# Practice M2: Network and System Security (openSUSE)

This practice assumes that you are working in an on-premise environment

All tasks can be achieved under different configurations (host OS and/or virtualization solution) with the appropriate adjustments

This practice is oriented mainly towards **openSUSE Leap 15.3**

## Part 1: Kernel Parameters. Resource Limits. ACLs

For this part we will need an infrastructure like this one:

Diagram

Description automatically generated

Machines can be with or without graphical environment

Network settings shown on the picture reflect the ones, used during the demonstration. You should adjust them according to your setup

### Kernel Parameters

Let’s log on to the machine

All steps can be executed without significant modifications on the other two machines (and distributions) as well

#### General

As we already know, we can see what’s going on with our system by checking the content of the **/proc** file system

If we list its content

**ls -l /proc**

We will see plenty of files and folders

We can identify files like **/proc/cpuinfo**, **/proc/uptime**, **/proc/version**, etc.

Some of the files here are read-only while others are writeable

Should we need more information about the **/proc** filesystem, we can ask for it with

**man 5 proc**

Search for example, for **/proc/sys**

When done, exit **man**

Let’s install the **tree** utility if not installed

**sudo zypper install tree**

And take a different look at the **/proc** filesystem and especially on the **sys** folder

**tree -L 1 /proc/sys**

Here, we can both read and write (or change) parameter values and alter the system’s behavior

Let’s first explore the **/proc/sys/kernel** folder

**ls -al /proc/sys/kernel**

We can ask for the content of **hostname** and **domainname** files

**cat /proc/sys/kernel/{host,domain}name**

We can see that the **hostname** is populated and the **domainname** is not. This is the **NIS/YP** domain name and should not be confused with the DNS domain name

Instead of browsing the files and folders in the **/proc/sys** folder, we can use a command to retrieve all variables and their values

**sysctl -a**

We can use it to filter the output. Let’s do it for the above two variables

**sysctl -ar name$**

What we can notice here is that even though the saw that for example the **hostname** is accessible via **/proc/sys/kernel/hostname**, the actual variable is **kernel.hostname**

Let’s experiment with the other one – **kernel.domainname**

There are several ways to interact with a kernel variable

For example, we can set the variable to a value with

**echo "lsaa" | sudo tee /proc/sys/kernel/domainname**

This should result in changed domain name. Let’s check

**sysctl -ar domainname**

Indeed, it has changed. Let’s use another approach to change it again

**sudo sysctl -w kernel.domainname='demo'**

This way, we see immediately that the variable has been changed

The changes made with both above approaches are temporary. After a reboot, the variable will be restored to its configured value

In order to make a change permanent, we must add it in a configuration file

This can be done either by editing **/etc/sysctl.conf** or adding a file in **/etc/sysctl.d/**

Let’s first check the content of the main configuration file

**cat /etc/sysctl.conf**

Instead of editing this file, it is a better idea to create a file and store it in **/etc/sysctl.d/**

Let’s first explore what is there

**ls -al /etc/sysctl.d/**

Depending on the distribution and the particular installation the content may vary. Let’s create a file

**sudo vi /etc/sysctl.d/50-kernel-domainname.conf**

And put inside the following text

**kernel.domainname=lsaa**

Save and close the file

Depending on where we stored our setting, we may follow different path to apply the changes

For example, if we edited the **/etc/sysctl.conf** file, we could execute just

**sudo sysctl -p**

If we created a file in **/etc/sysctl.d/** folder, we must execute  
**sudo sysctl -p /etc/sysctl.d/50-kernel-domainname.conf**

Alternatively, we could execute the following to read and apply the settings

**sudo sysctl --system**

#### ICMP Echo

We could use the above techniques to adjust the system’s behavior in terms of security as well

For example, we can turn off ICMP echo response functionality

Let’s first check what variables we have

**sysctl -ar icmp**

We can see that the **net.ipv4.icmp\_echo\_ignore\_all** is set to **0**

In order to activate it, we must set it to **1**

**sudo sysctl -w net.ipv4.icmp\_echo\_ignore\_all=1**

Now, if we try to ping ourselves, depending our network configuration we may or may not have success

