Test SSHd config on a different SSH port

Sometimes you need to tweak your SSH daemon on an important system and you just don't know if particular settings will break connectivity to the server or not. In such cases it's best to test new SSHd config using separate SSH daemon instance and separate SSH port – debug it there and only then apply new configs into your primary SSHd configuration.

Creating New SSHd Config

The easiest is to start by copying /etc/ssh/sshd_config file – you will need sudo/root privileges for that:

greys@s2:~ $ **sudo cp /etc/ssh/sshd_config /home/greys**

I then just remove everything I don't need from it, leaving bare minimum. These are the parameters I kept (I ended up renaming my config to /home/greys/sshd_config.minimal after edits)

greys@s2:~ $ **grep -v ^# /home/greys/sshd_config.minimal | uniq -u**
Port 2222

```
HostKey /etc/ssh/ssh_host_rsa_key

RSAAuthentication yes
PubkeyAuthentication yes

AuthorizedKeysFile /var/ssh/%u/authorized_keys

PasswordAuthentication no

UsePAM yes
```

I only updated the SSH Port parameter – you can pick any other number instead of 2222.

Starting SSH daemon with custom config file

There's a few rules for testing SSH configuration using separate file:

- you need to have sudo/root privilege (mostly to avoid mess with host SSH keys)
- it's better to increase verbosity level to see what's going on
- it's best to run SSHd in foreground (non-daemon) mode

With these principles in mind, here's the command line to test the config shown above:

```
greys@s2:~ $ sudo /usr/sbin/sshd -f /home/greys/sshd_config.minimal -ddd -D
debug2: load_server_config: filename /home/greys/sshd_config.minimal
debug2: load_server_config: done config len = 194
debug2: parse_server_config: config /home/greys/sshd_config.minimal len 194
debug3: /home/greys/sshd_config.minimal:1 setting Port 2222
debug3: /home/greys/sshd_config.minimal:10 setting HostKey /home/greys/ssh_host_rsa_key
```

```
debug3: /home/greys/sshd_config.minimal:12 setting RSAAuthentication yes
/home/greys/sshd_config.minimal line 12: Deprecated option RSAAuthentication
debug3: /home/greys/sshd_config.minimal:13 setting PubkeyAuthentication yes
debug3: /home/greys/sshd_config.minimal:18 setting AuthorizedKeysFile /var/ssh/%u/authorized_keys
debug3: /home/greys/sshd_config.minimal:20 setting PasswordAuthentication no
debug3: /home/greys/sshd_config.minimal:22 setting UsePAM yes
debug1: sshd version OpenSSH_7.4, OpenSSL 1.0.2k-fips 26 Jan 2017
debug1: private host key #0: ssh-rsa SHA256:g7xhev6zJefXRFc0ClAG4rzpFI1Ts8H7PhQ/h3PTmLM
debug1: rexec_argv[0]='/usr/sbin/sshd'
debug1: rexec_argv[1]='-f'
debug1: rexec_argv[2]='/home/greys/sshd_config.minimal'
debug1: rexec_argv[3]='-ddd'
debug1: rexec_argv[4]='-D'
debug3: oom_adjust_setup
debug1: Set /proc/self/oom_score_adj from 0 to -1000
debug2: fd 3 setting O_NONBLOCK
debug1: Bind to port 2222 on 0.0.0.0.
Server listening on 0.0.0.0 port 2222.
debug2: fd 4 setting O_NONBLOCK
debug3: sock_set_v6only: set socket 4 IPV6_V6ONLY
debug1: Bind to port 2222 on ::.
Server listening on :: port 2222.
```

That's it, the configuration is ready to be tested (assuming your firewall on server doesn't block port 2222).

