January 2023

Federated and decentralized metadata system

Philippe Ombredanne
NexB

Federated and decentralized metadata system.

And its application to open source software package origin, license and vulnerabilities metadata sharing and curation.

Abstract

This publication describes a new system and approach that supports decentralized and federated metadata aggregation and sharing to remove data silos, distribute control, improve availability and enable distributed and social review over this metadata.

Such a system is designed to effectively allow large scale sharing and exchange of metadata using open systems and to support the efficient curation, and peer review of the metadata through self-organizing networks of reputable or trusted peers based on social networking.

This system supports decentralized and distributed metadata collection and federated multi-aggregation of the metadata to avoid single points of control, avoid walled gardens and exclusive gated access, or the lack of availability due to system failure or shutdown of a central system with a goal to improve availability, resilience of the system while lowering the overall effort and compute cost required to achieve these results.

Keywords

federated, open source, metadata, curation, ActivityPub, FOSS, license, vulnerability, decentralized, OSPO, master data management, security, software composition, SBOM

Context

Metadata (and master reference data) are central to many software systems. The metadata could be about software itself such as software package and dependency manifests, or about music records or books. The metadata are typically scattered in several places and stored in different formats making it difficult and costly to build efficiently aggregated metadata collections. For example, in the software and software security domains, there are aggregators such as SoftwareHeritage, Repology, NVD, Libraries.io and ClearlyDefined; or there are aggregators such as Discogs, Musicbrainz, or Gracenote in the music domain These aggregated databases are centralized and are either proprietary, or curated in an opaque non-traceable manner, or too big to share.

Problem

More recently, a vulnerability database such as VulnerableCode [14] or a software metadata knowledgebase such as purldb [13] are essentially reproducing the status quo whereby they aggregate and normalize metadata in a centralized database that is then offered with an API access acting as a gate. These systems (and most current metadata systems) transform and combine several small scattered metadata silos in one bigger metadata silo with a centralized access. There is value and convenience for users in centralized access, but this comes at the expenses of surrendering control over the metadata to a single entity with its associated problems: higher costs, lower privacy, lower availability and resilience, loss of control, lack of customization, opaque decisions about depth and breadth of metadata made by faceless actors with non-

disclosed incentives, and storing many less useful or less relevant records for a given usage. Even when they redistribute whole database snapshots as open data (like is done with VulnerableCode or purldb), these systems do not have a straightforward and efficient way to disseminate and share their data beyond whole databases when any user may not need access to the whole metadata collection but only to a subset that is useful in a usage context.

As a software domain example, a central database may contain hundred thousand records for public GitHub repositories published by computer science students as programming assignments. These are interesting for students, but are mostly useless otherwise and highly unlikely to ever be reused: yet such records pollute existing databases and their users cannot elect not to include these in their queries.

And when multiple metadata database instances co-exist separately, each instance ends up re-doing the same data collection, processing, indexing work as all other instances. When data needs review and is manually updated for correction and curation in an instance, this data cannot be shared back easily or at all. Data that cannot be shared back and synchronized will eventually diverge. As an example in the software vulnerability domains, a tool like vulntotal [15] highlights that it is common for competing vulnerability databases to disagree: each database may report the same vulnerability with different affected software packages or the same package but with different affected and fixed versions.

Because of the difficulty to share (or the disincentives when the investment was significant), there are ever more systems that collect and aggregate data silos to re-create new bigger data silos and there is a significant duplication and waste of compute and human resources to create and maintain these metadata databases. Creating new silos is not a satisfying solution and contributes to further fragmentation and locking this metadata with several problems:

Inconsistent formats: These larger aggregated metadata silos are using different set of conventions, data formats, schemas and APIs even when they store metadata for the same domain.

Too big to share: Furthermore, the large size of these aggregated databases make them impossible to share as too big to replicate. This makes these aggregators important gatekeepers of any access to this data.

Metered or gated access: Effectively, these aggregators are turning silos of openly accessible data into new walled gardens of proprietary data only accessible with a "narrow straw" e.g., a throttled or metered API whether or not for a fee.

Incomplete: The data is also typically either incomplete or out of date because it is collected in filtered batches rather than on demand for specific records.

No private access: A central remote database cannot guarantee the confidentiality and privacy of who is accessing what data, which is a problem for several industrial, government and defense actors. The fact that these databases are too big to share prohibits privacy-preserving offline or "air gapped" usage and exists not only with proprietary and commercial aggregators but also with aggregators that are claiming to be open data providers.

Availability and resilience: These centralized systems are more prone to failure and availability problems; when controlled by a single country or organization such a system may be shutdown or de-funded with little advanced warning.

