REGISTRY MODE

Configuration Reference

Complete reference for Socket Registry Firewall configuration options.

Configuration Reference

Complete reference for Socket Registry Firewall configuration options. All options can be set in `socket.yml` or overridden via environment variables.

Configuration File (`socket.yml`)

The firewall reads configuration from `/app/socket.yml` inside the container. Mount your config file:

YAML

```yaml
volumes:
  - ./socket.yml:/app/socket.yml:ro
```

Core Socket Settings

YAML

```yaml
socket:
  # Socket.dev API endpoint (required)
  api_url: https://api.socket.dev

  # Behavior when Socket API is unreachable
  fail_open: true            # true = allow packages (default), false = block all

  # Behavior for unscanned/unknown packages (Socket returns purlError)
  fail_open_unscanned: true  # true = allow unscanned with warning (default), false = block
unscanned
  expose_unscanned_header: false  # true = add X-Socket-Unscanned response header (default:
false)

  # Console log level (controls which messages appear in logs)
  log_level: info            # error, warn, info (default), debug

  # Corporate egress proxy for all upstream connections
  outbound_proxy: http://proxy.company.com:3128
  no_proxy: localhost,127.0.0.1,internal.company.com

  # SSL verification for Socket API calls
  api_ssl_verify: true       # Verify SSL for Socket API (default: true)
  api_ssl_ca_cert: /path/to/corporate-ca.crt  # Custom CA cert

  # SSL verification for upstream registry connections
  upstream_ssl_verify: true  # Verify SSL for upstream registries (default: true, inherits
api_ssl_verify)
  upstream_ssl_ca_cert: /path/to/upstream-ca.crt  # Custom CA for upstreams

  # Socket API timeouts (seconds) — apply to Socket.dev API calls only.
  # Distinct from proxy.connect_timeout / send_timeout / read_timeout, which apply to
upstream registries.
  api_read_timeout: 300      # Response body budget (defaults to proxy.read_timeout when
unset)
  api_connect_timeout: 5     # TCP+TLS connect (clamped to ≤ api_read_timeout)
  api_send_timeout: 10       # Request send (clamped to ≤ api_read_timeout)

  # Request behavior
```

```
  request_id_header: X-Socket-Request-ID  # Custom request ID header name

  # Client auth gate (require clients to present credentials)
  bearer_token: SOCKET_AUTH_TOKEN  # Env var name containing the token
  bearer_token_type: env           # Resolve bearer_token from the env var
  # basic_auth_username: ${SOCKET_BASIC_AUTH_USERNAME}  # HTTP Basic auth username
  # basic_auth_password: ${SOCKET_BASIC_AUTH_PASSWORD}  # HTTP Basic auth password
```

Environment variables:

Bash

```
 SOCKET_SECURITY_API_TOKEN=${SOCKET_SECURITY_API_TOKEN}  # Required (scopes: packages:list,
entitlements:list)
SOCKET_API_URL=https://api.socket.dev
SOCKET_FAIL_OPEN=true
SOCKET_FAIL_OPEN_UNSCANNED=true
SOCKET_EXPOSE_UNSCANNED_HEADER=false
SOCKET_LOG_LEVEL=info
SOCKET_LOG_MAX_BODY_SIZE=3900
SOCKET_OUTBOUND_PROXY=http://proxy:3128
SOCKET_NO_PROXY=localhost,127.0.0.1
SOCKET_API_READ_TIMEOUT=300
SOCKET_API_CONNECT_TIMEOUT=5
SOCKET_API_SEND_TIMEOUT=10
SOCKET_BEARER_TOKEN=${SOCKET_AUTH_TOKEN} # Client auth gate (alternative to
socket.bearer_token in YAML)
```

Fail-Open Behavior

The firewall has two independent fail-open settings that control behavior in different error scenarios:

`fail_open` — API Errors

Controls behavior when the **Socket API is unreachable** or returns an HTTP error (timeout, 500, network failure).

Value	Behavior
`true` (default)	**Allow** the package with a warning. Error message appears in `warn_reason` field of decision logs, Splunk events, and webhooks.
`false`	**Block** the package. Error message appears in `block_reason` field. Returns 403.

`fail_open_unscanned` — Unscanned Packages

Controls behavior when the Socket API **doesn't recognize a package/version** (returns a `purlError` response). This happens when the package hasn't been scanned yet or doesn't exist in Socket's database.

Value	Behavior
`true` (default)	**Allow** the package with a warning. The purlError message appears in `warn_reason`. Decision logs include `unscanned: true`.
`false`	**Block** the package. The purlError message appears in `block_reason`. Returns 403. Decision logs include `unscanned: true`.

Why two settings? `fail_open` covers infrastructure issues (API down, network errors) while `fail_open_unscanned` covers data coverage gaps (new packages, private packages Socket hasn't indexed). Organizations may want different policies for each — for example, allowing packages during API outages but blocking unscanned packages in strict environments.

`expose_unscanned_header` — Response Header Visibility

When `true`, adds an `X-Socket-Unscanned: true` response header on requests for unscanned packages. Default: `false` (header not sent).

YAML

Socket Dashboard ›

```
socket:
  fail_open: true               # API errors: allow with warning (default)
  fail_open_unscanned: true     # Unscanned packages: allow with warning (default)
  expose_unscanned_header: false # X-Socket-Unscanned header: suppressed (default)
```

Client Auth Gate (`bearer_token` / `basic_auth`)

Optional feature to require all inbound requests to present valid credentials. Supports two methods:

1. **Bearer token**: `Authorization: Bearer <token>`
2. **Basic auth**: `Authorization: Basic <base64(username:password)>`

If both are configured, either method is accepted. When configured, requests without valid credentials receive a `401 Unauthorized` response.

Configuration

YAML

```
socket:
  # Bearer token auth:
  # Read token from an environment variable
  bearer_token: SOCKET_AUTH_TOKEN       # Name of the env var (NOT the value)
  bearer_token_type: env                # Tells config tool to resolve the env var

  # Basic auth:
  basic_auth_username: ${SOCKET_BASIC_AUTH_USERNAME}
  basic_auth_password: ${SOCKET_BASIC_AUTH_PASSWORD}
```

Setting	Values	Default	Description
`bearer_token`	string	*(empty)*	The token value, or env var name when `bearer_token_type` is `env`
`bearer_token_type`	`string` , `env`	`string`	How to interpret `bearer_token`
`basic_auth_username`	string	*(empty)*	Username for HTTP Basic authentication
`basic_auth_password`	string	*(empty)*	Password for HTTP Basic authentication

Or via environment variables (works without any YAML config):

Bash

```
SOCKET_BEARER_TOKEN=${SOCKET_AUTH_TOKEN}
# or
SOCKET_BASIC_AUTH_USERNAME=${SOCKET_BASIC_AUTH_USERNAME}
SOCKET_BASIC_AUTH_PASSWORD=${SOCKET_BASIC_AUTH_PASSWORD}
```

Behavior

- **All requests** (except `/health`) must include a valid `Authorization` header matching one of the configured methods
- Unauthenticated or mismatched requests receive `401` with appropriate `WWW-Authenticate` header and JSON error body
- The `/health` endpoint is **exempt** from auth — always accessible without credentials
- When auth succeeds, the client's `Authorization` header is **stripped** and NOT forwarded upstream. Upstream auth is handled separately via `upstream_token` if configured. The exception is a route with `forward_client_auth` set, which leaves the header in place
- Auth failures are logged (credentials are never logged)
- If both bearer token and basic auth are configured, the `WWW-Authenticate` header advertises both methods

Interaction with `upstream_token`

Integrations

Socket for GitHub

`bearer_token` / `basic_auth`	`upstream_token`	**Behavior**
—	—	Client auth passes through to upstream (default)
set	—	Client must match; **no** auth forwarded upstream
—	set	No inbound gate; upstream gets token from env var
set	set	Client must match; upstream gets token from env var

A route with `forward_client_auth` set overrides this table: the caller's own token goes upstream, and `upstream_token` is ignored.

CI-CD Integrations ›

Upstream Auth Token (`upstream_token`)

Inject a `Bearer` token on all upstream (firewall → registry) requests for a specific route. The token value comes from an environment variable — the YAML config specifies only the **env var name**, so secrets never appear in config files.

Configuration

Supported on both path-based routes and domain-based registries:

YAML

```yaml
# Path-based routing
path_routing:
  routes:
    - path: /pypi
      upstream: https://private-pypi.company.com
      registry: pypi
      upstream_token: PYPI_AUTH_TOKEN        # env var name → value used as Bearer token

    - path: /npm
      upstream: https://private-npm.company.com
      registry: npm
      upstream_token: NPM_AUTH_TOKEN

# Domain-based routing
registries:
  pypi:
    domains:
      - pypi.company.com
    upstream: https://private-pypi.company.com
    upstream_token: PYPI_AUTH_TOKEN          # same behavior for domain routes
```

Set the actual token value as an environment variable on the container:

Bash

```bash
# Bearer token (no colon in value)
PYPI_AUTH_TOKEN=${PYPI_AUTH_TOKEN}

# Basic auth (user:password format — auto-detected by the colon)
NPM_AUTH_TOKEN=${NPM_REGISTRY_CREDS}
```

Setting	Values	Default	Description
`upstream_token`	string	*(empty)*	Name of an environment variable containing the auth credential for upstream requests

Auth Scheme Auto-Detection

The firewall inspects the **value** of the env var at startup to choose the HTTP auth scheme:

Socket CLI ›

Socket MCP
Socket Firewall
Socket Firewall Overview
Socket Firewall Free
Socket Firewall Enterprise
Registry Mode
Artifactory Configuration
AWS CodeArtifact Configuration
Configuration Reference
Installation Guide
Nexus Configuration
Downstream Deployment Guide
Upstream Deployment Guide
Wrapper Mode
Socket for VS Code
Socket REST API
Socket Chrome Extension
Next steps
Advanced

Env var value	Detected scheme	Authorization header sent
`<token-value>` (no `:`)	Bearer	`Authorization: Bearer <token-value>`
`<username>:<password>` (contains `:`)	Basic	`Authorization: Basic <base64(username:password)>`

This is fully automatic — no additional configuration needed.

Behavior

- When `upstream_token` is set for a route, every upstream request on that route includes the auto-detected `Authorization` header — replacing any client-sent Authorization header
- Routes without `upstream_token` pass the client's Authorization header through unchanged (default behavior)
- The env var name must match `[A-Za-z_][A-Za-z0-9_]*` (standard env var naming)
- Token values are **pre-resolved at worker startup** for performance (no per-request `os.getenv()` overhead)
- Token values are **never logged** — only the env var name appears in init logs as `<redacted>`
- If the env var is empty or not set, a warning is logged and no Authorization header is injected

Security Notes

- Token values exist only in environment variables and process memory — never in config files or logs
- Each route can have a different token, enabling per-registry credential isolation
- Works independently from `bearer_token` (inbound client auth gate) — see interaction table above

---

Forwarding the Caller's Own Credential (`forward_client_auth`)

> **Requires version 2.4.1 or higher.**

`upstream_token` sends one shared credential upstream for everyone on a route. `forward_client_auth` sends each caller's own instead, so a private registry can apply its per-user permissions. Developers log in with their own token (`npm login` , or the equivalent for their ecosystem) and that token is what the registry sees.