Testing SSH connectivity using

Different SSH Port

Here's my login session in a separate window, connecting from my MacBook Pro to the s2 server on SSH port 2222 (I have masked my static IP with aaa.bbb.ccc.ddd and my s2 server's IP with eee.fff.ggg.hhh):

```
greys@MacBook-Pro:~ $ ssh s2 -p 2222
Warning: untrusted X11 forwarding setup failed: xauth key data
not generated
Last login: Fri May 24 15:53:59 2019 from aaa.bbb.ccc.ddd
debug3: Copy environment: XDG_SESSION_ID=14813
debug3: Copy environment: XDG_RUNTIME_DIR=/run/user/1000
Environment:
USER=greys
LOGNAME=greys
HOME=/home/greys
PATH=/usr/local/bin:/usr/bin
MAIL=/var/mail/greys
SHELL=/bin/bash
SSH_CLIENT=aaa.bbb.ccc.ddd 64168 2222
SSH_CONNECTION=aaa.bbb.ccc.ddd 64168 eee.fff.ggg.hhh 2222
SSH_TTY=/dev/pts/14
TERM=xterm-256color
XDG_SESSION_ID=14813
XDG_RUNTIME_DIR=/run/user/1000
SSH_AUTH_SOCK=/tmp/ssh-ajOUyvbR6i/agent.20996
greys@s2:~ $ uptime
16:18:08 up 86 days, 17:32, 2 users, load average: 1.00, 1.02,
1.05
```

Pretty cool, huh?

See Also

- Basic SSH configuration
- SSH port
- SSH port forwarding
- Getting started with Ansible
- How To Check SSH port status

Show List of Available SELinux Users

```
[greys@rhel8 ~]$ seinfo -u

Users: 8
   guest_u
   root
   staff_u
   sysadm_u
   system_u
   unconfined_u
   user_u
   xguest_u
```

I'm slowly improving my understanding of the **SELinux setup**, currently looking into controlling user access. As you know, there may be lots of different users created in your Linux system. For them to be controlled by the SELinux framework, we need to map all users to one of the users in SELinux policy.

Install SELinux Tools

The command we need is called **seinfo**, and it's not installed by default. We have to install the **setools-console** package first:

```
[greys@rhel8 ~]$ sudo yum install setools-console
[sudo] password for greys:
Updating Subscription Management repositories.
Updating Subscription Management repositories.
Red Hat Enterprise Linux 8 for x86_64 - AppStream Beta (RPMs)
```

```
3.0 kB/s | 4.1 kB 00:01
Red Hat Enterprise Linux 8 for x86_64 - BaseOS Beta (RPMs) 3.0
kB/s | 4.1 kB 00:01
Dependencies resolved.
================================================================
===========
Package Arch Version Repository Size
================================================================
===========

Installing:
setools-console x86_64 4.1.1-11.el8 rhel-8-for-x86_64-baseos-
beta-rpms 28 k

Transaction Summary
================================================================
===========

Install 1 Package

Total download size: 28 k
Installed size: 109 k
Is this ok [y/N]: y
Downloading Packages:
setools-console-4.1.1-11.el8.x86_64.rpm 15 kB/s | 28 kB 00:01
----------------------------------------------------------------
----------------------------------------------------------------
---
Total 15 kB/s | 28 kB 00:01
Running transaction check
Transaction check succeeded.
Running transaction test
Transaction test succeeded.
Running transaction
Preparing : 1/1
Installed: setools-console-4.1.1-11.el8.x86_64
Installing : setools-console-4.1.1-11.el8.x86_64 1/1
Installed: setools-console-4.1.1-11.el8.x86_64
Running scriptlet: setools-console-4.1.1-11.el8.x86_64 1/1
Verifying : setools-console-4.1.1-11.el8.x86_64 1/1
```

```
Installed:
setools-console-4.1.1-11.el8.x86_64

Complete!
```

List Available SELinux Users

Now that the package is installed, run the seinfo -u command to show list of SELinux users:

```
[greys@rhel8 ~]$ seinfo -u

Users: 8
guest_u
root
staff_u
sysadm_u
system_u
unconfined_u
user_u
xguest_u
```

