Security Decisions

- Path Handling, Normalization and Encoding
- Forwarded Headers and Client Identity
- Headers Traefik Sets for the Backend
- TLS and mTLS Enforcement
- Kubernetes References Across Namespaces and Providers
- Generated Keys and Shared State
- Ingress-NGINX Provider Compatibility
- Authentication Middleware Correctness
- Availability and Resource Consumption
- Configuration the Operator Controls
- Dependency and Standard Library Findings
- Non-GA Code

Some classes of report reach us repeatedly. This page records the positions we have already settled, so that you can check a finding against them before submitting, and so that we can answer with a reference instead of a fresh analysis.

It is organised by the surface a report touches. Each entry states our position, then **where the line is**: the neighbouring variant we do treat as a vulnerability, with the advisories that prove it. That line is the useful part. Almost every surface below has produced both closures and CVEs.

This page applies the threat model, which defines the boundary itself. Start there if you want the short version.

!!! note "Why only some cases are linked"

```
We reference published advisories only. Reports closed as not-a-vulnerability stay private to
the
reporter and the security team, so declined classes are described as rules rather than linked.
Where
a class has produced a CVE, it is cited: that is the evidence the line is real rather than a
```

```
way of
declining work.
```

Path Handling, Normalization and Encoding

What is usually reported. A crafted path (`%2e%2e`, `%2f`, double encoding, a literal `;` or backslash, a fragment) traverses or escapes a prefix and reaches a route or backend that a middleware was expected to protect.

Our position. This is our most productive surface and we treat it seriously. A normalization or encoding differential that reaches the backend **with stock entrypoint defaults** is a vulnerability, including when it only manifests as a desynchronisation between the decoded and raw forms of the path.

Where the line is. We decline variants whose escape depends on the operator having widened handling, or on one backend's own re-interpretation of a character the standard permits:

- The escape works only with encoded-character handling left permissive. That default is a deliberate, documented compatibility decision, revisited only at a major version.
- The vector is a character RFC 3986 permits in a path segment, such as a literal `;` or a backslash, and the escape exists only because a given backend re-interprets it. That re-interpretation varies by backend and is the backend's to define: path parameters introduced by `;` are standard behaviour across the Java servlet family, not one implementation's quirk. We do not extend CVE treatment to every literal-byte sibling of a published encoded-separator issue.

Confirm the bypass against a normalizing backend with stock entrypoint configuration before submitting.

Decided in this class. GHSA-xf64-8mw2-4gr2 (CVE-2026-48020, StripPrefix route-level auth bypass), GHSA-6jwx-7vp4-9847 (CVE-2026-40912, StripPrefixRegex, Path/RawPath desync), GHSA-46wh-3698-f2cx (CVE-2026-33186, dot-segment bypass in strip-prefix middleware), GHSA-gm3x-23wp-hc2c (CVE-2025-66490, router plus middleware rules), GHSA-vrch-868g-9jx5 (CVE-2025-47952, traversal via URL encoding), GHSA-6p68-w45g-48j7 (CVE-2025-32431, path matchers), GHSA-fvhj-4qfh-q2hm (CVE-2023-47106, URL fragment), and GHSA-cxjq-mrr5-89rv (ReplacePathRegex).

Forwarded Headers and Client Identity

What is usually reported. A middleware passes through, or fails to strip, an `X-Forwarded-*` or `Forwarded` header, letting a client influence what a backend believes about its identity, protocol, port or address. Often framed as an incomplete fix for a published issue, because a sibling code path does not repeat a strip added elsewhere.

Our position. The trust boundary for forwarded headers is **entrypoint-level**. `forwardedHeaders.trustedIPs` and `forwardedHeaders.insecure` decide once, at the entrypoint, before any middleware runs, whether a client's forwarding headers are trusted. We deliberately do not re-decide per middleware: re-deciding in each middleware is the design mistake we are avoiding, not an omission. A middleware passing through a header the entrypoint accepted is behaving as designed, and a sibling path that does not repeat a strip is **not automatically** an incomplete fix.

Where the line is. In scope:

- A path that **reintroduces or reconstructs a trusted value after entrypoint sanitisation**, so the entrypoint's decision no longer holds.
- A middleware that makes a **pre-authentication decision** on a header the operator declared untrusted.

"Not automatically an incomplete fix" is not "never": where a sibling genuinely reintroduces the same primitive, we have published it as one.