Works on path-based routes defined inline under `path_routing.routes` . Domain-based registries and routes loaded from an external routes file ignore it.

Configuration

YAML

```yaml
path_routing:
  enabled: true
  domain: firewall.example.com
  routes:
    - path: /npm-private
      upstream: https://registry.npmjs.org
      registry: npm
      forward_client_auth: true

metadata_filtering:
  response_cache_enabled: true
  cache_authenticated: true       # see Before you enable it, below
```

Setting	Values	Default	Description
`forward_client_auth`	boolean	`false`	Accept any well-formed bearer token on this route and forward it upstream unchanged

Behavior

Three things change on the route. Nothing else does.

- The client auth gate checks that a bearer token is present instead of matching `bearer_token`. No token, a bare `Bearer`, or another scheme still gets `401` with `WWW-Authenticate: Bearer`. Basic auth is not accepted here, even where it is configured.
- `Authorization` is not stripped, so the registry sees the caller's token
- `upstream_token` is ignored on this route. A route can forward the caller's credential or replace it, not both.

Packages are scanned the same as on any other route.

Before you enable it

- **The upstream decides who gets in.** Any well-formed bearer satisfies the gate, so the registry does the real access control, not the firewall.
- **This is not a way to add authentication.** With no `bearer_token` and no basic auth configured, the gate never runs and the route takes anonymous requests, same as any other route.
- **Turn on `metadata_filtering.cache_authenticated` if you use the filtered response cache** (`response_cache_enabled: true`). Every request on the route now carries `Authorization`, and by default the cache skips authenticated requests, so each one goes upstream. Setting it true keys the cache per credential and revalidates with the caller's token, so a revoked caller never gets cached private content.
- Credentials never reach the logs. The access log shows `[REDACTED]`.
- The scheme is matched case-sensitively, so a client sending `bearer` is rejected. npm and other mainstream clients send `Bearer`.

Response Tracking Headers

The firewall adds tracking headers to responses for downstream observability and end-to-end request correlation.

Request ID Header

Every response includes a request ID header for tracking. The header name is configurable via `socket.request_id_header` in `socket.yml` (default: `X-Socket-Request-ID`). The same header is also sent to upstream registries and the Socket API for end-to-end correlation.

YAML

```yaml
socket:
  request_id_header: X-Socket-Request-ID  # Default value; customize as needed
```

Header	Present On	Description
`X-Socket-Request-ID`	All responses	Unique request identifier (nginx `$request_id`). Header name configurable via `socket.request_id_header`.

Decision Headers

Security-checked requests (package downloads) include additional headers indicating the firewall's decision. Passthrough requests (metadata, checksums, default routes) do **not** include decision headers.

Header	Present On	Values / Description
`X-Socket-Decision`	Security-checked responses	`allowed` — Package passed security checks. `blocked` — Package blocked by security policy. `fail_open` — Socket API unavailable, package allowed due to fail-open policy.
`X-Socket-Block-Reason`	Blocked responses (403)	Comma-separated alert titles that caused the block (e.g., `malware,typosquat`).

Header	Present On	Values / Description
`X-Socket-Warn-Reason`	Allowed responses with warnings	Comma-separated alert titles with warn-level severity.
`X-Socket-Monitor-Reason`	Allowed responses with monitors	Comma-separated alert titles with monitor-level severity.
`X-Socket-Unscanned`	Unscanned package responses	`true` when the package/version was not found or not yet scanned by Socket. Only present when `socket.expose_unscanned_header: true` (default: false).

Examples:

Blocked package:

```
HTTP/1.1 403 Forbidden
X-Socket-Request-ID: a1b2c3d4e5f6...
X-Socket-Decision: blocked
X-Socket-Block-Reason: malware,typosquat
```

Allowed package with warnings:

```
HTTP/1.1 200 OK
X-Socket-Request-ID: a1b2c3d4e5f6...
X-Socket-Decision: allowed
X-Socket-Warn-Reason: protestware
```

Allowed package (no alerts):

```
HTTP/1.1 200 OK
X-Socket-Request-ID: a1b2c3d4e5f6...
X-Socket-Decision: allowed
```

Passthrough request (metadata/checksums):

```
HTTP/1.1 200 OK
X-Socket-Request-ID: a1b2c3d4e5f6...
```

Ports

YAML

```yaml
ports:
  http: 8080     # HTTP port (redirects to HTTPS)
  https: 8443    # HTTPS port
```

Environment variables:

Bash

```bash
HTTP_PORT=8080
HTTPS_PORT=8443
```

Deployment Mode

Controls path generation for different deployment topologies:

YAML

```yaml
# Default (downstream) - Client → Firewall → Registry
# Generates API paths for package manager clients
# No config_mode needed
```

```
# Upstream mode - Registry → Firewall → Public
# Generates direct paths for registry-to-registry communication
config_mode: upstream

# Middle mode - Registry → Firewall → Registry
# Generates both API and direct paths for multi-tier registries
config_mode: middle
```

Mode	Use When	Paths Generated	URL Rewriting
(default)	Client → FW → Registry	API paths	Yes
`upstream`	Private Registry → FW → Public	Direct paths	Yes
`middle`	Private Registry → FW → Private	Both API+Direct	No (proxy)

Environment variable:

Bash

```
CONFIG_MODE=upstream  # or 'middle'
```

Path-Based Routing

All registries behind a single domain with path prefixes. Recommended for most deployments.

YAML

```
path_routing:
  enabled: true
  domain: firewall.company.com

  routes:
    - path: /npm
      upstream: https://registry.npmjs.org
      registry: npm
      mode: rewrite  # 'rewrite' (default) or 'proxy'

    - path: /pypi
      upstream: https://pypi.org
      registry: pypi
      mode: rewrite

    - path: /maven
      upstream: https://repo1.maven.org/maven2
      registry: maven

    - path: /cargo
      upstream: https://index.crates.io
      registry: cargo

    - path: /rubygems
      upstream: https://rubygems.org
      registry: rubygems

    - path: /openvsx
      upstream: https://open-vsx.org
      registry: openvsx

    - path: /nuget
      upstream: https://api.nuget.org
      registry: nuget

    - path: /go
      upstream: https://proxy.golang.org
      registry: go

    - path: /conda
      upstream: https://repo.anaconda.com/pkgs/main
      registry: conda
```

Hostnames (`domain` and `allowed_domain`)

The firewall answers on the hostname in `domain` and, optionally, any extra hostnames listed in `allowed_domain`.

> **`allowed_domain` and `use_incoming_domain` require version 1.1.345 or higher.**

YAML

```yaml
path_routing:
  enabled: true
  domain: firewall.company.com     # Primary FQDN. Must be a single hostname.
  allowed_domain:                  # Extra hostnames the firewall answers on
    - lb.internal.company.com
    - '*.cluster.company.com'
    - localhost
```

Field	Default	Description
`domain`	(required)	Primary FQDN the firewall serves. Must be a single hostname: package download URLs in metadata responses are rewritten to this value, so multiple hostnames here produce malformed URLs that package managers reject (npm fails with `ERR_INVALID_URL`).
`allowed_domain`	`[]`	Additional hostnames the firewall answers on, beyond `domain`. Accepts a list or a single string. Entries may be exact hostnames, nginx wildcards (`*.cluster.company.com`), or an nginx regex (`~^.+$` matches any hostname). Entries containing whitespace or nginx metacharacters are dropped with a warning at startup. When empty, the firewall answers only on `domain`.
`use_incoming_domain`	`false`	Which hostname rewritten package URLs point at. With the default `false`, URLs are rewritten to `domain`, so downloads route back through the primary hostname even when the request arrived on an `allowed_domain` entry. Set `true` to rewrite using the incoming `Host` header instead.

Use case: the firewall sits behind a load balancer, and clients reach it through more than one hostname (the load balancer name, per-cluster names, or `localhost` during testing). Requests to hostnames not covered by `domain` or `allowed_domain` return 404.

> **Keep `domain` to exactly one hostname.** Put load balancer names, per-cluster hostnames, or `localhost` in `allowed_domain` instead. Listing several hostnames in `domain` breaks download URL rewriting for every route.

URL Rewrite Scheme

Control the URL scheme used when rewriting metadata URLs.

YAML

```yaml
path_routing:
  enabled: true
  domain: firewall.company.com
  rewrite_scheme: https            # Scheme for upstream connections (default: https)
  client_rewrite_scheme: http      # Scheme for client-facing URLs (optional)
```

Field	Default	Description
`rewrite_scheme`	`https`	Scheme used for upstream connections and URL rewriting
`client_rewrite_scheme`	(same as `rewrite_scheme`)	Scheme used in rewritten URLs returned to clients

Use case: When the firewall terminates SSL but clients connect via HTTP:

YAML

```yaml
path_routing:
  rewrite_scheme: https             # Upstream connections use HTTPS
  client_rewrite_scheme: http       # Rewritten URLs use HTTP for clients
```

The firewall also respects `X-Forwarded-Proto` and `X-Forwarded-Scheme` headers as fallbacks when `rewrite_scheme` is not set.

Environment variables:

Bash

```
PATH_ROUTING_REWRITE_SCHEME=https
PATH_ROUTING_CLIENT_REWRITE_SCHEME=http
```

Forward for Domain (Unmatched Path Handling)

Controls what happens to requests that don't match any configured route.

YAML

```yaml
path_routing:
  enabled: true
  domain: firewall.company.com
  upstream_fqdn: artifactory.company.com
  forward_for_domain: true    # Forward unmatched paths to upstream_fqdn (default: false)
```

`forward_for_domain`	`upstream_fqdn` set	Unmatched path behavior
`true`	Yes	Forwarded to `upstream_fqdn` uninspected (passthrough)
`true`	No	Returns 404
`false` (default)	Any	Returns 404

When auto-discovery is active (`private_registry` configured), `forward_for_domain` is **automatically enabled**. This means any repository type not explicitly routed (unsupported ecosystems, repos filtered by `include_pattern` / `exclude_pattern`, etc.) will still be forwarded to the upstream without Socket inspection.

