Trusted Publishers for All Package Repositories

Trusted Publishers is a new authentication mechanism using the OpenID Connect standard (OIDC) to exchange OIDC tokens for short-lived and tightly scoped API tokens for authenticating with package repository publishing APIs. Using short-lived API tokens removes the need to share long-lived and potentially highly privileged API tokens with external systems when publishing software. This capability is primarily beneficial for hosted, automated publishing workflows.

Why Trusted Publishers?

The goal of Trusted Publishers is to reduce the need for long-lived tokens or credentials to be shared with external systems when authenticating with package repositories. Managing the lifecycle of long-lived tokens poses additional security and management challenges for package indices and maintainers.

After configuration, Trusted Publishers require little to no user maintenance, providing an ergonomic and secure publishing experience for users, since they no longer need to manage long-lived secrets manually. Trusted publishers also allow maintainers to centralize authorization in a machine/workload identity, rather than additionally requiring managing users and authentication in the package repository.

Trusted Publishers are popular with users when they're available, for example PyPI added support for Trusted Publishers in April of 2023 and has since seen over 13,000 projects voluntarily adopt Trusted Publishers.

Trusted Publishers ensures a package is coming from a specific CI system, workflow, hosted machine or build pipeline, reducing the window for exfiltration abuse, unlike a long-lived API token.

For some Trusted Publishing providers, Trusted Publishers allow binding verifiable metadata like the source repository URL to a published artifact, allowing package repositories to avoid "Star-Jacking" and similar attacks to confuse users about the trustworthiness of a project.

Trusted Publishers are ideal for package repositories that accept user-built packages, like PyPI and RubyGems, as opposed to package repositories that have centralized build infrastructure like Homebrew.

Trusted Publishers pairs well with other technologies such as SLSA build provenance, as it is built on the same underlying technology, the OIDC standard.

Package repositories which don't host separate artifacts (such as pkg.go.dev) don't require authenticating with the repository, thus Trusted Publishers isn't applicable.

How Trusted Publishers works

Trusted Publishers allow a package repository like the Python Package Index (PyPI) to authenticate an identity from an Identity Provider (IdP) like GitHub Actions or GitLab Pipelines, and using a technology called OpenID Connect (OIDC).

Trusted Publishers works by requiring users to pre-configure a trust policy on the receiving package repository which will be later used as part of authenticating a publish request. When new artifacts are being published to the registry, the OIDC token (implemented as a JWT token) will be verified by checking the JWT claims and signature against the IdP's JSON Web Keyset (JWK) and checked to ensure the OIDC token's issuer and claims match the pre-configured trust policy for the package.

The trust policy doesn't necessarily need to have a *direct* link to a package, for example some package repositories may want to assign trust policies to package namespaces, the trust policy determines whether a given package can be published by a given workload identity.

The JWKs are discovered using the well-known OpenID Connect Discovery URL which is based on the JWT's issuer ("iss") claim. For example, GitHub's "iss" claim is "https://token.actions.githubusercontent.com" and thus it's well-known OpenID Connect Discovery URL is "https://token.actions.githubusercontent.com/.well-known/openid-configuration".

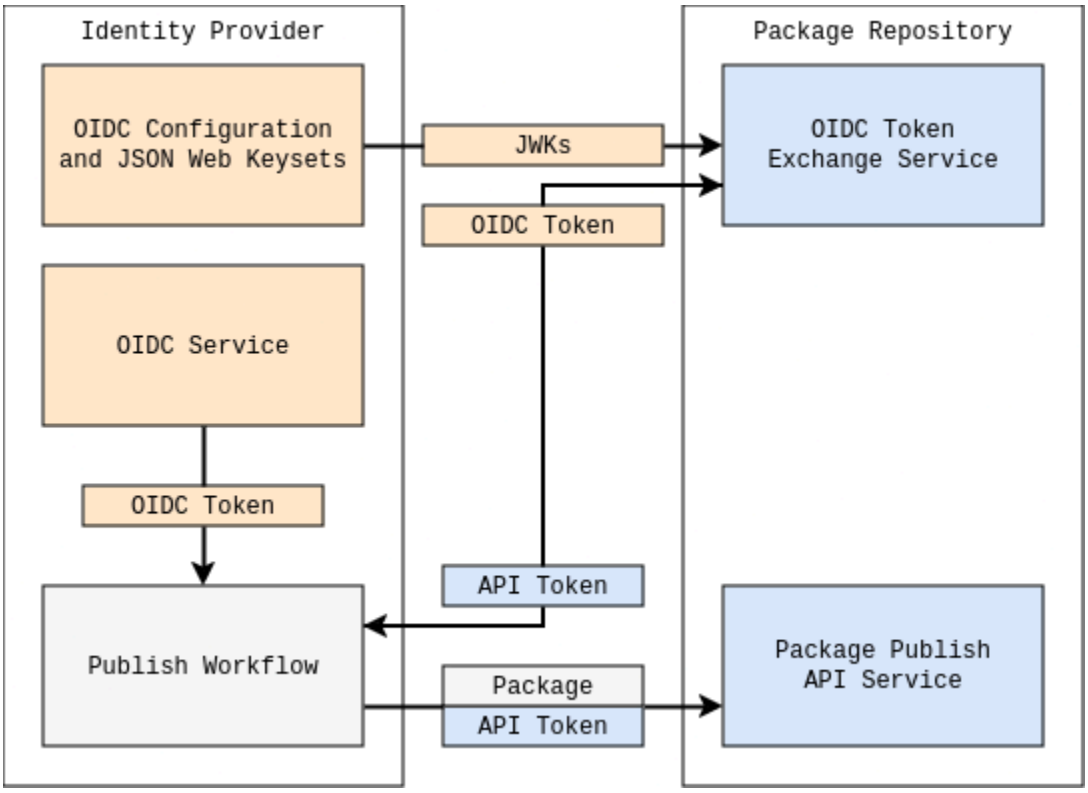
A high-level overview of how PyPI verifies the OIDC token against a pre-configured policy is below:

