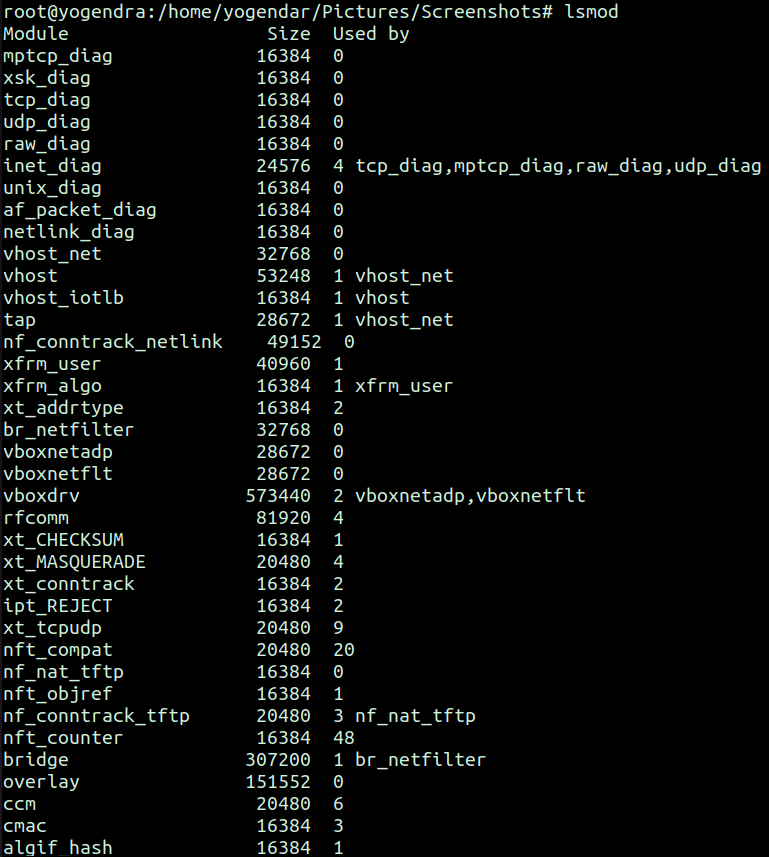
**Assignment**

**Question1: How to view modules in the kernel?**

**Answer: Link-** [**https://www.networkworld.com/article/3391362/looking-into-linux-modules.html**](https://www.networkworld.com/article/3391362/looking-into-linux-modules.html)

Kernel modules are chunks of code that are loaded and unloaded into the kernel as needed, thus extending the functionality of the kernel without requiring a reboot.

The easiest way to list modules is with the **lsmod** command.



In the output above:

* "Module" shows the name of each module
* "Size" shows the module size (not how much memory it is using)
* "Used by" shows each module's usage count and the referring modules

Clearly, that's a *lot* of modules. The number of modules loaded will depend on your system and distribution and what's running. We can count them like this:

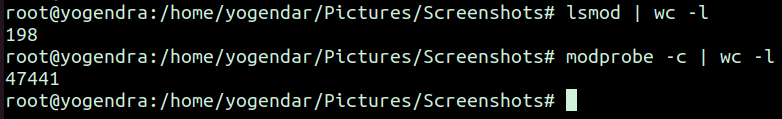
root@yogendra:/home/yogendar/Pictures/Screenshots# **lsmod | wc -l**

**198**

To see the number of modules available on the system (not just running):

root@yogendra:/home/yogendar/Pictures/Screenshots# **modprobe -c | wc -l**

**47441**



### **Other commands for examining modules**

Linux provides several commands for listing, loading and unloading, examining, and checking the status of modules.

* depmod -- generates modules.dep and map files
* insmod -- a simple program to insert a module into the Linux Kernel
* lsmod -- show the status of modules in the Linux Kernel
* modinfo -- show information about a Linux Kernel module
* modprobe -- add and remove modules from the Linux Kernel
* rmmod -- a simple program to remove a module from the Linux Kernel

======================================================================

**Question2: Add a new module in the Kernel.**

**Answer: Link-**

<https://medium.com/@bharathreddy_90228/writing-a-simple-hello-world-kernel-module-on-linux-ubuntu-137999d494f6>