Decided in this class. GHSA-6384-m2mw-rf54 (CVE-2026-35051, `trustForwardHeader=false` still honouring a spoofed prefix), GHSA-3q9r-p662-5j8m (CVE-2026-54764, ForwardAuth trusting a spoofed port), GHSA-92mv-8f8w-wq52 (CVE-2026-29054, case-sensitive Connection header), GHSA-62c8-mh53-4cqv (CVE-2024-45410, Connection header abused to strip `X-Forwarded-*`), GHSA-h924-8g65-j9wg (CVE-2024-52003, open redirect via prefix) and GHSA-6qq8-5wq3-86rp (CVE-2020-15129, prefix header not validated).

Headers Traefik Sets for the Backend

What is usually reported. A client pre-sets, or sets an alternative spelling of, a header that Traefik writes for the backend's benefit: the authenticated user an auth middleware puts in its

configured `headerField`, client certificate information from `PassTLSClientCert`, `X-Forwarded-Prefix` from a prefix middleware, `X-Replaced-Path`, or a custom request header. The backend trusts the value because Traefik set it, so forging it is the attack.

Our position. The same architecture as forwarded headers, for the same reason: **request header sanitisation is an entrypoint concern, not a per-middleware one.** A middleware removes the canonical spelling of the header it is about to set. Alternative spellings that a backend folds onto that same name are handled once, before routing, through the entrypoint's `aliasHeadersStrategy`.

The underlying mismatch is worth stating, because it is what makes the class non-obvious: Go canonicalises header names on `-` only, while backends that derive variable names from header names (CGI, WSGI, PHP, nginx, Tomcat) fold other characters to `_`. Two spellings that Traefik sees as different headers can therefore arrive at the backend as one. That is a property of the whole request rather than of any one middleware, which is precisely why it is resolved at the entrypoint.

Where the line is. A report showing that **one middleware, taken in isolation, does not sanitise a header it sets** is declined. The entrypoint owns that sanitisation, and re-deciding it inside each middleware is the design we are avoiding, not an omission. In scope:

- a spelling that survives the entrypoint sanitisation and still reaches the backend as the trusted name;
- a middleware that reconstructs or reintroduces the value **after** the entrypoint has sanitised it.

Decided in this class. GHSA-qr99-7898-vr7c (CVE-2026-33433, identity spoofing through a non-canonical `headerField`), GHSA-x677-9fxg-v5c5 (CVE-2026-54763, an accepted incomplete fix: an aliased spelling survived the strip added for the case above) and GHSA-5m6w-wvh7-57vm (CVE-2026-39858, alias spoofing ahead of a pre-authentication decision).

TLS and mTLS Enforcement

What is usually reported. A client reaches a route, or avoids presenting a certificate, because Traefik selected the wrong TLS configuration for the connection.

Our position. Enforcement that does not hold is in scope, and this surface has a specific recurring shape we now look for directly: **the TLS configuration chosen for a connection must match the configuration the operator intended for that host**, including wildcard mappings, mixed-case hosts, and every protocol version. A mismatch that drops client-certificate verification is a bypass even when each component behaves as written.

Where the line is. Configuring a permissive `TLSOption`, or omitting client authentication, is the operator's choice and not a defect. In scope is Traefik **not applying** the stricter configuration that

the operator did set, or applying it on one protocol but not another.

Decided in this class. GHSA-9cr8-q42q-g8m7 (CVE-2026-53622, HTTP/3, exact SNI lookup for wildcard and mixed-case hosts), GHSA-5r4w-85f3-pw66 (CVE-2026-48491, SNICheck ignoring wildcard mappings, domain-fronted bypass), GHSA-wvvq-wgcr-9q48 (CVE-2026-32305, fragmented ClientHello, pre-SNI fallback to the default), GHSA-gv8r-9rw9-9697 (CVE-2025-68121, ClientAuth on HTTP/3), GHSA-gxrv-wf35-62w9 (CVE-2024-39321, IP allow-lists bypassed via QUIC 0-RTT early data), GHSA-468w-8x39-gj5v (CVE-2022-46153, routes exposed with an empty `TLSOption`) and GHSA-hrhx-6h34-j5hc (CVE-2022-23632, wrong TLS configuration selected).

Kubernetes References Across Namespaces and Providers

What is usually reported. A resource in one namespace, or under one provider, reaches a service, middleware or transport it should not, through a `backendRef`, an `ExtensionRef`, a `ServersTransport`, or a cross-provider reference.

