wg-securing-software-repos

Principles for Package Repository Security

Authors: Jack Cable (CISA), Zach Steindler

Last updated: Feb 2024

Background

The Securing Software Repositories Working Group (WG) of the OpenSSF has identified a taxonomy of package repositories and a set of principles for their security capabilities. This is intended to offer a set of best practices that package repositories should strive to adhere to.

By package repository (sometimes also called package registry) we're referring to the service that stores and indexes packages, enabling users and clients to download packages. Often users and clients use a CLI tool to download packages, and some ecosystems have multiple popular CLI tools in use. Most of these security capabilities focus on the package repository, but there's also a section on security capabilities the CLI tooling can implement.

We include the below taxonomy because not all security advice applies to all package repositories. For example, https://proxy.golang.org/ does not manage user accounts, and so has no need of documented account recovery.

The roadmap of security capabilities can then be used by package repositories to assess gaps, put together fundable improvement lists (like Python Packaging WG), or write specific grant proposals that reference this guidance.

This is v0.1 of this document. You can give feedback for v0.2 at https://github.com/ossf/wg-securing-software-repos/pull/38.

Taxonomy of Package Repositories

Security capabilities will differ based on the services that the package repository offers. For instance, if the package repository has user accounts, it will need to enforce authentication securely. In this section, we lay out the various relevant aspects of package repositories.

Does the package repository …

… have user accounts?

- Yes: then you have to manage things like authentication and account recovery. For example, PyPI
- No: for example, {index, proxy, sum}.golang.org

… accept built packages, do builds on behalf of users, or only host source code?

- Accept built packages: for example, npm
- Builds on behalf of users: for example, Homebrew
- Only host source code: for example, {index, proxy, sum}.golang.org

Security Capabilities of Package Repositories

We define 4 levels of security maturity:

- Level 0 is defined as having very little security maturity.
- Level 1 is defined as having basic security maturity, which includes supporting basic security features like multi-factor authentication (MFA) and allowing security researchers to report vulnerabilities. All package management ecosystems should be working towards at least this level.
- Level 2 is defined as having moderate security, which includes actions like requiring MFA for critical packages and warning users of known security vulnerabilities.
- Level 3 is defined as having advanced security, which includes actions like requiring MFA for all maintainers and supporting build provenance for packages. This level is more aspirational, especially for smaller package management ecosystems.

These levels are split into four tracks enumerated in the below sections: Authentication, Authorization, General Capabilities, and CLI Tooling. All tracks may not apply to all package repositories, as detailed above. Having these tracks allows package repositories to assess their security across various dimensions.

Authentication

Applies to: package repositories that have user accounts.

- Level 1
  - The package repository requires users to verify their email address. This ensures that if a user loses access to their account, they are able to recover it.
  - The package repository documents their account recovery policy.
  - The package repository supports strong multi-factor authentication (MFA) via, at minimum, TOTP. Solely offering weaker forms of MFA such as SMS, email, or phone call-based MFA would not meet this requirement.
  - The package repository notifies maintainers via email for critical account security changes, such as password changes or disabling multi-factor authentication. This helps

users detect if their account has been compromised.
    - The package repository implements account security measures like brute force prevention (even with 2FA attempts)
- Level 2
    - To prevent domain resurrection for account takeover via the recovery process, the package repository detects abandoned email domains. This may look like doing a WHOIS lookup on all registered email domains, and removing the ability to recover an account via an email domain that has been abandoned.
    - The package repository supports MFA via phishing-resistant means such as WebAuthn.
    - The package repository requires MFA for packages deemed critical (e.g. top 1% of packages by downloads).
    - The package repository integrates with public leaked credential databases such as Have I Been Pwned to detect when users are using known leaked credentials upon login. If a user has a leaked credential, detected upon login, the package repository prompts them to change it. The package manager checks passwords at registration and prevents users from registering with known leaked credentials.
- Level 3
    - The package repository supports passwordless authentication, such as passkeys.
    - The package repository requires MFA for all maintainers.
    - The package repository requires phishing-resistant MFA for packages deemed critical (e.g., the top 1% of packages by downloads). If possible, the package repository supplies security keys to these maintainers.

Authorization

Applies to: package repositories that have user accounts and accept built packages.

- Level 1
    - The package repository allows maintainers to provision API keys scoped to specific packages. This allows maintainers to maintain packages via automated workflows without needing to provide their account password.
    - API tokens are prefixed with a package repository-specific identifier.
- Level 2
    - The package repository supports role-based access control (RBAC) for maintainers, allowing separate roles for managing users and publishing packages.
- Level 3
    - The package repository provisions short-lived API tokens via an OpenID Connect (OIDC) token exchange to prevent the need to provision long-lived API keys. Example: trusted publishers.
    - The package repository API tokens are integrated into common third-party secret scanning programs. This typically requires using a known prefix or pattern to reduce false positives. Furthermore, the package repository provides a secret-reporting

endpoint for these secret scanning programs to report leaked credentials, which automatically revokes valid credentials.
    - The package repository supports providing build provenance for packages.

General Capabilities

Applies to: all package repositories

- Level 1
    - The package repository has published a vulnerability disclosure policy allowing security researchers to identify and report vulnerabilities affecting the package repository, and receive legal safe harbor when doing so within the requirements of the policy. The package repository should strive to remediate reported vulnerabilities within 90 days.
    - The package repository takes steps to prevent typosquatting attacks around package names. This may include namespacing, detecting similar names, or other actions.
- Level 2
    - The package repository has an unpublish policy to prevent specific versions of a package from being replaced, and, if a package is deleted, prevent others from reusing the same package name and version.
    - The package repository allows users to report a package as suspicious or malicious via the registry UI, tooling CLI, and via API.
    - The package repository takes steps to detect malware, including scanning for known malware code snippets and hashes.
    - The package repository warns of known security vulnerabilities in dependencies in the package repository UI.
- Level 3
    - Undergo periodic security reviews of package repository infrastructure (and optionally publish remediations once they are complete)
    - The package repository publishes an event transparency log to enable detection of anomalous behaviors (e.g. replicate.npmjs.com).
    - Advisories for malicious packages are published using a standardized machine readable format, such as using the OSV schema and aggregation provided by the OSV service (e.g. here).

CLI Tooling

Applies to: all package management ecosystems, although here we are primarily focused on the CLI tooling instead of the registry itself.

- Level 1
    - The CLI allows installing dependencies that are pinned based on hash, version, etc.
- Level 2

    - The CLI warns of known security vulnerabilities in dependencies when installing packages.
- Level 3
    - The CLI has functionality to produce SBOMs.
    - The CLI has functionality to, where possible, automatically remediate known vulnerabilities in dependencies by upgrading them.
    - The package repository is able to reduce false positives of identified vulnerabilities using static analysis to understand whether a vulnerable code path is actually reachable. (e.g. Golang's vulncheck feature).

Resources / Inspiration

- https://github.com/psf/fundable-packaging-improvements/blob/master/FUNDABLES.md
- https://github.com/ossf/wg-securing-software-repos/blob/main/survey/2022/results.csv
- https://docs.npmjs.com/threats-and-mitigations
- https://www.microsoft.com/en-us/research/uploads/prod/2018/03/build-systems.pdf (as a framework for comparing options in a problem space)

Note: CISA does not endorse any commercial entity, product, company, or service, including any entities, products, or services linked within this document. Any reference to specific commercial entities, products, processes, or services by service mark, trademark, manufacturer, or otherwise, does not constitute or imply endorsement, recommendation, or favoring by CISA.