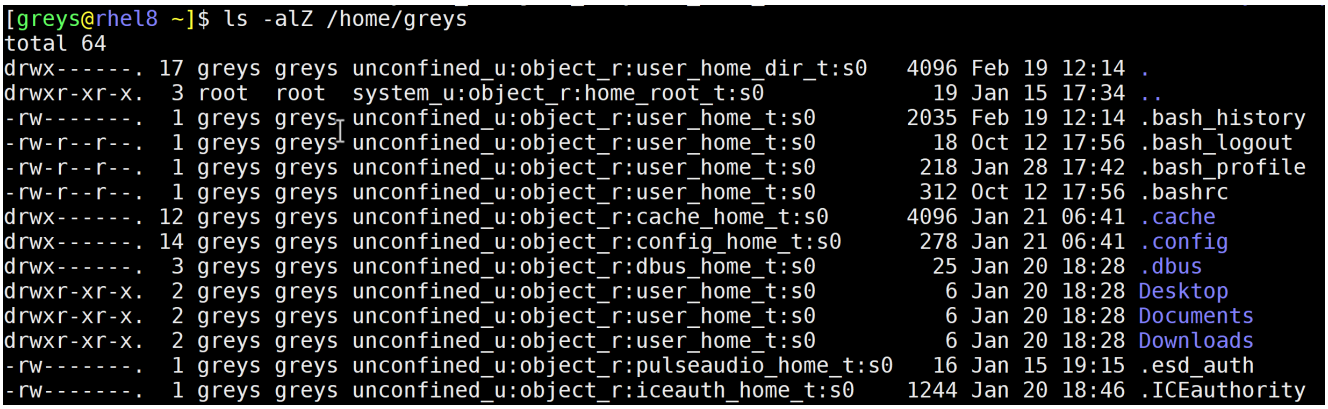
While we're at it, let's check the current user's SELinux context: usually you're mapped to the unconfined_u user:

```
[greys@rhel8 ~]$ id -Z
unconfined_u:unconfined_r:unconfined_t:s0-s0:c0.c1023
```

See Also

- List files with SELinux contexts
- SELinux status
- Advanced sestatus
- Enable SELinux
- Disable SELinux
- Linux Commands

How To: List Files with SELinux Contexts



When running a **SELinux** based setup, it might be useful to know how to quickly inspect files and directories to confirm their current SELinux context.

What is SELinux Context?

Every process and file in SELinux based environment can be labeled with additional information that helps fulfill RBAC (Role-Based Access Control), TE (Type Enforcement) and MLS (Multi-Level Security).

SELinux context is the combination of such additional information:

- user
- role
- type
- level

In the following example we can see that **unconfined_u** is the SELinux user, **object_r** is the role, **user_home_dir_t** is the

object type (home user directory) and the SELinux sensitivity (MCS terminology) level is **s0**:

```
drwx------.           17           greys           greys
unconfined_u:object_r:user_home_dir_t:s0 4096 Feb 19 12:14 .
```

Use ls -Z to show SELinux Context

Using ls command with -Z option will show the SELinux contexts. This command line option is totally made to be combined with other **ls command** options:

```
[greys@rhel8 ~]$ ls -alZ .
total 64
drwx------.           17           greys           greys
unconfined_u:object_r:user_home_dir_t:s0 4096 Feb 19 12:14 .
drwxr-xr-x. 3 root root system_u:object_r:home_root_t:s0 19
Jan 15 17:34 ..
-rw-------. 1 greys greys unconfined_u:object_r:user_home_t:s0
2035 Feb 19 12:14 .bash_history
-rw-r--r--. 1 greys greys unconfined_u:object_r:user_home_t:s0
18 Oct 12 17:56 .bash_logout
-rw-r--r--. 1 greys greys unconfined_u:object_r:user_home_t:s0
218 Jan 28 17:42 .bash_profile
-rw-r--r--. 1 greys greys unconfined_u:object_r:user_home_t:s0
312 Oct 12 17:56 .bashrc
drwx------.           12           greys           greys
unconfined_u:object_r:cache_home_t:s0 4096 Jan 21 06:41 .cache
drwx------.           14           greys           greys
unconfined_u:object_r:config_home_t:s0  278  Jan  21  06:41
.config
drwx------. 3 greys greys unconfined_u:object_r:dbus_home_t:s0
25 Jan 20 18:28 .dbus
drwxr-xr-x. 2 greys greys unconfined_u:object_r:user_home_t:s0
6 Jan 20 18:28 Desktop
drwxr-xr-x. 2 greys greys unconfined_u:object_r:user_home_t:s0
6 Jan 20 18:28 Documents
drwxr-xr-x. 2 greys greys unconfined_u:object_r:user_home_t:s0
6 Jan 20 18:28 Downloads
-rw-------.           1           greys           greys
unconfined_u:object_r:pulseaudio_home_t:s0  16  Jan  15  19:15
```