**Create Directory:**

**root@yogendra:/home/yogendra# mkdir /usr/local/src/kernel\_example**

**root@yogendra:/home/yogendra# cd /usr/local/src/kernel\_example**

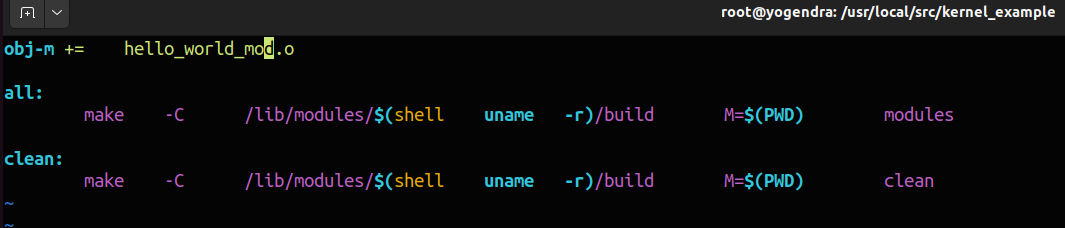
After creating the directory create file and insert the given below content:

root@yogendra:/usr/local/src/kernel\_example# **vim hello\_world\_mod.c**



Create another file and enter the following:

root@yogendra:/usr/local/src/kernel\_example# **vim Makefile**



After creating the file run make command:

**root@yogendra:/usr/local/src/kernel\_example# make**

make -C /lib/modules/5.15.0-53-generic/build M=/usr/local/src/kernel\_example modules

make[1]: Entering directory '/usr/src/linux-headers-5.15.0-53-generic'

warning: the compiler differs from the one used to build the kernel

The kernel was built by: gcc (Ubuntu 11.2.0-19ubuntu1) 11.2.0

You are using: gcc (Ubuntu 11.3.0-1ubuntu1~22.04) 11.3.0

CC [M] /usr/local/src/kernel\_example/hello\_world\_mod.o

MODPOST /usr/local/src/kernel\_example/Module.symvers

CC [M] /usr/local/src/kernel\_example/hello\_world\_mod.mod.o

LD [M] /usr/local/src/kernel\_example/hello\_world\_mod.ko

BTF [M] /usr/local/src/kernel\_example/hello\_world\_mod.ko

Skipping BTF generation for /usr/local/src/kernel\_example/hello\_world\_mod.ko due to unavailability of vmlinux

make[1]: Leaving directory '/usr/src/linux-headers-5.15.0-53-generic'

root@yogendra:/usr/local/src/kernel\_example# **insmod hello\_world\_mod.ko**

root@yogendra:/usr/local/src/kernel\_example# **lsmod | grep hello\_world\_mod**

**hello\_world\_mod 16384 0**

======================================================================

**Question3: How to specify parameters of the module?**

**Answer: Link-** [**https://www.kernel.org/doc/html/v4.12/admin-guide/kernel-parameters.html**](https://www.kernel.org/doc/html/v4.12/admin-guide/kernel-parameters.html)

Module parameters can be specified in two ways: via the kernel command line with a module name prefix, or via modprobe, e.g.:

(kernel command line) usbcore.blinkenlights=1

(modprobe command line) modprobe usbcore blinkenlights=1

Parameters for modules which are built into the kernel need to be specified on the kernel command line. modprobe looks through the kernel command line (/proc/cmdline) and collects module parameters when it loads a module, so the kernel command line can be used for loadable modules too.

Hyphens (dashes) and underscores are equivalent in parameter names, so:

log\_buf\_len=1M print-fatal-signals=1

can also be entered as:

log-buf-len=1M print\_fatal\_signals=1

Double-quotes can be used to protect spaces in values, e.g.:

param="spaces in here"

======================================================================

**Question4: What is PAM? How does it work?**

**Answer: Link-** [**https://developer.ibm.com/tutorials/l-pam/**](https://developer.ibm.com/tutorials/l-pam/)

[**https://www.redhat.com/sysadmin/pluggable-authentication-modules-pam**](https://www.redhat.com/sysadmin/pluggable-authentication-modules-pam)

PAM is an API that takes care of authenticating a user to a service. Before PAM, applications like login (and rlogin, telnet, rsh) looked for the username in /etc/passwd, then compared the two and authenticated the user-typed name. All applications used these shared services, although the implementation details and authority to configure them was not shared.

Next, application developers tried coding their own processes. With this came the need to separate the application and security module (a common security module can be shared by applications and can be configured as needed).

The PAM mechanism integrates multiple low-level authentication schemes into a high-level API that allows programs that rely on authentication to be written independently of the underlying authentication scheme. The principal feature of PAM is the dynamic configuration of authentication through either an /etc/pam.d or /etc/pam.conf file.

PAM can be configured to deny certain programs the right to authenticate users and to warn when certain programs attempt to authenticate. PAM programs make use of PAM modules (authentication modules): They are attached to applications at runtime in order to work.

