Blocking AI Crawlers on Istio: AuthorizationPolicy's Ceiling and Envoy's Full-String Regex Trap

2026-07-27

Blocking ~160 known AI-crawler user agents at an Istio ingress gateway without Lua: why AuthorizationPolicy cannot do substring header matching, and why a single safe_regex alternation would silently never match anyway.

Blocking a list of known AI-crawler user agents at an ingress gateway sounds like a five-minute job: match the `User-Agent` header against a list of bot names, return `403`. The obvious tool for that on Istio is its own `AuthorizationPolicy` CRD — until you actually try to express "header contains one of these ~160 substrings" in it, and discover the API simply cannot say that. The next obvious fallback — a Lua filter that runs the match by hand — works, but is more machinery than the problem needs. There is a third option: configure the same native Envoy filter Istio's own CRD compiles down to, directly, with the one matching primitive the CRD never exposes. Getting there also surfaces a second, unrelated gotcha in how Envoy's regex matching actually works, which would have made an obvious "just use one big regex" approach fail silently.

Setup

The list in question is ai.robots.txt, a community-maintained set of known AI-crawler/scraper user-agent substrings — around 160 entries at the time of writing (`GPTBot`, `ClaudeBot`, `Bytespider`, `PerplexityBot`, and so on), packaged as ready-made block rules for nginx, Apache, HAProxy, and a few others. None of those targets are Istio or Envoy natively — reasonable, since the project is server-agnostic, but it means the matching logic has to be built for the ingress gateway from scratch.

The gateway in question is an Istio ingress Gateway (Gateway API `Gateway`/`HTTPRoute`), fronting one public hostname (`app.example.com`). The requirement: return `403` for any request whose `User-Agent` header contains one of the ~160 known bot substrings, on that hostname only, except for `/robots.txt` itself — even a bot that's about to be blocked should still be able to read the rules it's ignoring.

AuthorizationPolicy's header matching ceiling

Istio's `AuthorizationPolicy` looks like the obvious tool: it has a `when` condition block that can match on `request.headers[<name>]`, and a `DENY` action with a negated match keeps the policy scoped to just the hosts it names (the same reasoning as in an earlier post on Istio Authorization-

Policy — an `ALLOW` policy would flip the entire gateway workload to default-deny for every other host on it). A first attempt looks like this:

```yaml
apiVersion: security.istio.io/v1
kind: AuthorizationPolicy
metadata:
  name: block-ai-bots
spec:
  selector:
    matchLabels:
      gateway.networking.k8s.io/gateway-name: ingress-gateway
  action: DENY
  rules:
    - when:
        - key: request.headers[user-agent]
          values: ["*GPTBot*", "*ClaudeBot*", "*Bytespider*"]
```

This does not do what it looks like it does. Istio's own conversion code for `AuthorizationPolicy` header conditions (`pilot/pkg/security/authz/matcher/header.go` in the Istio source) only ever produces an exact match, a prefix match (trailing `*`), or a suffix match (leading `*`) — never a "contains" match, regardless of how the value string is written. A value of `"*GPTBot*"` is not a wildcard-both-ends pattern to this code path; it is read the same way as `"*GPTBot"` followed by a separate suffix check, and the leading/trailing `*` handling in the matcher explicitly special-cases only a single wildcard at one end. There is no code path that produces a substring/contains match here at all.

That matters because real `User-Agent` strings are essentially never bare bot names. They look like:

```
Mozilla/5.0 (compatible; GPTBot/1.0; +https://openai.com/gptbot)
```

An exact, prefix, or suffix match against `GPTBot` cannot match a string where `GPTBot` sits in the middle. Every AI crawler on the list that identifies itself this way — most of them — would sail straight through a policy built this way, with no error, no warning, and a config that looks entirely reasonable on review.

The instinct at this point is to reach for a Lua filter, since Envoy's Lua HTTP filter can run arbitrary string matching against a header value. That works, but it trades a declarative, protobuf-validated filter for a small interpreted script embedded in an `EnvoyFilter` — a script that has to be hand-maintained against ~160 substrings, re-escaped by hand if any of them contain regex-special characters, and that runs in the request path with none of Envoy's own config validation behind it.