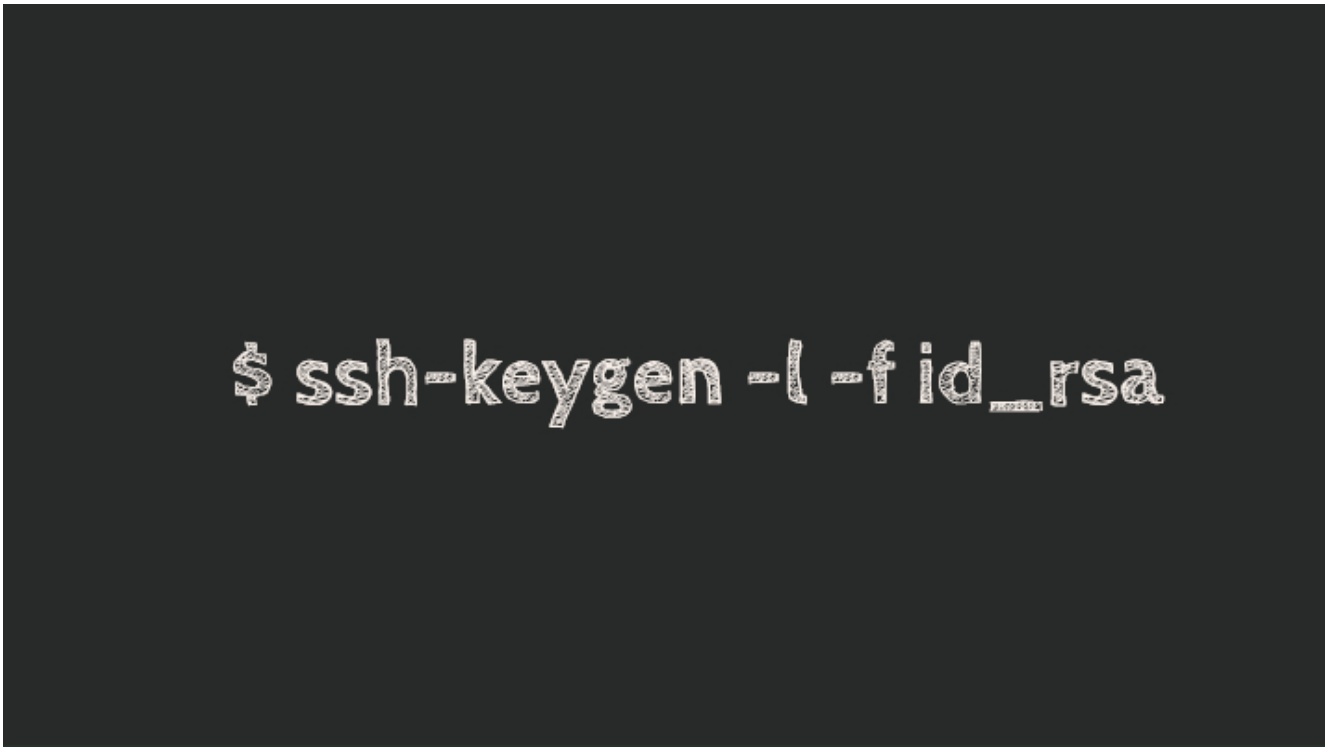
```
.esd_auth
-rw-------.           1           greys           greys
unconfined_u:object_r:iceauth_home_t:s0  1244  Jan  20  18:46
.ICEauthority
-rw-------. 1 greys greys unconfined_u:object_r:user_home_t:s0
3434 Jan 22 18:06 id_rsa_4k
-rw-r--r--. 1 greys greys unconfined_u:object_r:user_home_t:s0
737 Jan 22 18:06 id_rsa_4k.pub
-rw-rw-r--. 1 greys greys unconfined_u:object_r:user_home_t:s0
21 Jan 28 17:53 infile2.txt
-rw-------. 1 greys greys unconfined_u:object_r:user_home_t:s0
38 Jan 22 18:05 .lesshst
drwxr-xr-x.           3           greys           greys
unconfined_u:object_r:gconf_home_t:s0 19 Jan 20 18:28 .local
drwxr-xr-x.           2           greys           greys
unconfined_u:object_r:audio_home_t:s0 6 Jan 20 18:28 Music
-rw-rw-r--. 1 greys greys unconfined_u:object_r:user_home_t:s0
0 Jan 22 18:01 newkey
drwxr-xr-x. 2 greys greys unconfined_u:object_r:user_home_t:s0
6 Jan 20 18:28 Pictures
drwxrw----. 3 greys greys unconfined_u:object_r:home_cert_t:s0
19 Jan 20 18:28 .pki
drwxr-xr-x. 2 greys greys unconfined_u:object_r:user_home_t:s0
6 Jan 20 18:28 Public
drwxrwxr-x. 4 greys greys unconfined_u:object_r:user_home_t:s0
165 Jan 16 11:00 screenFetch
-rw-------.           1           greys           greys
unconfined_u:object_r:xauth_home_t:s0  150  Jan  20  18:44
.serverauth.1859
-rw-------.           1           greys           greys
unconfined_u:object_r:xauth_home_t:s0  50  Jan  20  18:39
.serverauth.1893
drwx------. 2 greys greys unconfined_u:object_r:ssh_home_t:s0
70 Jan 22 18:07 .ssh
-rw-rw-r--. 1 greys greys unconfined_u:object_r:user_home_t:s0
0 Jan 21 07:49 system_u:object_r:shell_exec_t:s0
drwxr-xr-x. 2 greys greys unconfined_u:object_r:user_home_t:s0
6 Jan 20 18:28 Templates
drwxr-xr-x. 2 greys greys unconfined_u:object_r:user_home_t:s0
6 Jan 20 18:28 Videos
-rw-------. 1 greys greys unconfined_u:object_r:user_home_t:s0
```

```
2874 Jan 29 04:40 .viminfo
-rw-------.           1           greys           greys
unconfined_u:object_r:xauth_home_t:s0  260  Feb  19  12:14
.Xauthority
```

