Envoy external dependency policy

Author: htuch@google.com
Input from: michael@sooper.org, mattklein123@gmail.com, asraa@google.com
Status: Review
Last updated: 2020-12-3
This document is publicly shared with the Envoy community

Overview

The goal of this document is to formulate a policy for Envoy's external dependencies; what criteria must an external dependency meet to be incorporated in Envoy's build?

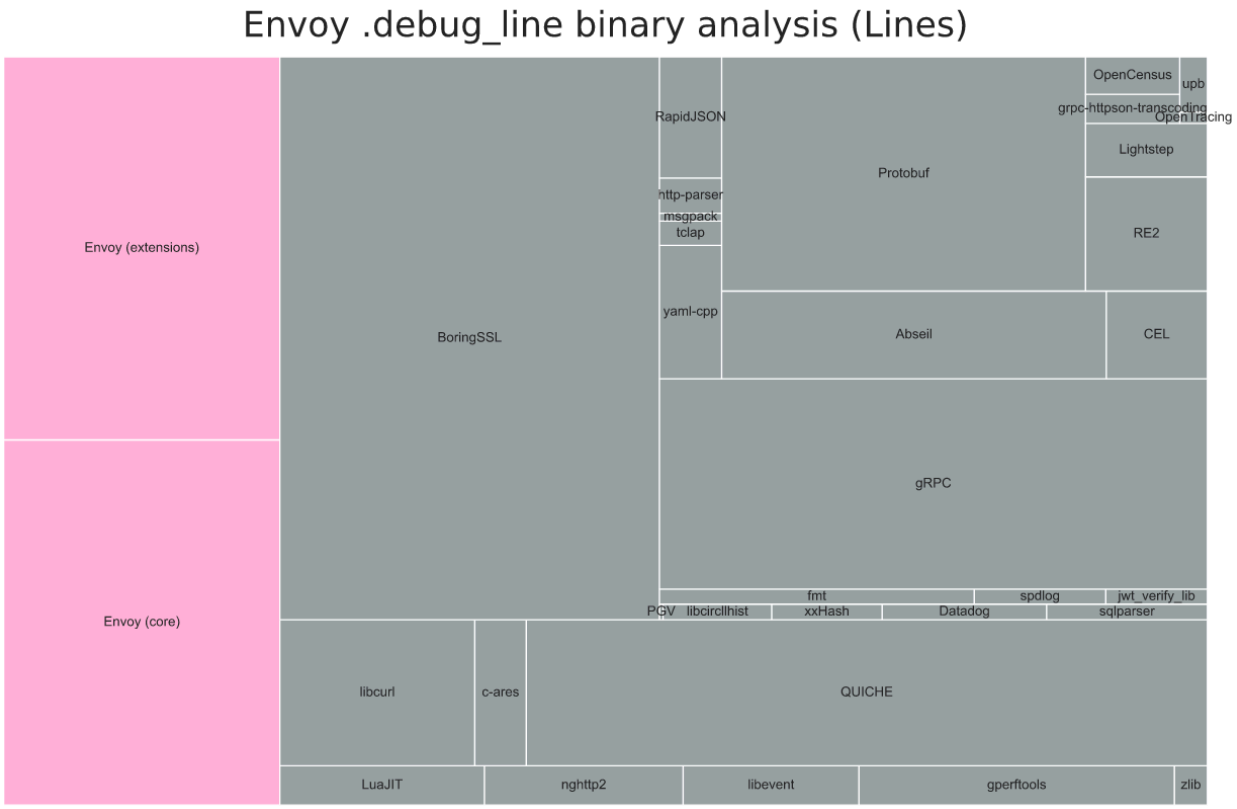
Until recently, we've had no stance on external dependencies or criteria for determining if a new external dependency is acceptable. Given the criticality of the supply chain in Envoy's overall security posture and a 3x growth in external dependencies in the past 3 years, it's desirable to have an explicit policy, ideally machine checkable/enforced, to ensure a high quality bar for new dependencies, to slow unnecessary growth and to address structural concerns (duplicated dependencies, known problematic dependencies).

