TF-A LTS Proposal

Authors: Okash Khawaja (Google), Varun Wadekar (NVIDIA)
Draft: 2022-08-05

Document History

Date	Author	Description
2022-07-20	Okash Khawaja, Varun Wadekar	Initial draft.
2022-07-21	Varun Wadekar	Refine the Maintainership guidelines and planning sections. Introduce a new section documenting a day in the life of a LTS branch maintainer
2022-08-05	Varun Wadekar, Okash Khawaja	Merge two drafts (draft 1 and 2), address comments made by both authors, cosmetic changes to the content all over the document

Summary

This document proposes a plan for long-term support (LTS) of the Trusted Firmware A (TF-A) project.

Why is LTS required?

LTS is needed for commercial reasons. More specifically, on the device side, when a product is released, the companies have to support that in-market product such that the amount of changes to the firmware are kept to a minimum to avoid the risk of regression. At the same time the companies don't want to exclude critical patches such as those for security advisories. Similarly on server side, companies want to minimize the churn when deploying fixes during incident response, e.g. due to critical security bugs.

This means that those companies have to maintain and backport critical updates to old branches internally. As this effort is duplicated across different companies using TF-A, it makes sense to factor out this effort into a community-wide LTS.

What does LTS mean for TF-A?

In this section we will define exactly what constitutes LTS for TF-A. Specifically, we will define the following characteristics:

- criteria for selecting patches which will be backported to LTS branches
- lifetime and frequency of LTS branches

Criteria

We must have an objective criteria for selecting patches to be backported to LTS branches. This will make maintenance easy because:

a) there will be less -- ideally no -- discussion when selecting patches to backport
b) large parts of the process can be automated

Below is the criteria

1. No features will be backported.
2. Security advisories: Any patch that makes it into TF-A security advisories is automatically selected for back porting. This includes patches to external components too, e.g. libfdt.
3. Workarounds for CPU and other ARM IP errata
4. Workarounds for non-ARM IP errata, e.g. TI UART
5. Fixes for platform bugs. These patches must not modify any code outside of the specific

platform that the fix applies to.
6. Patches can only be backported from the master branch. In other words, the master branch will be a superset of all the changes in any LTS branch.

Lifetime and frequency

This section approaches three questions: for how long should an LTS release be supported, how frequently should LTS releases be made and at which time(s) of the year should the releases be made.

1. **For how long should an LTS release be supported?**

   Linux kernel supports an LTS branch for 5 years. Since firmware tends to have less churn and longer lifetime than HLOS, TF-A should support at least 5 years for its LTS. We should leave the room open for discussions about extending it to 7 years.

2. **How frequently should LTS releases be made?**

   Given that many products that have a release cycle, have a yearly release cycle, it would make sense to have yearly TF-A releases.

3. **Which time(s) of the year should the releases be made?**

   TF-A releases are cut twice a year: May and November. Basing LTS release on November TF-A release has a few benefits. First, it aligns with Linux LTS releases which happen towards the end of each year. Second, it aligns with Android releases which tend to fall in Q3 each year. Since product releases are timed with Android release, this gives enough time to harden the TF-A LTS release during development so that it's ready for launch in Q3 following year. On the other hand, if the May release of TF-A is chosen as the basis for LTS then developers will have very little time -- about a month, taking into account the test-and-debug phase before LTS is cut (see below) -- before Android release.

To summarize, there will be one LTS release per year. It will be supported for 5 years and we can discuss extending it to 7 years later on. The LTS release will be based on the November release of TF-A.

Testing Criteria

Every patch merged to the LTS branch will complete the following tests before getting approved.

1. TFTF tests currently running in the testing farm
2. CI/CD static analysis scans
3. Coverity scans

4. Platform tests

Platforms that are not maintained upstream will undergo testing downstream in a pre-defined window. The platform maintainer will complete the testing and provide a verified score on the patch once testing is completed.

TFTF branching

A note about testing here. After a patch is backported to an LTS branch, that branch will need to be regression tested. Since TFTF moves forward with latest TF-A changes, newer TFTF tests may not apply to old LTS branches. Therefore TFTF will also need to be branched, in-sync with TF-A LTS branches. In other words, there will be one TFTF LTS branch corresponding to each TF-A LTS branch. The TFTF LTS branch will be used to regression test the corresponding TF-A LTS branch.