If we have IPv6 configured, and execute

**ping localhost**

The ping will succeed because it will resolve the localhost to its IPv6 address

And thus, we can use

**ping -4 localhost**

Now, there should not be any reply

Of course, we can achieve the same effect using other means. For example, change a firewall setting

### Resource Limits

First, we can check if the **pam\_limits.so** module is configured

**grep pam\_limits.so /etc/pam.d/\***

As we can see, it is required by the used authentication mechanisms

The main tool to explore and interact with the resource limits is the **ulimit** built-in command. As such, we can use **man** to ask for information about the **bash** and then navigate to **ulimit**

Alternatively, the usual approach can be used

**ulimit --help**

If we execute the command without any options, it will show us the **file size** (**-f**) limit

Should we want to see all limits at once, we can execute

**ulimit -a**

Let’s examine just the maximum number of user processes

**ulimit -u**

Now, let’s set the soft limit for the maximum number of user processes to **5120**

**ulimit -u 5120**

The above will set both **soft** and **hard** limits. If the number is bigger than the current value, the operation won’t succeed. So we can try with a smaller value

If we want to modify the **soft** limit, we can execute

**ulimit -S -u 2048**

Should we want to modify the **hard** limit, we must use **-H** instead

And ask again for its value. It appears that if nothing is specified, we are seeing the **soft** limit

Changes made this way are for the current session only. Should we want persistent changes, we must either edit **/etc/security/limits.conf** file or add a file to **/etc/security/limits.d/** folder

Let’s first examine the main configuration file

**cat /etc/security/limits.conf**

Detailed information can be obtained from **man** with

**man 5 limits.conf**

Now, that we know the structure of the limit rules, let’s create a file **10-maxlogins.conf** in **/etc/security/limits.d/**

**sudo vi /etc/security/limits.d/10-maxlogis.conf**

Add the following line to limit maximum logins for all users to **2**

**\* - maxlogins 2**

Save and close the file

Now, open or try to open a few more sessions. After the second, you should not be able to open additional sessions

### File Mode

By now, we should know that there are two types of file and folder permissions – **standard** and **special**

When a new filesystem object is created, its permissions are set based on the defined **umask**

We can use the **umask** tool to either check (when run without parameters) or set (a number is provided as parameter) the default umask

Being logged with a regular user and being in your home directory, check what’s the current umask and set it to **0002** if something else (this change will go away when you close your session)

**umask 2**

We can omit the leading zeroes

Create a folder

**mkdir temp1**

Now, create one more

**mkdir --mode 774 temp2**

And check permissions of both folders

**ls -ld temp\***

Now, turn on the **sticky bit** on both folders

**chmod o+t temp\***

And ask again for their permissions. Folder **temp1** has a small letter **t**, while folder **temp2** has a capital letter **T**

This is because of the different state of the **execute** permission for **others**

The same behavior can be seen when setting the **SGID** and **SUID** permissions. They will depend on the execute permission for **group** and **user**. The letter will be either **S** (execute not set) or **s** (execute was set)

Should we need to traverse the whole filesystem for files that have for example the SGIDbit set, we can execute

**sudo find / -type f -perm -2000**

Let’s create an empty **file** with

**touch file**

And check its permissions and a lot of additional information with

**stat file**

We can narrow down the returned information to just the permissions in **octal**

**stat -c %a file**

Or **symbolic** notation

**stat -c %A file**

### Access Control Lists

File mode permissions are limiting in a way. We can set permissions for a single user and group

Access control lists are the answer to this

We can install the necessary tools for ACL if they aren’t installed already

**sudo zypper install acl**

In order to check if our system has ACL functionality enabled, we can execute

**grep -i acl /boot/config-$(uname -r)**

Add two additional users

**sudo useradd demo1**

**sudo useradd demo2**

Now, create a top-level folder

**sudo mkdir -m 700 /secret**

And try to change into it with the current (regular) user

**cd /secret**

It must return you an error. Let’s correct it by setting an ACL

**sudo setfacl -m u:<user-name>:rwx /secret**

If we try to change in the new folder, now we will succeed. Return to our folder

We can check how the permissions information for the folder changed

**ls -ld /secret**

We can notice a small **+** symbol just after the permissions block. This is an indication that there are ACL entries on this item

Detailed information for ACL entries can be received with

**getfacl /secret**

We can set default permissions via ACL. This way every new file in this folder will be accessible by our user

