Better External Authorization

Shared with Istio Community

|  | |
| --- | --- |
| **Owner**: Yangmin Zhu ([ymzhu@google.com](mailto:ymzhu@google.com)) Shriram Rajagopalan ([rshriram@tetrate.io](mailto:rshriram@tetrate.io)), Jianfei Hu ([jianfeih@google.com](mailto:jianfeih@google.com))  **Working Group:**  Security | **Status**: WIP |In Review | **Approved** | Obsolete **Created**: 2020-09-16  **Approvers**: Limin Wang [X], Louis Ryan[X], Sven Mawson[X], Neeraj Poddar[x], Lin Sun [x], Mandar Jog[x ] |

# **TL;DR**

This design proposes to support a new "CUSTOM" action in authorization policy to allow customers to integrate their custom authorization system with Istio.

# **Background**

Today the AuthorizationPolicy provides the ALLOW and DENY action to enforce finer-grained access control evaluated locally inside the Envoy sidecar. This provides declarative-style access control with good performance.

For many customers, this type of local authorization policy is not enough and lacks the flexibility of integrating custom authorization systems with Istio.

One example of custom authorization system is the **OIDC** flow ([blog: Setting Up oauth2-proxy with Istio](https://www.paraesthesia.com/archive/2020/09/03/setting-up-oauth2-proxy-with-istio/)) that allows the customer to "*verify the identity of the End-User based on the authentication performed by an Authorization Server, as well as to obtain basic profile information about the End-User in an interoperable and REST-like manner*." ([OIDC spec](https://openid.net/connect/))

Another common use case is that many customers have their own authorization systems (e.g. **OPA**) that are usually very specific to their organization and AuthorizationPolicy doesn't satisfy the need. It is much easier to integrate the external authorization system with Istio seamlessly instead of trying to convert it to Istio authorization policy which is not always possible. For example, Yelp is using the Envoy ext\_authz filter to delegate the authorization decision to their own custom authorization DSL in the production environment ([KubeCon 2019 presentation](https://static.sched.com/hosted_files/kccncna19/c4/kubecon_2019_how_yelp_moved_security_from_the_app_to_the_mesh_with_envoy_and_opa.pdf)).

Today, the most common solution to the above use cases is to use the Envoy ext\_authz filter with the Istio EnvoyFilter CRD. However, this user experience is not only difficult and hard to maintain, but also not a reliable solution for any serious enterprise users who are looking for a maintainable long-term solution (see [pain points](#u67wcd1rt97v) below for more details).

# **Pain Points**

The current flow of using Envoy ext\_authz filter with EnvoyFilter is detailed in [Configuring ext authZ as first class entity in Istio](https://docs.google.com/document/d/1LuBRI1xTVzAGW-0k3yA3vKTpD5TTvCB75b5ZFO_UrlY/edit?ts=5ed542a3#).

There are many frequently complained pain points of this solution:

* It does not support triggering the external authorization flow conditionally. Either all requests or none requests are enabled with ext-authz. In reality, a lot of requests (e.g. health check) just do not need the authorization check.
  + "*The envoy.ext\_authz is a very powerful and useful feature, but in most of the cases the user needs to add an exception, so some "operations" should be allowed to bypass it. In my case it is a health check and liveness probe endpoints which should remain "public".*" ([issue](https://github.com/istio/istio/issues/24462))
  + "*I want to avoid routing requests to OPA authz sidecar for kubernetes health check requests.*" ([issue](https://github.com/istio/istio/issues/26805))
  + "*I wish to use the envoy envoy.ext\_authz filter with various unique configurations, each targeting a specific domain name.*" ([issue](https://github.com/istio/istio/issues/23058))
  + "*Would like to enable filters on ingress gateway instead of individual pod. However, is it possible that this filter is run/executed/applied to specific kubernetes service/pod? Tried routeconfig with domain name , yet it applied to all services with all domain names. please help*" ([issue](https://github.com/istio/istio/issues/24761))
* EnvoyFilter is essentially a transparent API that exposes the full functionality of the Envoy API, which also includes the instability part of the API.
  + A small change in the upstream Envoy API could cause all the user applied EnvoyFilter to fail, sometimes even failing silently
  + "*Envoy filter format is in flux. The object defined here is really dependent on the version of Envoy that Istio is using. This was a huge pain.*" -- ([user blog](https://www.paraesthesia.com/archive/2020/09/03/setting-up-oauth2-proxy-with-istio/))
* Configuration depends on pilot internal implementation details
  + User needs to understand the some Istio internal implementation details in order to configure the ext\_authz filter correctly with Istio, which is hard for beginners and sometime even Istio experts
    - Istio internally generates the cluster name in the format of "<DIRECTION>|<PORT>|<SUBSET>|<HOSTNAME>" ([github](https://github.com/istio/istio/blame/195a47c43cd7738f847b635b415aaf0caa8473b1/pilot/pkg/model/service.go#L531-L535)) where the pipe "|" character is actually invalid to be used in the ext\_authz gRPC client, the final result is that the user just cannot use the auto generated cluster name in ext\_authz.
    - [ext\_authz gRPC service doesn't work with the cluster name generated by pilot](https://github.com/istio/istio/issues/21841)
    - [External Auth not functional when auth service inside mesh](https://github.com/istio/istio/issues/16676)

* General difficulties in using and debugging the EnvoyFilter CRD
  + Customer mistyped url in the filter config
    - [Cannot get EnvoyFilter ext\_authz to work](https://discuss.istio.io/t/cannot-get-envoyfilter-ext-authz-to-work/7095)
  + Bug in EnvoyFilter causing wrong protobuf merge results
    - [(ext\_authz) per\_filter\_config not allowing 'disabled: false' values](https://github.com/istio/istio/issues/24548)
  + [Configuring an HTTP external authz service for EnvoyFilter resource](https://github.com/istio/istio/issues/9097)
  + [Envoy.ext\_authz filter](https://github.com/istio/istio/issues/8295)
  + [Adding header-based routing EnvoyFilter patch discards per\_filter\_config for ext\_authz](https://github.com/istio/istio/issues/21213)
  + [Istio Envoy (EnvoyFilter) skipping ext\_authz filter (Istio 1.5.5)](https://discuss.istio.io/t/istio-envoy-envoyfilter-skipping-ext-authz-filter-istio-1-5-5/7486/3)

# **Use Cases**

This section describes the new user flow of the proposed change.