As we work with the LTS branch of TFTF, we might also need fixes for TFTF itself to be ported to LTS. However, decision-making about those patches need not be as stringent as for TF-A.

Release details

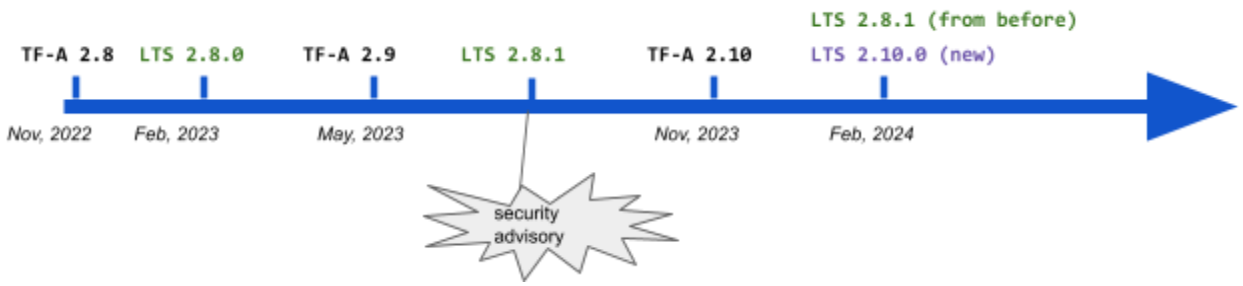
This section goes into details of what the LTS release process will look like.

Test-and-debug period

Since the LTS branch will be used in product releases, it is expected that more testing and debugging will be done on the November release of TF-A. Therefore it would make sense to leave at least a month after the November release and then cut the LTS branch. We recommend about two months, given that one of the months is December which tends to be slowed down due to holidays. So an end-of-November TF-A release would result in a beginning-of-February LTS release. Note that the LTS branch will be created at the same time as the TF-A November release but it will be officially released at the end of January or early February. Going forward we should strive to make the period smaller and smaller until ideally it coincides with TF-A November release which means that our test and CI/CD infra is good enough to allow that to happen.

Example timeline

Below is an example timeline starting from the November 2022 release of TF-A.



- **Nov 2022:** TF-A 2.8 is released towards the end of Nov, 2022. Not shown in the diagram, at the same time LTS release candidate branch is made which is based on TF-A 2.8. This means new features going in 2.8 won't go in the LTS branch. We can call it `LTS 2.8-rc`.
- **Feb 2023:** After testing and debugging LTS 2.8-rc for a couple of months, LTS 2.8.0 is officially released in early Feb 2023.
- **May 2023:** TF-A 2.9 is released but since this is not an LTS branch it doesn't affect LTS.
- **Somewhere between May and Nov of 2023:** A security advisory comes up and the related patches go into TF-A master branch. Since these patches fall under LTS criteria, they are backported to LTS 2.8.0 which results in LTS 2.8.1 being released. Note that here we don't allow the extra testing and debugging time that we had between Nov 2022 and early Feb 2023. This is because there isn't as much to test and debug as an annual LTS release has. Also companies might want to deploy critical patches soon.
- **Nov 2023:** TF-A 2.10 is released. Not shown in the diagram, at the same time LTS 2.10-rc is made. It's tested by partners for a couple of months.
- **Feb 2024:** LTS 2.10.0 is released in early Feb. Now there are two LTS branches: 2.8.1 and 2.10.0.

Note that TFTF will follow similar branching model as TF-A LTS, i.e. there will be TFTF LTS 2.8.0 in Feb 2023, 2.8.1 (if new TFTF tests need to be added for the security advisory) when there is TF-A LTS 2.8.1 and so on.

