# Contributing to Falco

- [Contributing to Falco](#contributing-to-falco)

- [Code of Conduct](#code-of-conduct)

- [Issues](#issues)

- [Triage issues](#triage-issues)

- [More about labels](#more-about-labels)

- [Slack](#slack)

- [Pull Requests](#pull-requests)

- [Commit convention](#commit-convention)

- [Rule type](#rule-type)

- [Coding Guidelines](#coding-guidelines)

- [C++](#c)

- [Unit testing](/tests/README.md)

- [Developer Certificate Of Origin](#developer-certificate-of-origin)

## Code of Conduct

Falco has a

[Code of Conduct](CODE\_OF\_CONDUCT.md)

to which all contributors must adhere, please read it before interacting with the repository or the community in any way.

## Issues

Issues are the heartbeat ?? of the Falco project, there are mainly three kinds of issues you can open:

- Bug report: you believe you found a problem in Falco and you want to discuss and get it fixed,

creating an issue with the \*\*bug report template\*\* is the best way to do so.

- Enhancement: any kind of new feature need to be discussed in this kind of issue, do you want a new rule or a new feature? This is the kind of issue you want to open. Be very good at explaining your intent, it's always important that others can understand what you mean in order to discuss, be open and collaborative in letting others help you getting this done!

- Failing tests: you noticed a flaky test or a problem with a build? This is the kind of issue to triage that!

The best way to get \*\*involved\*\* in the project is through issues, you can help in many ways:

- Issues triaging: participating in the discussion and adding details to open issues is always a good thing,

sometimes issues need to be verified, you could be the one writing a test case to fix a bug!

- Helping to resolve the issue: you can help in getting it fixed in many ways, more often by opening a pull request.

### Triage issues

We need help in categorizing issues. Thus any help is welcome!

When you triage an issue, you:

\* assess whether it has merit or not

\* quickly close it by correctly answering a question

\* point the reporter to a resource or documentation answering the issue

\* tag it via labels, projects, or milestones

\* take ownership submitting a PR for it, in case you want ?

#### More about labels

These guidelines are not set in stone and are subject to change.

Anyway a `kind/\*` label for any issue is mandatory.

This is the current [label set](https://github.com/falcosecurity/falco/labels) we have.

You can use commands - eg., `/label <some-label>` to add (or remove) labels or manually do it.

The commands available are the following ones:

```

/[remove-](area|kind|priority|triage|label)

```

Some examples:

\* `/area rules`

\* `/remove-area rules`

\* `/kind kernel-module`

\* `/label good-first-issue`

\* `/triage duplicate`

\* `/triage unresolved`

\* `/triage not-reproducible`

\* `/triage support`

\* ...

### Slack

Other discussion, and \*\*support requests\*\* should go through the `#falco` channel in the Kubernetes slack, please join [here](https://slack.k8s.io/).

## Pull Requests

Thanks for taking time to make a [pull request](https://help.github.com/articles/about-pull-requests) (hereafter PR).

In the PR body, feel free to add an area label if appropriate by typing `/area <AREA>`, PRs will also

need a kind, make sure to specify the appropriate one by typing `/kind <KIND>`.

The list of labels is [here](https://github.com/falcosecurity/falco/labels).

Also feel free to suggest a reviewer with `/cc @theirname`, or to assign an assignee using `/assign @nickname`.

Once your reviewer is happy, they will say `/lgtm` which will apply the

`lgtm` label, and will apply the `approved` label if they are an

[owner](/OWNERS).

Your PR will be automatically merged once it has the `lgtm` and `approved`

labels, does not have any `do-not-merge/\*` labels, and all status checks (eg., rebase, tests, DCO) are positive.

### Commit convention

As commit convention, we adopt [Conventional Commits v1.0.0](https://www.conventionalcommits.org/en/v1.0.0/), we have an history

of commits that do not adopt the convention but any new commit must follow it to be eligible for merge.

#### Rule type

Besides the classic types, we adopt a type for rules, `rule(<scope>):`.

Example:

```

rule(Write below monitored dir): make sure monitored dirs are monitored.

```

Each rule change must be on its own commit, if a change to a macro is done while changing a rule they can go together but only one rule per commit must happen.

If you are changing only a macro, the commit will look like this:

```

rule(macro user\_known\_write\_monitored\_dir\_conditions): make sure conditions are great

```

## Coding Guidelines

### C++

\* File `userspace/engine/banned.h` defines some functions as invalid tokens. These functions are not allowed to be used in the codebase. Whenever creating a new cpp file, include the `"banned.h"` headers. This ensures that the banned functions are not compiled.

A complete list of banned functions can be found [here](./userspace/engine/banned.h).

## Developer Certificate Of Origin

The [Developer Certificate of Origin (DCO)](https://developercertificate.org/) is a lightweight way for contributors to certify that they wrote or otherwise have the right to submit the code they are contributing to the project.

Contributors to the Falco project sign-off that they adhere to these requirements by adding a `Signed-off-by` line to commit messages.

```

This is my commit message

Signed-off-by: John Poiana <jpoiana@falco.org>

```

Git even has a `-s` command line option to append this automatically to your commit message:

```

$ git commit -s -m 'This is my commit message'

```