1. (Primary) Integrate custom external authorizer on Istio Ingress gateway

This is the primary use case since the ingress gateway is the most common place for doing end-user related custom external authorization. Example: <https://www.paraesthesia.com/archive/2020/09/03/setting-up-oauth2-proxy-with-istio/>

1. (Secondary) Integrate custom external authorizer on Istio sidecar

This is similar to the JWT policy where the primary use case was mainly on ingress gateway but later we find people are also using it on sidecar for various reasons (e.g. the ingress gateway is not terminating and just a passthrough). Example: <https://github.com/istio-ecosystem/authservice/blob/master/bookinfo-example/README.md>



Prerequisites

1. **Mesh admin** defines the external authorizer including details like where and how to talk to it
2. The external authorizer must be added to the Istio service registry so that it is accessible by Envoy sidecar in the mesh. For example, it can be deployed in the mesh with a normal k8s service or out of the mesh with a ServiceEntry.

In configuration time

1. **Mesh admin** or **Namespace/Workload owner** applies the AuthorizationPolicy with action CUSTOM on workloads. The AuthorizationPolicy decides when to trigger and which external authorizer to use.

In runtime

1. A request hits Envoy and will be evaluated to decide whether or not to trigger the external authorization flow
2. If the external authorization flow is triggered, Envoy will send an external authorization check request to the external authz server
3. The external authz server will make the decision (pass or reject). Additionally, the external authz server can modify the header of the original request
4. If the decision is DENY, the request will be rejected immediately. If it's a pass, the request will continue to be evaluated by the normal ALLOW and DENY authorization policies.

# **Examples**

This section gives examples of the e2e user flow, refer to [Design](#215sxl5j39wv) for details of the API.

Step 0. (**Prerequisites**) Mesh admin deploy the external authorizer in the istio-system namespace, expose it to Istio with k8s service and enable mTLS and access control to only allow ingress gateway to access it. (see the [example ext-authz server deployment](#8roanpz6wxo))

Step 1. Mesh admin adds the external authorizer to MeshConfig:

| extensionProviders:  # the unique name for the authorizer, referred to by AuthorizationPolicy.  - name: "my-custom-ext-authz-service"  # Specifies the concrete type of the external authorizer as Envoy ext\_authz filter using  # HTTP API. Note the HTTP API means the protocol used to talk to the external server, not  # the user traffic. The other supported API is gRPC.  # See [Exposed ext\_authz API](#4xtquij55jo9) for a detailed list of the ext\_authz fields exposed here.  envoyExtAuthzHttp:  service: "ext-authz.istio-system.svc.cluster.local"  port: "8000"  # Send the check request to ext-authz server at the path /check  pathPrefix: "/check"  # Includes the header "x-foo" in the check request to the external server.  includHeadersInCheck: ["x-foo"]  # Forwards the Authorization header from the external server to upstream (backend).  headersToUpstreamOnAllow: ["Authorization"]  # Forwards the Set-cookie header from the external server to downstream (user).  headersToDownstreamOnDeny: ["Set-cookie"] |
| --- |

The following is another example of defining the external authorizer using gRPC API

| extensionProviders:  - name: "my-custom-ext-authz-service"  envoyExtAuthzGrpc:  service: "ext-authz.istio-system.svc.cluster.local"  port: 9000 |
| --- |

Step 2. The namespace/workload owner (whoever has the necessary permission) applies the AuthorizationPolicy with action CUSTOM to use with the above external authorizer:

| apiVersion: security.istio.io/v1beta1 kind: AuthorizationPolicy  metadata:  name: ext-authz-external  namespace: istio-system spec:  selector:  matchLabels:  app: istio-ingressgateway  action: CUSTOM  provider:  name: "my-custom-ext-authz-service"  rules:  - to:  - operation:  paths: ["/admin/\*"] |
| --- |

Step 3. **Optionally**, the external authorizer may authorize the original user request and return another credential (e.g. JWT token) that can be used within the mesh.

The user can apply the following additional RequestAuthentication and AuthorizationPolicy to verify the new JWT token.

| # Define RequestAuthentication to validate the exchanged JWT token. apiVersion: security.istio.io/v1beta1 kind: RequestAuthentication metadata:  name: jwt  namespace: istio-system spec:  selector:  matchLabels:  app: istio-ingressgateway  jwtRules:  - issuer: "issuer@ext-authz.istio-system.svc.cluster.local"  jwksUri: "http://ext-authz.foo.svc.cluster.local/jwks.json" --- # Define AuthorizationPolicy to enforce access control on the exchanged JWT token. apiVersion: security.istio.io/v1beta1 kind: AuthorizationPolicy  metadata:  name: allow-group-admin  namespace: istio-system spec:  selector:  matchLabels:  app: istio-ingressgateway  action: ALLOW  rules:  - from:  - source:  requestPrincipals: ["issuer@ext-authz.istio-system.svc.cluster.local/\*"]  when:  - key: request.auth.claims[group]  values: ["admin"] |
| --- |

# **Design**

**Overview**

* A new ExtensionProvider is added in MeshConfig to define the extension providers that are allowed to be used in the AuthorizationPolicy
  + Currently at most 1 extension provider is allowed, meaning all authorization policy will use the same external authorizer.
  + The configuration will also specify the details of where and how to talk to the external authorizer
  + The current only supported concrete external authorizer is the Envoy ext\_authz filter of both its HTTP and gRPC API.
    - It's worth mentioning that we are **intentionally exposing a small API surface** in the configuration instead of exposing the full capabilities of the Envoy ext\_authz filter. The interface is restricted to auth purpose, and should not become a general mixer style checker API.
    - See the list of [exposed Envoy ext\_authz API](#4xtquij55jo9) for a detailed list of fields we exposed.
  + It's possible that more other authorizer concrete implementations can be added in the future in addition to the Envoy ext\_authz filter.
    - They will have their own configuration per needed.
* A new "CUSTOM" action is added to the authorization policy
  + The "CUSTOM" action and related new fields will be explicitly commented as **experimental feature** in both the API and documentation on istio.io
  + Users should clearly understand the implications of using the experimental feature, which mainly means there may be breaking API changes in later versions based on user feedback.
* A new "provider" field is added to specify the detail configuration for the CUSTOM action, currently this only includes the **name** of the authorizer provider defined in the MeshConfig.