##### Figure: PAM library parses the config file and loads modules to it

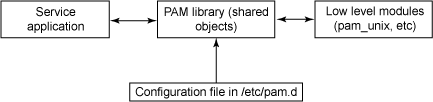


Figure: shows the basic flow of the PAM model.

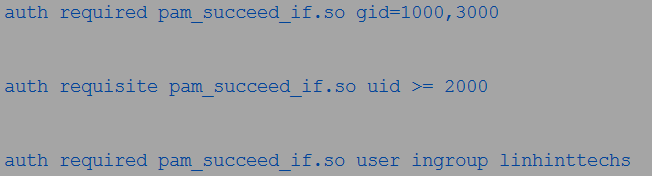
**Question5: What are the modules of PAM?**

**Answer: Link-** [**https://linuxhint.com/basic-linux-pam-modules/**](https://linuxhint.com/basic-linux-pam-modules/)

The following PAM inbuilt modules exist in our systems, and we should be conversant with each of them for the proper use of Linux PAM:

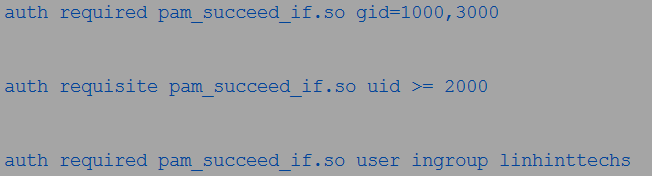
**1. pam\_succeed\_if module**

This module controls the access to users and groups. For example, you can validate the user accounts using this command:



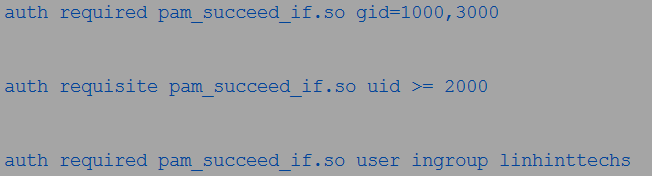
The previous example signifies that only the users whose IDs are 1000 or 3000 can log in.

Another example is as in the following command:



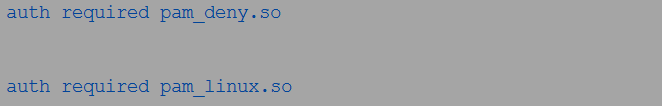
The previous example specifies that only the users with the user IDs equal to or greater than 2000 can access the service or program.

An example of using an ingroup parameter is as seen in the following:



**2. pam\_deny module**

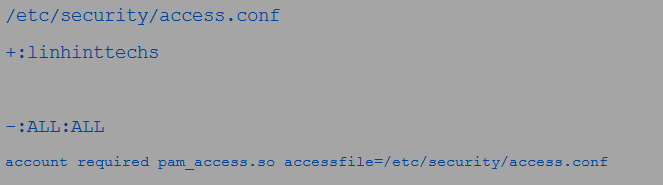
The pam\_deny module is commonly used for denying or restricting access. When used, the module will return a non-OK result upon processing. Using this module at the end of your module stack protects any possible misconfiguration. However, using it at the beginning of a module stack will disable your service, as seen in the following figure:



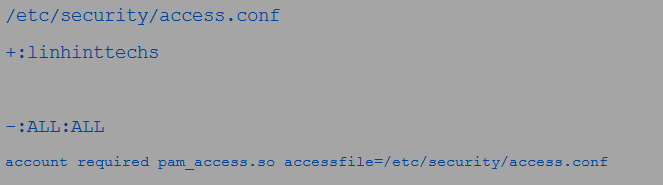
Interestingly, you can use this module with the account, auth, password, and session management groups.

**3. pam\_access module**

The pam\_access module is another module that you can use with all the management groups. It functions the same way as the pam\_succeed\_if module. However, the pam\_succeed\_if module does not check the login details from the networked hosts, while the pam\_access module focuses on that.



You can then type the access rules as seen in the following figures:



And



The rules state that only the users within linhinttecks can log in. The + and – signs in the rule allow and deny, respectively. This module is also useable with all the management groups.

**4. pam\_nologin module**

This module is selective and only allows the root to log in should the file exist. Unlike in the previous modules, which you can use with all the management groups, this module is only useable with auth and account management groups.





**5. pam\_cracklib module**

Cybercrime is on the rise, and strong passwords are mandatory. This module sets the rules for how strong your passwords can get. In the following example, the module provides you up to 4 chances to pick a strong password failure to which it will exit. Again, the module provides that you can only pick a password of 12 or more characters.



**6. pam\_localuser module**

This module is often used to check if a user is in the /etc/passwd. You can use this module with all the management groups including auth, password, session, and account.