Our position. A reference that **escapes a boundary the operator established**, reaching a namespace or provider the configuration did not grant, is a vulnerability. This surface has produced a large share of our CVEs.

Where the line is. Check whether the allow-list trust anchor is genuinely bypassable for the route in question. If the traversal is gated by an opt-in the operator enabled, such as `crossProviderNamespaces` or `allowCrossNamespace`, that is the documented contract of the option. If it works **without** that opt-in, or defeats it, it is in scope. Equally, a report requiring the ability to create or edit the Kubernetes objects involved is not a boundary crossing: that actor already controls routing in the namespace.

Decided in this class. GHSA-62fc-8686-hfmq (CVE-2026-71325, `allowCrossNamespace=false` bypassed via `@kubernetescrd`), GHSA-3g6v-2r68-prfc (CVE-2026-54761, `crossProviderNamespaces` allow-list defeated), GHSA-42cj-m3vj-89wv (CVE-2026-65602, IngressRouteTCP ServersTransport), GHSA-qq9q-x9w4-chhj (CVE-2026-65601, ExtensionRef namespace confusion), GHSA-xhjw-95fp-8vgq (CVE-2026-41174, cross-namespace middleware binding), GHSA-96qj-4jj5-wcjc (CVE-2026-44774, `rest@internal` reachable as a backend), GHSA-67jx-r9pv-98rj (CVE-2026-32695, host restriction bypassed) and GHSA-8q2w-wr49-whqj (CVE-2026-29777, rule injection through unescaped values in Ingress and HTTPRoute).

Generated Keys and Shared State

What is usually reported. Two distinct objects collide in an internal key, or share a cache, connection pool or in-flight request, so one reaches the other's context.

Our position. The distinction that decides these reports is **who generates the colliding key**.

- A collision **the reporter must author themselves**, by creating or editing objects whose names collide, is not a boundary crossing. The internal keyspace naming routers, services and middlewares is not a tenant isolation boundary on a shared instance.
- A key **Traefik generates** that collides for inputs the operator kept separate is a vulnerability. So is state shared through a cache, pool or de-duplication mechanism keyed too loosely.

Where the line is. Severity follows the boundary the collision actually crosses, not the mechanism: a generated-key collision reaching across a namespace or tenant on a shared control plane is high, while one confined inside a single credential cache with no cross-tenant reach is low. Both are still vulnerabilities.

Decided in this class. GHSA-fgjj-px3w-67xx (CVE-2026-71327, Gateway API route identity collision, cross-namespace backend hijacking, high), GHSA-6765-c87h-8mrf (CVE-2026-71326, BasicAuth singleflight key collision, low), GHSA-6p8f-p8j2-rqmv (CVE-2026-54765, backendRef filters leaking across routes sharing a `Service:port`) and GHSA-3ccp-42pg-hgv6 (CVE-2026-71324, cross-user response poisoning through the shared backend keep-alive pool).

Ingress-NGINX Provider Compatibility

What is usually reported. A default or annotation behaviour of the Kubernetes Ingress-NGINX provider is insecure, with a working reproduction.

Our position. This provider's contract is annotation compatibility with ingress-nginx. Where it faithfully reproduces documented upstream semantics, we keep the compatible behaviour **even when the upstream outcome is insecure**, and we improve the documentation instead: changing it would silently break the migrations the provider exists to serve. Check the upstream behaviour first. If Traefik matches it, expect a documentation change rather than an advisory. Reports we have closed on these grounds include permissive CORS defaults, per-ingress `auth-url` precedence, and `auth-url` variable interpolation differences.

Where the line is. Compatibility covers **semantics the operator opted into**. It does not cover **authentication or mTLS enforcement silently not happening**. Where the provider fails open, rather than faithfully reproducing an insecure upstream default, that is a vulnerability.

Decided in this class. GHSA-4mr2-fg2p-w63c (CVE-2026-54762, fails open when `auth-secret` resolution fails), GHSA-7vww-mvcr-x6vj (CVE-2025-66491, inverted TLS verification logic, so client certificates were not verified) and GHSA-8rxv-jg7p-wvg3 (`rewrite-target` path traversal defeating route-level authentication).

Authentication Middleware Correctness

What is usually reported. An authentication middleware leaks information, accepts an identity it should not, or forwards credentials somewhere unintended.

