INSTITUTE FOR DEFENSE ANALYSES

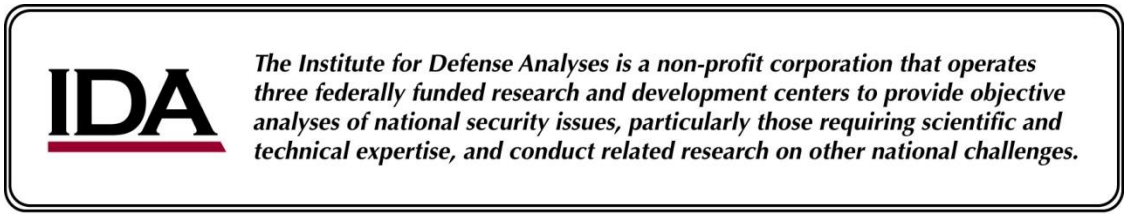
Core Infrastructure Initiative (CII) Best-Practices Badge Criteria

David A. Wheeler

June 28, 2016

IDA Non-Standard
NS D-8054
Log: H 2016-000794
Copy

INSTITUTE FOR DEFENSE ANALYSES
4850 Mark Center Drive
Alexandria, Virginia 22311-1882



About This Publication

This work was conducted by the Institute for Defense Analyses (IDA) under Task LX-5-3968, "Linux Foundation Core Infrastructure Initiative," for The Linux Foundation. The views, opinions, and findings should not be construed as representing the official position of either the Department of Defense or the sponsoring organization.

Acknowledgments

Margaret E. Myers

Core Infrastructure Initiative (CII) Best-Practices Badge Criteria

David A. Wheeler

The Linux Foundation Core Infrastructure Initiative (CII) recently announced the general availability of its best-practices badge project, which is meant to help projects follow practices that will improve their security. I'm the technical lead of the project, which is also known as the "badging project." In this article I'll focus on what the badge criteria currently are, including how they were developed and some specific examples, as well as talk about the project as a whole. But first, a little history.

In 2014, the Heartbleed vulnerability was found in the OpenSSL cryptographic library. This vulnerability raised awareness that there are some vitally important free/libre and open source software (FLOSS) projects that have serious problems. In response, the Linux Foundation created the CII to fund and support critical elements of the global information infrastructure. The CII has identified and funded specific important projects, but it cannot fund all projects. So the CII is also funding some approaches to generally improve the security of FLOSS.

The badge

The latest CII project, which focuses on improving security in general, is the "best-practices badge" project. CII believes that FLOSS projects that follow best practices are more likely to be healthy and to produce better software in many respects, including having better security. Most project members want their projects to be healthy, and users prefer to depend on healthy projects. Without a list of best practices, it's easy to overlook something important.

FLOSS projects that adequately follow the best practices can get a badge to demonstrate that they do. It costs no money to get a badge, and filling in the form takes less than an hour. Note that the CII best-practices badge is for a project, not for an individual, since project members can change over time.

There really is a problem today; some projects are *not* applying the hard-learned lessons of other projects. Many projects are not released using a FLOSS license, yet their developers often appear to (incorrectly) think they are FLOSS projects. Ben Balter's 2015 presentation *Open source licensing by the numbers* suggested that on GitHub, 23% of the projects with 1,000 or more stars had no license at all. Omitting a FLOSS license tends to inhibit project use, co-development, and review (including security reviews).

Some projects (like american fuzzy lop) do not have a public version-controlled repository, making it difficult for others to track changes or collaborate. Some projects only provide unauthenticated downloads of their code using HTTP, making it possible for attackers to subvert software downloads en route. Some projects don't provide any information on how to submit vulnerability reports (are you supposed to use the usual bug tracker?); this can create unnecessary delays in vulnerability reporting and handling. Many projects don't use any static source code analysis tools, even though these tools can find defects (including vulnerabilities).

OpenSSL before Heartbleed is an example. The OpenSSL project at the time of Heartbleed had a legion of problems. For example, its code was hard to read (there was no standard coding style and its code was notoriously complex), making it difficult to review. Unsurprisingly, vulnerabilities (like Heartbleed) are more likely to slip in when code is difficult to review. The best-practices criteria were not created with OpenSSL specifically in mind, but one of its project members went back and found that the OpenSSL project before Heartbleed failed to meet about one-third of the current best-practices criteria.

