TUTORIAL

How to Configure SELinux Policies on CentOS 8

Configure SELinux policies on CentOS 8 by preparing the system, identifying denials, creating or adjusting policy rules, and validating the result before production use.

Operating Systems - July 19, 2026

Introduction

When a service works in permissive mode but fails in enforcing mode, the problem is often not the daemon itself but its SELinux policy context, labels, or permissions. On CentOS 8, that distinction matters operationally because SELinux is meant to keep enforcing while you tune the minimum policy required for the service to function.

This tutorial shows how to configure SELinux policies on CentOS 8 in a practical workflow: confirm the system state, decide whether a policy change is really needed, create or adjust the policy safely, validate the outcome, and verify the host is still suitable for production. By the end, you will know how to identify the right policy path, apply it with minimal privilege, and check that the service remains stable under enforcement.

What you are building

The finished state is a CentOS 8 host where:

- SELinux remains in enforcing mode.
- The affected service runs successfully without broad policy disablement.
- File labels, booleans, or custom policy rules are applied intentionally.

- Validation steps confirm the change is effective and limited in scope.
- You have evidence to support production rollout or rollback.

If your goal is only to make one specific service work, do not treat SELinux as a global on/off switch. For some cases, a boolean is enough; in others, you need file context correction or a custom policy module. If the root cause is not yet confirmed, use CentOS SELinux Troubleshooting: Fix Denials and Enforcing Mode to identify the denial before changing policy.

Prerequisites and stop-here checks

Before you change policy, verify these conditions.

Goal

Make sure the host is in a state where SELinux policy work can be done safely and reproducibly.

Action

Check the current SELinux mode and install the policy utilities if needed:

If the tools are missing, install them with your standard package workflow.

Expected output

- getenforce should usually report Enforcing on a production-like system.
- sestatus should confirm the SELinux status and policy type.
- The policy utilities should be installed so you can inspect labels, booleans, and audit logs.

Validation

Confirm that the host is actually managed by SELinux and not relying on a different security layer for access control. Also verify that you can collect audit logs, because policy work without logs turns into guesswork.

Common failure

A common mistake is changing policy before proving SELinux caused the problem. If the service is failing because of file ownership, wrong port binding, or a firewall rule, SELinux policy changes will only hide the real issue.

Stop-here-if warning

Stop here if you cannot confirm the denial source from logs, or if you are planning to set SELinux to permissive just to get the service running. That is a temporary diagnostic step, not a production fix.

Decide which SELinux change is appropriate

Not every SELinux issue requires a custom policy module. Choosing the smallest effective change is the main operational goal.

Goal

Select the least invasive correction that preserves enforcement.

Action

Use the denial details to choose one of three paths:

- Fix file contexts if the service is accessing content with the wrong label.
- Adjust a boolean if the service needs one specific policy behavior changed.
- Create a custom policy rule or module if the access pattern is valid but not permitted by existing policy.

For network-facing services, it is also worth checking whether the port is correctly opened in the firewall. If you need a refresher on service exposure and zone handling, How to Configure FirewallD on CentOS 8 Step by Step is useful for separating firewall issues from SELinux denials.

Expected output

You should know whether the fix belongs to labels, booleans, or custom policy.

Validation

A good decision rule is this: if changing a boolean resolves the denial cleanly, use it instead of writing policy. If relabeling the path resolves the issue, do that before creating a module. Reserve custom policy for cases where the required access is legitimate and stable.

Common failure

The most common failure is treating audit2allow output as an automatic recipe. Generated rules can be broader than necessary. Review them before deploying anything beyond a test host.

Inspect denials and labels

Before editing policy, confirm exactly what SELinux is blocking and what context the affected files currently carry.

Goal

Identify the denied action and the mislabeled object, if any.

Action

Review recent AVC denials and labels:

If the service uses files under a custom path, inspect both the parent directory and the target files. Labels often inherit from the parent, so one incorrect directory label can affect an entire tree.

Expected output

- AVC logs identify the denied source domain and target type.
- File and process labels show whether the service is running in the expected SELinux domain.
- The audit summary points to a specific class of access, such as file read, write, or network connection.

Validation

Cross-check the denial against the service design. If the service should only read content from a data directory, but the log shows write access to /var/www or another default location, the application may be pointing at the wrong path rather than needing a policy exception.

Common failure

A frequent problem is reading an old denial after the underlying issue has already changed. Clear your assumptions by collecting a fresh denial after reproducing the exact failure.

Fix file contexts first when the label is wrong

Incorrect labels are one of the safest and most common SELinux problems to correct.