- Parse the JWT without verifying the claims or signature to extract the issuer claim (`iss`). Be sure that this parsed JWT is discarded and that the JWT isn't marked as verified by the service at this stage.
- Checking this issuer against the supported list of OIDC identity providers for the package repository, select the implementation for verifying the token (e.g "GitHub" or "GitLab").
- Fetch the OpenID Connect discovery URL and JWKs for the issuer.
- Using the JWKs, verify the JWT signature and claims (claims like `nbf`, `exp`, `jti`, etc).
- Verify that the audience claim (`aud`) is equal to a service-specific value (i.e. `pypi` and `testpypi`). This requires that the IdP supports configuring the audience of the emitted OIDC token.
- Using the claims, check the values against the pre-configured trust policy for the Trusted Publisher. For example, a GitHub workflow would check the following claims:
    - `sub` (Subject) is of the form "example-owner/example-repo:.*"
    - repository is "example-repo"
    - repository_owner is "example-owner"

- repository_owner_id is “12345”
- job_workflow_ref is “example-owner/example-repo/.github/workflows/publish.yml@abcdef”

Once this is complete, the package repository can authorize publications by delegating to a repository-managed token.

This document recommends exchanging the OIDC token for a separate repository-managed token, but this step is not required and the OIDC token may be used directly for authenticating the publish request. See the “Resulting Design Decisions” section for further reasoning behind this recommendation.



Overview of how PyPI exchanges the OIDC token for a PyPI API token

Every unique IdP that a package repository wishes to support will require its own implementation as the claims on each OIDC token vary depending on the provider (beyond the standard issuer, expiration, and audience claims of JWT tokens). The sets of claims chosen to check must constitute sufficiently trusted metadata to be checked against the trust policy on packages. Trusted Publisher implementations must not rely exclusively on a generic set of claims (i.e. only `sub` + `iss` + `aud`) as these do not constitute a sufficiently trusted set for many IdPs.

Resulting Design Decisions

Security model of Trusted Publishers

Trusted Publishers represent a more secure authentication model than long-lived API tokens, however Trusted Publishers are not a panacea. The following security model must be followed by users to maintain a secure configuration:

- Short-lived tokens like OIDC tokens or repository-managed API tokens are still sensitive material and should not be disclosed. Either of these tokens being disclosed means they can be used by an attacker to publish malicious packages to the repository until they expire.
- Configuring a Trusted Publisher is a means of establishing trust in some external state. That external state (both the IdP itself and the resources managed by the IdP) must not be controllable by untrusted parties.

It's recommended to document this security model alongside provider-specific guides on how to adopt Trusted Publishers for the package repository.

Data model of Trusted Publishers

The intuitive answer for the data model of Trusted Publisher configurations to packages is one-to-one, with one source of artifacts for each package. However, for many packaging ecosystems, a many-to-many approach will be needed to support the complete range of existing use-cases for authenticating a source of artifacts with a project.

"Monorepos" are where multiple separate projects' source code is stored in a single source repository. Monorepos are common across many packaging ecosystems and their existence requires a one-to-many relationship between a Trusted Publisher configuration and packages on a package repository.