Lack of dissemination: The metadata are not universally available and accessible even though they are typically needed by all their users.

Redundant collection and processing: Each aggregator and user end up doing redundant data collection

where multiple organizations are re-generating and re-collecting the same data on a daily basis using the same tools, independently from each other in their own silos and not sharing these outputs. This is a waste of computing (and energy) resources.

These are redundant because many of the processing tasks are not unique to a business or organization: for instance in software domain, company A and B may be both scanning package C for license, origin and vulnerabilities with the same version of tool D using the same settings and these will therefore see the exact same scan results. Ideally such a scan should only need to run once and then cached and shared for everyone else to reuse. Or when the same scan has been launched multiple times independently, each output could be shared and help validate the integrity of the results.

Redundant review and curation: Each user of these data tends not only to run the same analysis tools on the same metadata but also they each review and curate the same data independently to reach the same or similar conclusions. Or in some cases, different actors of the same domain may reach different conclusions from the same facts, but can never confront these differences: these disconnected reviews of the same metadata independently without ever sharing these reviews can never benefit from the concordance or conflict between these curations.

For instance in the software domain, multiple organizations will scan the same software packages for origin, license and vulnerabilities with the same tools and will review and curate these scans to assert eventually the same conclusion on the facts with regard to provenance, licensing and applicable vulnerabilities on these software packages. In some cases, they may disagree and reach different conclusions on complex licensing.

Tragedy of the commons: But still in the software domain, initiatives in the industry at large to share this information have not taken up for various reasons like fear of leaking data, worry about liabilities ("what if we provide these results and they turn out to be wrong?"), financial arguments ("why should we pay for doing this work so X doesn't have to?"), wrong scope (full legal analysis vs. just providing the scanner results). It is another example of the "tragedy of the commons".

Solution

The solution described here is a new approach where the metadata is **distributed and decentralized over multiple versioned data stores** where the data provenance, evolution and eventual curation is fully traceable with a **complete and auditable history**. As an overlay to these multiple data stores, there is a **federated system to advertise and publish metadata updates** and a mechanism for client users and systems to **subscribe** to these updates as well as **participate in the review, validation and curation** of the accuracy of these metadata.

Each metadata consumer can selectively aggregate and receive updates for the subset of the metadata they care for, accept curation from actors they trust -- and if they elect -- share back contributions in the form of new data analysis, reviews, and curations.

Architecture

Note: To illustrate the proposed approach, this publication will use the open source software metadata domain (to document the provenance, licensing and security of open source software); but this also applies to any reference metadata in other domains than software, such as books, movies or music records.

This proposed design for this new approach will use a combination of three critical components:

- **multiple independent user systems consuming and producing metadata.** These systems would typically be a local private installation of a software vulnerability database or software dependency and composition management system in the software domain. Each of these systems could aggregate in their local database the data from the system described below: for instance, they could fetch the curated licensing information of a software package, or the actual version ranges affected by a security bug. They could also contribute the output of tool runs (e.g., software scans) to the commons and have their users participate in the shared metadata review and curation.

- **multiple distributed and versioned document databases** to store structured metadata in a decentralized way.

One of the possible implementations would be to use a version control system such as Git [3] or Mercurial [4] for this purpose. Such systems provide a suite of comprehensive mechanisms to record a complete traceable history, authenticating contributors and contributions; they are decentralized by design and have enabled large scale content sharing and multi-way synchronizations.

- **an advertising, dissemination, notification and sharing federated system** such that interested users can discover and subscribe to metadata updates; and provide feedback, reviews, discussions and curation of these metadata.

One of the possible implementations would be to use the W3C ActivityStreams [2] and ActivityPub [1] standards. ActivityPub description is: *The ActivityPub protocol is a decentralized social networking protocol based upon the ActivityStreams 2.0 data format. It provides a client to server API for creating, updating and deleting content, as well as a federated server to server API for delivering notifications and content.*

This usage of ActivityPub for other purposes than social networking is not new: even though ActivityPub was originally designed for decentralized social networking, it can be used and extended for other purposes such as federated forge [6] [7]. An additional benefit of using ActivityPub would enable the system to be plugged into the fediverse [5].

In this model, each metadata entity or logical group of metadata entities (i.e., each software package or each software project in our example domain):

- have their own dedicated versioned data store. For instance in the software metadata domain, each software package or project would get **its own dedicated metadata Git repository** that would store metadata not code, as well tool outputs (such as scans or SBOMs [9]). There, time series of repeated data collections or scans, as well contributions or curations are stored with full traceability and expose the metadata for decentralized replication using well known Git protocols and tools. This means concretely that there would be a large number of such distributed data repositories (potentially a few million) where each could store the metadata for origin, provenance, licensing, quality and security of all versions of one package or project (or small group of packages) together with the output of tools used to perform automated analysis.