See Also

- Enable SELinux
- Disable SELinux
- Confirm SELinux Status
- Get SELinux mode with getenforce
- Advanced sestatus Usage
- Linux Commands
- **SELinux Reference**

How To Inspect SSH key fingerprints



As you can imagine, SSH keypairs – combinations of private and public keys – are vital elements of your digital identity as a sysadmin or a developer. And since they can be used for accessing source code repositories and for deploying changes to production environments, you usually have more than one SSH key. That's why it's important to know how to inspect SSH key fingerprints.

SSH Key Fingerprints

Key fingerprints are special checksums generated based on the public SSH key. Run against the same key, **ssh-keygen command** will always generate the same fingerprint.

Because of this property, you can use SSH key fingerprints for three things:

1. Identify SSH key – fingerprint will stay the same even if you rename the file
2. Confirm integrity of the SSH key – if you get the same fingerprint from your private SSH key, you can be sure it's still valid and intact

3. Validate identity of the SSH key – same fingerprint means you're dealing with the same key (that you or your solution trusted for specific functionality)

How to Check SSH Fingerprint of a Key

ssh-keygen command takes the identity (SSH key) filename and calculates the fingerprint.

You can start by changing directory into .ssh and checking if you have any SSH keys there already. If not, you should generate a new SSH key.

```
greys@server:~$ cd .ssh
greys@server:~/.ssh$ ls -la
total 24
drwx------  3 greys greys 4096 Feb 17 21:11 .
drwxr-xr-x 15 greys greys 4096 Feb 17 21:13 ..
-rw-------  1 greys greys 1766 Feb 17 21:11 id_rsa
-rw-r--r--  1 greys greys  394 Feb 17 21:11 id_rsa.pub
```

Let's run **ssh-keygen** to confirm the fingerprint of the id_rsa keypair:

```
greys@server:~/.ssh$ ssh-keygen -l -f id_rsa
2048  SHA256:z96jtEGIqfLoaq1INIBFI/3K2M+f9xZUyupsm3itgvI  no comment (RSA)
```

Check Fingerprint of the Private SSH Key

By default this command looks for the public key portion (id_rsa.pub file), so it's not a very good test of integrity or identity of the private key. There is a very real possibility that you have one private key and a separate public key, that are not related to each other.