There is a more direct route. Istio's own `AuthorizationPolicy` doesn't invent a new enforcement mechanism — it compiles down to a plain `envoy.filters.http.rbac` HTTP filter, the same filter type available to any `EnvoyFilter` patch. The CRD's limitation is in what it exposes, not in what the underlying filter can do: Envoy's `Principal` type (`envoy/config/rbac/v3/rbac.proto`) has a `header` identifier backed by a full `StringMatcher` (`envoy/type/matcher/v3/string.proto`), and `StringMatcher` has a native `contains` field — plain substring matching,

no regex, no escaping. `Principal` also has `and_ids`/`or_ids` (a `Principal.Set`, matched as AND/OR respectively) and `not_id` (negation), which is enough to express the whole requirement — host scope, bot-name match, `/robots.txt` exemption — as one declarative tree, no scripting:

```
apiVersion: networking.istio.io/v1alpha3
kind: EnvoyFilter
metadata:
  name: block-ai-bots
spec:
  workloadSelector:
    labels:
      gateway.networking.k8s.io/gateway-name: ingress-gateway
  configPatches:
    - applyTo: HTTP_FILTER
      match:
        context: GATEWAY
        listener:
          filterChain:
            filter:
              name: envoy.filters.network.http_connection_manager
              subFilter:
                name: envoy.filters.http.router
      patch:
        operation: INSERT_BEFORE
        value:
          name: envoy.filters.http.rbac
          typed_config:
            "@type": type.googleapis.com/
envoy.extensions.filters.http.rbac.v3.RBAC
            rules:
              action: DENY
              policies:
                block-ai-bots:
                  permissions:
                    - any: true
                  principals:
                    - and_ids:
                        ids:
                          - header:
                              name: ":authority"
                              string_match:
                                exact: "app.example.com"
                          - or_ids:
                              ids:
                                - header:
                                    name: "user-agent"
```

```
                    string_match:
                      contains: "GPTBot"
                - header:
                    name: "user-agent"
                    string_match:
                      contains: "ClaudeBot"
                - header:
                    name: "user-agent"
                    string_match:
                      contains: "Bytespider"
            - not_id:
                header:
                  name: ":path"
                  string_match:
                    exact: "/robots.txt"
```

`permissions: [{any: true}]` matches every request regardless of path or method — all the actual selectivity comes from `principals`, which is where the host scope, the bot-name OR-list, and the `/robots.txt` exemption all live. The filter is inserted with `INSERT_BEFORE` immediately ahead of `envoy.filters.http.router`, the same insertion point used for any other custom HTTP filter added via `EnvoyFilter`. In a full deployment, the `or_ids` list under `user-agent` has one entry per bot name — a template loop over the maintained list, not a hand-written block, but the same shape either way. Scoping the policy to a specific `:authority` value is a deliberate choice for gateways where more than one hostname shares the same Envoy listener (the common case behind a wildcard certificate, and the same shared-listener setup discussed in the earlier post on HTTP/2 connection coalescing): without it, the policy would apply gateway-wide.

Envoy's full-string regex trap

Before settling on `contains`, the more obvious design was a single `safe_regex` alternation covering the whole list — `(?i)(GPTBot|ClaudeBot|Bytespider|...)` — one `Principal`, one field, no OR-list to maintain. It compiles cleanly. It would also never match anything.

Envoy's `RegexMatcher` (`envoy/type/matcher/v3/regex.proto`) documents this explicitly, in one sentence easy to skim past: *"The regex is matched against the full string, not as a partial match."* A `safe_regex` field is an anchored, whole-string match — functionally `^(?:...)$` — not a "find this pattern anywhere in the string" search the way `grep` or a browser's regex literal behaves by default. Matching `(?i)(GPTBot|ClaudeBot|Bytespider)` against `GPTBot` alone succeeds. Matching it against the actual header value — `Mozilla/5.0 (compatible; GPTBot/1.0; +https://openai.com/gptbot)` — fails, because the alternation only covers a fragment of the string and the match requires the whole thing to fit the pattern.