- have their own dedicated publish/subscribe account on a federation of ActivityPub servers. For instance in the software metadata domain, **each software package or project would get its own ActivityPub** account typically identified by a package-url [8]. This account would **share and advertise updates to the package** and project such as the release of new versions, the availability of new tool scans, the discovery of new vulnerabilities, the contribution of new or updated curations and reviews, etc. with pointers to the versioned data store where the actual metadata are stored.
When a software package becomes its own **fediverse [5] "user"**, an ActivityPub-based solution would support not only metadata updates advertising, but also the important social interactions to have humans (and

possibly tools) effectively review and discuss these reviews and curations to reach a consensus when needed, for instance in the case of ambiguous licensing, or to qualify the context in which a given security issue may apply. Each package metadata could be "followed", "liked" and commented. Such an approach can also capture diverse discussions; or can capture data that are not present in the original data, enriching the data with another layer.

Application Examples

Software packages origin and license

In this example, a software team uses an internal system to track and vet the open source packages reused in their systems to validate their origin and license. When they use a new package, they will first lookup for a fediverse user with the package-url of this package. This could happen using a web interface or could be mediated by their system interface using ActivityPub APIs.

In the account of the software package identified by its package-url, they will find actual summaries on the origin and license for this package as reviewed by other users and pointers to detailed metadata and detailed scans in the package Git repository with a complete history. They can decide to trust or not these curations based on the posted reviews, shares and "likes" and who did these.

In another scenario, after running scans locally that identified package dependencies used in their system, they can directly request to lookup the detailed curated information from the fediverse for each package-url to enrich the data collected with their scan. The inclusion of published metadata may enhance the prominence, reputation or trust attached to this piece of data and that of the users that contributed it. The trust could be based on the fact that a review was contributed by a user account that is "followed". For instance, groups may follow each other and therefore trust their respective contributions and curations: these could be a network of peer OSPOs (open source program offices) in a large organization or the participants in an industry consortium.

When the package-url does not exists yet as a fediverse account, they would create (either directly or mediated through their system) this new software package account and contribute the scan in a new public Git repository and contribute their scanner runs for this package as well as their review and curation of the license and origin. From this point forward, other users may benefit directly from these scans and reviews like described above in the first example. They may also discuss, like and share these reviews and curation: this is an eminently social activity and will be well supported by a social networking application.

Software packages vulnerabilities

In this example, a security team is responsible to track new vulnerabilities in the open source software packages that are used in their systems whether custom or third-party software. They know the inventory of software package origins from provided SBOMs or scans.

They subscribe to the account of each package by package-url, either using a standard ActivityPub server web interface, or through the integration in their SIEM [10] system or a threat sharing and management system such as MISP [11].

They receive notifications when a new vulnerability is published that impacts one of their package, or when there is a new package release that fixes a security bug or that is still impacted by known security bugs.

They then review the new vulnerability information and may validate the package vulnerable version range, enhancing this metadata reputation. They also post a comment that explains how to work around the vulnerability in a specific context with a simple configuration change without requiring a software update: this comment becomes available to all users of this package and can help mitigate and remediate the issue faster and more efficiently.

In a variation or refinement of this example, each published software vulnerability could get its own ActivityPub account, typically identified by CVE [12]. Here security researchers could collaborate and each contribute comments, evaluate the severity and share mitigation and remediation instructions, as well as helping determine the effective package version ranges that are affected by a security bug and which ones are not.

References

[1] https://www.w3.org/TR/activitypub/
[2] https://www.w3.org/TR/activitystreams-core/
[3] https://git-scm.com/
[4] https://www.mercurial-scm.org/
[5] https://en.wikipedia.org/wiki/Fediverse
[6] https://github.com/forgefed
[7] https://lab.forgefriends.org/forgefriends/forgefriends
[8] https://github.com/package-url
[9] https://en.wikipedia.org/wiki/Software_supply_chain
[10] https://en.wikipedia.org/wiki/Security_information_and_event_management
[11] https://www.misp-project.org/
[12] https://cve.mitre.org/
[13] https://github.com/nexB/purldb/
[14] https://public.vulnerablecode.io/
[15] https://github.com/nexB/vulnerablecode/tree/main/vulntotal

Author

Philippe Ombredanne, nexB Inc. pombredanne@nexb.com