**sudo setfacl -m d:u:<user-name>:rwx /secret**

If we ask once again for the ACL entries, we will see a block for the **default** set of entries

Let’s create a file with

**touch /secret/file**

And then ask for its properties

**ls -al /secret/file**

And for the ACL entries

**getfacl /secret/file**

Of course, we can add entries for other users as well. We can go even further, we can add multiple entries at once

**sudo setfacl -m u:demo1:rwx,u:demo2:rw /secret/file**

If we did add an entry by mistake, we can revoke (or delete it)

**sudo setfacl -x u:demo1 /secret/file**

We can check the result

Don’t forget that ACL entries go with the file object. So, if we move it to another folder, they will remain untouched

Of course, we can remove all ACL entries at once with

**sudo setfacl -b /secret/file**

## Part 2: MAC (AppArmor)

For this part we will again a single machine. So, we can continue on the one used during the first part

### AppArmor

Log on to the machine with a regular user account

Check if **AppArmor** is active with

**sudo aa-enabled**

Then check its status with

**sudo aa-status**

We have two sets of commands. In fact, there is a main set with **aa-xxxx** and an alternative one, which starts with **apparmor\_xxxx** and every utility is a symbolic link to the appropriate main tool

In order to enrich our toolset, we can install additional packages

**sudo zypper install apparmor-utils**

Now, if we check, we should notice that there are much more tools

Let’s check where the profiles are stored

**ls -l /etc/apparmor.d/**

Okay. Let’s pretend that we want to create a profile for the **ping** command

Unfortunately, there is one by default, so let’s delete (or move it somewhere else) it first

And now create a new one

**sudo aa-autodep ping**

Now, if list again content of the **/etc/apparmor.d/** folder, we will notice the **usr.bin.ping** file

Let’s examine its content

**sudo cat /etc/apparmor.d/usr.bin.ping**

We can see that the **ping** command is in **complain** mode

Let’s try to use the **ping** command

**ping softuni.bg**

We should see an error. Okay, then try with **sudo**

**sudo ping softuni.bg**

Again, no luck

Let’s check what is in the log with

**sudo aa-logprof**

When asked, hit the **A** key as many times as needed and finally, press the **S** key

Now, our **ping** related profile should have been updated

Let’s check if we can ping

**ping softuni.bg**

We can check again the profile and compare it with the previous one

Finally, let’s switch it to **enforce** with

**sudo aa-enforce /usr/bin/ping**

Now, if we check again the profile, we will see the difference

**sudo cat /etc/apparmor.d/usr.bin.ping**

Try to ping again. It must succeed

## Part 3: Audit and Packet Filtering

For this part we will again a single machine. So, we can continue on the one used during the second part

### Audit

Let’s log on to the machine with a regular account

Add one new **demo31** user with

**sudo useradd demo31**

Let’s check if, how this was written to the logs

**sudo journalctl -xe**

Here, we can see (almost) the whole information about the operation

Audit package is installed by default on the recent **openSUSE** versions

Let’s check if the audit service has been started

**systemctl status auditd**

We can use another command to check the **auditd** status

**sudo auditctl -s**

Now, if we add another user, we should see the difference

**sudo useradd demo32**

And let’s use the new set of tools to explore the information that has been captured

**sudo ausearch -m ADD\_USER --start recent**

We can ask for another message type, for example user logons

**sudo ausearch -m USER\_LOGIN**

No results are being returned. The reason is that we didn’t do a logoff/logon cycle yet

Let’s logoff and during login, let’s make one or two mistakes

Now, that we are back in, let’s repeat the last command. We should see a few related messages

Let’s list only the unsuccessful login attempts

**sudo ausearch -m USER\_LOGIN --success no**

We can check what other message types we can use to filter the audit log by passing an invalid argument to **-m** for example **HELP**

**ausearch -m HELP**

Main configuration file is stored in **/etc/audit/audit.conf**

**sudo cat /etc/audit/auditd.conf**

Help for the file can be get via

**man auditd.conf**

The rules are stored in **/etc/audit/rules.d/** folder

We can check if there are any rules with

**sudo auditctl -l**

No rules by default

Let’s create one. Assume that there is a super important file in the root folder and we want know who has read it