This setting can also be used **without auto-discovery** — for example, with manual routes where you want unmatched paths forwarded to an Artifactory or Nexus instance:

YAML

```yaml
path_routing:
  enabled: true
  domain: firewall.company.com
  upstream_fqdn: artifactory.company.com
  forward_for_domain: true     # Forward docker, helm, raw, etc. without inspection
  routes:
    - path: /artifactory/api/npm/npm-remote
      upstream: https://artifactory.company.com/artifactory/api/npm/npm-remote
      registry: npm
    # Only npm is inspected; everything else passes through
```

Environment variable:

Bash

```
PATH_ROUTING_FORWARD_FOR_DOMAIN=true
```

Route Options

Field	Required	Description	Values
`path`	Yes	URL path prefix (must start with `/`)	`/npm`, `/pypi`, etc.
`upstream`	Yes	Upstream registry URL	HTTPS URL
`registry`	Yes	Registry type/ecosystem	npm, pypi, maven, etc.
`mode`	No	URL rewriting mode	`rewrite` (default), `proxy`
`upstream_token`	No	Env var holding the credential sent upstream. See Upstream Auth Token	Env var name
`forward_client_auth`	No	Forward the caller's own token upstream. See Forwarding the Caller's Own Credential	`false` (default), `true`

Route Mode: rewrite vs proxy

`mode: rewrite` (default) - Rewrites package URLs to point back through the firewall:

- Use for: Downstream, Upstream deployments
- URL in metadata → `https://firewall.company.com/npm/package.tgz`
- Clients fetch packages through firewall

`mode: proxy` - Passes URLs through unchanged:

- Use for: Middle deployments (Registry → FW → Registry)
- URL in metadata → `../../packages/package.tgz` (relative) or original upstream URL
- Downstream registry resolves relative URLs
- Required when using `config_mode: middle`

External Routes File

For 50+ routes or dynamic route management, use an external file:

YAML

```yaml
path_routing:
  enabled: true
  domain: firewall.company.com
  routes_file: /config/routes.yml
```

routes.yml format:

YAML

```yaml
routes:
  - path: /npm-public
    upstream: https://registry.npmjs.org
    registry: npm
  - path: /npm-internal
    upstream: https://nexus.company.com/repository/npm-internal
    registry: npm
  # ... many more routes
```

Mount the routes file:

YAML

```yaml
volumes:
  - ./routes.yml:/config/routes.yml:ro
```

Auto-Discovery (Artifactory/Nexus)

Automatically sync routes from your artifact manager. Routes update on interval without restarting the firewall!

YAML

```yaml
path_routing:
  enabled: true
  domain: firewall.company.com
  mode: artifactory  # or 'nexus'

  private_registry:
    api_url: https://artifactory.company.com/artifactory
    api_key: ${PRIVATE_REGISTRY_KEY}      # Token auth (takes precedence)
    # OR use basic auth with separate fields:
    username: ${PRIVATE_REGISTRY_USERNAME} # Basic auth username
    password: ${PRIVATE_REGISTRY_PASSWORD} # Basic auth password
    interval: 5m                          # Auto-sync interval (e.g., 30s, 5m, 1h)
    ignore_ssl_errors: false              # Disable verification of SSL when connecting to the
Private Registry
    include_pattern: ".*"                  # Include all repos (default)
    exclude_pattern: "(tmp|test)-.*"       # Exclude temp/test repos
    supported_ecosystems_only: true       # Skip unsupported package types (default: true)
```

Artifactory Auto-Discovery

Discovers REMOTE, LOCAL, and FEDERATED repositories in Artifactory and creates firewall routes automatically. VIRTUAL repositories are excluded by default (see `include_virtual`). Only REMOTE repos pointing to known public registries get native Socket scanning routes.

Supported repository types:

- npm
- pypi
- maven
- cargo
- rubygems
- nuget
- go
- conda (experimental support)

Example discovered routes:

```
/npm-public       → https://registry.npmjs.org
/pypi-public      → https://pypi.org
/maven-central    → https://repo1.maven.org/maven2
/cargo-crates     → https://index.crates.io
```

Nexus Auto-Discovery

Discovers proxy and hosted repositories in Nexus and creates firewall routes automatically. Group repositories are excluded by default (see `include_virtual`).

YAML

```yaml
path_routing:
  mode: nexus
  private_registry:
    api_url: https://nexus.company.com
    api_key: ${PRIVATE_REGISTRY_KEY}
    interval: 5m
```

Supported repository formats:

- npm
- pypi

- maven2
- cargo
- rubygems
- nuget
- go
- conda

Route naming: Routes are named after the repository name in Nexus (e.g., `/npm-proxy`, `/pypi-proxy`)

Auto-Discovery Configuration Options

All auto-discovery settings are configured in `socket.yml` under `path_routing.private_registry`:

YAML

```yaml
private_registry:
  api_url: https://artifactory.company.com    # Repository manager base URL (required)
  api_key: ${PRIVATE_REGISTRY_KEY}            # API key/token (takes precedence)
  # OR use basic auth with separate fields:
  username: ${PRIVATE_REGISTRY_USERNAME}       # Basic auth username
  password: ${PRIVATE_REGISTRY_PASSWORD}       # Basic auth password
  interval: 5m                                # Sync interval (default: 5m)
  ignore_ssl_errors: false                    # Disable SSL cert verification (default:
false)
  include_pattern: ".*"                        # Regex to include repos (default: all)
  exclude_pattern: "(tmp|test)-.*"             # Regex to exclude repos (default: none)
  supported_ecosystems_only: true             # Only route supported ecosystems (default:
true)
  include_virtual: false                      # Include VIRTUAL/group repos (default: false)
```

The `api_key` can also be provided via the `PRIVATE_REGISTRY_KEY` environment variable.

Authentication priority: `api_key` (or `PRIVATE_REGISTRY_KEY` env var) takes precedence. If `api_key` is not set, `username` and `password` are combined as `username:password` for basic auth.

`supported_ecosystems_only` (default: `true`)

Controls which discovered repositories get explicit firewall routes:

Value	Supported ecosystems (npm, pypi, maven, etc.)	Unsupported ecosystems (docker, helm, etc.)
`true`	Native route with Socket PURL scanning	No explicit route — forwarded uninspected via `forward_for_domain` catchall
`true` + `external_registry_cooldown.enabled: true`	Native route with Socket PURL scanning	Cooldown route **only if** repo matches an `external_registry_cooldown.registries` entry; otherwise skipped
`false`	Native route with Socket PURL scanning	Explicit passthrough route (forwarded without inspection)

REMOTE repository host validation: For REMOTE repos with supported package types, auto-discovery also verifies the remote URL points to a known public registry (e.g., `registry.npmjs.org` for npm). If the remote URL points elsewhere (e.g., Google AOSS, a private Artifactory, or any non-public host), the repo is:

- Downgraded to a **cooldown route** when `external_registry_cooldown.enabled: true`
- **Skipped** (no route) otherwise — traffic still reaches the upstream via the `forward_for_domain` catchall

This prevents repos that proxy non-public registries (like Google AOSS themes) from being treated as native ecosystems that Socket can scan.

`include_virtual` (default: `false`)

Controls whether VIRTUAL (Artifactory) or group (Nexus) repositories are included in auto-discovery.

Virtual/group repos aggregate multiple member repositories behind a single endpoint. They are excluded by default because:

- Routing through a virtual aggregator bypasses per-member allowlist and cooldown gating
- The concrete member repos (REMOTE, LOCAL, etc.) are already discovered individually

Set to `true` only when clients need to route through the virtual/group endpoint directly (e.g., downstream topologies).

Domain-Based Routing

Each registry gets its own subdomain. Requires multiple DNS records (or wildcard DNS) and certificates (or wildcard cert).

YAML

```yaml
registries:
  npm:
    domains: [npm.company.com]
    upstream: https://registry.npmjs.org  # Optional - defaults to public registry

  pypi:
    domains: [pypi.company.com, python.company.com]  # Multiple domains supported
    upstream: https://pypi.org

  maven:
    domains: [maven.company.com]

  cargo:
    domains: [cargo.company.com]

  rubygems:
    domains: [rubygems.company.com]

  openvsx:
    domains: [vsx.company.com]

  nuget:
    domains: [nuget.company.com]

  go:
    domains: [go.company.com]

  conda:
    domains: [conda.company.com]
```

Client usage:

Bash

```bash
npm config set registry https://npm.company.com/
pip config set global.index-url https://pypi.company.com/simple
```

DNS requirements:
Each domain needs an A or CNAME record pointing to the firewall host.

SSL requirements:
Either provide individual certs for each domain, or use a wildcard cert (`*.company.com`).

Caching

Local In-Memory Cache (Default)

```yaml
cache:
  ttl: 600  # Freshness window in seconds (10 minutes default)
  revalidation_lock_lease_seconds: 0  # 0 = auto (api_read_timeout + 30s)
  revalidation_jitter_seconds: 120    # Per-package spread on freshness boundary (0 =
disabled)
  revalidation_async: false           # false: revalidate expired entries live; true: serve
stale + refresh in background
  confirm_allow_mode: wait            # How an expired ALLOW is handled (see Confirm-stale-
ALLOW below)
```

Cached results are stored in nginx shared memory. Fresh for `ttl` seconds (plus optional jitter), then becomes stale but is retained for revalidation.

Environment variables:

```bash
SOCKET_CACHE_TTL=600
CACHE_REVALIDATION_LOCK_LEASE_SECONDS=0
CACHE_REVALIDATION_JITTER_SECONDS=120
CACHE_REVALIDATION_ASYNC=false
SOCKET_PURL_CONFIRM_ALLOW_MODE=wait
```

Redis Cache (Distributed)

For multi-instance deployments or persistent caching across restarts:

```yaml
redis:
  enabled: true
  host: redis.company.com
  port: 6379
  password: ${REDIS_PASSWORD}  # Optional
  db: 0  # Redis database number (default: 0)
  ttl: 86400   # Stale window in seconds (24 hours default)

  # SSL/TLS settings
  ssl: true
  ssl_verify: true
  ssl_ca_cert: /path/to/redis-ca.pem
  ssl_client_cert: /path/to/client-cert.pem  # For mTLS
  ssl_client_key: /path/to/client-key.pem   # For mTLS
  ssl_server_name: redis.company.com  # SNI hostname
```

Stale-while-revalidate behavior:

- **Fresh zone** (0 to `cache.ttl` + per-package jitter): Return cached value immediately
- **Stale zone** (`cache.ttl` to `redis.ttl` seconds): Serve stale immediately; one caller per package refreshes in the background (when `revalidation_async: true`) or synchronously (when `false`). Concurrent requests for the same stale package do not duplicate Socket API calls.
- **Expired** (after `redis.ttl`): Key removed by Redis, fetch fresh from Socket API

Async revalidation (`revalidation_async`): When `true` , every request receives a cached decision without waiting for the Socket API; the decision may be briefly stale until the background refresh completes. When `false` (the default), the lock holder refreshes synchronously on expiry (concurrent callers still get served stale to avoid a stampede). Neither setting fails open or closed — it serves the last known decision; true fail-open/closed only applies on cache **miss** when the API is unavailable.

Confirm-stale-ALLOW (`cache.confirm_allow_mode`)

An expired **BLOCK** is always safe to serve stale (a stale block is conservative). This setting controls how an expired **ALLOW** is handled, so a stale ALLOW cannot hide a package that has since become malicious. The circuit breaker state decides whether the Socket API is "degraded."

Mode	API healthy (breaker closed)	API degraded (breaker open)
`serve_stale`	Serve the stale ALLOW, refresh in background	Serve the stale ALLOW
`wait` *(default)*	Revalidate live; never serve the stale ALLOW	Serve the stale ALLOW — never call a degraded API
`confirm_when_degraded`	Revalidate live; never serve the stale ALLOW	Confirm live against the degraded API; on failure apply fail policy — never serve the stale ALLOW
`fail_when_degraded`	Revalidate live; never serve the stale ALLOW	Apply fail policy (`fail_open` / fail-closed); never call the API and never serve the stale ALLOW

The default `wait` confirms ALLOWs whenever the API is healthy, and never calls a degraded API. Choose `confirm_when_degraded` when you would rather re-check a stale ALLOW against a struggling API than serve it unverified. Choose `fail_when_degraded` (with `fail_open: false`) when a stale ALLOW during an outage should be blocked outright.