That’s why for checking the private key you must take it a step further and copy private key (id_rsa) into some other directory where you can use ssh-keygen again:

```
greys@server:~/.ssh$ cp id_rsa ..
greys@server:~/.ssh$ cd ..
```

this time, because there’s no public key file found nearby, the ssh-keygen command will have to open private key. And if it’s passphrase protected (as it always should be), you’ll be asked for the SSH key passphrase:

```
greys@server:~$ ssh-keygen -l -f id_rsa
Enter PEM pass phrase:
2048 SHA256:z96jtEGIqfLoaq1INIBFI/3K2M+f9xZUyupsm3itgvI no comment (RSA)
```

Old-school SSH fingerprints

If you’ve been using Linux/Unix for more than a couple of years, you probably noticed that ssh-keygen now shows you a different looking fingerprints: they used to be these semicolon-delimited sequences like this:

```
06:6e:bc:f4:4e:03:90:b7:ba:99:8d:a5:71:1e:dc:22
```

… instead they now are shown as this:

```
z96jtEGIqfLoaq1INIBFI/3K2M+f9xZUyupsm3itgvI
```

The reason for this is that by default fingerprints are shown as SHA256 sequences, while in the past they were MD5.

In order to show the SSH fingerprint in MD5 format, just specify this in the command line:

```
greys@server:~$ ssh-keygen -l -E md5 -f id_rsa
Enter PEM pass phrase:
2048 MD5:06:6e:bc:f4:4e:03:90:b7:ba:99:8d:a5:71:1e:dc:22 no comment (RSA)
```

See Also

- How To: Generate SSH key
- How To: Change SSH key passphrase
- SSH port
- SSH port forwarding
- SSH
- Important SSH server configuration options

Disable portmapper in CentOS 7

If you don’t have any other network services running on your Linux system, you probably don’t need portmapper running. Here are the steps to check and to disable portmap.

What portmapper does

Portm apper is a special Unix/Linux service that runs on networked systems that provide RPC (Remote Procedure Call) based services, like NFS.

Port mapper service is called portmapper and always runs on TCP and UDP ports 111.

IMPORTANT: back in 2015 portmapper was confirmed as vulnerable for Distributed Denial of Service attacks (DDoS) – so it’s considered a good practice to disable it or at least protect using firewall.

List RPC services

You can use rpcinfo command to list currently active RPC services on your system.

In my example below there's nothing else running RPC, just the portmapper itself:

```
root@s5:~ # rpcinfo -p
program vers proto port service
100000 4 tcp 111 portmapper
100000 3 tcp 111 portmapper
100000 2 tcp 111 portmapper
100000 4 udp 111 portmapper
100000 3 udp 111 portmapper
100000 2 udp 111 portmapper
```

Stop portmapper in CentOS 7

Somewhat confusing, the service providing portmapper functionality is always called **rpcbind**.

First, let's stop the portmapper service:

```
root@s5:~ # systemctl stop rpcbind
Warning: Stopping rpcbind.service, but it can still be activated by:
rpcbind.socket
root@s5:~ # systemctl stop rpcbind.socket
```

Prevent portmapper from restarting upon reboot

Now, let's make sure the service is also disabled:

```
root@s5:~ # systemctl disable rpcbind
Removed symlink /etc/systemd/system/multi-user.target.wants/rpcbind.service.
```

And just to confirm it's all done correctly, let's run rpcinfo

again, it will return an error now:

```
root@s5:~ # rpcinfo -p
rpcinfo: can't contact portmapper: RPC: Remote system error -
Connection refused
```

See Also

- Unix commands
- Advanced Unix commands
- Unix Tutorial

How To Enable SElinux

SElinux – Security Enhanced Linux

If you're using RedHat or CentOS Linux distros (or sporting a Fedora Linux desktop), you probably have **SELinux** enabled by default. But if it's been disabled for some reason and you want it back – here's how you can enable **SELinux** in your Linux system.

Confirm current SELinux mode

Run the getenforce command to confirm that SELinux is actually disabled:

```
[root@rhel8 ~]# getenforce
Disabled
```

Check SElinux status with sestatus

sestatus normally shows verbose SElinux status information, but if SELinux is disabled, you'll only get one line of output, like this:

```
root@rhel8 ~]# sestatus
SELinux status: disabled
[root@rhel8 ~]#
```

If **sestatus** shows that SELinux is disabled, you'll need to enable it via /etc/selinux.png/config file and reboot the server as shown below.