**Security model**

The security model is NOT changed with the introduction of the CUSTOM action. If there is a namespace admin role in the mesh, the namespace admin would have the permission to apply the CUSTOM action just like they apply the ALLOW/DENY policy.

The CUSTOM action will send the user request often with security sensitive information (e.g. the Authorization header) to the external authorization server, the same thing could be achieved with the use of VirtualService, EnvoyFilter or modifications of the application code directly since the namespace admin would have full control over the workloads in the namespace.

If it's needed to enforce finer grained policy admission, the mesh admin could use the k8s RBAC to control who could apply what in the namespace, and if it needs to be content-based (e.g. who could apply EXTERNAL policy in the namespace), the mesh admin can use other admission controllers like Gatekeeper [OPA](https://www.openpolicyagent.org/docs/v0.12.2/kubernetes-admission-control/).

**Deployment model**

The external authorization server is not provided by Istio and is usually under the control of customers. The customer needs to make sure it is properly deployed and accessible by Istio. This is similar to the jwksUri field in the RequestAuthentication policy: the customer needs to make sure the jwks server is accessible by Istio.

The external server can be deployed either within the mesh or out of the mesh, and must be registered to Istio service registry via either k8s service or ServiceEntry (or any other supported methods), this should be done by **mesh admin** as it requires access to the MeshConfig.

| Example 1: deployed as a standalone service in the mesh | The external server can also sit behind the Envoy sidecar so that other Istio APIs like PeerAuthentication can be used to enable mTLS, AuthorizationPolicy can be used to control who can call the external server, and even the DestinationRule if needed. |
| --- | --- |
| Example 2: deployed in the same pod of the Envoy as a separate container | In this case, the external authorization request never leaves the pod itself that provides better security and also performance. A ServiceEntry is needed to add the (localhost) service to Istio service registry. |
| Example 3: deployed out of the mesh | It's strongly recommended that the customer enables at least the HTTPS to protect the traffic to the external server. A ServiceEntry is needed to add the service to Istio service registry. |

The customer chooses the deployment type that best suits their use cases, this often involves tradeoffs between performance, security and matinatinility. Istio doesn't force the user to use any specific deployment type but just provides the flexibility and necessary API to support it.

**Performance**

The **latency will be much larger than the local authorization** because of the external network request. We consider this acceptable for existing users if they are already using the ext\_authz filter via the EnvoyFilter CRD.

Additionally, many external authorization use cases are latency insensitive applications such as the OIDC flow. The extra network request is unavoided in this case in order to redirect users to the 3rd authorization page.

Some customers believe the latency is good enough for their current use case and they care more about improving the maintainability and flexibility of the external authorization flow instead of focusing on the performance.

Last, the customer will now be able to enable the external flow conditionally which should improve the overall performance a lot by skipping the extra network request on unnecessary traffic.

Based on this, we do not plan to do performance optimization (e.g. caching) in this design. It could be done later when needed.

**Evaluation order**

The CUSTOM action will be evaluated before the JWT policy and the ALLOW/DENY action.

We choose this evaluation order for the following reasons

* The CUSTOM action is considered as an optional extension to the local access control (ALLOW/DENY) policy. And it could behave much differently than the local access control and is completely up to the extension provider, this means it is considered a "blackbox" style extension. It is considered a complementary but not a full replacement to the native access control and thus should not be able to bypass the decisions made by the native access control policies;
* This also means the dependency relationship is clear and simple since the native access control policies are always evaluated after the CUSTOM action, it can rely on the output of the CUSTOM action instead of vice versa;
* We will focus on improving conditions to enable optimization and not add any ordering beyond what is already in the spec given that this is effectively now a general purpose 'trigger' API we'll have to address ordering issues in more detail later;
* Last, from the implementation perspective, this is also the most natural choice as the ext\_authz is a separate filter that is executed independently from the rbac filter.

A special note on the JWT policy part, this means we currently won't support using JWT-related matching rule in the CUSTOM policy. This is more of an implementation optimization problem to figure out a better way to make the JWT matching rule available beforehand as we would need to inject the JWT filter twice and it could potentially evaluate the JWT token twice, there is **no behavior change** but just performance overhead and implementation complexity, like mentioned above we will focus on improving the conditions to better support this and avoid the need to evaluate JWT token twice.

The ext\_authz Envoy filter will be added to the filter chain in front of the JWT and RBAC filter:



~~If there is any need to evaluate the (local) ALLOW/DENY action before the EXTERNAL action, the user can configure the matching rule to control when the EXTERNAL action should be triggered to achieve the same final result, in other words, the user can use the matching rule to only trigger the request that indeed needs external authorization and all other requests can bypass to be handled by the normal ALLOW/DENY action.~~

The following diagram shows the evaluation flow of the policies:



The CUSTOM policies are evaluated independently from the DENY/ALLOW policies.

* A request will be rejected if the CUSTOM policies return deny
* A request could still be rejected if the CUSTOM policies return allow because the DENY/ALLOW policies are evaluated after

In other words, the CUSTOM policies will never bypass the DENY/ALLOW policies, but it could short-cut the DENY/ALLOW policies by rejecting the request early. The other way to think about this is to think of CUSTOM policies like some additional DENY policies.

A request is allowed if and only if "CUSTOM not rejected" **AND** "DENY not matched" **AND** "ALLOW matched".

**Matching Mechanism (conditionally trigger)**

The CUSTOM action is only triggered when the matching rule evaluates to true. This means the matching rule will now also control the behavior of the Envoy ext\_authz filter in addition to the RBAC filter.

We have investigated multiple options for this support and choose the option 1, see below for more details.