Environment variables:

```bash
REDIS_ENABLED=true
REDIS_HOST=redis
REDIS_PORT=6379
REDIS_PASSWORD=${REDIS_PASSWORD}
REDIS_DB=0
REDIS_TTL=86400
REDIS_SSL=true
REDIS_SSL_VERIFY=true
```

Proactive Cache Warming (optional)

> **Requires Redis.** Disabled by default (opt-in); no-op when Redis is disabled.

Cache warming refreshes cached decisions in the background **before** they leave the fresh zone, so more requests hit the fresh cache and Socket API load is spread evenly instead of spiking as many keys expire together.

```yaml
cache:
  warm_enabled: false           # Enable proactive warming (default: false; requires Redis)
  warm_interval: 60             # Seconds between warm cycles (default: 60)
```

Warming also honors the circuit breaker: while the Socket API breaker is open, warm cycles are skipped so a degraded API does not receive background warm traffic.

Environment variables:

```bash
CACHE_WARM_ENABLED=false
CACHE_WARM_INTERVAL=60
```

Circuit Breaker

Protects against a slow or erroring Socket API. When the verdict API trips the breaker, the firewall stops calling it and fast-fails — serving cached results immediately and applying your `fail_open` policy to uncached packages at once, instead of blocking each request for the full API timeout. A background health probe closes the breaker once the API recovers, backing off exponentially while it stays down. **Enabled by default.**

Example: During a Socket API latency spike, package installs can hang until each request hits `api_read_timeout`. With the circuit breaker enabled, after sustained high latency or error rate the

breaker opens: cached decisions continue to be served, and uncached packages follow `fail_open` immediately instead of waiting on a degraded API. When the API recovers, probes close the breaker and normal checking resumes.

YAML

```yaml
socket:
  resilience:
    circuit_breaker:
      enabled: true                      # Master switch (default: true)
      latency_ewma_threshold_ms: 10000   # Trip when smoothed latency stays at/above this
      ewma_alpha: 0.3                    # Latency smoothing factor (0..1)
      error_rate_threshold: 0.5          # Trip when error ratio reaches this...
      error_min_requests: 20             # ...once at least this many calls are in the window
      error_window_s: 10                 # Rolling error/total counter window
      open_window_s: 30                  # Base fast-fail duration before a recovery probe
      max_open_window_s: 300             # Cap on the backed-off fast-fail duration
      backoff_factor: 2                  # Open-window multiplier per failed probe sequence
      halfopen_probes: 3                 # Sequential healthy probes required to close
      halfopen_probe_spacing_s: 1        # Spacing between recovery probes
      probe_timeout_ms: 10000            # Recovery probe API timeout
```

Field	Default	Description
`enabled`	`true`	Master switch. Set `false` to disable the breaker entirely.
`latency_ewma_threshold_ms`	`10000`	Trip when smoothed API latency stays at or above this (milliseconds).
`ewma_alpha`	`0.3`	Latency smoothing factor (`0` – `1`). Higher values react faster but are noisier.
`error_rate_threshold`	`0.5`	Trip when the error ratio in the window reaches this (`0` – `1`).
`error_min_requests`	`20`	Minimum calls in the window before error-rate tripping is considered.
`error_window_s`	`10`	Rolling window (seconds) for error/total counters.
`open_window_s`	`30`	Base fast-fail duration before the first recovery probe.
`max_open_window_s`	`300`	Cap on the backed-off fast-fail duration.
`backoff_factor`	`2`	Multiplier applied to the open window after each failed probe sequence.
`halfopen_probes`	`3`	Consecutive healthy probes required to close the breaker.
`halfopen_probe_spacing_s`	`1`	Delay between sequential recovery probes (seconds).
`probe_timeout_ms`	`10000`	Per-probe API timeout so a probe cannot hang indefinitely.

Most deployments only need `enabled`; every threshold has a safe built-in default. Tripping is driven by **either** sustained latency **or** a sustained error rate — a one-off slow call or single error will not trip it.

Environment variables:

Bash

```bash
SOCKET_BREAKER_ENABLED=true              # Kill-switch: set false to disable entirely
SOCKET_BREAKER_LATENCY_EWMA_THRESHOLD_MS=10000
SOCKET_BREAKER_EWMA_ALPHA=0.3
SOCKET_BREAKER_ERROR_RATE_THRESHOLD=0.5
SOCKET_BREAKER_ERROR_MIN_REQUESTS=20
SOCKET_BREAKER_ERROR_WINDOW_S=10
SOCKET_BREAKER_OPEN_WINDOW_S=30
SOCKET_BREAKER_MAX_OPEN_WINDOW_S=300
SOCKET_BREAKER_BACKOFF_FACTOR=2
SOCKET_BREAKER_HALFOPEN_PROBES=3
```

```
SOCKET_BREAKER_HALFOPEN_PROBE_SPACING_S=1
SOCKET_BREAKER_PROBE_TIMEOUT_MS=10000
```

Nginx Performance

YAML

```yaml
nginx:
  worker_processes: 2        # Number of worker processes (match CPU cores)
  worker_connections: 4096   # Max concurrent connections per worker
  http2: true                # Enable HTTP/2 on HTTPS listeners (default: true)
```

HTTP/2 is enabled by default on all HTTPS listeners. Modern package managers (for example `uv`, `pnpm`, and `bun`) can multiplex many parallel requests over a single TLS connection, reducing connection overhead. HTTP/2 is negotiated only on HTTPS; HTTP/1.1 clients continue to work unchanged. Upstream connections to registries remain HTTP/1.1. Disable with `nginx.http2: false` if your environment requires HTTP/1.1 only.

Resource-Based Recommendations

Resources	worker_processes	worker_connections	Est. Throughput
1 CPU / 1 GB RAM	1	512	~30 req/s
2 CPU / 2 GB RAM	2	1024	~60 req/s
4 CPU / 4 GB RAM	4	4096	~100 req/s
8 CPU / 8 GB RAM	8	8192	~170 req/s
16 CPU / 16 GB	16	16384	~300 req/s

Environment variables:

Bash

```
WORKER_PROCESSES=2
WORKER_CONNECTIONS=4096
```

Proxy Timeouts

Configure timeouts for upstream registry connections. These are separate from `socket.api_*_timeout` settings, which apply only to Socket.dev API calls.

YAML

```yaml
proxy:
  connect_timeout: 60  # Seconds to establish connection
  send_timeout: 60     # Seconds to send request
  read_timeout: 60     # Seconds to read response

  # Buffer sizes (advanced)
  buffer_size: 4k      # Initial buffer for response headers
  buffers_count: 8     # Number of buffers for response body
  buffers_size: 4k     # Size of each buffer
  busy_buffers_size: 8k  # Buffers that can be sent to client while reading
```

For large packages (e.g., Maven artifacts > 100MB):

YAML

```yaml
proxy:
  connect_timeout: 120
  send_timeout: 300
  read_timeout: 300
```

Environment variables:

Bash

```
PROXY_CONNECT_TIMEOUT=60
PROXY_SEND_TIMEOUT=60
PROXY_READ_TIMEOUT=60
```

Forward Proxy

Optional HTTP `CONNECT` listener for placing the firewall behind a CASB-style forward proxy (for example Netskope or Zscaler). When enabled, the CASB tunnels registry traffic to the firewall; the firewall terminates the tunnel and scans the inner request as usual. Leave disabled unless you need this topology.

YAML

```yaml
forward_proxy:
  enabled: false                   # Enable CONNECT listener (default: false)
  port: 3128                       # External CONNECT listener port
  proxy_protocol_port: 8081        # Internal loopback port that terminates TLS
  max_connections_per_source: 64   # Concurrent tunnels allowed per client IP
  max_tunnel_lifetime_seconds: 600 # Hard deadline before a tunnel is closed
```

Field	Default	Description
`enabled`	`false`	Enable the HTTP CONNECT forward-proxy listener
`port`	`3128`	Port the CASB (or other forward proxy) dials for CONNECT tunnels
`proxy_protocol_port`	`8081`	Internal loopback port used to terminate TLS for the inner request (not exposed externally)
`max_connections_per_source`	`64`	Maximum concurrent tunnels allowed per client IP
`max_tunnel_lifetime_seconds`	`600`	Maximum lifetime of a single tunnel before it is force-closed

SASE Resolve Endpoint

Lets a SASE / Secure Web Gateway (for example Zscaler) steer blocked registry traffic into the firewall **without reconfiguring developer machines**. Many gateways can block a URL and, instead of returning an error, redirect the user to a host you control — passing the originally requested URL along in a query parameter. The firewall exposes a `/socket/resolve` endpoint that reads that URL, looks up the protected path for the same package, and redirects the client to it. The request then flows through the firewall's normal scanning and blocking pipeline exactly as if the client had been pointed at the firewall directly.

The endpoint is generated automatically from your `path_routing` configuration — every route that has both an `upstream` and a `path` becomes resolvable. There is no separate feature flag to enable.

Flow:

```
Developer / CI  --requests-->  registry.npmjs.org/lodash
      |
      v
SASE / Secure Web Gateway (e.g. Zscaler)   URL blocked -> redirect to
      |                                     firewall.example.com/socket/resolve?url=
<original, url-encoded>
      v
Socket Firewall  /socket/resolve   maps original URL -> protected path, 302 -> /npm/lodash
      v
Socket Firewall  /npm/lodash       scan -> allow / block (normal pipeline)
```

How the gateway must call it. The endpoint reads the originally requested URL from a `url` query parameter (`/socket/resolve?url=<original absolute URL>`). That is the only supported form.

Zscaler is compatible. Create a URL policy for the public registry hosts you want to protect (for example `registry.npmjs.org` , `pypi.org`), set the action to **block**, and set the redirect URL to:

```
https://firewall.example.com/socket/resolve
```

Zscaler appends `?url=...` itself — do not add it manually.

Responses:

Situation	Response
`url` matches a configured route	`302` redirect to the protected path (e.g. `/npm/lodash`)
`url` parameter missing or empty	`400 Missing url parameter`
`url` present but matches no configured route	`400 Cannot redirect <url>`

Only URLs for registries you have added to `path_routing` resolve; a URL for an unconfigured registry returns `400`. Direct requests to the firewall are unaffected — only the gateway auto-redirect path depends on the route being present.

Client IP Detection (`client_ip`)

When the firewall sits behind a load balancer or reverse proxy, `$remote_addr` will be the proxy's IP address — not the real client. The `client_ip` section configures nginx's `ngx_http_realip_module` to extract the true client IP from a trusted header.

Once configured, **all** logging, telemetry, webhook events, Splunk HEC events, and the `SOCKET_DECISION` log field `client_ip` automatically reflect the real client IP. No Lua code changes are needed — the module transparently replaces `$remote_addr`.

