited task as root or other users, while not giving them full access to modify the system.
- Use visudo command to open up sudoers in the default editor. ($EDITOR)

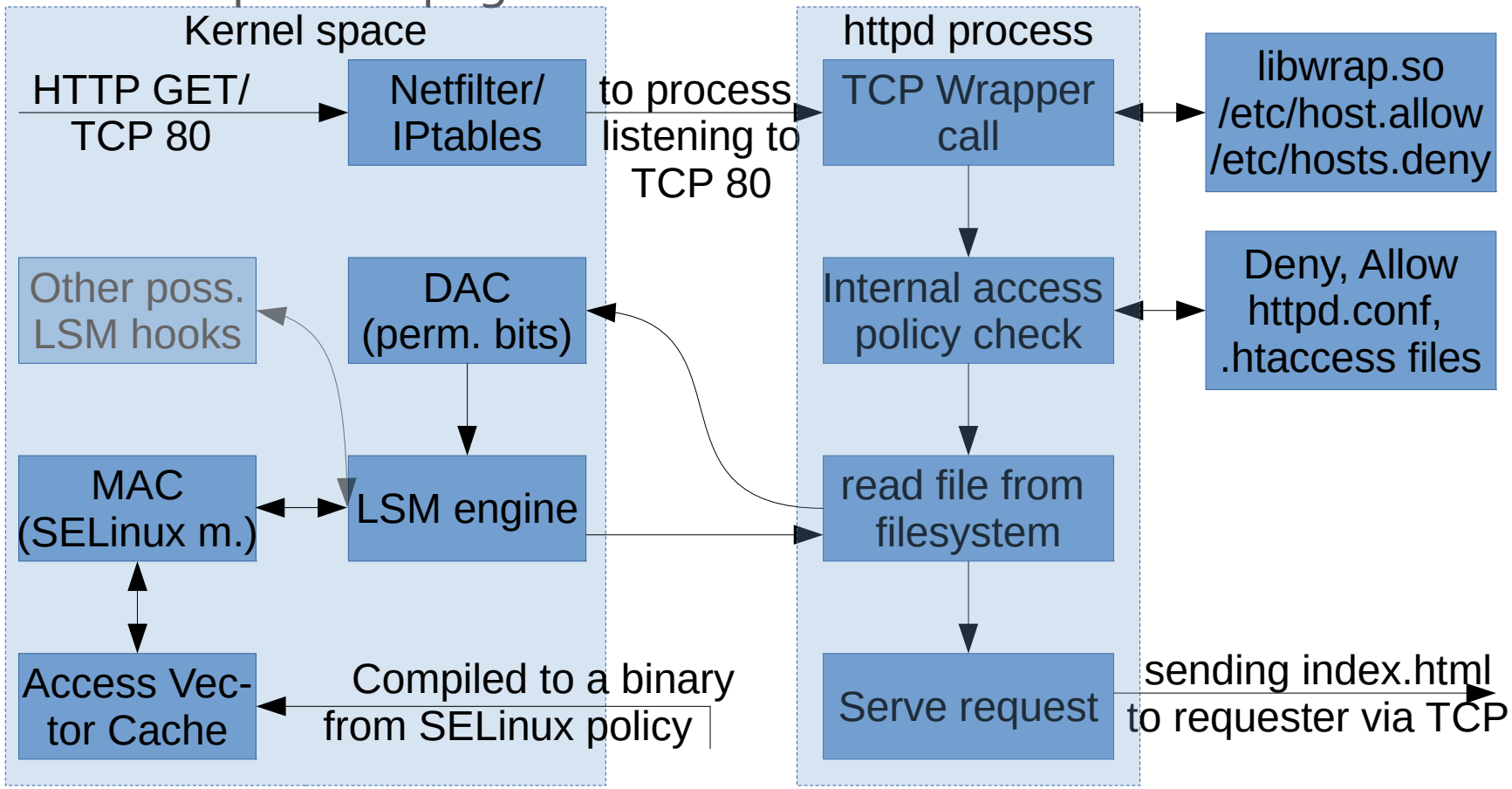
SELinux

Lecture 4

Chapter 3

General flow of security measures for network resources

- In Linux several checks are performed before a resource (e.g. file) is served to a requester. Consider this diagram depicting the flow of events when a user wants to access a simple web page over the network.



SELinux

- All actions taken on the system which invokes Linux kernel calls are passed through LSM (Linux Security Modules) framework to decide whether a call is to be allowed or not.
- Security Enhanced Linux is an implementation of the US DoD style Mandatory Access Control model for Linux.
- SELinux has LSM hooks, therefore it takes part in the LSM decision process.
- It's based on several earlier projects of US NSA.
- The whole set of rules is called the SELinux policy. It can be quite complex, therefore it is compiled to a binary object (Access Vector Cache) for speed considerations.
- A given call's context is compared in the kernel against the AVC to rule either grant or denial.