**Option 1 (chosen) - Use RBAC filter shadow mode to evaluate matching rules**



In this option, a minimal change is made to the ext\_authz filter to add the metadata matcher( [PR](https://github.com/envoyproxy/envoy/pull/13404)). In runtime, Istio will inject an extra RBAC filter in shadow mode to evaluate the matching rules before the ext\_authz filter, and store the matching result to a well-known dynamic metadata namespace. The shadow mode means the RBAC filter will only evaluate but not enforce.

Later in the ext\_authz filter, it will read from the dynamic metadata via the metadata matcher to decide whether or not to trigger the external flow based on the matching result.

**Pros**

* Minimal API and code changes in Envoy, only metadata matcher in ext\_authz, this makes it much more likely to be accepted by Envoy community while some other long term solutions are being finalized and implemented (see option 2 and 3)
* 100% consistent evaluation result of the matching rule because we reuse the same RBAC filter for the evaluation
* Supports every field in the matching rule (header, source IP, JWT, source principal, etc.)
* Small changes in istiod as we can easily reuse the existing code for converting the matching rule to the RBAC filter config, the only change is to use the shadow\_rules field (same type as the normal rules field)
* Can be easily supported with other filters and extensions, the other filter or extension will just need to add a the metadata matcher so that it can leverage the AuthorizationPolicy matching rules just like the ext\_authz filter. And it's worth to mention that the metadata matcher will be added as part of the ongoing unified matching API work in Envoy, this means this option can actually be used as a good long term solution.

**Cons**

* Small performance overhead for reading from metadata, should be minimal as the data is very simple (essentially just a string) and not a big deal compared to the external request overhead
* Also a small overhead from running the extra RBAC filter

**Option 2 - Use the generic unified matching API in Envoy**

The Composite HTTP Filters ([design](https://docs.google.com/document/d/1dOa_v1eMA4F61f0jdFkxtbcO-_TYD76oMfGoY7nzZhk/edit#heading=h.9v2rpr8krgni)) proposes to add the generic unified matching API in filter chain level so that every filter can use it without requiring any changes in the filter config itself, this is a more general solution but requires much larger changes and more time to implement.

This generic solution is very unlikely to be finalized any soon and will take more time for the implementation and improvement to be ready for the replacement of the RBAC filter matching API.

After some discussions with Envoy maintainers (Matt Klein and Snow Pettersen), we agreed to go with option 1 as a short term solution to unblock the better external authorization in Istio.

Also note, the generic unified matching API will include the support of metadata matcher (may not be in the first implementation, see [the comment in this PR](https://github.com/envoyproxy/envoy/pull/13404#pullrequestreview-503194761))

**Option 3 - Add the matching API directly to the ext\_authz filter**

The Unified Matching in Access Control ([design](https://shorturl.at/gkrM3)) proposes to add the unified matching API directly in ext\_authz filter and RBAC filter. The unified matching API is a newly designed matching API based on the TAP filter and supports more capabilities (like request body matching and streaming support) and is supposed to be used by different filters easily with a clear defined interface.

However, this approach is mostly replaced by an even more general filter chain level matching API (option 3), the original [PR](https://github.com/envoyproxy/envoy/pull/12964) was thus put on hold for this reason.

In addition to the long term conflict with option 2, this option also requires much larger work in both ext\_authz and RBAC filter, and correspondingly requires many changes in the control plane to use the new matching API.

This is also risky because any inconsistencies in the matching evaluation could result in some severe bugs or CVEs as this is used for access control policy enforcement.

**Multiple policies**

Multiple CUSTOM policies can be applied to the same workload, the CUSTOM policies will be triggered once per provider name. In the case of ext\_authz, it means we will make one call per provider name.

Currently we only allow at most 1 provider, it's a configuration error if multiple provider names are used for now, and we will generate a default deny-all config until the policy in error is fixed by users. This means all the CUSTOM policies will be using the same provider. We will make only 1 external request to the specified provider no matter how many CUSTOM policies are matched.

For example, let's say you have 2 CUSTOM policies A and B and they both use provider X. policy A is triggered if request path is "/abc", and policy B is triggered if the request method is "GET". A GET request at path "/abc" will trigger the CUSTOM action **once** and make only 1 external check request to the provider X even if they are both matched. In Envoy, both policy A and B are coalesced into a into single ext\_authz filter.

When multiple providers are supported, policy A and B could be using different providers that point to different addresses or the same address but with different configuration, in either case it is considered a logically different extension provider, and we will start to make **multiple external requests** instead of only 1.

Also note we haven't decided the order when multiple different providers are used, we will get more community feedback to evaluate various options (e.g. no order, implicit order defined in Istio, explicit order defined by users).

# **API Options**

* External authorizer definition
  + **Option 1 (chosen) - Store the detail configuration in the MeshConfig**
    - The (platform or mesh) admin is responsible for defining what external authorizer is allowed to be used in the mesh and how to talk to them
    - Initially only allow at most 1 external authorizer defined and used in AuthorizationPolicy
    - Most API change in MeshConfig which allows an easier depreciation or promotion to separate API if needed
  + Option 2 - Inline in the AuthorizationPolicy with EXTERNAL action
    - All external server configurations are defined just in the Authz policy, will be a little bit easier to use (less yaml to create and maintain) but also makes the authz policy a little bit bloated
    - Cannot share the same configuration among multiple AuthorizationPolicy
  + Option 3 - Store the detail configuration in ConfigMap
    - Similar to option 1 but doesn't provide the formal protobuf definition in the istio/api repo

* Server endpoint design
  + **Option 1 - Use a simple service string**
    - Examples: "ext-authz.istio-system.svc.cluster.local:8000"
    - The service must first be in the Istio service registry (we will just use the cluster created from the k8s service or ServiceEntry and not create the cluster ourselves)
  + **Option 1 - Use the "**[**Destination**](https://github.com/istio/api/blob/master/networking/v1beta1/virtual_service.proto#L529-L555)**" (also used in VirtualService)**
    - The host, subset and port field will be used to construct the proper cluster name
      * This also means the authz code will need to access the service registry information (or some variant) in order to construct the cluster name to
        + get the port if it's omitted in the Destination
        + get the protocol (http, https or grpc)
    - We will need an extra field (e.g. "path") to specify the specific endpoint of the ext\_authz service (e.g. "/check")
  + **Option 2 - Use a FQDN URL (e.g. "http://ext-authz.foo.svc.local:8000/")**
    - The host of the URL (e.g. "ext-authz.foo.svc.local:8000") will be used to construct the corresponding cluster-name, we don't necessarily need to do a lookup to check if the cluster is really created, we just need to follow the same naming convention of the cluster name
    - The scheme of the URL (e.g. "http", "https", or "grpc) will decide the protocol to talk to the external server
    - The optional path part (e.g. "/check") will tell which specific endpoint of the ext\_authz service