Configuration

YAML

```yaml
client_ip:
  header: X-Forwarded-For       # Header containing the real client IP
  trusted_proxies:               # CIDR ranges of trusted proxies
    - 10.0.0.0/8
    - 172.16.0.0/12
    - 192.168.0.0/16
  recursive: true                # Walk the header chain to find first untrusted IP (default:
true)
```

Options

Setting	Type	Default	Description
`header`	string	*(none)*	HTTP header to read client IP from. Common values: `X-Forwarded-For`, `X-Real-IP`, `CF-Connecting-IP` (Cloudflare), `True-Client-IP` (Akamai)
`trusted_proxies`	string[]	`[]`	List of CIDR ranges whose IPs are trusted to set the client IP header. **Required** when `header` is set.
`recursive`	bool	`true`	When `true` and `header` is `X-Forwarded-For`, nginx walks the comma-separated IP chain from right to left, skipping IPs that match `trusted_proxies`, and uses the first untrusted IP.

Environment Variables

Bash

```bash
CLIENT_IP_HEADER=X-Forwarded-For
CLIENT_IP_TRUSTED_PROXIES=10.0.0.0/8,172.16.0.0/12,192.168.0.0/16
CLIENT_IP_RECURSIVE=true
```

Examples

Behind an AWS ALB:

```yaml
client_ip:
  header: X-Forwarded-For
  trusted_proxies:
    - 10.0.0.0/8       # VPC CIDR
  recursive: true
```

Behind Cloudflare:

```yaml
client_ip:
  header: CF-Connecting-IP
  trusted_proxies:
    - 173.245.48.0/20
    - 103.21.244.0/22
    - 103.22.200.0/22
    - 103.31.4.0/22
    - 141.101.64.0/18
    - 108.162.192.0/18
    - 190.93.240.0/20
    - 188.114.96.0/20
    - 197.234.240.0/22
    - 198.41.128.0/17
    - 162.158.0.0/15
    - 104.16.0.0/13
    - 104.24.0.0/14
    - 172.64.0.0/13
    - 131.0.72.0/22
```

Behind a single known proxy (e.g., nginx ingress controller):

```yaml
client_ip:
  header: X-Real-IP
  trusted_proxies:
    - 10.0.1.5/32       # Proxy IP
```

How It Works

1. **No `client_ip` configured** — `$remote_addr` is used as-is (direct client connection).
2. **`client_ip.header` + `trusted_proxies` set** — nginx's `ngx_http_realip_module` generates:

   ```nginx
   set_real_ip_from 10.0.0.0/8;
   set_real_ip_from 172.16.0.0/12;
   real_ip_header X-Forwarded-For;
   real_ip_recursive on;
   ```

3. When a request arrives from a trusted proxy IP, nginx replaces `$remote_addr` with the value from the configured header. Lua's `ngx.var.remote_addr` reflects this automatically.

Security Notes

- **Always restrict `trusted_proxies`** to your actual proxy/LB CIDR ranges. If set too broadly (e.g., `0.0.0.0/0`), any client can spoof their IP via the header.
- The `header` value is only trusted when the request comes from an IP in `trusted_proxies`.
- Without `trusted_proxies`, the `header` setting is ignored (fail-safe).

Metadata Filtering

> **Requires a Redis instance for caching.**

> **Requires version 1.1.176 or higher.**

Remove blocked or warned package versions from registry metadata responses before clients see them, preventing installation attempts of unsafe packages entirely. Supports all 9 ecosystems with ecosystem-appropriate filtering granularity.

YAML

```yaml
metadata_filtering:
  enabled: true
  filter_blocked: true              # Remove blocked/error packages from metadata
  filter_warn: false                # Keep warned packages visible (show warnings only)
  include_unchecked_versions: true  # Include versions not yet checked by Socket (default:
true)
  max_versions: 100                 # Max versions to check per package (default: 100, newest
first)
  cache_ttl: 3600                   # Cache TTL for metadata PURL lookups (default: 3600s = 1
hour)
  batch_size: 100                   # Max PURLs per version-check batch (default: 100, capped
at 100)
  max_body_size: 500m              # Max metadata response body size for filtering (default:
500m)
  package_filter_timeout: 60        # Time budget in seconds per filter operation (default:
60) (v1.1.323+)
  semaphore_wait_timeout: 60        # Max wait for a filter slot (default: min(60,
package_filter_timeout))
  package_filter_retry: 0           # Retries per failed batch; 0 = unlimited within the time
budget (v1.1.323+)
  max_concurrent: 3                 # Concurrent filter operations per nginx worker (default:
3) (v1.1.323+)
  metadata_concurrent_batch: 1      # Concurrent version-check batches per operation
(default: 1) (v1.1.327+)
  prefetch_enabled: true            # Enable/disable background prefetch for conda metadata
(default: true)
  prefetch_ttl: 600                 # Prefetch refresh interval in seconds (0 = always check
upstream, >0 = check every N seconds)
  prefetch_max_concurrent: 2        # Max concurrent prefetch operations across all workers
(default: 2)
  prefetch_batch_concurrency: 4     # Max concurrent PURL batch API calls during metadata
filtering (default: 4)

  # Filtered response body cache (disabled by default — opt-in)
  response_cache_enabled: false     # Cache filtered metadata responses
  response_cache_ttl: 600           # Max age (seconds) before a full re-filter; 0 = inherit
cache_ttl
  response_cache_fresh: 5           # Serve-without-upstream window (seconds)
  cache_authenticated: false        # Cache responses that forward Authorization upstream
  response_cache_excluded_ecosystems: []  # Ecosystems to skip for response caching
  confirm_allow_mode: wait          # How a stale ALLOW body is revalidated (see below)
```

Options

Field	Default	Description
`enabled`	`false`	Enable metadata filtering
`filter_blocked`	`true`	Remove packages with `block` or `error` actions from metadata
`filter_warn`	`false`	Remove packages with `warn` actions from metadata
`include_unchecked_versions`	`true`	Keep versions not yet scanned by Socket (`false` = strict security posture)
`max_versions`	`100`	Max versions to check per package (newest first; older versions kept as-is)
`cache_ttl`	`3600`	Cache TTL in seconds for metadata PURL lookups (separate from download TTL)

Field	Default	Description
`batch_size`	`100`	Max PURLs per version-check batch. Values above 100 are capped to 100.
`max_body_size`	`500m`	Max metadata response body size for filtering (supports `k`, `m`, `g` suffixes)
`package_filter_timeout`	`60`	Time budget in seconds for one metadata filter operation (v1.1.323+). When it expires, in-flight checks are killed and the remaining versions are treated as unchecked, following `include_unchecked_versions`. Raise this for very large packages whose indexes list thousands of files.
`semaphore_wait_timeout`	`min(60, package_filter_timeout)`	Max seconds a request waits for a free filter slot before giving up. Independent of `package_filter_timeout`: defaults to about 60s (a typical client request timeout) so waiters stop once the client itself would give up, instead of queuing for the full work budget. Capped at `package_filter_timeout`.
`package_filter_retry`	`0`	Max retries per failed PURL batch (v1.1.323+). `0` = unlimited retries within the time budget.
`max_concurrent`	`3`	Max metadata filter operations running at once per nginx worker (v1.1.323+). Additional requests wait for a slot, up to `semaphore_wait_timeout`.
`metadata_concurrent_batch`	`1`	PURL batches checked concurrently within one filter operation (v1.1.327+). Prefetch operations use `prefetch_batch_concurrency` instead.
`prefetch_enabled`	`true`	Enable/disable background prefetch for conda metadata. Set to `false` to disable all prefetch timers.
`prefetch_ttl`	`600`	Prefetch refresh interval in seconds. `0` = always check upstream for changes (does NOT disable prefetch). `>0` = check every N seconds.
`prefetch_max_concurrent`	`2`	Max concurrent prefetch operations across all nginx workers. Prevents worker exhaustion on startup.

Field	Default	Description
`prefetch_batch_concurrency`	`4`	Max concurrent PURL batch API calls during metadata filtering. Higher values speed up large metadata like lodash or conda repodata.
`response_cache_enabled`	`false`	Cache filtered metadata response bodies (opt-in; disabled by default).
`response_cache_ttl`	`600`	Max age (seconds) before a cached filtered body is fully re-filtered. `0` = inherit `cache_ttl`.
`response_cache_fresh`	`5`	Seconds to serve a cached body without contacting the upstream registry (coalesces rapid repeats).
`cache_authenticated`	`false`	When `true`, also cache responses that forward `Authorization` upstream (namespaced per credential).
`response_cache_excluded_ecosystems`	`[]`	Ecosystems excluded from response-body caching.
`confirm_allow_mode`	`wait`	How a stale ALLOW filtered body is revalidated. Same modes as `cache.confirm_allow_mode`: `wait` (default), `serve_stale`, `confirm_when_degraded`, `fail_when_degraded`.

Filtered Response Cache

When `response_cache_enabled` is `true`, the firewall caches the *filtered* metadata document so repeated requests for the same package are served quickly instead of being re-downloaded and re-filtered. Disabled by default — enable only when you understand the freshness tradeoff.

- **`response_cache_fresh`**: Within this window, cached bodies are served without contacting the upstream registry.
- **`response_cache_ttl`**: After the freshness window, the body may still be served until this max age, subject to `confirm_allow_mode`. Past `response_cache_ttl`, the response is fully re-filtered against current Socket verdicts.
- **`confirm_allow_mode`**: Parallel to the PURL cache setting. The default `wait` re-checks an ALLOW body when the API is healthy, and serves stale when the circuit breaker reports a degraded API. Set `serve_stale` for lower latency under load when listings may lag verdict changes by up to `response_cache_ttl`.

Downloads are always verified against current verdicts at request time — response caching only affects how quickly verdict changes appear in metadata *listings*.

How It Works

1. Client requests package metadata (e.g., `npm install lodash`, `pip install requests`)
2. Firewall fetches the full metadata response from the upstream registry
3. Extracts all package versions/artifacts and builds Package URLs (PURLs)
4. Calls the Socket API in batches to check security status of each version
5. Removes blocked (and optionally warned) versions from the response
6. Returns sanitized metadata to the client

When filtering is disabled, responses stream through unchanged with no buffering.