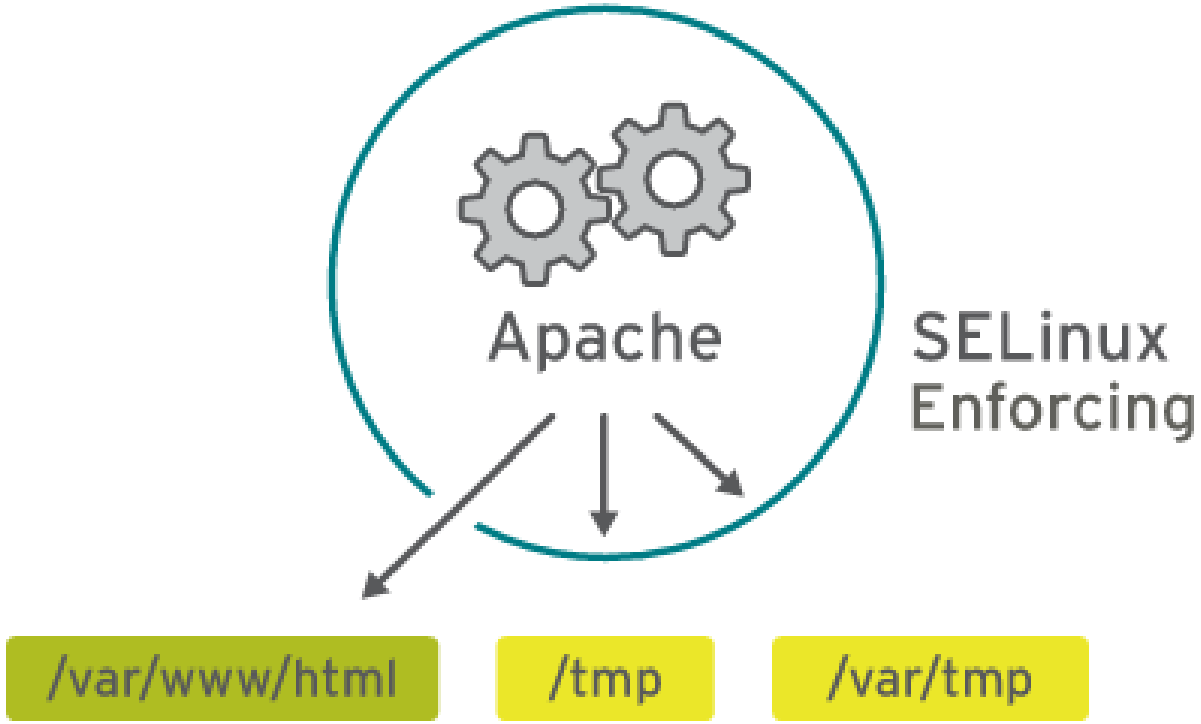
SELinux context

- Every network port, filesystem object and process has an SELinux context.
- The mapping of context to FS objects is called labeling.
- The SELinux context involves 4 SELinux attributes:
  - **user:** SELinux user is independent of the Unix user model, though a mapping can be done. Most users and files belong to the same SELinux user.
  - **role:** a user is authorized by the policy for a defined set of roles. A role is authorized for specific set of domains. This is an intermediary layer, seldom used.
  - **domain (type):** the most involved attribute in the targeted policy. (More on this later)
  - **level (category):** a level of severity with an optional category specifier. Involved in MLS/MCS.
- Refer to **RH134 CH 7.1.** for more.

SELinux policies

- There are several ways to structure the policy (ruleset). The most common ones are listed here. You can install a specific policy with the package manager. Throughout this course you'll **not** need to modify the policy itself.
  - **targeted policy:** this is the default in RH. It is most involved with types. An example rule translated to plain English should express something like this: allow httpd_t type processes to access httpd_log_t type files for open and append operations.
  - **mls policy:** users and processes are subjects, labeled with a clearance level. Files and passive components are objects labeled with classification. (level and categ. attributes.) The ruleset defines the allowed data flows (e.g. write only, read only, both) between subj. and obj. of specific levels.
  - **minimal:** a reduced/simplified version of targeted.

SELinux isolation concept



SELinux modes

- There are two enable modes for SELinux and also a disabled setting. You can switch between modes at run-time, but can only enable or disable across reboots.
  - **enforcing mode:** all request are evaluated against the policy and result is forced in effect. Logging of AVC denials also happens.
  - **permissive mode:** all request are evaluated against the policy, but only logging happens in effect. Actual denial never occurs. This mode is only recommended for troubleshooting.
  - **disabled:** the subsystem is completely disabled, the SELinux kernel modules are not loaded at all.
- You can switch modes with the **setenforce** command and check with **getenforce**. The default mode and policy is set in the **/etc/selinux/config** file.
- Refer to **RH134 CH7.2.** and **7.3.** for more.

SELinux modes

- A reboot is required if you set to disable or choose to enable SELinux. The newly booted system will assume the settings in this file. The selinux= and enforcing= kernel boot time arguments override this file.

