CII badging review

A former CII badging evaluation for OpenDaylight was posted at https://bestpractices.coreinfrastructure.org/en/projects/617

by Casey Cain for Boron
(NB the first one at https://bestpractices.coreinfrastructure.org/en/projects/78 posted by Daniel Farell is older and less advanced.)

Below is a first analysis of what has been posted and what must be modified, with security at a first target.

Basics

Identification

What is the human-readable name of the project?
opendaylight
-> shouldn't it be OpenDaylight rather ?

What is the URL for the version control repository (it may be the same as the project URL)?
https://github.com/opendaylight

-> shouldn't it be https://git.opendaylight.org/ rather ?

The project website MUST provide information on how to: obtain, provide feedback (as bug reports or enhancements), and contribute to the software.
docs.opendaylight.org ask.opendaylight.org

-> ask.opendaylight.org does not exist anymore: wiki.opendaylight.org ?

The information on how to contribute MUST explain the contribution process (e.g., are pull requests used?) (URL required) [contribution]
Project uses pull requests. http://docs.opendaylight.org/en/stable-boron/developer-guide/index.html

-> it is not really described in developer guide and boron is EOL
should rather be https://docs.opendaylight.org/en/latest/#contributing-to-opendaylight

The information on how to contribute SHOULD include the requirements for acceptable contributions (e.g., a reference to any required coding standard). (URL required) [contribution_requirements]
http://docs.opendaylight.org/en/stable-boron/developer-guide/pulling-and-pushing-the-code-from-the-cli.html (a) The contribution was created in whole or in part by me and I have the right to submit it under the open source license indicated in the file; or

-> http://docs.opendaylight.org/en/stable-boron/developer-guide/index.html

The project MUST have one or more mechanisms for discussion (including proposed changes and issues) that are searchable, allow messages and topics to be addressed by URL, enable new people to participate in some of the discussions, and do not require client-side installation of proprietary software. [discussion]
All meetings, with the exception of the Board meeting are open to the public, recorded and archived on the OpenDaylight Wiki. https://wiki.opendaylight.org/view/Meetings

-> Board was moved at the LFN level, https://wiki.opendaylight.org/display/ODL/Community+Meetings

Security

Secure development knowledge

The project MUST have at least one primary developer who knows how to design secure software. (See 'details' for the exact requirements.) [know_secure_design]

At least one of the project's 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.

-> yes

Use basic good cryptographic practices

The software produced by the project MUST use, by default, only cryptographic protocols and algorithms that are publicly published and reviewed by experts (if cryptographic protocols and algorithms are used).

-> yes

If the software produced by the project is an application or library, and its primary purpose is not to implement cryptography, then it SHOULD only call on software specifically designed to implement cryptographic functions; it SHOULD NOT re-implement its own.

-> yes

All functionality in the software produced by the project that depends on cryptography MUST be implementable using FLOSS.

-> yes

The security mechanisms within the software produced by the project MUST use default keylengths that at least meet the NIST minimum requirements through the year 2030 (as stated in 2012). It MUST be possible to configure the software so that smaller keylengths are completely disabled.

-> yes

The default security mechanisms within the software produced by the project MUST NOT depend on broken cryptographic algorithms (e.g., MD4, MD5, single DES, RC4, Dual_EC_DRBG), or use cipher modes that are inappropriate to the context, unless they are necessary to implement an interoperable protocol (where the protocol implemented is the most recent version of that standard broadly supported by the network ecosystem, that ecosystem requires the use of such an algorithm or mode, and that ecosystem does not offer any more secure alternative). The documentation MUST describe any relevant security risks and any known mitigations if these broken algorithms or modes are necessary for an interoperable protocol.

-> yes but not sure about doc

The default security mechanisms within the software produced by the project SHOULD NOT depend on cryptographic algorithms or modes with known serious weaknesses (e.g., the SHA-1 cryptographic hash algorithm or the CBC mode in SSH).

-> yes and no, SHA-1 is still supported but not activated in Mina-SSHD and has been temporarily reintroduced for compatibility with some other controller. Also, it's one of the few diffie hellman SHA-1 variant whose vulnerability cannot be exploited.

The security mechanisms within the software produced by the project SHOULD implement perfect forward secrecy for key agreement protocols so a session key derived from a set of long-term keys cannot be compromised if one of the long-term keys is compromised in the future. [crypto_pfs]

-> yes

If the software produced by the project causes the storing of passwords for authentication of external users, the passwords MUST be stored as iterated hashes with a per-user salt by using a key stretching (iterated) algorithm (e.g., Argon2id, Bcrypt, Scrypt, or PBKDF2). See also OWASP Password Storage Cheat Sheet). [crypto_password_storage]

-> ?

The security mechanisms within the software produced by the project MUST generate all cryptographic keys and nonces using a cryptographically secure random number generator, and MUST NOT do so using generators that are cryptographically insecure.

-> ?

Secured delivery against man-in-the-middle (MITM) attacks

The project MUST use a delivery mechanism that counters MITM attacks. Using https or ssh+scp is acceptable

-> yes

A cryptographic hash (e.g., a sha1sum) MUST NOT be retrieved over http and used without checking for a cryptographic signature.

-> yes

Publicly known vulnerabilities fixed

There MUST be no unpatched vulnerabilities of medium or higher severity that have been publicly known for more than 60 days.

-> yes

Projects SHOULD fix all critical vulnerabilities rapidly after they are reported.

-> yes

The public repositories MUST NOT leak a valid private credential (e.g., a working password or private key) that is intended to limit public access.

-> yes

Static code analysis

At least one static code analysis tool (beyond compiler warnings and "safe" language modes) 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.

-> yes SonarCloud and Spotbugs

It is SUGGESTED that at least one of the static analysis tools used for the static_analysis criterion include rules or approaches to look for common vulnerabilities in the analyzed language or environment.

-> yes Sonarcloud

All medium and higher severity exploitable vulnerabilities discovered with static code analysis MUST be fixed in a timely way after they are confirmed

-> yes

It is SUGGESTED that static source code analysis occur on every commit or at least daily.

-> no weekly report

Dynamic code analysis

It is SUGGESTED that at least one dynamic analysis tool be applied to any proposed major production release of the software before its release.

-> no

It is SUGGESTED that if the software produced by the project includes software written using a memory-unsafe language (e.g., C or C++), then at least one dynamic tool (e.g., a fuzzer or web application scanner) be routinely used in combination with a mechanism to detect memory safety problems such as buffer overwrites. If the project does not produce software written in a memory-unsafe language, choose "not applicable" (N/A)

-> N/A Java is memory-safe

It is SUGGESTED that the project use a configuration for at least some dynamic analysis (such as testing or fuzzing) which enables many assertions. In many cases these assertions should not be enabled in production builds. [dynamic_analysis_enable_assertions]

-> no

All medium and higher severity exploitable vulnerabilities discovered with dynamic code analysis MUST be fixed in a timely way after they are confirmed.

-> N/A