Supported Ecosystems

Per-artifact filtering — individual artifacts within a version can be selectively removed:

Ecosystem	Metadata Format	Notes
PyPI	HTML (PEP 503) and JSON (PEP 691) `/simple/{package}/`	Filters tar.gz, wheels, eggs, zips independently
Maven	HTML directory listings and `maven-metadata.xml`	Filters by classifier and type (e.g., `?classifier=sources&type=jar`)

Per-version filtering — if any artifact is blocked, the entire version is removed:

Ecosystem	Metadata Format	Notes
npm	Package JSON (`/{package}`)	Filters `versions`, `dist-tags`, and `time` objects
NuGet	Registration catalog JSON (`/v3/registration5-gz-semver2/`)	Removes entries from catalog pages
RubyGems	CompactIndex (`/info/{gem}`) and JSON API (`/api/v1/versions/`)	Line-based and JSON array formats
Cargo	Sparse index NDJSON (`/{2-chars}/{2-chars}/{crate}`)	One version per NDJSON line
Go	`/@v/list` (newline-separated) and `/@latest` (JSON)	Source-only, no binary artifacts
Conda	`repodata.json` (`packages` and `packages.conda` objects)	Uses PyPI PURLs (Socket API fallback)
OpenVSX	Extension detail JSON (`/api/{namespace}/{extension}`)	Single .vsix per version

Use Cases

- Prevent accidental installation of malicious or vulnerable packages
- Remove flagged packages from search results and dependency resolution
- Enforce strict security posture by excluding unchecked versions (`include_unchecked_versions: false`)
- Pre-warm the PURL cache from metadata lookups, speeding up subsequent download checks

Environment Variables

Bash

```
METADATA_FILTERING_ENABLED=true
METADATA_FILTER_BLOCKED=true
METADATA_FILTER_WARN=false
METADATA_INCLUDE_UNCHECKED_VERSIONS=true
METADATA_MAX_VERSIONS=100
METADATA_CACHE_TTL=3600
METADATA_FILTER_BATCH_SIZE=100
METADATA_MAX_BODY_SIZE=524288000       # 500MB in bytes (default)
METADATA_PACKAGE_FILTER_TIMEOUT=60
METADATA_SEMAPHORE_WAIT_TIMEOUT=60
METADATA_PACKAGE_FILTER_RETRY=0
METADATA_MAX_CONCURRENT=3
METADATA_CONCURRENT_BATCH=1
METADATA_RESPONSE_CACHE_ENABLED=false
METADATA_RESPONSE_CACHE_TTL=600
METADATA_RESPONSE_CACHE_FRESH=5
METADATA_CACHE_AUTHENTICATED=false
METADATA_RESPONSE_CACHE_EXCLUDED_ECOSYSTEMS=
METADATA_CONFIRM_ALLOW_MODE=wait
```

Per-Ecosystem `recentlyPublished` Downgrade

> **Requires version 1.1.134 or higher.**

Control which ecosystems have the `recentlyPublished` alert downgraded from its API-assigned action to `warn`, allowing the package through instead of blocking it.

By default (empty or omitted), the API action is respected as-is for all ecosystems. When one or more ecosystems are listed, `recentlyPublished` alerts are downgraded to `warn` **only** for those ecosystems; all other ecosystems continue to use the API-assigned action.

YAML

```yaml
socket:
  # Downgrade recentlyPublished to warn only for npm and pypi;
  # all other ecosystems use the API-assigned action
  recently_published_enabled_ecosystems:
    - npm
    - pypi
```

Options

Field	Default	Description
`recently_published_enabled_ecosystems`	`[]` (empty)	List of ecosystems where `recentlyPublished` alerts are downgraded to `warn`. Ecosystems not listed respect the API-assigned action. When empty or omitted, all ecosystems respect the API action.

Valid Ecosystem Values

`npm`, `pypi`, `maven`, `cargo`, `rubygems`, `nuget`, `go`, `conda`, `openvsx`

Behavior

Configuration	Effect
Empty or omitted	All ecosystems use the API-assigned action (default — no downgrade)
One or more ecosystems listed	`recentlyPublished` is downgraded to `warn` for listed ecosystems; all others use the API-assigned action

When a `recentlyPublished` alert is downgraded:

- The package is **allowed** through (not blocked)
- A warning is logged with the original and downgraded action
- The `X-Socket-Warn-Reason` response header includes `recentlyPublished`
- Splunk, webhook, and telemetry events reflect the downgraded `warn` action

Examples

Downgrade recentlyPublished to warn only for npm:

YAML

```yaml
socket:
  recently_published_enabled_ecosystems:
    - npm
```

Downgrade recentlyPublished to warn for npm, pypi, and maven:

YAML

```yaml
socket:
  recently_published_enabled_ecosystems:
    - npm
```

```
    - pypi
    - maven
```

Environment Variable

Bash

```
RECENTLY_PUBLISHED_ENABLED_ECOSYSTEMS=npm,pypi   # Comma-separated list
```

Per-Ecosystem Parameters

Fine-tune behavior for specific ecosystems using the `ecosystem_params` section under `socket:`. Currently supports conda-specific settings for managing packages that lack native Socket API coverage.

YAML

```
socket:
  ecosystem_params:
    conda:
      use_private_created_at: true  # Use private registry timestamps for cooldown
enforcement (default: true)
      allow_unknown: true           # Allow unscanned packages with a warn alert (default:
true)
```

Options

Field	Default	Description
`use_private_created_at`	`true`	When enabled, the conda parser queries the private registry (Nexus/Artifactory) for package publish timestamps and applies cooldown enforcement. Requires `external_registry_cooldown.enabled: true`. Set to `false` to skip private registry cooldown checks for conda.
`allow_unknown`	`true`	Overrides the global `fail_open_unscanned` setting for this ecosystem. When `true`, unscanned packages (Socket API returns `purlError`) are allowed with a `warn` action and Splunk/telemetry events. When `false`, unscanned packages are blocked regardless of the global setting. When not set, falls back to `fail_open_unscanned`.

Environment Variable

Bash

```
SOCKET_ECOSYSTEM_PARAMS='{"conda":{"use_private_created_at":true,"allow_unknown":true}}'  #
JSON
```

External Registry Cooldown

For registries not natively supported by Socket — or private registries where Socket hasn't scanned packages — the cooldown system blocks recently-published packages. Packages published within a configurable cooldown window are blocked, protecting against supply-chain attacks that rely on recently-published malicious packages.

The cooldown system is a **replacement** for the Socket PURL API for unsupported registries. For ecosystems Socket doesn't natively support, the cooldown system is the security check.

Modes

The cooldown system supports two modes:

API mode (default when `socket.deployment` is configured):

- Sends PURLs to the centralized Socket cooldown API endpoint

- Each component must include both a `registry-name` and a `purl`, e.g. `{components: [{"registry-name": "<name>", "purl": "pkg:..."}]}` — unlike the standard batch PURL API, the cooldown endpoint additionally requires a `registry-name` per component
- Falls back to the local daemon if the API is unreachable
- Does **not** require Redis (uses Redis for caching when available)

Local mode (legacy, used when deployment is not configured):

- Runs a local Python daemon that queries external registries directly for publish dates
- Communicates with nginx via Redis queue
- Requires Redis

How It Works

1. Auto-discovery (Artifactory/Nexus) detects repos with unsupported package types
2. Routes for those repos are tagged as `cooldown`-checked instead of `passthrough`
3. When a package is requested, the firewall checks Redis cache for the cooldown result
4. On cache miss:
   - **API mode**: HTTP POST to `POST /v0/orgs/{org}/firewall/{deployment}/cooldown`
   - **Local mode**: Push to Redis queue → daemon queries registry → response via Redis
5. If `recentlyPublished` alert with `action: error` → **blocked** (403)
6. If no `recentlyPublished` alert → **allowed** (proxied upstream)

Configuration

YAML

```yaml
external_registry_cooldown:
  enabled: true                    # Enable cooldown checks (default: false)
  mode: api                        # "api" (default when deployment set) or "local" (legacy daemon)
  cooldown_period: 7d              # Block packages published within this window (default: 7d)
  check_interval: 60               # Queue polling interval in seconds (default: 60, local mode only)
  redis_key_prefix: "cooldown:"    # Redis key prefix (default: "cooldown:")
  cache_ttl: 86400                 # Cache TTL for cooldown results (default: 86400 = 24h)
```

Options

Field	Default	Description
`enabled`	`false`	Enable cooldown checks
`mode`	auto	`api` (Socket API), `local` (daemon). Auto-detected from `socket.deployment` if not set
`cooldown_period`	`7d`	Block packages published within this window (e.g., `30s`, `5m`, `1h`, `3d`, `7d`)
`check_interval`	`60`	Seconds between queue polling cycles (local mode only)
`redis_key_prefix`	`"cooldown:"`	Prefix for Redis keys storing cooldown data
`cache_ttl`	`86400`	Cache TTL in seconds for cooldown results (24 hours)
`fallback`	`""`	Use private registry as publish-date source (see Publish-Date Fallback)

Explicit External Registries

Query external registries directly for package publish dates:

YAML

```yaml
external_registry_cooldown:
  enabled: true
```

```yaml
  cooldown_period: 7d

  registries:
    - name: internal-pypi
      url: https://pypi.internal.company.com
      ecosystem: pypi
      auth_type: bearer_token       # none (default), bearer_token, basic
      auth_credential: INTERNAL_PYPI_TOKEN  # Env var name
      rate_limit_header: X-RateLimit-Remaining  # Optional header for remaining rate limit
      ignore_ssl_errors: false      # Skip TLS verification for this registry (default:
false)
      timeout: 30                   # HTTP timeout for publish-date queries (seconds)
      cooldown_period: 3d           # Per-registry override (optional)

    - name: private-npm
      url: https://npm.internal.company.com
      ecosystem: npm
      auth_type: basic
      auth_credential: NPM_REGISTRY_CREDS  # Env var (user:pass format)
```

Field	Required	Description
`name`	Yes	Unique name for this registry (used in logs and cache keys)
`url`	Yes	Base URL of the registry API
`ecosystem`	Yes	Package ecosystem: `npm`, `pypi`, `maven`, `cargo`, `rubygems`, `nuget`, `go`, `conda`
`auth_type`	No	Authentication type: `none` (default), `bearer_token`, `basic`
`auth_credential`	No	Name of environment variable containing the credential
`rate_limit_header`	No	Optional response header name used to read remaining rate-limit budget
`ignore_ssl_errors`	No	Skip TLS certificate verification for this registry (default: `false`)
`timeout`	No	HTTP timeout in seconds for publish-date queries (default: `30`)
`cooldown_period`	No	Override the global cooldown period for this registry

Private Registry Auto-Discovery

Reuses the existing Artifactory/Nexus connection from `path_routing.private_registry` to discover unsupported repos and check their import timestamps:

YAML

```yaml
external_registry_cooldown:
  enabled: true

  private_registry:
    enabled: true                     # Enable auto-discovery for cooldown
    source: auto                      # auto | artifactory | nexus
    auth_type: bearer_token           # bearer_token (default) | basic
    auth_credential: ARTIFACTORY_TOKEN  # Env var name (falls back to
path_routing.private_registry credentials when unset)
    include_unsupported_only: true    # Only repos skipped by supported_ecosystems_only
    include_pattern: ".*"             # Regex for repo names to include
    exclude_pattern: "^$"             # Regex for repo names to exclude
```

Field	Default	Description
`enabled`	`false`	Enable cooldown via private registry auto-discovery
`source`	`auto`	Registry type: `auto` (detect from `path_routing.mode`), `artifactory`, `nexus`
`auth_type`	`bearer_token`	Authentication type for cooldown queries: `bearer_token` or `basic`

Field	Default	Description
`auth_credential`	*(none)*	Name of environment variable containing the credential. When unset, falls back to `path_routing.private_registry` credentials.
`include_unsupported_only`	`true`	`true` = only repos with unsupported ecosystems; `false` = all repos
`include_pattern`	`".*"`	Regex filter for repo names to include
`exclude_pattern`	`"^$"`	Regex filter for repo names to exclude

Both modes (explicit registries and private registry auto-discovery) can be active simultaneously. Explicit entries take precedence by name.