Envoy's supply chain

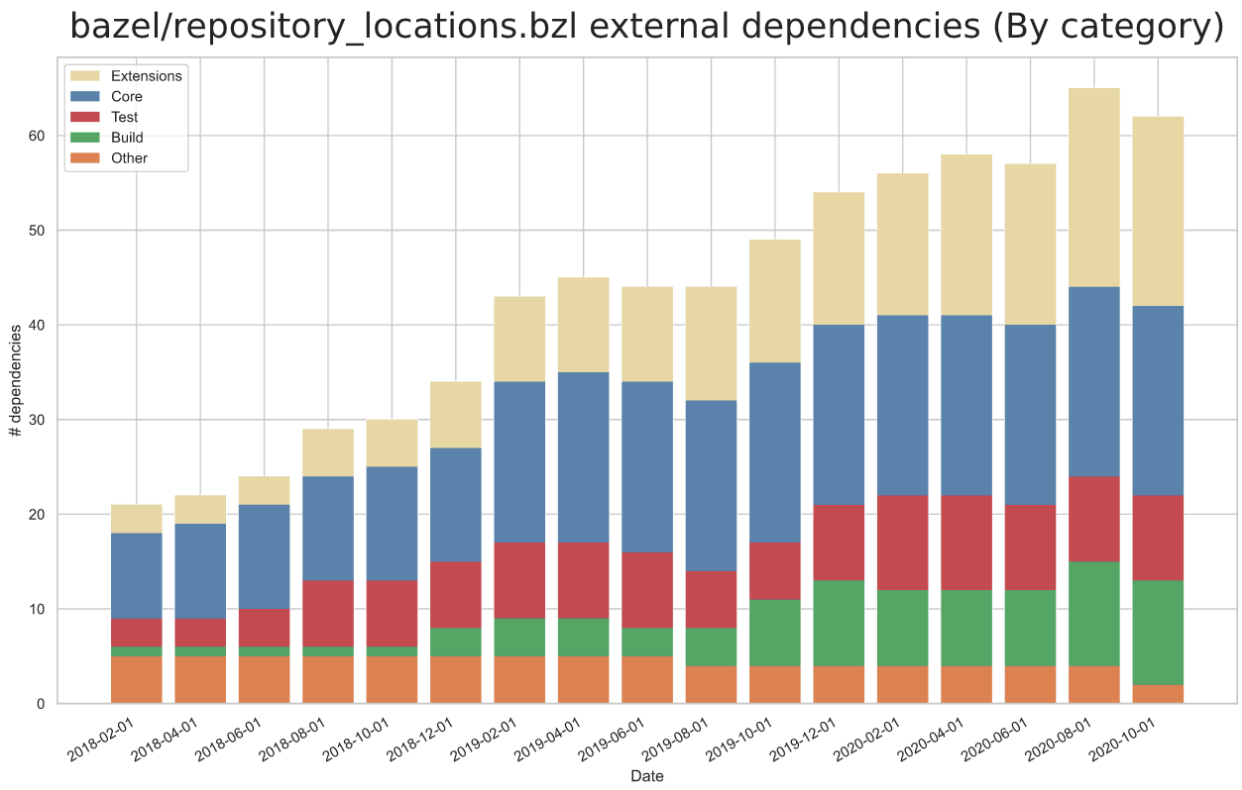
A recent survey of Envoy's supply chain is provided by the "Understanding, maintaining and securing Envoy's supply chain" EnvoyCon 2020 talk (slides, video). Key points on growth are recapped below.

A starting point is to observe that an Envoy binary is predominantly composed of code belonging to 3rd party dependencies. Not all of this code is exercised in practice, but it dwarfs

the Envoy-specific code.



We've seen 3x growth in these dependencies in the past few years. Even excluding test/build/miscellaneous dependencies, we now have ~40 external dependencies, roughly half of which are core and half in extensions. There is no single maintainer who can confidently speak to usage and project status of all these dependencies.



This growth is problematic, since it is challenging to keep track of:

- Why a dependency exists in Envoy and where it is used. Is it a critical data plane dependency, or something that is used with trusted inputs?
- How stale a dependency is relative to its upstream.
- Whether any known Common Vulnerabilities and Exposures (CVE) apply to Envoy's dependencies at the versions we consume.
- How closely the external dependencies follow best practices, e.g. code reviews, multi-party governance, two factor authentication (2FA), vulnerability disclosure process, regular release practices, etc.