Goal

Restore the expected label so SELinux can authorize access without policy relaxation.

Action

Check the file context mapping and apply the correct label:

Replace the example type with the correct type for your service. Use existing policy conventions whenever possible. If the path belongs to a persistent application data directory, define the mapping with semanage fcontext rather than relying only on a one-time relabel.

Expected output

The target files inherit the expected SELinux type, and the service can access them under enforcement.

Validation

Verify the applied labels:

Then restart the service and confirm no new AVC denials appear for the same path.

Common failure

If restorecon appears to succeed but the label reverts later, the path probably lacks a persistent fcontext rule. Apply the mapping first, then relabel again.

Use a boolean when the policy already supports the access pattern

Booleans are the preferred option when SELinux already has a documented switch for the required behavior.

Goal

Enable only the supported policy behavior that the service needs.

Action

Inspect booleans relevant to your service and enable the specific one:

The -P flag makes the change persistent across reboots. Use it only after you have confirmed the boolean is the correct control and the smallest safe adjustment.

If you are working with daemon-specific access limits, booleans are often a better choice than a custom module. For examples of how this approach helps preserve least privilege, see Configure SELinux Booleans on CentOS to Enforce Least Privilege.

Expected output

The needed capability becomes available without changing the broader SELinux domain.

Validation

Reproduce the original action and confirm the denial disappears. Then check the boolean state:

Common failure

A boolean can solve the symptom while leaving a label problem unresolved. If the access still fails after the boolean change, stop and re-check the denial rather than adding more permissive controls.

Create a custom policy module only when necessary

If labels and booleans do not fit the case, create a narrowly scoped policy module based on evidence.

Goal

Allow a specific, justified access path without disabling enforcement.

Action

First capture the denial, then generate a candidate module from the audit log. A common workflow is:

Use this only after you have reviewed the rules. Generated policy may include access that is wider than intended, so inspect the module source before installation.

A safer practice is to start from a very small set of denials and confirm that each proposed rule matches the intended service behavior.

Expected output

The module loads successfully, and the service gains only the access it needs to function.

Validation

Check that the module exists and is active:

Then reproduce the service action and confirm that the same denial no longer appears in audit logs.

Common failure

The common failure here is overfitting to one denial while ignoring a second, related access path. If the service still logs denials, do not keep adding broad rules blindly. Reassess the workflow and tighten the policy scope.

Validate the policy change

Validation is not optional. A policy change is not complete until the service works and the system still enforces SELinux as intended.

Goal

Prove the change solved the right problem and did not weaken the host more than necessary.

Action

Run these checks after the change:

If the service exposes a web endpoint or network port, test the real operational path rather than only the process state. A running daemon is not enough if it still cannot read its files or accept traffic.

Expected output

- SELinux remains in enforcing mode.
- The service starts cleanly.
- The original operation succeeds.
- No new AVC denials appear for the same scenario.

Validation

Repeat the exact user or application action that originally failed. This is the most important test because it proves the policy supports the production use case, not just a restart.

Common failure

A service can show active (running) while still failing at runtime due to denied file, socket, or network access. Always validate the functional path, not just the daemon status.

Operational follow-up for production use

Policy work should end with a record of what changed and how it was verified.

Goal

Make the SELinux adjustment maintainable and safe to audit later.

Action

Record the following in your change log or ticket:

- The original denial or file label issue.
- The chosen fix path: context, boolean, or module.
- The exact commands used.
- The validation performed.
- Any rollback step, if the change must be reversed.

If the change introduced a custom module, keep the source file and note why the module was necessary instead of a boolean or relabel.

Expected output

You have a repeatable record of why the policy exists and how to verify it after upgrades or configuration drift.

Validation

After a reboot or package update, recheck the service and the SELinux state. Policy changes that appear stable immediately after deployment can still fail later if file contexts drift or a configuration management job rewrites paths.

Common failure

The most common long-term failure is undocumented policy drift. Months later, no one remembers whether the fix came from a relabel, a boolean, or a custom module, and troubleshooting becomes slower than it should be.

Final takeaway

To configure SELinux policies on CentOS 8 correctly, start with evidence, choose the least invasive fix, and validate the exact service path under enforcing mode. If the issue is a wrong label, relabel it. If a documented boolean exists, enable only that behavior. If neither fits, create a narrow custom module and review it before installation. The production-ready outcome is not just a working service, but a working service that still benefits from SELinux enforcement.

Use this guidance together with CentOS 7 hardening with SELinux and Firewalld to connect the workflow with related operational context already available on the site.