Supported Ecosystem Plugins

Each ecosystem has a dedicated plugin that knows how to query the registry API for publish dates:

Ecosystem	API Used
npm	Packument `.time.{version}` field
PyPI	JSON API `/pypi/{name}/json` → `upload_time_iso_8601`
Maven	`maven-metadata.xml` `<lastUpdated>` + POM Last-Modified
Cargo	crates.io API `.version.created_at`
RubyGems	Versions API `.created_at`
NuGet	Registration index `.catalogEntry.published`
Go	Proxy `.info` endpoint `.Time`
Conda	`repodata.json` `.timestamp` field
Artifactory	File Info API + AQL search (import timestamps)
Nexus	Components API + Search Assets API

Redis Communication

The firewall uses a cache-first strategy for low latency:

1. **Cache hit** (fast path, ~1ms): Direct Redis lookup `cooldown:{ecosystem}:{name}:{version}`
2. **Cache miss**: Push request to `COOLDOWN_QUEUE`, wait for daemon response (configurable timeout, default 2s)
3. **Timeout**: Allow through (fail-open) — no blocking on transient daemon failures

Decision Logging

Cooldown decisions are logged with full parity to Socket API decisions:

- `[SOCKET_DECISION]` log entries include `block_source: cooldown`
- Splunk HEC events include a synthesized `recentlyPublished` alert
- Webhook payloads include cooldown metadata
- Socket telemetry events include cooldown status

Publish-Date Fallback

When the primary ecosystem plugin can't determine a package's publish date, the cooldown system can fall back to querying the configured private registry (Artifactory or Nexus) to determine when the artifact was first imported.

This is useful when:

- The ecosystem plugin's public registry is unreachable
- The package exists only in the private registry (no public registry entry)
- The plugin can't parse the upstream response

The fallback uses `path_routing.private_registry` connection config, so no extra auth is needed.

YAML

```yaml
external_registry_cooldown:
  enabled: true
  fallback: external    # Use private registry as primary date source
```

Fallback Modes

Value	First lookup	Fallback (if first returns None)
`""`	Ecosystem plugin only	— (no fallback)
`"external"`	Private registry (auto-detect type)	Ecosystem plugin
`"artifactory"`	Ecosystem plugin	Artifactory AQL
`"nexus"`	Ecosystem plugin	Nexus REST Search API

`external` (recommended): Auto-detects whether to use Artifactory AQL or Nexus REST from `path_routing.mode`. Queries the private registry **first** — if it has the artifact's import date, that's used. Falls back to the ecosystem plugin if the private registry returns nothing.

`artifactory` / **`nexus`**: Queries the ecosystem plugin **first** (e.g., Maven Central metadata), and only tries the private registry if the plugin returns no date.

How Each Registry Is Queried

Artifactory AQL: Queries `items.find()` with a name/version pattern. Searches both the repo and its `-cache` variant (remote repos store cached artifacts in `<repo>-cache`). Returns the earliest `created` timestamp. Compatible with Artifactory Pro and OSS.

Nexus REST: Queries `/service/rest/v1/search?repository={repo}&name={name}&version={version}`. Returns the earliest `blobCreated` timestamp (falls back to `lastModified` if `blobCreated` is unavailable).

Environment Variables

Bash

```bash
COOLDOWN_ENABLED=true
COOLDOWN_PERIOD=7d
COOLDOWN_CHECK_INTERVAL=60
COOLDOWN_REDIS_KEY_PREFIX=cooldown:
COOLDOWN_CACHE_TTL=86400
COOLDOWN_FALLBACK=external
COOLDOWN_REGISTRIES='[{"name":"internal-
pypi","url":"https://pypi.internal.company.com","ecosystem":"pypi"}]'
COOLDOWN_PRIVATE_REGISTRY_ENABLED=true
COOLDOWN_PRIVATE_REGISTRY_SOURCE=auto
COOLDOWN_PRIVATE_REGISTRY_UNSUPPORTED_ONLY=true
```

Splunk Integration

Forward security events to Splunk HTTP Event Collector (HEC):

YAML

```yaml
splunk:
  enabled: true
  hec_url: https://splunk.company.com:8088/services/collector/event
  hec_token: ${SPLUNK_HEC_TOKEN}
  index: security           # Splunk index name (optional, no default)
  source: socket-firewall   # Splunk source name
  sourcetype: socket:firewall:event  # Splunk sourcetype (default: socket:firewall:event)

  # SSL settings
  ssl_verify: true
  ssl_ca_cert: /path/to/splunk-ca.pem
```

```
  # Event batching
  batch_size: 1           # Events per batch (default: 1)
```

Event types logged:

- Package blocks (malicious/supply-chain attacks)
- Package warnings
- API errors
- Cache hits/misses
- Request/response metadata

Example Splunk event:

JSON


```
{
  "time": 1709078400,
  "event": {
    "event_type": "package_check",
    "purl": "pkg:npm/malicious-package@1.0.0",
    "decision": "blocked",
    "action": "block",
    "response_code": 403,
    "upstream_status": null,
    "block_source": "download",
    "block_reason": "Known Malware",
    "warn_reason": "",
    "repo": "npm-remote",
    "client_ip": "203.0.113.10",
    "user_agent": "npm/8.19.2",
    "request_id": "abc123xyz",
    "upstream_host": "registry.npmjs.org",
    "source_path": "/repository/npm/malicious-package/-/malicious-package-1.0.0.tgz",
    "cached": false,
    "stale": false,
    "socket_api_response_code": 403,
    "purl_check_latency_ms": 142,
    "private_registry_request_id": "ecb06b92c7f89c93:ecb06b92c7f89c93:0000000000000000:0",
    "reason": "security_policy",
    "alerts": [
      {"type": "knownMalware", "severity": "critical", "category": "security", "action":
"error"}
    ],
    "alert_count": 1,
    "blocked_alerts": [
      {"type": "knownMalware", "severity": "critical", "category": "security", "action":
"error"}
    ],
    "blocked_alert_count": 1,
    "score": 0.1,
    "versions": {}
  },
  "source": "socket-firewall",
  "sourcetype": "socket:firewall:event"
}
```


Environment variables:

Bash

```
SPLUNK_ENABLED=true
SPLUNK_HEC_URL=https://splunk.company.com:8088/services/collector/event
SPLUNK_HEC_TOKEN=${SPLUNK_HEC_TOKEN}
SPLUNK_SOURCE=socket-firewall
```

Unified Event Fields

All three event systems — **console logging** (`[SOCKET_DECISION]`), **Splunk HEC**, and **Socket telemetry** — share the same core event fields built by a single function. This guarantees consistent

observability regardless of which system is consuming the events.

Core Fields (all systems)

Field	Type	Description
`request_id`	string	Unique request identifier for correlation
`purl`	string	Package URL (e.g., `pkg:npm/lodash@4.17.21`)
`decision`	string	`"blocked"` or `"allowed"`
`action`	string	Overall severity: `"block"`, `"warn"`, `"monitor"`, `"ignore"`
`response_code`	number	HTTP status code sent to client (e.g., 200 or 403)
`upstream_status`	number or null	HTTP status from upstream registry (null for blocked packages)
`source_path`	string	Request URI path
`upstream_host`	string or null	Upstream registry hostname
`upstream_path`	string or null	Upstream registry request path
`repo`	string or null	Route/repository name (from `socket_route_name`)
`client_ip`	string	Client IP address
`user_agent`	string	Client User-Agent header
`socket_api_response_code`	number	Socket API HTTP status (200 allowed, 403 blocked)
`cached`	boolean	Whether result was served from cache
`stale`	boolean	Whether cached value was stale (revalidation attempted)
`block_source`	string or null	`"download"` (artifact check) or `"metadata"` (metadata filtering)
`block_reason`	string	Comma-separated alert titles for block/error alerts
`warn_reason`	string	Comma-separated alert titles for warn alerts
`api_error`	string or null	Error message if Socket API call failed
`unscanned`	boolean or null	`true` when the package/version was not found or not yet scanned by Socket (purlError response)
`purl_check_latency_ms`	number or null	Milliseconds to check package via Socket API
`private_registry_request_id`	string or null	Trace/request ID from private registry (Jaeger `uber-trace-id` or `X-Request-Id` from Artifactory/Nexus)

Platform-Specific Fields

Splunk HEC adds these fields on top of the core:

Field	Type	Description
`event_type`	string	Always `"package_check"`
`reason`	string	Result reason string from Socket API
`alerts`	array	Structured array of alert objects (`type`, `severity`, `category`, `action`)
`alert_count`	number	Total number of alerts

Field	Type	Description
`blocked_alerts`	array	Structured array of blocked/error alert objects
`blocked_alert_count`	number	Number of blocked/error alerts
`score`	number	Package security score
`versions`	object	Component versions from `.versions` file

Socket telemetry adds these fields on top of the core:

Field	Type	Description
`input_purl`	string	Decoded PURL sent for readability
`event_sender_created_at`	string	HTTP date timestamp
`socket_client_version`	string	Socket client library version
`event_type`	string	Always `"firewall_package_encountered"`
`event_category`	string	Always `"proactive"`
`registryFqdn`	string	Registry hostname from request
`machine_id`	string	SHA256-based machine identifier
`parser_name`	string	Ecosystem parser name
`parser_version`	string	Ecosystem parser version
`artifact_purl`	string	Decoded PURL for the artifact
`alert_action`	string	Alias for `action`
`client_action`	string	Alias for `action`
`purlCheckLatencyMs`	number	Alias for `purl_check_latency_ms` (camelCase)
`versions`	object	Component versions from `.versions` file

SOCKET_DECISION (`[SOCKET_DECISION]` JSON log line) includes the core fields plus:

Field	Type	Description
`monitor_reason`	string	Comma-separated alert titles for monitor alerts

Webhook Events

Send package decision events to any HTTP endpoint. Useful for custom dashboards, alerting systems, or SIEM integrations beyond Splunk.

YAML

```yaml
webhook:
  enabled: true
  url: https://siem.company.com/api/events
  auth_header: "Bearer ${WEBHOOK_AUTH_TOKEN}" # Authorization header (optional)
  ssl_verify: false                     # Verify TLS certificate (default: false)
  timeout: 5000                         # Request timeout in ms (default: 5000)
  on_block: true                        # Fire on block decisions (default: true)
  on_warn: true                         # Fire on warn decisions (default: true)
  on_monitor: true                      # Fire on monitor decisions (default: true)
  on_ignore: true                       # Fire on ignore decisions (default: true)
  # Batch delivery (off by default — sends one POST per event)
  batch_enabled: false                  # Accumulate events and flush as NDJSON
  batch_size: 524288                    # Max bytes before immediate flush (default: 512KB)
  batch_period: 5                       # Max seconds to hold events before flushing
```

Field	Default	Description
`enabled`	`false`	Enable webhook event delivery

Field	Default	Description
`url`	(none)	Webhook endpoint URL (required when enabled)
`auth_header`	(none)	Value for the `Authorization` header (optional)
`ssl_verify`	`false`	Verify TLS certificate of webhook endpoint
`timeout`	`5000`	HTTP request timeout in milliseconds
`on_block`	`true`	Send events for blocked packages
`on_warn`	`true`	Send events for warned packages
`on_monitor`	`true`	Send events for monitored packages
`on_ignore`	`true`	Send events for ignored packages
`batch_enabled`	`false`	Accumulate events and send as NDJSON batches
`batch_size`	`524288`	Max accumulated NDJSON bytes before immediate flush
`batch_period`	`5`	Max seconds to hold events before periodic flush

Delivery Modes

Immediate (default): Each decision event is sent as an individual HTTP POST with `Content-Type: application/json`. One request per event.