We have made some progress recently on a number of fronts:

- Issue #10471 was opened to explore adding a policy to Envoy governing new external dependencies. This policy proposed in this document aims to fulfil this goal.
- We now track extended metadata in repository_locations.bzl that indicates a number of useful pieces of information listed below. These are validated using Bazel build graph analysis and with the help of Continuous Integration (CI) presubmit checks. A table of external dependencies is included in Envoy's documentation.
    - `use_category`: indicating whether a dependency is used in core vs. extensions, data vs. control plane, etc.
    - `release_date`: when the version Envoy used was released upstream. This is validated with the help of the GitHub API.
    - `project_{name,desc,url}`

    - `version:` Machine parsable version identifier.
    - `cpe`: project Common Platform Enumeration (CPE) (where available)
- We run a heuristic script hourly on AZP to scan against the latest CVE database, for dependencies where a CPE is available. This is a homebrew effort due to the lack of standard package management in C++/Bazel and surrounding tools.
- We have opened issues to track some evidently problematic dependencies:
    - Moving from http-parser for the HTTP/1 codec to llhttp (#5155). http-parser is deprecated and maintainerless, we have discovered security vulnerabilities in the past here.
    - nghttp2 (HTTP/2 codec) has been identified as problematic for security reasons. It lacks a strong review culture, has single developer project governance and an insufficient security vulnerability disclosure process.
    - Eliminating the zlib/zlib-ng duplicated dependencies (#13261).
    - Eliminating MoonJIT (maintainerless, duplicated) in favor of LuaJIT (#13539).
    - Replacing RapidJSON with nlohmann/json (#4705).
    - Removing Curl as a dependency (#11816).
- We have formulated an initial external dependency policy that covers the mechanics of dependency maintenance and some criteria to consider when evaluating a new dependency.
- A dedicated group of dependency shepherds is responsible for approving new dependencies and updates to existing dependencies.

The existing criteria that we informally work with are as follows:

- Does the project have release versions? How often do releases happen?
- Does the project have a security vulnerability disclosure process and contact details?
- Does the project have effective governance, e.g. multiple maintainers, a governance policy?
- Does the project have a code review culture? Are patches reviewed by independent maintainers prior to merge?
- Does the project enable mandatory GitHub 2FA for contributors?
- Does the project have evidence of high test coverage, fuzzing, static analysis (e.g. CodeQL), etc.?

Towards a new supply chain policy

A key problem with our existing policy is that it relies exclusively on subjective assessment, has no automation and is only weakly enforced. We would benefit from having the policy criteria appear explicitly as metadata and with additional tooling to enforce properties such as “dependencies without code reviews are not used on the core data plane”. The policy also does not distinguish between dependency usage, i.e. the same criteria applies no matter whether we are considering a dependency for processing gRPC configuration fetches or an HTTP codec.

A number of initiatives in the wider Open-source software (OSS) supply chain security arena have been working to formulate scorecards and methods for evaluating OSS projects for best practices. A brief summary:

- OSSF Security Scorecards: These are fully automatable OSS security best practice checks that run against a GitHub repository via the GitHub API. Examples include validating the existence of a security policy, signed releases, OSS-fuzz, pull requests for changes, etc.
- CII Best Practices: Set of criteria for Free/Libre and Open Source Software (FLOSS) projects. Self-certifying compliance providing a badge to attest. This provides an expansive list of criteria which cover everything from project website to cryptographic algorithms. Relevant criteria includes release versions/notes, CVE assignment, vulnerability process, static/dynamic analysis, test quality.
- OSSF metrics dashboard brainstorming: Collection of potential criteria for evaluating OSS project supply chain health with discussion on pros/cons.
- Our Software Dependency Problem (Russ Cox): Reflections on manual criteria for evaluating software dependencies, e.g. licensing, maintenance, CVE history, code quality, testing.

Moving beyond Envoy's existing informal considerations and manual evaluation, we can supplement the existing Envoy dependency policy with:

- Criteria expressed in the Envoy repository_locations.bzl metadata.
- Machine checking/enforcement of compliance, e.g. via GitHub API.
- An explicit gating function (e.g. code reviews via a PR trail are mandatory), with a process for making exceptions (e.g. code reviews are known to take place in some other forum X).