Of course, there are a massive number of practices that could together be called "best practices." The term "best practices" is really just a commonly used term for some set of recommended practices.

Let's first admit the limitations on any set of practices. No set of practices can *guarantee* that software will never have defects or vulnerabilities. Even formal methods can fail if the specifications or assumptions are wrong. Nor is there any set of practices that can guarantee that a project will sustain a healthy and well-functioning development community.

However, following best practices can help improve the results of projects. For example, some practices enable or encourage multi-person review, or can make review more effective at finding defects (including defects that lead to vulnerabilities).

Perhaps the most important step toward developing the criteria (and the web application that implements them) was the decision that the project would *itself* be developed as a FLOSS project. The web application is under the MIT license; all text (including the criteria) are dual-licensed under the MIT or CC-BY version 3.0 (or later)

licenses. The CII publicly set up the project on GitHub, created some early draft criteria, and invited feedback.

Producing the criteria

The initial criteria were primarily based on reviewing a lot of existing documents about what FLOSS projects should do, and those were in turn based on observing existing successful projects. A good example, and probably the single most influential source, is Karl Fogel's book *Producing Open Source Software*. Many people provided feedback or contributed to the badging project, including Dan Kohn, Emily Ratliff, Karl Fogel, Greg Kroah-Hartman (the Linux kernel), Rich Salz (OpenSSL), Daniel Stenberg (curl), Sam Khakimov, Doug Birdwell, Alton Blom, and Dale Visser.

A web application was developed for FLOSS project members to use to fill in information; that web application project fulfilled the criteria, so it got its own badge. This effort helped steer the project away from impractical criteria. The project also got some early "alpha tester" projects to try out early drafts and provide feedback, in particular to ensure that the criteria would apply to both big projects (like the Linux kernel) and small projects (like curl). For example, there is no criterion requiring 100% statement coverage for tests; that can be a useful goal, but on many projects that's impractical (especially if it requires unusual hardware) or not worth pursuing.

Getting a badge intentionally doesn't require or forbid any particular services or programming languages. A lot of people use GitHub, and in those cases the web application automatically fills in some of the form based on data from GitHub, but projects do not *have* to use GitHub.

Scale is also a key issue. An evaluation process that takes a year or more, or costs hundreds of thousands of dollars, cannot be applied to all of the vast number of FLOSS projects. In-depth evaluation is not bad, of course, but the project is trying to be useful for a large set of FLOSS projects. Instead of requiring expensive third-party assessment, the focus is on self-assessment combined with automation.

Self-assessment can have its problems, but it scales much better and there are several approaches to help counter the problems of self-assessment, such as false claims. First, all the results are made public, so anyone can check the claims. Second, the web application also includes automation that checks entries before they are saved — and in some cases it overrides user information if it's false or inadequately justified. Finally, the CII does review project entries (particularly if they claim to be passing) and can delete or fix entries (e.g., if they are false or irrelevant). This emphasis on self-assessment does mean that the badging project had to try to write criteria that could be clearly understood directly by the projects.

Currently, the focus is on identifying best practices that well-run projects typically *already* follow. The project leads decided that it was more important to come up with a smaller set of widely applied best practices. That way, all projects can be helped to reach some minimum bar that is widely accepted. The project was especially interested in criteria that help enable multi-person review or tend to improve security. The criteria also had to be relevant, attainable by typical FLOSS projects, and clear. It was also preferred to add criteria if at least one project didn't follow the practice. After all, if everyone does it without exception, it would be a waste of time to add it as a criterion.

In the longer term, there are plans to add *higher* badge levels beyond the current "passing" level, tentatively named the "gold" and "platinum" levels. Projects that are widely depended on and are often attacked, such as the Linux kernel or any cryptographic library, should, of course, be doing *much* more than a minimum set of widely applied best practices. However, the project team decided to create the criteria in stages.

There is the expectation that once a number of projects get a passing badge (and provide feedback), the badging project will be in a better position to determine the criteria for higher levels. You can see a list of some of the proposed higher-level criteria in the "other" criteria documentation. If you think of others, or think some are especially important, please let the badging project know.