**7. pam\_rootok module**

Only the root users can run this service since it checks if the UID is 0. Thus, this module comes in handy when a service is dedicated to the root users only. It is useable with no other management group except the auth management group.



**8. pam\_mysql module**

You can use the pam\_mysql module to validate the users rather than checking their credentials against the /etc/shadow. It is useable to validate the users with the pam\_mysql parameters. You can install it using the following command if you do not have it in your system. This is another module that you can use with all the management groups:



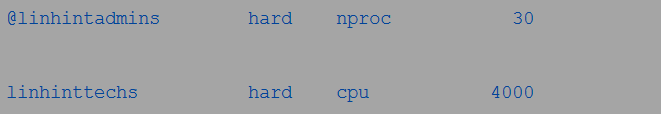
**9. pam\_limits module**

If you need to set the limits on your system resources, the pam\_limits module is what you need. This module affects everyone, including the root users using the limits configuration file available in the /etc/security/limits.d/ directory. It is beneficial in protecting the system resources and is only useable in the session management group.



The limits set in the /etc/security/limits.conf file can either be hard or soft. Only the root users can change the limit value in hard limits, while the ordinary users cannot. On the other hand, even ordinary users can also change the limit value.

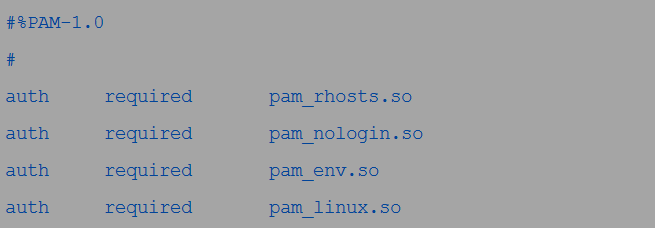
Again, limits can be classified as cpu, fsize, data, nproc, and many more. A good example is shown in the following figure:



The first limit for the linhintadmins members sets the number of processes for each member at 30. On the other hand, the second limit is for the linhintechs members and sets the CPU duration for them at 4000 minutes.

**10. pam\_rhosts module**

It performs the standard network authentication for services and programs often traditionally implemented in rsh and rlogin, among others. The three available options include debug, superuser, and silent. It is only useable with the auth management group and features in the following example:



======================================================================

**Question6: What is the sudo module?**

**Answer: Link-** [**https://www.oreilly.com/library/view/learning-puppet-security/9781784397753/ch06s04.html**](https://www.oreilly.com/library/view/learning-puppet-security/9781784397753/ch06s04.html)

The sudo module manages all aspects of your sudoers configuration. The module has options to leave the system configuration alone, as well as not purging unmanaged sudoers entries. The recommended path is to manage all the sudoer resources; however, the options are there if needed.

To install the sudoers module, we'll issue the following command:

s**udo puppet module install saz-sudo**

======================================================================

**Question7: How does the sudo module work when we provide a user sudo access?**

**Answer: Link-** [**https://unix.stackexchange.com/questions/126914/how-does-sudo-really-work**](https://unix.stackexchange.com/questions/126914/how-does-sudo-really-work)

sudo is a so called "SetUID binary", as we can see in the output of ls -l:

**$ ls -l /usr/bin/sudo**

**-rwsr-xr-x 1 root root 159016 Mar 21 20:40 /usr/bin/sudo**

The s in the fourth column (where you'd normally find an x on executable files) tells you that the SetUID bit is set. This bit has one significant meaning: When a binary with the SetUID bit set is executed, it does not run with the user ID of the invoking user, but the user ID of the binary's owner (in this case root).

And that's the clue. sudo is always run with superuser privileges (as root). Thus sudo has the ability to do some privileged tasks like calling system functions only allowed for root. One of those system calls (the essential one) are setuid(2) and friends. By calling setuid() a process can *change* its UID to any UID it wants (thus impersonating another user).

What sudo does is:

* read and parse /etc/sudoers, look up the invoking user and its permissions,
* ask the invoking user for a password (this is usually the user's password, but can also be the target user's password or skipped as with NOPASSWD)
* create a child process in which it calls setuid() to change to the target user
* execute a shell or the command given as arguments in this child

======================================================================

**Question8: Code of ls command?**

**Answer: Link-** [**https://github.com/wertarbyte/coreutils/blob/master/src/ls.c**](https://github.com/wertarbyte/coreutils/blob/master/src/ls.c)

ls is part of coreutils. All these basic commands are part of the coreutils package.

Only 4984 code lines for a command **ls** is Written by Richard Stallman and David MacKenzie. We can see the code in the above given link.

======================================================================

**Thank You**