We focus the dependency policy on the data plane below, since this is in the scope of Envoy's threat model, while the control plane is generally trusted. One result of this focus is that only Envoy's C/C++ dependencies are considered in scope, since our Go/Python dependencies are typically used in build/test. We also deliberately exclude tooling such as Bazel build rules and the compiler (Clang and ecosystem); while these are clearly potential attack vectors, the shared fate with other projects and low rate of historical CVEs suggests that our efforts are best focussed on linked data plane code.

Proposal

Dependencies must be git canonical or mirrored

Most of Envoy's dependencies are GitHub-canonical or mirrored on GitHub. Unfortunately, a handful (e.g. QUICHE, BoringSSL-FIPS, V8, googleurl) exist in Google Cloud Storage tarball snapshots. This is highly problematic, since we are unable to easily bump versions and have limited ability to validate the dependency criteria below using standard tooling. All new C++ dependencies MUST be git-based and directly reference the git repository, with no manual

update steps, unless there is a compelling reason beyond expediency provided. Mirrors are acceptable providing they automatically sync against some upstream source. GitHub is the strong preference for hosting since it provides a common project API for compliance checking, e.g. validated how far from the last released version we have drifted.

Criteria for new Envoy dependencies

Building on ossf/scorecard, we will develop tooling to validate the following criteria, and augment with manual checks. These apply to all dependencies with use category:

- `dataplane_core`
- `dataplane_ext`
- `controlplane`
- `observability_core`
- `observability_ext`

Criteria	Requirement	Mnemonic	Automated	Weight	Rationale
>=2 contributors responsible for a non-trivial number of commits	MUST	Contributors	Yes - custom	Normal	Avoid bus factor of 1
Tests run in CI	MUST	CI-Tests	Yes - OSSF Scorecard	Normal	Changes gated on tests
Code review before merge	MUST	Code-Review	Yes - OSSF Scorecard	Normal	Consistent code reviews
PRs for all changes (for GH repos)	MUST	Pull-Requests	Yes - OSSF Scorecard	Normal	Audit trail of reviews
CVE history appears reasonable, no pathological CVE arcs	MUST	SoundCVEs	No	**High**	Avoid dependencies that are CVE heavy in the same area (e.g. buffer overflow)
High test coverage (also static/dynamic analysis, fuzzing)	MUST	Test-Coverage	No[1]	Normal	Key dependencies must meet the same quality bar as Envoy
No duplication of existing	MUST	NoDuplication	No	**High**	Avoid maintenance

[1] This is the most time consuming criterion to verify; projects do not commonly report test coverage or have readily accessible coverage reports. Fuzzing is easier to understand when an OSS-fuzz integration exists.

dependencies					cost of multiple JSON parsers etc
Cloud Native Computing Foundation (CNCF) approved license	MUST	License	Yes	**High**	
Security vulnerability process exists	MUST	SecPolicy	No	**High**	Lack of a policy implies security bugs are open zero days
Dependencies must not substantially increase the binary size unless they are optional (i.e. confined to specific extensions)	MUST	BinarySize	Yes	**High**	Envoy Mobile is sensitive to binary size. We should pick dependencies that are used in core with this criteria in mind.
Envoy can obtain advanced notification of vulnerabilities	SHOULD	SecPolicy-Compat	No	**High**	Coordinated security releases possible, but most dependencies do not feature this
Do other significant projects have shared fate by using this dependency?	SHOULD	SharedFate	No	**High**	Increased likelihood of security community interest, many eyes.
Releases (with release notes)	SHOULD	Releases	Semi	Normal	Discrete upgrade points, clear understanding of security implications. We have many counterexamples today (e.g. CEL, re2).
Commits/releases in last 90 days	SHOULD	Active	Yes - OSSF Scorecard	Normal	Avoid unmaintained deps, not compulsory since some code bases are

					“done”

Dependencies that fail any “MUST” criteria will require discussion amongst maintainers. When exempted, a rationale will be added to repository_locations.bzl metadata tracking the reasoning behind exceptions. “SHOULD” criteria boost confidence in a project’s suitability; they are worth evaluating and tracking but do not block.

New dependencies should be competitively evaluated against alternative candidates where available best on this criteria.

Dependencies that change use category should be re-evaluated against this criteria.

