Here is a robust **Chef InSpec** profile to verify that your configuration is working correctly.

This test focuses on the **"Enabled"** scenario (since that is the high-risk state) and proves that fapolicyd is running but **not** blocking your critical applications.

InSpec Control: verify_mq_splunk_fapolicyd.rb

You can save this file in your test/integration/default/ directory or run it as a standalone InSpec profile.

Ruby

```ruby
control 'mq-splunk-fapolicyd-access' do
  title 'Verify IBM MQ and Splunk execution under fapolicyd'
  desc 'Checks that fapolicyd is active but specifically allows MQ and Splunk binaries to execute'
  impact 1.0

  # ------------------------------------------------------
  # 1. Verify Security Service Status
  # ------------------------------------------------------
  describe service('fapolicyd') do
    it { should be_installed }
    it { should be_enabled }
    it { should be_running }
  end

  # ------------------------------------------------------
  # 2. Verify Rule Configuration
  # ------------------------------------------------------
  # Ensure the MQ rule file exists and has the correct relaxed trust level
  describe file('/etc/fapolicyd/rules.d/80-ibm-mq.rules') do
    it { should exist }
    its('content') { should match %r{allow perm=any all : dir=/opt/mqm/ all trust=0} }
  end

  # Ensure the Splunk rule file exists
  describe file('/etc/fapolicyd/rules.d/81-splunk.rules') do
    it { should exist }
    its('content') { should match %r{allow perm=any all : dir=/opt/splunkforwarder/ all trust=0} }
  end
```

```
# ------------------------------------------------------
# 3. Functional Execution Tests (The Real Proof)
# ------------------------------------------------------

# TEST A: IBM MQ
# We attempt to run a basic MQ binary (dspmq).
# If fapolicyd is blocking it, exit_status will be 126 or 1 (Operation not permitted).
describe command('/opt/mqm/bin/dspmq') do
  its('exit_status') { should eq 0 }
  its('stdout') { should match /QMNAME/ } # Assumes at least one QM exists, or just checks that it runs
  its('stderr') { should_not match /Operation not permitted/ }
end

# TEST B: Splunk
# We attempt to query the Splunk status.
describe command('/opt/splunkforwarder/bin/splunk status') do
  its('exit_status') { should eq 0 }
  its('stdout') { should match /splunkd is running/ }
  its('stderr') { should_not match /Operation not permitted/ }
end

# ------------------------------------------------------
# 4. Check for Recent Denials in Audit Logs
# ------------------------------------------------------
# This queries the audit log for any recent 'fanotify' denials related to MQ
# If this command returns output, it means something was blocked recently.
describe command('ausearch -m fanotify -ts recent | grep "/opt/mqm"') do
  its('exit_status') { should_not eq 0 } # Exit status 1 means grep found nothing (Good!)
end
end
```

How to use this

1. **If using Test Kitchen:** Save this in test/integration/default/fapolicyd_spec.rb and run kitchen verify.
2. **If running manually:** Copy the code to a file named check_mq.rb on the server and run:
   ```
   Bash
   inspec exec check_mq.rb
   ```

What if I want to test the "Disabled" scenario?

If you applied the **Disable** recipe, the test above will fail (because fapolicyd service is stopped). You can use this simpler control for that scenario:

Ruby

```ruby
control 'mq-splunk-disabled-fapolicyd' do
  title 'Verify fapolicyd is disabled'

  # 1. Service should be dead
  describe service('fapolicyd') do
    it { should_not be_running }
  end

  # 2. Apps should still work (via standard permissions)
  describe command('/opt/mqm/bin/dspmq') do
    its('exit_status') { should eq 0 }
  end
end
```

Next Step

Would you like me to combine all these logic pieces (Enable Recipe, Disable Recipe, and Splunk Fix) into a single **Chef Wrapper Cookbook** structure so you can see how the directory layout should look?