* Config customization design
  + **Option 1 (chosen) - Create a new minimal API that abstracts the Envoy ext\_authz filter API**

(example in this [commit 3](https://github.com/istio/api/pull/1674/commits/7602328beb86948983b1ba5a403e760891f30efd))

* + - Pros
      * This provides better stability in Istio API level since it's a totally different abstraction and can give more flexibility if we ever want to integrate with a different "ext\_authz" filter
      * The Istio abstraction will also be more user friendly and consistent with other Istio API conventions (e.g. the AuthZ API uses the wildcard in string field to represent prefix/suffix/present matching)
    - Cons
      * Most parts of the ext\_authz filter API is very straightforward business related logic and already easy to use (e.g. what headers to include in the check request, fail open or fail close, what headers to modify in the response, etc.)
      * Users will still need to use EnvoyFilter in some rare cases not covered in the first-class API
  + **Option 2 - Use a generic (passthrough) type "google.protobuf.Struct"**

(it will be converted to [HTTP.ExtAuthz](https://github.com/envoyproxy/envoy/blob/3de2b81ab366c73772b51690b3e8a626da0c3b58/api/envoy/extensions/filters/http/ext_authz/v3/ext_authz.proto#L27) or [TCP.ExtAuthz](https://github.com/envoyproxy/envoy/blob/3de2b81ab366c73772b51690b3e8a626da0c3b58/api/envoy/extensions/filters/network/ext_authz/v3/ext_authz.proto#L27) in runtime)

* + - Pros
      * Maximal flexibility, user has control over any field in the ext\_authz filter
    - Cons
      * A raw struct is hard to use for customers as the real definition is in a different repo (Envoy)
      * The bigger issue is the stability. A breaking change in the ext\_authz API (or in its dependent types) will cause all user policies to fail in the passthrough mode.
        + This could be addressed by adding some intermediate convention logic but then we lose the benefits of passthrough.
  + **Option 3 - Mirror and fork the Envoy ExtAuthz API**

The forked version will be stored just in istio/api repo and can be converted directly to the Envoy ext\_authz API. (example in this [commit 1](https://github.com/istio/api/pull/1674/commits/d7e8162567e18265f72d3a6e98c88e6c58ea567b))

* + - Pros
      * Provide an indirection of the Envoy API for smooth transition to newer ext\_authz API in the future. Even if Envoy deprecates the v3 API, it won't break Istio and Istio should be able to convert to the new version without problems because the new envoy API is assumed to be backward compatible.
      * Clear type and definition of the API, easier to use as everything is in istio/api.
      * There is a precedent in Istio operator API that embeds the k8s API in the operator API. See the definition of the [KubernetesResourcesSpec](https://github.com/istio/api/blob/44a8342c376c259730a6bd1f551d0681ec71a4fa/operator/v1alpha1/operator.proto#L218) in IstioOperator API.
    - Cons
      * It is quite cumbersome (a lot of dependent types must be forked as well) to maintain the forked API, but probably not a big deal since it's not going to change very often.
      * There will still be stability issues in the long term because this option needs to fork a lot of the dependent types in addition to the ext\_authz filter API itself.
  + **Option 4 - Do not include the "config" field at all**

(example in this [commit 2](https://github.com/istio/api/pull/1674/commits/099b62329c4aba9b33799b3fc8c54877d843c28e))

* + - Pros
      * Simplifies the API by not over-designing the very first version of the API, for most users it may be good enough to just use the default configuration and rely on the EnvoyFilter CRD to fine-tune it if needed
        + Note, this is very different from the current EnvoyFilter user experience because the 2 biggest pain points (endpoint configuration and conditionally triggering) are solved regardlessly
      * Allows us to move fast in the first version of the API, and we can always add the config field (option 1-3) in later versions without breaking the API (since it's a new field)
    - Cons
      * If some field is widely configured to use a non-default value, many users may find they still need a separate EnvoyFilter in this case.

**Summary**

The external authorization is a natural complement to the current security policy (especially the authorization policy), the user experience should be as consistent as possible between external and local authorization.

**Goal 1: Consistent with current Istio AuthN/Z API and user experience**

In Istio, AuthorizationPolicy already provides a very powerful matching rule for local authorization access control. Our design goal is to reuse this part as much as possible so that the user will have a consistent experience when configuring when the external authN/Z flow should be triggered, which would be the same as they already did in the AuthorizationPolicy.

Additionally, the same workload label selector model, policy scope hierarchy should apply seamlessly to the external authN/Z flow to provide a real consistent user experience.

**Goal 2: Minimal API changes and no new CRD to make it easier to use**

It's hard, sometimes controversial and usually very time consuming for adding any new major CRD in Istio today as we strive to stabilize the API and reduce the number of CRDs to make Istio easier to learn and use for customers.

In summary, the goal of this design is to improve the overall user experience by solving the mentioned pain points with minimal API changes:

* Less error prone by not using EnvoyFilter and with strict (typed) validation in config apply time
* Consistent user experience with current security API
  + Workload selector and policy scope model
  + Conditionally triggers the external auth flow

In the short term, this design should greatly improve the user experience and unblock new use cases of the external authorization in Istio, and we can have enough time to get feedback from the community in the long term while the minimal API change gives us enough flexibility to in designing a more potentially more suitable new CRD if needed.