Our position. In scope: credential or identity handling that leaks across requests or users, observable timing differences that disclose whether a principal exists, and credentials forwarded to a destination the operator did not authorise.

Where the line is. Choosing a weak authentication mechanism, or configuring it permissively, is the operator's decision. The middleware failing to deliver what its documentation promises is ours.

Decided in this class. GHSA-6x2q-h3cr-8j2h (CVE-2026-41263, BasicAuth timing side channel) and GHSA-g3hg-j4jv-cwfr (CVE-2026-32595, BasicAuth timing side channels, username enumeration), GHSA-p6hg-qh38-555r (CVE-2026-41181, Errors middleware forwarding `Authorization` and `Cookie` to a separate service) and GHSA-h2ph-vhm7-g4hp (CVE-2022-23469, `Authorization` in debug logs).

Availability and Resource Consumption

What is usually reported. A request pattern that exhausts memory, connections, file descriptors or CPU, or that stalls a connection indefinitely.

Our position. Unbounded consumption, panics, and missing timeouts reachable from unauthenticated requests are in scope. Buffering and queuing that stays within documented,

configurable limits is not.

Where the line is. We require a reproduction rather than a reading of the code, and so should you. This class repeatedly turns on whether a per-stream limit actually bounds the per-connection aggregate, which code reading gets wrong in both directions. A resource claim with no reproduction against a capped environment cannot be assessed, and we will ask for one.

Decided in this class. GHSA-4hjq-9h5c-252j (CVE-2026-27141, HTTP/2 frames panicking the server), GHSA-fw45-f5q2-2p4x (CVE-2026-26998, ForwardAuth unbounded response body), GHSA-xw98-5q62-jx94 (CVE-2026-26999, stalled TCP router connections) and GHSA-cwjm-3f7h-9hwq (CVE-2026-22045, handshake stalls without timeouts), GHSA-89p3-4642-cr2w (CVE-2026-25949, `readTimeout` bypassed via STARTTLS), GHSA-4vwx-54mw-vqfw (CVE-2024-28869, Content-Length handling), GHSA-7hj9-rv74-5g92 (CVE-2023-29013, header parsing), GHSA-c6hx-pjc3-7fqr (CVE-2022-39271, HTTP/2 connection management) and GHSA-8g85-whqh-cr2f (CVE-2023-47124, ACME challenge amplification).

Configuration the Operator Controls

What is usually reported. A behaviour reachable after enabling a documented option, or in a default deployment the documentation tells operators to harden. Dashboard and API reachability, snippet annotations, error request headers, and variable interpolation are the recurring examples.

Our position. Configuration is trusted input, and so is every provider that supplies it. Anyone able to write configuration Traefik reads already controls routing. A report whose precondition is configuration-write access, or operator or cluster-admin privilege, is not a vulnerability. Where an option is explicitly opt-in, its documented effect is its contract.

Where the line is. In scope: an unprivileged, untrusted client achieving the same effect **without** that configuration access, or a documented option that does not do what it says.

Dependency and Standard Library Findings

What is usually reported. A scanner flags a CVE in a Go module or in the standard library Traefik builds against.

Our position. A dependency finding is an exposure only if Traefik reaches the vulnerable code path at runtime. Build-time-only and unreachable paths are not vulnerabilities in Traefik, and we confirm reachability with `govulncheck` before answering. Where the path is reachable we publish, which is why several standard library CVEs appear in our advisory history.

Where the line is. If you can show a reachable call path from a Traefik entrypoint, say so explicitly: that is the part that determines the answer.

Non-GA Code

Vulnerabilities affecting only release candidates, betas, or development branches are fixed without a CVE, as stated in the CVE policy.

Open a normal issue for these, not a security advisory. Once you have confirmed the defect is absent from every GA release, there is no released version to protect, so there is nothing to keep private, and an issue reaches the maintainers faster than the advisory queue. Reporting during a release candidate is also the most useful moment to reach us: the fix costs a single commit, rather than an advisory, a backport across every maintained branch, and exposure for everyone who already upgraded.

If the same defect is **also** reachable in a GA release, it is not a non-GA finding, and it belongs in a security advisory. Check the released branches before concluding that a finding is release-candidate only, and tell us which ones you checked.

□□□ #2
□ □□□ □□□ 2026-10-31 13:58:08 UTC
□ □□□ □□□ 2026-10-31 13:58:09 UTC