The fix at the regex level would be to wrap the whole alternation in `.*(...).*`, restoring find-anywhere semantics explicitly. That works, but re-introduces exactly the escaping burden a `contains`-based `or_ids` list avoids for free: every bot name has to be regex-escaped before being inserted into the alternation, or a name containing a regex metacharacter (a literal `.` in `bigsur.ai`,

a `/` in `iaskspider/2.0` — both real entries on the list) silently changes what the pattern matches. `StringMatcher.contains` needs no escaping at all, because it never treats its input as a pattern in the first place — which is also why the fix above builds the block list as one `or_ids` entry per name rather than one `safe_regex` for all of them.

Nothing here would have failed loudly. A `safe_regex` alternation missing the `.*` wrapper compiles, deploys, and reports healthy — it just never matches a real-world header, which reads identically to "nobody's crawling this site with a blocked bot," right up until someone tests it with an actual bot's user-agent string instead of the bot's name alone.

Verification

```
# Bot user-agent on the scoped host → expect 403
curl -A "GPTBot" -o /dev/null -w "%{http_code}\n" https://app.example.
com/

# robots.txt stays reachable even for a blocked bot → expect 200
curl -A "GPTBot" -o /dev/null -w "%{http_code}\n" https://app.example.
com/robots.txt

# realistic, embedded user-agent string, not just the bare bot name →
expect 403
curl -A "Mozilla/5.0 (compatible; GPTBot/1.0; +https://openai.com/
gptbot)" \
     -o /dev/null -w "%{http_code}\n" https://app.example.com/

# ordinary browser user-agent → expect a normal response
curl -A "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36" \
     -o /dev/null -w "%{http_code}\n" https://app.example.com/

# a host outside the scoped :authority, same bot UA → expect no 403
(scope test)
curl -A "GPTBot" -o /dev/null -w "%{http_code}\n" https://other.example.
com/
```

The third check is the one that actually exercises `contains` rather than an accidental exact match — a bare `curl -A "GPTBot"` would pass even against a broken `safe_regex` alternation missing its `.*` wrapper, since `GPTBot` alone happens to satisfy a whole-string match too. Testing with only the bare bot name would have hidden exactly the bug described above.

Takeaways

- Istio's `AuthorizationPolicy` header conditions (`when.values`) only ever compile to exact, prefix, or suffix matches — never a substring/contains match, no matter how the wildcard characters in the value string are arranged. Confirm this against the actual conversion code rather than the value syntax alone if a header condition is behaving unexpectedly.

- `AuthorizationPolicy` compiles to Envoy's native `envoy.filters.http.rbac` filter. When the CRD can't express something the underlying filter can, configuring that same filter type directly via `EnvoyFilter` is not a workaround outside the mesh's own model — it's the same mechanism, used with the one primitive (`Principal.header.string_match.contains`) the CRD never surfaces.
- Envoy's `safe_regex` is a whole-string match, not a substring search. A pattern that looks correct against a bare test value can still never match a real header, silently, with no error at any layer — verify against the actual header shape a real client sends, not an idealized substring of it.
- Prefer `StringMatcher.contains` over a `safe_regex` alternation for plain substring matching against a list of literal strings — no escaping burden, no whole-string-match trap, and no regex engine in the request path for something that was never a pattern-matching problem to begin with.

Summary

Two Envoy behaviors that look like edge cases and turn out to matter for anything short of a literal-string block list: `AuthorizationPolicy`'s header conditions cap out at exact/prefix/suffix regardless of how the value is written, and `safe_regex` matches the entire header value rather than searching within it. Both are avoidable with the same fix — reach one layer below the CRD, to the `envoy.filters.http.rbac` filter it compiles to, and use `StringMatcher.contains` instead of a regex for what was always a plain substring check.