**Pros**

* Minimal API changes and no new CRD (only a new action type in authz API)
  + solve immediate pain points
  + collect feedback for guiding long term API design which is hard to finalize, could align with the WASM extension API later
* Consistent user experience with existing AuthorizationPolicy and matching API
* In common simple use cases, users can simplify specify the user-friendly `server` field to and no longer need to worry about the cluster\_name formats
  + Some most commonly used customization can be done inline with the AuthorizationPolicy so no need to use EnvoyFilter in most cases

**Cons**

* Still need to use EnvoyFilter in some non-common use cases to fine-tune the ext\_authz filter configuration. Note this is still easier than the user experience today because the user can just use MERGE operation to precisely tweak the configurations without worrying about the Istio related logic (e.g. cluster name)

# **Formal API definition**

**API Changes in AuthorizationPolicy**

| message AuthorizationPolicy {  enum Action {  // <omitted>   // THe CUSTOM action delegates the authorization decision to an external authorizer.  // An authorization check request  // will be sent to the external authorizer and the response will decide whether or not the  // original request should be allowed or denied.  //  // The CUSTOM action is evaluated before the ALLOW and DENY action and is currently an  // \*\*experimental feature\*\* in Istio.  CUSTOM = 3;  }   message Provider {  // Specifies the name of the extension provider. The list of available providers are defined  // in the MeshConfig.  string name = 1;  }   oneof action\_detail {  // Specifies detailed configuration of the CUSTOM action. Must be used only with CUSTOM action.  Provider provider = 4;  } } |
| --- |

**API Changes in MeshConfig**

| message MeshConfig {  message ExtensionProvider {  // REQUIRED. A unique name identifying the extension provider.  string name = 1;   // REQUIRED. Specifies detailed configuration for the extension provider.  oneof provider {  // Configures an external authorizer that implements the Envoy ext\_authz filter  // authorization check service using the HTTP API.  EnvoyExternalHttpAuthorizationProvider envoy\_ext\_authz\_http = 2;  // Configures an external authorizer that implements the Envoy ext\_authz filter  // authorization check service using the gRPC API.  EnvoyExternalGrpcAuthorizationProvider envoy\_ext\_authz\_grpc = 3;  }   message EnvoyExternalHttpAuthorizationProvider {  // REQUIRED. Specifies the service that implements the Envoy ext\_authz HTTP authorization service.  // The format is "[<Namespace>/]<Hostname>". The specification of <Namespace> is required only when  // it is insufficient to unambiguously resolve a service in the service registry. The <Hostname> is  // a fully qualified host name of a service defined by the Kubernetes service or ServiceEntry.  //  // Example: "my-ext-authz.foo.svc.cluster.local" or "bar/my-ext-authz.example.com".  string service = 1;  // REQUIRED. Specifies the port of the service.  uint32 port = 2;   // Sets a prefix to the value of authorization request header \*Path\*.  // For example, setting this to "/check" for an original user request at path "/admin" will cause the  // authorization check request to be sent to the external server at the path "/check/admin"  // instead of "/admin".  string path\_prefix = 3;   // List of headers that should be included in the authorization request sent to the  // external authorizer. Note that in addition to the headers supplied by users:  // 1. \*Host\*, \*Method\*, \*Path\* and \*Content-Length\* are automatically sent.  // 2. \*Content-Length\* will be set to 0 and the request will not have a message body.  repeated string include\_headers\_in\_check = 4;   // List of headers from the external server that should be added to the original request and  // forwarded to the upstream when the authorization check result is allowed (HTTP code 200).  // If not specified, the original request will not be modified and forwarded to backend as-is.  // Note, any existing headers will be overridden.  repeated string headers\_to\_upstream\_on\_allow = 5;   // List of headers from the external authorizer that should be forwarded to downstream when  // the authorization check result is not allowed (HTTP code other than 200).  // If not specified, all the authorization response headers, except \*Authority (Host)\* will  // be in the response to the downstream.  // When a header is included in this list, \*Path\*, \*Status\*, \*Content-Length\*,  // \*WWWAuthenticate\* and \*Location\* are automatically added.  // Note, the body from the external authorizer is always included in the response to downstream.  repeated string headers\_to\_downstream\_on\_deny = 6;  }   message EnvoyExternalGrpcAuthorizationProvider {  // The format is "[<Namespace>/]<Hostname>". The specification of <Namespace> is required only when  // it is insufficient to unambiguously resolve a service in the service registry. The <Hostname> is  // a fully qualified host name of a service defined by the Kubernetes service or ServiceEntry.  //  // Example: "my-ext-authz.foo.svc.cluster.local" or "bar/my-ext-authz.example.com".  string service = 1;  // REQUIRED. Specifies the port of the service.  uint32 port = 2;  }  }   // Defines a list of extension providers that provide extra functionality to Istio.  // For example, An extension provider can be configured as an external authorizer used with the  // AuthorizationPolicy to integrate Istio with a custom authorization system.  // Note, currently at most 1 extension provider is allowed.  repeated ExtensionProvider extension\_providers = 57; } |
| --- |

# **Test** **Plan**

An example ext-authz server will be added to allow testing the end to end flow in the test, the example server will be deployed in the cluster with a sidecar, and both HTTP and gRPC API will be implemented in the example server.

E2E tests

The E2E tests will send requests from client to server and the request will be checked by the example ext-authz server in the cluster.

We will verify the following CUJs in the e2e tests:

* Verify the CUSTOM action is working as expected in AuthZ policy
  + Make sure the request is indeed checked by the ext-authz server
* Verify the matching rule in authz policy is working as expected
  + Make sure the request is bypassed and not sent to the ext-authz server if the rule is not matched
  + Make sure the request is checked by the ext-authz server if the rule is matched
* Verify the e2e flow for example token exchange or group resolving using both AuthZ and RequestAuthN policy
  + Make sure the exchanged token from the ext-authz server can be used in following AuthorizationPolicy
* Verify that multiple authorization policy from different level (mesh-level, namespace-level and workload-level) are merged correctly for the CUSTOM action
* Verify an incorrect CUSTOM action policy is deny-by-default

Unit tests:

* Verify the generated Envoy filter config from the CUSTOM action, including the config of the ext-authz filter

Validation tests:

* Verify that bad configuration are caught in the validation webhook correctly
  + Incorrect cluster-name and other internal fields
  + Missing matching rule for EXTERNAL action
  + Etc.