Realistically, we expect that many dependencies (including those that exist today) will require exemptions. The objective is that we are deliberative about these exceptions and avoid dependencies that require a gross number of them. This list of exceptions will provide a set of criteria that we can use when evaluating competing dependencies and allow us to work constructively with projects to improve their compliance (e.g. by filing upstream issues, reaching out to CNCF, engaging contractors).

This policy will add friction to adding new dependencies; in part this is desirable, since we want to ensure that the non-trivial risk that dependencies add to Envoy carries with it a responsibility to think seriously about dependency quality. At the same time, we aim to be pragmatic; if a dependency is required for essential functionality and no better alternative exists, maintainers may exercise wide discretion.

An example evaluation of some of Envoy’s existing data plane dependencies against this criteria. Each evaluation required roughly 20 minutes of time on average.

	BoringSSL[2]	**Protobuf**	**c-ares**	**xxhash**	**zlib**
Contributors	Yes	Yes	Yes	Yes	Maybe (Claims 2 contributors, git history indicates only 1, huge backlog of PRs)
CI-Tests	Yes, but not on GitHub, see here.	Yes	Yes	Yes	No
Code-Review	Yes, but not on GitHub, see here.	Yes (but fails on scorecard)	Commits without review trail	Commits without review trail	No
Pull-Requests	N/A	Yes	No	~1k commits,	No

[2] This repository was not Github canonical, so OSSF scorecard automation failed.

				~300 PRs	
Active	Yes	Yes	Yes	Yes	No (not since 2017)
Test-Coverage	65%+ coverage, fuzzing, *SAN sanitizers	Unit tests exist and cover major functionality. No coverage scripts, low test:source SLOC ratio. Fuzzing exists.	88.7% test coverage, fuzzing, *SAN sanitizers, valgrind	Some tests/benchmarks, no coverage CI or scripts. No OSS-fuzz integration, but Envoy has good coverage (~60%, we don't use all of xxhash)..	67% coverage, fuzzing
NoDuplication	Yes	Yes	Yes	Yes	No (zlib-ng)
Releases	No	Yes	Yes	Yes	Yes
SecPolicy	No, but disclosure form exists.	No	Yes	No	No
SecPolicy-Compat	No	No	No[3]	No	No
SharedFate	Yes	Yes	Yes	Yes	Yes
License	Yes	Yes	Yes	Yes	Yes
SoundCVEs	Yes	Yes	Yes	N/A	Yes[4]

An evaluation of the OSSF Scorecard against all of Envoy's eligible criteria is available here.

Extend repository_locations.bzl metadata

The above criteria will be captured in repository_locations.bzl metadata. Each item will be assigned a short mnemonic and a list of failing criteria added to each dependency.

CI enforcement

Dependencies will be required to declare the above metadata when used on the data plane or control planes. Tooling run in CI presubmits will validate metadata when automatically checkable (e.g. with OSSF Scorecard) and the existence of exemption rationale for violations.

[3] Similar to nghttp2, c-ares has adopted a vulnerability disclosure policy based around https://oss-security.openwall.org/wiki/mailing-lists/distros. This is a distro-first model that doesn't have room for static binary build models commonly used with Envoy.
[4] Longish history of gnarly CVEs, but not many since 2016.

Existing dependencies

Existing dependencies will be incrementally covered by this policy. When they are found in violation, issues will be filed in the Envoy and upstream repositories. We anticipate prioritizing core dependencies and those dependencies used by extensions tagged as stable and robust in untrusted environments.