Create the **/readme.txt** file

Create new user **demo33** and set a known password for it

Now, let’s create the rule with

**sudo auditctl -w /readme.txt -p r -k super-secret**

If we ask again for the available rules, we will see our new one listed

Let’s switch to the newly created user and read the file. Then exit back to our user

Now, we can check if anything is captured in the log

**sudo ausearch -k super-secret**

Yes, there are events

This rule that we created will last until we reboot the system or the **auditd** service

Should we want to preserve the rule, we must either add it to the **/etc/audit/rules.d/audit.rules** file or create a new one, for example **/etc/audit/rules.d/super-secret.rules** and we must store only the last part

**-w /readme.txt -p r -k super-secret**

We can remove a rule from the run-time configuration with

**sudo auditctl -W /readme.txt -p r -k super-secret**

### FirewallD

Log on to the machine with a regular user account

Check if the **firewalld** is running with

**sudo firewall-cmd --state**

Or with

**systemctl status firewalld**

If the software is there and not running, enable and start it with

**sudo systemctl enable --now firewalld**

Check which zone is set as the default zone

**sudo firewall-cmd --get-default-zone**

We can check all zones with interfaces in them

**sudo firewall-cmd --get-active-zones**

In a similar way we can get the list of the available zones

**sudo firewall-cmd --get-zones**

We can change the zone of our interface via the **nmcli** command or with the help of **firewall-cmd**

Let’s use the **firewall-cmd**. Here we can first use the **--remove-interface** and then the **--add-interface** or directly

**sudo firewall-cmd --change-interface eth0 --zone internal**

In order to avoid the necessity to specify the zone for every command, we can adjust the default zone as well

**sudo firewall-cmd --set-default-zone internal**

All commands affect the run-time configuration and those changes will be gone if the service or the system is restarted

In order to save the run-time configuration into the permanent one, we can execute

**sudo firewall-cmd --runtime-to-permanent**

Or we can add the **--permanent** option to every command that we want to change the persistent configuration. Then if we want to apply the changes to the run-time configuration as well we usually execute one additional command

**sudo firewall-cmd --reload**

Let’s compare the settings for two different zones

**sudo firewall-cmd --list-all --zone internal**

**sudo firewall-cmd --list-all --zone external**

Let’s remove a service from our default zone and change the permanent configuration

**sudo firewall-cmd --remove-service dhcpv6-client --permanent**

Now, if we check again the configuration of the default zone with

**sudo firewall-cmd --list-all**

We will see that it is unchanged. In order to apply the changes, we must reload the configuration files with

**sudo firewall-cmd --reload**

Now, the changes are reflected and active

In a similar fashion, we can add services. Let’s add the **http** service to our default zone

**sudo firewall-cmd --add-service http --permanent**

**sudo firewall-cmd --reload**

We can add or remove more than one service as well. Let’s remove a few more

**sudo firewall-cmd --remove-service={mdns,samba-client} --permanent**

**sudo firewall-cmd --reload**

And ask for the list of services configured for the default zone

**sudo firewall-cmd --list-services**

We can check what services are defined and available on our system by listing the folder

**ls -al /usr/lib/firewalld/services**

Let’s ask for the content of the **http.xml** file

**cat /usr/lib/firewalld/services/http.xml**

And now, let’s create our own empty service

**sudo firewall-cmd --new-service demo --permanent**

And check the target folder

**sudo ls -al /etc/firewalld/services**

Open the file for editing

**sudo vi /etc/firewalld/services/demo.xml**

And make sure that the content is like this

**<?xml version="1.0" encoding="utf-8"?>**

**<service>**

**<short>Demo</short>**

**<port protocol="tcp" port="8000" />**

**<port protocol="tcp" port="8080" />**

**</service>**

Save and close the file

Now, let’s apply our new service to the default zone and check the result

**sudo firewall-cmd --add-service demo --permanent**

**sudo firewall-cmd --reload**

**sudo firewall-cmd --list-all**

### iptables

*Please note that is not supposed to use* ***iptables*** *directly. Instead, we must use* ***firewalld*** *and should we want, we can switch its backend in the main configuration file* ***/etc/firewalld/firewalld.conf***

Log on to the machine with a regular user account

**iptables** is fading away but should we want to use it, we are free to do it

Before we remove the **firewalld** let’s check if there is at least some part of **iptables** present