# **Tool and Documentation**

Tool

* Provide an offline tool to convert EnvoyFilter of ext\_authz to the AuthorizationPolicy automatically
  + The convert will be at its best efforts as long as it's supported by the AuthorizationPolicy semantics (constrained by the workload label selector, the choice of the config customization option and etc.)

Debug

* Envoy ext\_authz filter will log in the access log when a request rejected by ext\_authz filter, currently this is only logged in HTTP, we can add it also to TCP
  + The ext\_authz filter also has useful stats: "ok", "denied", "error", "timeout" and "failure\_mode\_allowed"
* The `istioctl authz check` command will be updated to also show the EXTERNAL policies applied to the workload, together with ALLOW, DENY and AUDIT

Documentation:

* Task for plugging an external authz server (HTTP and gRPC)
* Task for token exchange or group resolving with example external ext\_authz server
* Update troubleshooting guide to include common issues in ext\_authz integration

# **Appendix**

# **Example ext-authz server deployment**

| # Deploy the external server in the istio-system namespace. apiVersion: apps/v1 kind: Deployment metadata:  name: ext-authz-server  namespace: istio-system spec:  replicas: 1  selector:  matchLabels:  app: ext-authz-server  template:  metadata:  labels:  app: ext-authz-server  spec:  containers:  - image: gcr.io/my-company/ext-authz-server:0.3  name: ext-authz-server  ports:  - containerPort: 8000  - containerPort: 9000 ---  # Use k8s service to expose the external server to Istio. apiVersion: v1 kind: Service metadata:  name: ext-authz  namespace: istio-system  labels:  app: ext-authz spec:  ports:  - name: http  port: 8000  targetPort: 8000  - name: grpc  port: 9000  targetPort: 9000  selector:  app: ext-authz-server --- # Apply PeerAuthentication to enable STRICT mTLS for access to the external server. apiVersion: "security.istio.io/v1beta1" kind: "PeerAuthentication" metadata:  name: ext-authz-mtls  namespace: istio-system spec:  selector:  matchLabels:  app: istio-ingressgateway  mtls:  mode: STRICT --- # Apply AuthorizationPOlicy to allow only ingress-gateway to access the external server. apiVersion: security.istio.io/v1beta1 kind: AuthorizationPolicy  metadata:  name: ext-authz-ingress  namespace: istio-system spec:  selector:  matchLabels:  app: ext-authz-server  rules:  - from:  - source:  principals: ["cluster.local/ns/istio-system/sa/istio-ingressgateway"] |
| --- |

# **Exposed Envoy ext\_authz API**

The following table lists all the existing fields in the Envoy ext\_authz API and also the decision about whether or not it is exposed in the MeshConfig.

We expose the most commonly used and business-related fields so that users can configure all they need in most cases, if there are some fields the user wants to configure but not exposed, they can continue to use EnvoyFilter to customize it (much simpler than injecting the whole config).

If needed, we can always add it to the MeshConfig, this should be easy to do in most cases as we now have a dedicated provider for Envoy ext\_authz, it might be tricky in some cases if the Envoy API involves too much internal types, in this case we will make adapt it to follow Istio conventions so that it will still be conceptually consistent with Istio.