For packaging ecosystems where there are multiple artifacts per release or multiple publishing workflows to support multiple platforms or CPU architectures (such as Python), the opposite "one-to-many" relationship is needed to support multiple Trusted Publisher configurations for a single package, to allow publishing from multiple build systems

Identity Providers and Claims

OIDC tokens are implemented as JSON Web Tokens (JWTs) and come with a set of claims, which are a mix of standard OIDC claims and provider-specific claims

IdPs must support customizing the audience claim (`aud`) and Trusted Publisher implementations must only accept OIDC tokens with a service-specific audience claim (ie "pypi").

IdPs all have their own claim sets for their OIDC tokens. Creating a Trusted Publisher for a provider requires knowledge about the provider's internal implementation and behavior when determining which subsets constitute a sufficiently trusted set to provide to users. Trusted Publisher implementations must not rely exclusively on a generic set of claims (i.e. only `sub` + `iss` + `aud`) as these do not constitute a sufficiently trusted set for many IdPs.

IdPs vary their sets of claims depending on the situation the OIDC token is used in, for example GitHub has special claims for reusable workflows or for different workflow trigger types. Providers have also been known to change their claims with varying amounts of fore-warning or documentation about the changes.

Trusted Publisher implementations may utilize the `jti` claim to prevent replaying of OIDC tokens by rejecting tokens which have been previously used.

IdPs may allow customizing other claims, like validity time of the OIDC token. Trusted Publisher implementations may decide to add further requirements to IdPs and their OIDC tokens (such as maximum validity time) to enforce more secure OIDC token handling where this is possible with the IdP.

OIDC tokens and repository-controlled tokens

Package repositories should not use OIDC tokens for directly authenticating with package publishing APIs. Instead, package repositories should exchange the OIDC token for a repository-controlled API token that is used for authenticating with publishing APIs. Below are a list of reasons for the exchange:

- If the package repository already supports token-based authentication, OIDC tokens won't plug into the existing authentication and authorization implementations. This will lead to needing to reimplement them for OIDC tokens, effectively having two token types and potentially leading to more bugs.
- JWTs can contain any information that the provider decides to include, like private or personally identifiable information (i.e. names and email addresses). Using a separate token avoids holding on to this information for extended periods of time to reduce potential exposure or persistence.
- Identity Providers can change the expiration and other policies around their tokens without notice. Creating your own temporary token makes the repository resilient to these changes.
- Repository-managed tokens are more pluggable into existing tooling for publishing packages.
- Repository-managed tokens benefit from external infrastructure, such as Secret Scanning.

Package repositories that don't support authenticating with API tokens are recommended to implement this capability as a part of their Trusted Publisher implementation.

Resurrection and Rename Attacks

Many third-party resources have human-readable attributes that are more discoverable (such as mutable GitHub owner and repository names) that relate to attributes that aren't as accessible to humans but constitute a much stronger guarantee about the resource (such as immutable GitHub owner IDs).

Resource names are typically mutable, meaning if they are solely depended on for a trust policy then that policy could be abused if resources are renamed or deleted on the third-party service without updating the corresponding Trusted Publisher.

Trusted Publisher policies are typically configured by humans, so it's more ergonomic to input the resource names rather than their IDs. To mitigate this issue, part of configuring a Trusted Publisher for many services is reaching out to the third-party service to "resolve" IDs from the configured names and then persist those IDs as a part of the Trusted Publisher policy.

There are limits to this approach, as not all resources will have IDs (for example, GitHub's "workflow file name" doesn't have an ID), but constraining the problem as tightly as possible should prevent the most common issues.

"Pending" Trusted Publishers

Trusted Publishers as described above are linked to *existing* resources in a package repository, either packages themselves or package namespaces, etc. But this leaves a casualty dilemma for package repositories that can only *create a new package* upon the first upload of an artifact (PyPI does this, for example). For package repositories with this property, it wouldn't be possible to create a new package using a Trusted Publisher, because the package has to exist before a Trusted Publisher is configured for it. Leaving this situation as-is would mean that other, less secure publishing methods need to be used before users can adopt Trusted Publishers which is both an ergonomics and security problem.

One solution to this issue is to implement "Pending" Trusted Publishers, which are only tied to a specific account or organization until after they're used for the first time, at which point they are bound further to a specific package or package namespace. When being used to authenticate an upload for the first time, the "pending" model is converted into a normal Trusted Publisher and associated with the authenticated resource in the database. This is done within the same database transaction as persisting the models to avoid scenarios where there's partial success during the upload process.

References

As of April 2024, the Python Package Index, RubyGems.org, and Dart's pub.dev support Trusted Publishers.

- https://docs.pypi.org/trusted-publishers/internals/
- https://docs.pypi.org/trusted-publishers/security-model/
- https://docs.google.com/presentation/d/e/2PACX-1vTJ2k4yr4tzug5Nf4HIrJ9Am2vMX2I1b0wUlreDPZc2dbWow9SRm78fmuvlyhEggtdk1Mefme3nmtd_/pub