One intentional omission is anything actually *requiring* an active development community, multi-person review, or multiple developers (e.g., a high "bus factor"). Obviously, having more reviewers or developers within an active community is much better for a project, and users should normally prefer such projects. However, in many cases this is not directly under a project's control. For example, some projects are so specialized that they're not likely to attract many reviewers or developers and new projects often can't meet such criteria. For the initial badge level, the focus is, instead, on things that project members can directly control. Meeting the badge criteria should help projects grow and sustain a healthy, well-functioning, and active development community. Higher badge levels will almost certainly add criteria requiring a larger active community and a minimum bus factor (at least more than one).

The criteria

Once the initial criteria were identified, they were grouped into the following categories: basics, change control, reporting, quality, security, and analysis. Below, a few of the 66 criteria (including their identifiers) are described, along with why they're important.

The "basics" group includes basic requirements for the project. This includes requiring either a project or repository URL (the web application uses this to automatically fill in some information). Examples include:

- Criterion floss_license states that "the software MUST be released as FLOSS" and FLOSS is defined as software "released in a way that meets the Open Source Definition or Free Software Definition." These criteria were designed for FLOSS projects and are meant to encourage collaborative development and review. That doesn't make sense when there's no legal basis for the collaboration.
- Criterion sites_https says that "the project sites (web site, repository, and download URLs) MUST support HTTPS using TLS" This is obviously a more security-oriented requirement. It's sparked some controversy, because GitHub pages do not fully support HTTPS. Although users can retrieve *.github.io pages using HTTPS, these pages are still vulnerable to interception and malicious modification because, at this time, they are retrieved via CloudFlare, which retrieves these files *without* using HTTPS. In addition, many projects have a custom domain (typically the project's name) with a web site served via GitHub pages and these cannot currently be protected by HTTPS at all. One compromise being discussed is to only require that the repository and download URLs use HTTPS, since that would at least protect the software while it's downloaded.

The "change control" group focuses on managing change, including ways to report problems, issue/bug trackers, and version-control software. Examples include:

- The repo_public criterion says that "the project MUST have a version-controlled source repository that is publicly readable and has a URL." Version control greatly reduces the risks of changes being dropped or incorrectly applied and makes it much easier to apply changes.
- Criterion vulnerability_report_process says: "The project MUST publish the process for reporting vulnerabilities on the project site." This makes it much easier for security researchers to provide their reports—and thus makes it more likely that that will happen. Many bug reporting systems are public, and it's not obvious to outsiders if projects will want security bug reports to be public or not. A surprising number of projects didn't meet this criterion, even though this can be as simple as putting one sentence on the project web site.

The "quality" group focuses on general software quality, including a project's build process and automated test suite. Examples include:

- Criterion test: "The project MUST have at least one automated test suite that is publicly released as FLOSS (this test suite may be maintained as a separate FLOSS project)." An automated test suite makes it much easier to detect many mistakes before users have to deal with them. Test suites can always be improved; the key is to have one that *can* be improved.
- The warnings criterion says: "The project MUST enable one or more compiler warning flags, a 'safe' language mode, or use a separate 'linter' tool to look for code quality errors

or common simple mistakes, if there is at least one FLOSS tool that can implement this criterion in the selected language." These flags and tools can detect some defects, some of which may be security vulnerabilities. In addition, these mechanisms can warn about awkward constructs that make code hard to read.

The "security" group lists criteria specific to improving software security. Examples include:

- The know_secure_design criterion states: "The project MUST have at least one primary developer who knows how to design secure software. This requires understanding the following design principles, including the 8 principles from Saltzer and Schroeder...." There are a number of well-known design principles for designing secure software, such as using fail-safe defaults (access decisions should deny by default and installation should be secure by default). Knowing these principles can reduce the likelihood or impact of vulnerabilities.
- Criterion know_common_errors: "At least one of the primary developers MUST know of common kinds of errors that lead to vulnerabilities in this kind of software, as well as at least one method to counter or mitigate each of them." Most vulnerabilities stem from a small set of well-known kinds of errors, such as vulnerabilities from SQL injections and buffer overflows. Knowing what they are (and how to counter or mitigate them) can result in an order-of-magnitude reduction in the number of vulnerabilities. It woud be best if all of the developers knew this; but if one does, that person can teach the others. The biggest problem is when *no* developer knows this information.
- Criterion crypto_published states: "The project's cryptographic software MUST use only cryptographic protocols and algorithms that are publicly published and reviewed by experts." Home-grown cryptography is vulnerable cryptography. You need to have an advanced degree in mathematics or a related field and have specialized for years in cryptography, before you know enough to create new cryptographic protocols and algorithms that can stand up to today's aggressive adversaries.