Maintainership

Guidelines & Responsibilities

1. Maintainers shall be impartial and strive to work for the benefit of the community
2. Objective and well-defined merge criteria to avoid confusion and discussions at random points in time when there is a "candidate" patch
3. The maintainers shall explain the lifecycle of a patch to the community, with a detailed description of the maximum time spent in each step
4. Automate, automate, automate
5. Reviewers should not focus too much on "what" and instead focus on "how"
6. Constantly refine the merge criteria to include more partner use cases

7. Ensure that all candidate patches flow from the main branch to all LTS branches

Options

These are some options in the order of preference.

1. Current set of maintainers from tf.org (or hired contractor) take care of the LTS
2. From the community, create a set of maintainers focused solely on the LTS branches

A day in the life of a maintainer

This section documents the daily tasks that a maintainer might perform to support the LTS program. It is expected that a maintainer follows clearly laid down steps and does not have to make policy level decisions for merge, testing, or candidate patch selection.

1. Monitor the main branch to identify candidate patches for the LTS branches
2. Inform the mailing list of a new candidate patch for LTS and solicit feedback
3. Start the review process and CI/CD cycle for the patch
4. Review the CI/CD output to ensure that the quality bar is met
5. After reviews are complete, merge the patch and bump the minor version, if required.
6. Monitor the mailing list for any LTS related issues
7. Propose or solicit patches to the main branch and tag them as candidates for LTS

Execution Plan

This section lists the steps needed to put in place the LTS system. However, in order to kick start LTS in Nov ‘22, only a few steps are needed. The rest can follow in the background.

Initial release steps

The following steps are necessary to kickstart the project and potentially create the first LTS from the Nov’22 release.

1. Create a TF-A LTS release-candidate branch and a TFTF LTS branch immediately after the Nov’22 release
2. Request all platform-owners to test and debug the RC branch
3. Gather feedback from the test and debug cycle
4. Mark the TF-A LTS branch ready by the end of January
5. Announce the official LTS release availability on the mailing lists

Long term release plan

Above will buy us time to then work on the rest of the execution plan which is given below.

1. The review criteria for LTS patches must be the same as TF-A patches.
2. The maintainers shall publish the well-defined merge criteria to allow the community to choose candidate patches
3. The maintainers shall publish a well-defined test specification for any patch entering the LTS branch
    a. Tests required to pass in the CI/CD flow
    b. Static analysis scans
    c. Coverity scans
4. The maintainers shall publish a mechanism to choose candidate patches for the LTS branch
5. The maintainers shall publish a mechanism to report bugs[1] seen with a LTS branch
6. The maintainers shall publish a versioning mechanism for the LTS branch
    a. Bump minor version for every “logical”[2] fix that gets merged
7. The CI/CD infrastructure shall provide test support for all “live” LTS branches at any given point in time
8. The CI/CD infrastructure shall provide means to
    a. notify all maintainers that a patch is ready for review
    b. automatically cherry-pick a patch to a given LTS branch
    c. get it through the CI/CD testing flow
    d. send nag emails to maintainers at regular intervals to ensure reviews keep moving

Future plans

In our discussions, in addition to the above points we also considered some questions but deferred answers to those questions to a later point when they become a real problem. Some of them are listed below.

- What happens when a bug fix applies just to a LTS branch and not to the master branch?
- When testing a backported patch, what if one of the partners needs more time while the patch fix is time-critical and hence slowing other partners?
- Too many CPU errata workarounds resulting in too many LTS releases. We propose bumping the version number for each logical fix as described in the section “Long term release plan” above because that will help accurately track what changes have been

[1] The plan is to create a system where reviewers can tag a patch on mainline which gets automatically rebased on LTS and pushed to gerrit. On seeing this patch, the CI/CD starts tests and provides a score. In parallel, the system also send an email to the maintainers announcing the arrival of a candidate patch for the LTS branch.

[2] Logical will be a patch or patches implementing a certain fix. For example, if a security mitigation is fixed with the help of three patches, then all of them are considered as one "logical" fix. The version is incremented only after all these patches are merged.

deployed in-field.
- What if LTS support duration needs to be extended to longer than 5 years?

At this stage we don’t want to address those questions. When they become real problems, then we will have concrete data and be better able to address them. This means that our LTS definition as presented in this document is not the final one. We will constantly be discussing it and deciding how to adapt it as we see practical problems.