/etc/selinux/config

```
# This file controls the state of SELinux on the
system.
# SELINUX= can take one of these three values:
#     enforcing
#     permissive
#     disabled - No SELinux policy is loaded.
SELINUX=enforcing
# SELINUXTYPE= can take one of these two values:
#     targeted - Targeted processes are protected,
#     minimum - Modification of targeted policy.
#     mls - Multi Level Security protection.
SELINUXTYPE=targeted
```

SELinux booleans

- Some of the rules also evaluate binary state variables.
- Take the following example. Some systems admins need to enable the possibility of running cgi scripts under the web server because it's needed by some web applications, others for e.g. only allow static content for enhanced security.
- When a system administrator decides on this question, he/she may be tempted to modify the policy to allow/disallow this feature. With SE booleans the choice can be brought in effect without modifying the policy itself. Only the state of a variable has to be switched.
- The corresponding rule in the policy for this example would say: allow execute operation on httpd_script_content_t files for httpd_t type processes **if** the httpd_enable_cgi boolean is set to true.
- Refer to **RH134 CH 7.6.** for more, and complete **CH 7.7.**

Get SELinux related information

- SE booleans can be used to fine tune the policy to make selective adjustments.
- You can list all the SE booleans with the **getsebool -a** command. You can set them with **setsebool**. You can use the -P switch for setting a value as permanent. (across reboots). Also note semanage boolean -l and -C.
- You can use the **sesearch -A | grep [keyword]** command to search the policy for expressions, types, rules. Use the -C -b switch in addition to restrict your search to SELinux booleans and print conditions.
- Try the system-config-selinux graphical tool for managing SELinux.
- Use the -Z switch in general to get SELinux context information. E.g.: **ls -Z**, **ps -Z**, **id -Z**
- You can use **sesearch -T** to look for transitional rules.

SELinux useful tips

- Transitional rules define what attribute can be changed to what. E.g.: a root shell of unconfined_u: unconfined_r:unconfined_t type can transition to almost any context but not the other way around: a confined type cannot transition back to higher privileges.
- Use the **runcon** command to run a command with specific SELinux context. This command, like others also falls under the rules of transition. Use of runcon is only recommended for troubleshooting/debugging purpose.
- Use the **man -k selinux_keyword** to search for man pages containing the keyword. E.g. a type, a boolean, …
- Create a file in the root directory called **/.autorelabel** to automatically restore the default context of all files in the filesystem at boot time.

Managing SELinux context

- Typically the location of a newly created file's context is determined through a way of applying defaults, which are described in a database. When moving existing objects to different locations the original context is preserved. This may be unintended in the new location.
- To explicitly set a file's context use the **chcon** command. Use the -t switch to only alter the type attribute. Use the **restorecon** command to reset a files context to the database defined, location dependent, default value. The -R switch stands for recursive, -v stands for verbose.

Shell command line

```
# chcon -t httpd_sys_content_t /var/www/html/app1
# restorecon -Rv /var/www/html
restorecon reset /var/www/html/file1 context
unconfined_u:object_r:user_tmp_t:s0
      -> system_u:object_r:httpd_sys_content_t:s0
```

- Refer to **RH134 CH 7.4.** for more.

Managing SELinux context defaults

- To change the database of default file-contexts use the **semanage fcontext** command. Also note **semanage port** for network ports. Use the -l switch to list the current database. Use -d to delete a line, use -a to add a line. Note that the order of evaluation is the same as the order of entry (overlapping rules for subdirs).

Shell command line

```
# semanage fcontext -l
...
/var/www(/.*)? all files sys_u:obj_r:httpd_sys_cont_t:s0
...
# semanage fcontext -a -t httpd_sys_content_t '/srv(/.*)?'
# restorecon -RFvv /srv
```

- The database is stored under /etc/selinux/targeted/contexts/files directory within both text source files and binaries. If you edit the text file directly, it has to be recompiled. (sefcontext_compile)
- Complete practice **RH134 CH 7.5.** (required for 7.9.)

Troubleshoot SELinux

- The most common reason for AVC denials is an incorrectly set file context. (E.g. file moved in a new location) → use **restorecon**. This has the most narrow impact on system security, it only affects the given file.
- In some cases a too restrictive policy can cause legitimate access intents to fail. → Look a related boolean with **getsebool -a** or **semanage boolean -l** and enable it with **setsebool**. This has a wider impact on the system.
- In can happen on very rare occasions that the policy installed by the package manager has bugs. → Add an allow rule with careful consideration. This can have a very wide impact on system security.
- Use **sealert -l [uuid]** command to get more details on a specific AVC denial. Its ID will be in the system log.
- Complete practice **RH134 CH.7.9-7.10 .** (Do 7.5. fisrt!)

Recommended reading

Lecture 4

Chapter 4

Relevant chapters in RH:

- RH 134 CH7 (SELinux)
- RH 124 CH7 (Manage linux processes)
- RH 134 CH5 (nice)

Read by next lecture:

- RH 124 CH14 (Linux filesystems)
- RH 134 CH9 (Partitions and filesystems)
- RH 134 CH10 (LVM)

Use your rhlearn.gilmore.ca login to access the learning materials.

Thank you for your attention!

Lecture 4, PM-TRTNB319, Linux System Administration

Zsolt Schäffer, PTE-MIK