**sudo iptables -L**

The command gets executed without an error but returns an empty result set

The reason for this is that in the recent versions of **SUSE** like distributions **iptables** are substituted with **nftables**

Stop and disable the **firewalld** service

**sudo systemctl disable --now firewalld**

Install the **iptables** package

**sudo zypper install iptables**

There is no iptables service anymore

We can check what is the default configuration by executing

**sudo iptables -L**

We can extend the last command to include statistics and interface information

**sudo iptables -L -v**

Should we want to have a numeric representation where possible, we can add one more argument

**sudo iptables -L -v -n**

Let’s flush all rules

**sudo iptables -F**

And check what we have now

**sudo iptables -L**

Let’s add one rule for accepting connections on **22/tcp**

**sudo iptables -I INPUT -m tcp -p tcp --dport 22 -j ACCEPT**

And one more for port **80/tcp**

**sudo iptables -I INPUT -m tcp -p tcp --dport 80 -j ACCEPT**

Now list the rules but with an extended syntax

**sudo iptables -L --line-numbers**

Let’s remove the rule for **http**

**sudo iptables -D INPUT 1**

And let’s save our rules but just the part related to the **filter** table

**sudo iptables-save -t filter**

And redirect the output to a file of our choice

### nftables

*Please note that is not supposed to use* ***nftables*** *directly. Instead, we must use* ***firewalld*** *and should we want, we can switch its backend in the main configuration file* ***/etc/firewalld/firewalld.conf***

Log on to the machine with a regular account

Ensure that all other firewall related solutions are either stopped or uninstalled. Let’s stop them

If you have **firewalld** installed, then execute the following

**sudo systemctl disable --now firewalld**

**sudo systemctl mask firewalld**

To be sure that we are starting with clean set, let’s flush all rules with

**sudo nft flush ruleset**

And then if we ask for the rules, we must receive an empty result set

**sudo nft list ruleset**

Now, we can create a table named **filter** that will cover both **ipv4** and **ipv6**

**sudo nft add table inet filter**

Then, we can add a chain named **INPUT** to the table **filter** table created earlier

**sudo nft add chain inet filter INPUT { type filter hook input priority 0 \; policy accept \; }**

We set the default policy as well

Now, we can ask for the list of all tables and rule sets

**sudo nft list tables**

**sudo nft list ruleset**

We can see what we have defined so far

Let’s add a rule to our chain that will allow all traffic on the **lo** interface

**sudo nft add rule inet filter INPUT iif lo accept**

Then add a rule that will accept all connections based on their state

**sudo nft add rule inet filter INPUT ct state established,related accept**

We can add a rule to accept connections based on their protocol and port

**sudo nft add rule inet filter INPUT tcp dport {22, 80} accept**

Add a rule to count the packets

**sudo nft add rule inet filter INPUT counter**

Finally, add a rule to drop everything else

**sudo nft add rule inet filter INPUT drop**

Instead of two separate instructions we could achieve this with one

**sudo nft add rule inet filter INPUT counter drop**

Now, let’s check our ruleset

**sudo nft list ruleset**

We can either check that the ping won’t work from outside

Or we can install a http server and test that it will be reachable both from inside and outside

The configuration that we have so far will go away if system is restarted

In order to make it persistent, we can save it to a file

We can save (or dump) the rules to a file

**sudo nft list ruleset > demo.rules**

It should look familiar

**cat demo.rules**

Now, we can flush (or delete) all rules from the run-time configuration

**sudo nft flush ruleset**

Of course, if we ask for the rules now, we will receive an empty data set

Let’s load them back with

**sudo nft -f demo.rules**

If we ask for the list of rules, they should be there

To insert a rule after an existing one or delete a rule, we need the **handle**

It can be listed with the following command

**sudo nft --handle list ruleset**

Let’s add one dummy rule just before the **counter**

**sudo nft insert rule inet filter INPUT handle 6 tcp dport 9090 accept**

Should we want to insert a rule after a handle, we must use **add** instead of **insert**

Check the list again

**sudo nft --handle list ruleset**

Now, delete the desired handle

**sudo nft delete rule inet filter INPUT handle 8**

And check again

**sudo nft --handle list ruleset**