Permanently Enable SELinux

Do the following two steps to enable SELinux:

1. Update **/etc/selinux.png/config** file (change **SELINUX=disabled** to **SELINUX=enforcing**)
2. Reboot your Linux system (**shutdown -r now**)

Once your server comes back online, run **sestatus** again to make

sure **SElinux is enabled** now:

```
[root@rhel8 ~]# sestatus
SELinux status: enabled
SELinuxfs mount: /sys/fs/selinux.png
SELinux root directory: /etc/selinux.png
Loaded policy name: targeted
Current mode: enforcing
Mode from config file: enforcing
Policy MLS status: enabled
Policy deny_unknown status: allowed
Memory protection checking: actual (secure)
Max kernel policy version: 31
```

See Also

- SELinux Status
- How To Disable SELinux
- Advanced Unix Commands
- Linux Commands
- SELinux Reference

SElinux: Advanced sestatus usage

I learned something new today! Apparently, **sestatus command** can report security contexts of the key system files – really neat for quickly recognising possible security compromise.

Files and processes in

/etc/sestatus.conf

The way this works is you must use the **/etc/sestatus.conf** file which contains list of files and list of processes that are checked for **SElinux** contexts. These are the most common security attack vectors, so SElinux notes them and helps you to quickly confirm their contexts using **sestatus -v** command.

VERY IMPORTANT: at this stage **sestatus command** does NOT highlight or warn you about any non-standard contextual changes. So the only thing is does is show you all the important files you selected and report their current contexts – if some of these have been changed, the task of recognising or fixing this is still on you.

You can add any files and process you like here, but here's the default list in RHEL8:

```
[greys@rhel8 ~]$ cat /etc/sestatus.conf
[files]
/etc/passwd
/etc/shadow
/bin/bash
/bin/login
/bin/sh
/sbin/agetty
/sbin/init
/sbin/mingetty
/usr/sbin/sshd
/lib/libc.so.6
/lib/ld-linux.so.2
/lib/ld.so.1

[process]
/sbin/mingetty
/sbin/agetty
/usr/sbin/sshd
```

Files and processes contexts with sestatus

```
[greys@rhel8 ~]$ sestatus -v
SELinux status: enabled
SELinuxfs mount: /sys/fs/selinux.png
SELinux root directory: /etc/selinux.png
Loaded policy name: targeted
Current mode: enforcing
Mode from config file: enforcing
Policy MLS status: enabled
Policy deny_unknown status: allowed
Memory protection checking: actual (secure)
Max kernel policy version: 31

Process contexts:
Current context: unconfined_u:unconfined_r:unconfined_t:s0-
s0:c0.c1023
Init context: system_u:system_r:init_t:s0

File contexts:
Controlling terminal: unconfined_u:object_r:user_devpts_t:s0
/etc/passwd system_u:object_r:passwd_file_t:s0
/etc/shadow system_u:object_r:shadow_t:s0
/bin/bash system_u:object_r:shell_exec_t:s0
/bin/login system_u:object_r:login_exec_t:s0
/bin/sh         system_u:object_r:bin_t:s0         ->
system_u:object_r:shell_exec_t:s0
/sbin/agetty system_u:object_r:getty_exec_t:s0
/sbin/init       system_u:object_r:bin_t:s0       ->
system_u:object_r:init_exec_t:s0
/usr/sbin/sshd system_u:object_r:sshd_exec_t:s0
```

See Also

- Disable SElinux
- Check SElinux Status
- Security commands in Unix

How To: Generate SSH Key

SSH is such an integral part of everyday Linux/Unix life now, that it makes sense to use it for as many remote access and automation tasks as you can. As you probably know, you shouldn't be using password SSH authentication unless you have a pretty good reason to do so. By default, always use SSH keys. Today I'll show you how to generate SSH keys.

Generate SSH key with ssh-keygen

ssh-keygen is a standard utility supplied with SSH package. If you have ssh command on your system, you probably have the ssh-keygen command as well.

Without any command line options, ssh-keygen will ask you a few questions and create the key with default settings:

```
[greys@rhel8 ~]$ ssh-keygen
Generating public/private rsa key pair.
Enter file in which to save the key (/home/greys/.ssh/id_rsa):
Created directory '/home/greys/.ssh'.
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/greys/.ssh/id_rsa.
Your public key has been saved in /home/greys/.ssh/id_rsa.pub.
The key fingerprint is:
SHA256:Seu7UBogeX+g9+iv01CDJqiXAby740JKZGrZtu1T3oQ greys@rhel8
The key's randomart image is:
+---[RSA 2048]----+
|. |
|.. . |
| .+.o ... |
| +oo.+oooo |
|+.+o.o+.S. |
|o*oo ..E . |
```

```
|=.o o *o= |
|oo . +.o.o |
|o.. ..+++. |
+----[SHA256]-----+
[greys@rhel8 ~]$
```

Specify SSH key size for ssh-keygen

Most likely you'll have your preferences for SSH keys and it is much easier to just specify them when running the ssh-keygen command.