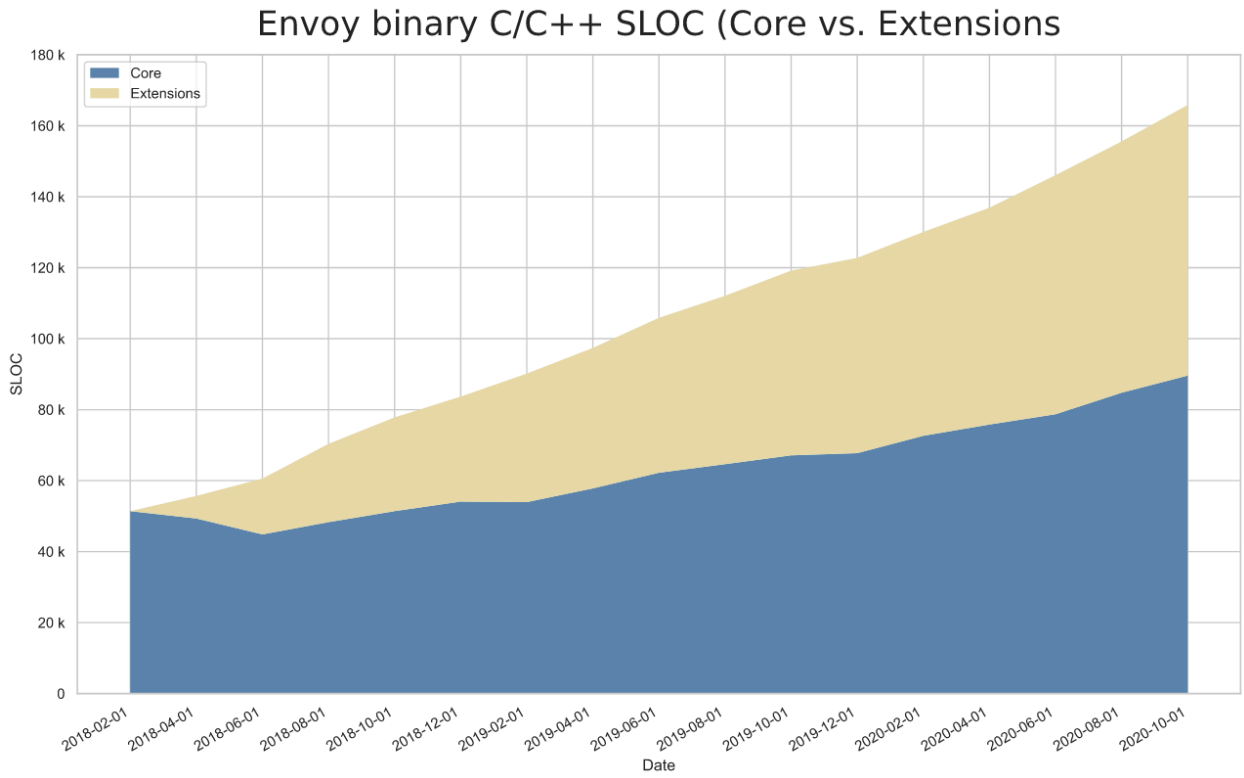
Dependency maintenance

The addition of a new dependency will imply a willingness to take on support of this dependency in Envoy. Each dependency will be tracked in a DEPENDENCY_OWNERS.md file. Issues will be filed against dependency owners when CI tooling detects stale dependencies (already possible via the release date metadata). For non-core dependency use, if dependency owners do not maintain their dependencies, the dependency and any related extension will be removed.

Dependency shepherds will continue to be responsible for enforcing this policy.

Non-core extension repository separation

One of the significant contributors to Envoy's continued dependency and SLOC growth is C++ extensions.



Many of these extensions have limited support, maintenance and production use. Some are labeled alpha or have unknown security postures (see this list). One reason for the growth in

C++ extensions is that Envoy has an unstable API internal C++ API, and upstreaming an extension ensures build maintenance as this API changes.

To structurally eliminate the dependencies implied by these extensions, we can ask that new external dependencies that are immature, not widely used or that are not in some sense “core” to Envoy, be added to an independent extension repository. Details are beyond the scope of this policy document and discussion is tracked at envoyproxy/envoy/issues/14078.

Bazel improvements

Some of the capabilities we desire around metadata (e.g. CPE, release date tracking) are traditionally performed by package managers in languages with standardized package managers, e.g. NPM, Go, PyPi. This does not exist in the C++ world, the closest we have is Bazel. There is an effort underway to add module support to Bazel, ideally we can reduce the number of places we have ad hoc tooling (e.g. the CVE scanner, stale release tracking) by influencing this effort with Envoy as a C++ exemplar project.

GitHub improvements

We’d like to be able to obtain the following from GitHub in the future:

- Dependency 2FA/MFA usage. This is currently only visible to project maintainers. We have reached out to GitHub and they suggest working with the OSSF Security Metrics project. This would be used to add a 2FA/MFA criteria and drive conversations with dependencies on adopting best practices to reduce contributor and maintainer account hijacking risk.