The "analysis" group lists criteria specific to analyzing software. Examples include:

- Criterion static_analysis requires: "At least one static code analysis tool MUST be applied to any proposed major production release of the software before its release, if there is at least one FLOSS tool that implements this criterion in the selected language. A static code analysis tool examines the software code (as source code, intermediate code, or executable) without executing it with specific inputs. For purposes of this criterion, compiler warnings and 'safe' language modes do not count as static code analysis tools (these typically avoid deep analysis because speed is vital)." Static code-analysis tools (designed for that purpose) can dig deep into code and find a variety of problems. It's true that these tools can't find everything, but the idea is to try to find to fix the problems that *can* be found this way.
- Criterion dynamic_analysis says: "It is SUGGESTED that at least one dynamic analysis tool be applied to any proposed major production release of the software before its release." Dynamic-analysis tools can find vulnerabilities that static-analysis tools often

> miss (and vice versa), so it's best to use both. It would be nice to use them on every commit, but on some projects that's impractical; typically, though, they can be applied to every release.

The criteria will change slowly, probably annually, as the project gets more feedback and the set of best practices in use changes. The current plan is to add proposed criteria as "future" criteria, which are added to the web application but are initially ignored. That will give projects time to meet the new criteria (and show that they do), justify modifying the criteria, or justify removing it from the set of proposed criteria.

For example, the hardening criterion is currently a planned addition; it would require that "hardening mechanisms be used so software defects are less likely to result in security vulnerabilities." The current plan is that this criterion would be added at the "passing" level for *all* projects in 2017. Projects that don't meet the updated criteria by the update deadline would lose their "passing" status until they fix the problem. This process is similar to a "recertification" process but is hopefully less burdensome.

FLOSS projects that have already achieved the badge include the Linux kernel, curl, Node.js, GitLab, OpenBlox, OpenSSL, and Zephyr. I encourage all FLOSS project members to go to the site and get their badges. If you have comments on the criteria (including for higher levels to be developed), please submit comments using the GitHub issue tracker or project mailing list.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE (DD-MM-YY)	2. REPORT TYPE	3. DATES COVERED (From – To)
2016-06-28	Non-Standard	

4. TITLE AND SUBTITLE

Core Infrastructure Initiative (CII) Best-Practices Badge Criteria

5a. CONTRACT NUMBER

N/A

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBERS

6. AUTHOR(S)

David A. Wheeler

5d. PROJECT NUMBER

LX-5-3968

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESSES

Institute for Defense Analyses
4850 Mark Center Drive
Alexandria, VA 22311-1882

8. PERFORMING ORGANIZATION REPORT NUMBER

NS D-8054
H 2016-000794

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Marcus Streets
Core Infrastructure Initiative (CII) Programs Director
The Linux Foundation
1 Letterman Drive
Building D
Suite D4700
San Francisco CA 94129

10. SPONSOR'S / MONITOR'S ACRONYM

Linux

11. SPONSOR'S / MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION / AVAILABILITY STATEMENT

Approved for public release; distribution is unlimited.

13. SUPPLEMENTARY NOTES

Project Leader: David A. Wheeler

14. ABSTRACT

The Linux Foundation Core Infrastructure Initiative (CII) recently announced the general availability of its best-practices badge project, which is meant to help projects follow practices that will improve their security. This article focuses on what the badge criteria currently are, including how they were developed and some specific examples, and talks about the project as a whole.

15. SUBJECT TERMS

open source software, free software, free/libre/open source software, FLOSS, best practices, Linux Foundation, LF, Core Infrastructure Initiative, CII, badge, badging, badges, LWN.net

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			Marcus Streets
Unclassified	Unclassified	Unclassified	Unlimited	7	19b. TELEPHONE NUMBER (Include Area Code) +44 7411 711101