This is how one can generate 4096-bit key, for example:

```
[greys@rhel8 ~]$ ssh-keygen -b 4096
Generating public/private rsa key pair.
Enter file in which to save the key (/home/greys/.ssh/id_rsa):
/home/greys/.ssh/rsa-4k
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/greys/.ssh/rsa-4k.
Your public key has been saved in /home/greys/.ssh/rsa-4k.pub.
The key fingerprint is:
SHA256:4rf1AGIc99L57/xC1PWu7pJpwhkn5YCmZQqua/XdmGA greys@rhel8
The key's randomart image is:
+---[RSA 4096]----+
| |
| .|
| . .. .o|
| .. o=o... . o|
| . .=*S ++ . . |
| ooEo oo.o . .|
| o o.o.=o=.+ . |
| o ..+o=o=oo |
| ... . o.=*o.|
+----[SHA256]-----+
```

See Also

- SSH

- SSH port
- SSH command
- SSH port forwarding
- Change SSH key passphrase

SElinux Status

SELinux

This post shows you how to confirm current **SElinux status** before you decide to disable SELinux.

SElinux Enforcing vs Permissive

The most burning question usually is: does my RedHat/CentOS Linux enforce SELinux (and prevent some of my applications from running out of the box) or is it in the permissive state (which means it logs security concerns but doesn't block anything from running).

Answering this is very easy with the help of the **getenforce command**:

```
[greys@rhel8 ~]$ getenforce
Enforcing
```

SElinux status with sestatus

If you're more curious about the way SELinux is configured, then **sestatus command** will be much more useful:

```
[greys@rhel8 ~]$ sestatus
SELinux status: enabled
SELinuxfs mount: /sys/fs/selinux.png
SELinux root directory: /etc/selinux.png
Loaded policy name: targeted
Current mode: enforcing
Mode from config file: enforcing
Policy MLS status: enabled
Policy deny_unknown status: allowed
Memory protection checking: actual (secure)
Max kernel policy version: 31
```

How to read the sestatus output

Although the output of **sestatus** is fairly standard, you'll appreciate how useful it is once you start making changes to your SELinux policies.

- **Loaded policy name** is useful because you can make SELinux load a strict policy as well, and it's important to understand which one is currently in use.
- **Current mode**: will confirm if SELinux is running in enforcing or permissive mode.
- **Policy MLS status**: must research more! I know MLS is Multi Level Security, but need to understand why it's separate option here.
- **Memory protection checking** – must come back to this as I'm not finding enough information. This is a flag confirming that SELinux still protects certain memory access syscalls in your Linux.

See Also

- How To: Disable SELinux
- How To: Enable SELinux
- **SELinux Reference**

How To: Disable SElinux

SElinux – Security Enhanced Linux

If you're using RedHat or CentOS Linux distros (or sporting a Fedora Linux desktop), you probably have SELinux enabled by default. SELinux is a Security-Enhanced Linux – a framework for securely managing processes, users and files on your

RedHat OS.

Confirm current SElinux mode

Just run the getenforce command to see what the story is. Most likely it will say “Enforcing” which is really good – means your OS is under solid protection:

```
[root@rhel8 ~]# getenforce
Enforcing
```

Temporarily Disable SELinux

If you need to disable SELinux just for a few minutes to debug some issue (mind you, there are better ways to debug than disabling SELinux!), you should use the setenforce command:

```
[root@rhel8 ~]# setenforce 0
```

As you can see, getenfore will now report that your system is running in a Permissive mode – not very safe:

```
[root@rhel8 ~]# getenforce
Permissive
```

IMPORTANT: This change won’t survive a reboot, so next time you restart your system it will come back with SELinux enabled and enforcing again.

Permanently Disable SELinux

If you’re serious about disabling SELinux altogether, you’ll have to do two things:

1. Update **/etc/selinux.png/config** file (change **SELINUX=enforcing** to **SELINUX=disabled**)
2. Reboot your Linux system

See Also

- make iptables survice a reboot
- **SELinux Reference**
- Advanced Unix Commands
- Linux Commands