| Envoy ext\_authz API | Description | **Exposed** |
| --- | --- | --- |
| ***Common fields of both HTTP and gRPC:*** | | |
| bool [failure\_mode\_allow](https://github.com/envoyproxy/envoy/blob/9ff2d695ff12512fb7c838fd8b7afc3e853d3e0c/api/envoy/extensions/filters/http/ext_authz/v3/ext_authz.proto#L62); | a simple bool field that can be easily added and would be useful in some cases. | **YES** |
| [BufferSettings with\_request\_body](https://github.com/envoyproxy/envoy/blob/9ff2d695ff12512fb7c838fd8b7afc3e853d3e0c/api/envoy/extensions/filters/http/ext_authz/v3/ext_authz.proto#L67); | Not exposed to keep the API and runtime behavior simpler | **NO** |
| bool [clear\_route\_cache](https://github.com/envoyproxy/envoy/blob/9ff2d695ff12512fb7c838fd8b7afc3e853d3e0c/api/envoy/extensions/filters/http/ext_authz/v3/ext_authz.proto#L79); | Not exposed to limit the use case of the ext\_authz server. | **NO** |
| [HttpStatus status\_on\_error](https://github.com/envoyproxy/envoy/blob/9ff2d695ff12512fb7c838fd8b7afc3e853d3e0c/api/envoy/extensions/filters/http/ext_authz/v3/ext_authz.proto#L83); | Could be exposed with some simplification, the original HttpStatus includes a huge enum list of 50+ HTTP status code. | **YES** |
| repeated string [metadata\_context\_namespaces](https://github.com/envoyproxy/envoy/blob/9ff2d695ff12512fb7c838fd8b7afc3e853d3e0c/api/envoy/extensions/filters/http/ext_authz/v3/ext_authz.proto#L97); | This is not supported as we put the ext-authz filter before the JWT filter | **NO** |
| [RuntimeFractionalPercent filter\_enabled](https://github.com/envoyproxy/envoy/blob/9ff2d695ff12512fb7c838fd8b7afc3e853d3e0c/api/envoy/extensions/filters/http/ext_authz/v3/ext_authz.proto#L105); | Rarely used, not exposed. | **NO** |
| [MetadataMatcher filter\_enabled\_metadata](https://github.com/envoyproxy/envoy/blob/9ff2d695ff12512fb7c838fd8b7afc3e853d3e0c/api/envoy/extensions/filters/http/ext_authz/v3/ext_authz.proto#L109); | Internally used by Istio, not exposed. | **NO** |
| [RuntimeFeatureFlag deny\_at\_disable](https://github.com/envoyproxy/envoy/blob/9ff2d695ff12512fb7c838fd8b7afc3e853d3e0c/api/envoy/extensions/filters/http/ext_authz/v3/ext_authz.proto#L118); | Rarely used, not exposed. | **NO** |
| bool [include\_peer\_certificate](https://github.com/envoyproxy/envoy/blob/9ff2d695ff12512fb7c838fd8b7afc3e853d3e0c/api/envoy/extensions/filters/http/ext_authz/v3/ext_authz.proto#L124); | Could be exposed as a simple bool flag. | **Maybe** |
| string [stat\_prefix](https://github.com/envoyproxy/envoy/blob/9ff2d695ff12512fb7c838fd8b7afc3e853d3e0c/api/envoy/extensions/filters/http/ext_authz/v3/ext_authz.proto#L141); | Not included for now as we may want to configure it by Istio. | **NO** |
| ***gRPC-only fields:*** | | |
| [EnvoyGrpc envoy\_grpc](https://github.com/envoyproxy/envoy/blob/9ff2d695ff12512fb7c838fd8b7afc3e853d3e0c/api/envoy/config/core/v3/grpc_service.proto#L277); | Not exposed as it's configured implicitly by Istio from the Istio service registry, the Envoy gRPC client supports clusters with EDS. | **NO** |
| [GoogleGrpc google\_grpc](https://github.com/envoyproxy/envoy/blob/9ff2d695ff12512fb7c838fd8b7afc3e853d3e0c/api/envoy/config/core/v3/grpc_service.proto#L282); | Not exposed as we don't need the advanced google gRPC client feature (e.g. STS), and it also doesn't support clusters with EDS. | **NO** |
| [Duration timeout](https://github.com/envoyproxy/envoy/blob/9ff2d695ff12512fb7c838fd8b7afc3e853d3e0c/api/envoy/config/core/v3/grpc_service.proto#L287); | Not exposed. We can set this to a very large value and rely on the DestinationRule to control the timeout. | **Maybe** |
| repeated [HeaderValue initial\_metadata](https://github.com/envoyproxy/envoy/blob/9ff2d695ff12512fb7c838fd8b7afc3e853d3e0c/api/envoy/config/core/v3/grpc_service.proto#L294); | Not exposed, not sure use cases. | **Maybe** |
| ***HTTP-only fields:*** | | |
| [HttpUri server\_uri](https://github.com/envoyproxy/envoy/blob/9ff2d695ff12512fb7c838fd8b7afc3e853d3e0c/api/envoy/extensions/filters/http/ext_authz/v3/ext_authz.proto#L200); | Not exposed as it's configured implicitly by Istio from the Istio service registry. | **NO** |
| [Duration timeout](https://github.com/envoyproxy/envoy/blob/9ff2d695ff12512fb7c838fd8b7afc3e853d3e0c/api/envoy/config/core/v3/http_uri.proto#L52); | Not exposed. We can set this to a very large value and rely on the DestinationRule to control the timeout. | **Maybe** |
| string [path\_prefix](https://github.com/envoyproxy/envoy/blob/9ff2d695ff12512fb7c838fd8b7afc3e853d3e0c/api/envoy/extensions/filters/http/ext_authz/v3/ext_authz.proto#L203); | exposed, very useful feature for the HTTP API. | **YES** |
| [ListStringMatcher allowed\_headers](https://github.com/envoyproxy/envoy/blob/9ff2d695ff12512fb7c838fd8b7afc3e853d3e0c/api/envoy/extensions/filters/http/ext_authz/v3/ext_authz.proto#L229); | exposed, allow finer control of what headers to be sent to the external server. | **YES** |
| repeated [HeaderValue headers\_to\_add](https://github.com/envoyproxy/envoy/blob/9ff2d695ff12512fb7c838fd8b7afc3e853d3e0c/api/envoy/extensions/filters/http/ext_authz/v3/ext_authz.proto#L233); | Not exposed but could be, this is a fixed list of headers to be added to the request to the external server. | **Maybe** |
| ListStringMatcher [allowed\_upstream\_headers](https://github.com/envoyproxy/envoy/blob/9ff2d695ff12512fb7c838fd8b7afc3e853d3e0c/api/envoy/extensions/filters/http/ext_authz/v3/ext_authz.proto#L243); | exposed, enables the use case of token exchange, etc. | **YES** |
| [ListStringMatcher allowed\_upstream\_headers\_to\_append](https://github.com/envoyproxy/envoy/blob/9ff2d695ff12512fb7c838fd8b7afc3e853d3e0c/api/envoy/extensions/filters/http/ext_authz/v3/ext_authz.proto#L248); | Not exposed. Could be included but not sure the use case. | **Maybe** |
| [ListStringMatcher allowed\_client\_headers](https://github.com/envoyproxy/envoy/blob/9ff2d695ff12512fb7c838fd8b7afc3e853d3e0c/api/envoy/extensions/filters/http/ext_authz/v3/ext_authz.proto#L255); | exposed, enables the use case of OIDC flow. | **YES** |

# **Brainstorm**

Just some brainstorming ideas for discussions, this is NOT part of this design.

Depending on the user feedback, there are multiple possible options in the long term

* If the current proposed design is good enough for most users, we may just use it as the long term plan.
* If the user wants more flexibility, a dedicated API is also possible but that requires careful planning and alignment with the upcoming WASM extension API in Istio.
  + Basic Auth WASM module: <https://github.com/bianpengyuan/wasm-extensions/tree/addfcab3ad8fe76de2fc33ab33da106e2b0c5f4c/extensions/basic_auth>)
  + ECDS example server: <https://github.com/istio/proxy/blob/master/tools/extensionserver/README.md>

Nevertheless, it's probably desirable to have a more general API to control the enabling/disabling of filters in the filter chain, the important difference between this general API and the current EnvoyFilter CRD is that the API's responsibility is more dedicated in inserting filters to the filter chain that is greatly simplified, easier to use and much less error prone.

A strawman (brainstorm) idea about this general API:

| apiVersion: filter.istio.io kind: FilterConfig  metadata:  name: my-group-resolver  namespace: istio-system spec:  selector:  matchLabels:  app: istio-ingressgateway  filter:  # kind is one of {LOCAL\_ACTION, EXTERNAL, JWT, WASM}  kind: LOCAL\_ACTION  configSource:  inline: <...>  order: 1   triggerRules:  - to:  - operation:  paths: ["/admin"] |
| --- |