Batched (`batch_enabled: true`): Events accumulate in a per-worker queue. A batch is flushed (sent as a single HTTP POST) when either threshold is reached:

- **`batch_size`** bytes of accumulated NDJSON payload, OR
- **`batch_period`** seconds have elapsed since the first queued event

Batch payloads use NDJSON format (`Content-Type: application/x-ndjson`) — one JSON object per line.

Events are delivered asynchronously (non-blocking) and include all core event fields:

JSON

```
{
  "event_type": "package_decision",
  "timestamp": 1709078400.123,
  "request_id": "abc123xyz",
  "purl": "pkg:npm/malicious-package@1.0.0",
  "decision": "blocked",
  "action": "block",
  "response_code": 403,
  "upstream_status": null,
  "block_source": "download",
  "block_reason": "Known Malware",
  "warn_reason": "",
  "client_ip": "203.0.113.10",
  "user_agent": "npm/8.19.2",
  "repo": "npm-remote",
  "source_path": "/repository/npm/malicious-package/-/malicious-package-1.0.0.tgz",
  "upstream_host": "registry.npmjs.org",
  "cached": false,
  "stale": false,
  "socket_api_response_code": 403,
  "purl_check_latency_ms": 142,
  "private_registry_request_id": "ecb06b92c7f89c93:ecb06b92c7f89c93:0000000000000000:0"
}
```

Environment variables:

Bash

```
WEBHOOK_ENABLED=true
WEBHOOK_URL=https://siem.company.com/api/events
WEBHOOK_AUTH_HEADER="Bearer ${WEBHOOK_AUTH_TOKEN}"
WEBHOOK_SSL_VERIFY=false
WEBHOOK_TIMEOUT=5000
WEBHOOK_ON_BLOCK=true
```

```
WEBHOOK_ON_WARN=true
WEBHOOK_ON_MONITOR=true
WEBHOOK_ON_IGNORE=true
WEBHOOK_BATCH_ENABLED=false
WEBHOOK_BATCH_SIZE=524288
WEBHOOK_BATCH_PERIOD=5
```

Log Level

Controls which messages appear in console output. The default level is `info`, which shows all security decisions. Splunk HEC events, Socket telemetry events, and webhook deliveries are **always sent** regardless of log level — this setting only affects console (stderr) output.

YAML

```yaml
socket:
  log_level: info  # error, warn, info (default), debug
```

Level	Console Output
`error`	Only block/error decisions (`[SOCKET_DECISION]` at ERR level)
`warn`	Block/error + warn decisions
`info`	All decisions including monitor/ignore (default)
`debug`	All decisions + verbose debug traces (automatically enables `debug_logging_enabled`)

SOCKET_DECISION Log Level Mapping

Each security decision action maps to a specific log level:

Action	Log Level	When Visible
`block` / `error`	ERR	Always (all log levels)
`warn`	WARN	`log_level: warn` or lower
`monitor`	INFO	`log_level: info` or lower (default)
`ignore`	INFO	`log_level: info` or lower (default)

Integration with Debug Logging

Setting `log_level: debug` automatically enables `debug_logging_enabled` , which provides verbose HTTP request/response header logging. You can also enable debug logging independently via `socket.debug_logging_enabled: true` without changing the log level.

Environment variable:

Bash

```bash
SOCKET_LOG_LEVEL=info  # error, warn, info (default), debug
```

Log Max Body Size

Controls the maximum byte length of a `[SOCKET_DECISION]` JSON body in a single `ngx.log()` call. OpenResty has a hard 4096-byte buffer (`NGX_MAX_ERROR_STR`) — messages exceeding this limit are **silently truncated**. When a decision body exceeds the configured limit, it is automatically split across multiple continuation log lines.

YAML

```yaml
socket:
  log_max_body_size: 3900  # bytes per log line (default: 3900, 0 = disable splitting)
```

Value	Behavior
`3900` (default)	Split long bodies into `[SOCKET_DECISION 1/N]`, `[SOCKET_DECISION 2/N]`, ... continuation lines
`0`	No splitting — output is truncated by nginx at 4096 bytes
Custom (≥100)	Use the specified chunk size (must leave room for ~80 bytes of prefix overhead)

Example output (split message)

```
[SOCKET_DECISION 1/2]
{"request_id":"abc123","purl":"pkg:npm/lodash@4.17.21","decision":"blocked",...
[SOCKET_DECISION 2/2] ...,"blocked_alerts":["Malware"],"score":0.1}
```

Environment variable:

Bash

```
SOCKET_LOG_MAX_BODY_SIZE=3900  # default; set to 0 to disable splitting
```

Debug Logging

Enable verbose request/response header logging for troubleshooting. Disabled by default.

YAML

```yaml
socket:
  debug_logging_enabled: false              # Enable debug logging (default: false)
  debug_user_agent_filter: "*artifactory*"  # Glob pattern to match user-agents (optional)
```

Field	Default	Description
`debug_logging_enabled`	`false`	Enable verbose HTTP header logging
`debug_user_agent_filter`	(none)	Glob pattern to limit debug logging to matching user-agents only

When `debug_user_agent_filter` is set, only requests whose `User-Agent` header matches the glob pattern will produce debug log output. The match is case-insensitive. Standard glob syntax is supported (`*` matches any characters, `?` matches a single character).

Examples:

YAML

```yaml
# Log all requests
socket:
  debug_logging_enabled: true

# Log only Artifactory traffic
socket:
  debug_logging_enabled: true
  debug_user_agent_filter: "*artifactory*"

# Log only npm client traffic
socket:
  debug_logging_enabled: true
  debug_user_agent_filter: "npm/*"
```

Environment variables:

Bash

```
SOCKET_DEBUG_LOGGING_ENABLED=true
SOCKET_DEBUG_USER_AGENT_FILTER="*artifactory*"
```

Health Check Logging

The firewall exposes a `/health` endpoint on every server block (default, per-registry, and path-routing). Health check requests are **automatically excluded from console output** to prevent log noise from load balancers and Kubernetes probes.

What is suppressed

Log source	Suppressed?	How
nginx access log	Yes	`access_log off;` on every `/health` location block
Debug logging (`[DEBUG]`)	Yes	`should_debug_log()` returns `false` for `/health` requests
Splunk HEC / Socket telemetry	N/A	Health checks do not trigger security decisions

Health check response

```
GET /health HTTP/1.1

HTTP/1.1 200 OK
Content-Type: text/plain
Server: SocketFirewall/1.2.3

SocketFirewall/1.2.3 - Health OK
```

Per-registry and path-routing health endpoints include additional context:

```
SocketFirewall/1.2.3 - Health OK - npm (npm.company.com)
SocketFirewall/1.2.3 - Health OK - path-routing (firewall.company.com)
```

No configuration is required — health check log suppression is always active.

Decision Log (SOCKET_DECISION)

Every package security check emits a `[SOCKET_DECISION]` JSON log entry for audit and observability. These entries appear in the firewall's standard error log.

Example log entry:

```
[error] [REQUEST_ID: abc123] [SOCKET_DECISION]
{"request_id":"abc123","purl":"pkg:npm/malicious-
package@1.0.0","decision":"blocked","action":"block","response_code":403,"upstream_status":null,"s
package/-/malicious-package-
1.0.0.tgz","upstream_host":"registry.npmjs.org","repo":"npm","client_ip":"10.0.0.5","socket_api_r
```

Decision Log Fields

Field	Type	Description
`request_id`	string	Unique request identifier
`purl`	string	Package URL (decoded, e.g., `pkg:npm/lodash@4.17.21`)
`decision`	string	`"allowed"` or `"blocked"`
`action`	string	Overall severity: `block` , `warn` , `monitor` , `ignore` , `error`
`response_code`	number	HTTP status returned to client (`200` or `403`)
`upstream_status`	number/null	HTTP status from upstream registry (null for blocked requests)
`source_path`	string	Request URI path

Field	Type	Description
`upstream_host`	string/null	Upstream registry hostname
`repo`	string/null	Route name (e.g., `npm`, `pypi-remote`)
`client_ip`	string/null	Client IP address
`socket_api_response_code`	number	HTTP status from Socket API response
`cached`	boolean	Whether the result was served from cache
`stale`	boolean	Whether the cached result was stale (revalidation attempted)
`block_source`	string/null	`"download"` (artifact check) or `"metadata"` (filtering)
`block_reason`	string	Comma-separated alert titles that caused a block
`warn_reason`	string	Comma-separated alert titles at warn level
`api_error`	string/null	Error message if Socket API call failed
`private_registry_request_id`	string/null	Trace/request ID from private registry (`uber-trace-id` or `X-Request-Id`)

Log Level by Action

Action	Log Level	When
`block` / `error`	ERROR	Package blocked or API error in fail-closed
`warn`	WARN	Package has warn-level alerts (still allowed)
`monitor` / `ignore`	NOTICE	Package allowed with monitor alerts or clean

Filtering Logs

Bash

```
# All security decisions
docker compose logs socket-firewall | grep SOCKET_DECISION

# Only blocked packages
docker compose logs socket-firewall | grep SOCKET_DECISION | grep '"decision":"blocked"'

# Decisions for a specific package
docker compose logs socket-firewall | grep SOCKET_DECISION | grep 'lodash'

# Metadata filtering decisions
docker compose logs socket-firewall | grep SOCKET_DECISION | grep '"block_source":"metadata"'
```

Access Log Format

The firewall uses a custom access log format that includes timing, upstream, and authentication fields for operational monitoring.

Log format:

```
$remote_addr - $remote_user [$time_local] "$request_method $request_uri $server_protocol"
  $status $body_bytes_sent "$http_referer"
  "$http_user_agent" "$http_x_forwarded_for"
  rt=$request_time
  upstream=$upstream_addr us=$upstream_status ut=$upstream_response_time
  auth=$sanitized_authorization
  req=$request_id trace=$sent_http_x_trace_id
```

Access Log Fields

Field	Description
`rt=`	Total request time in seconds (includes upstream + processing)
`upstream=`	Upstream server address (IP:port)
`us=`	Upstream HTTP status code
`ut=`	Upstream response time in seconds
`auth=`	Authorization header (redacted to `[REDACTED]` for security)
`req=`	NGINX-generated unique request ID (32-char hex, correlates with `[REQUEST_ID: ...]` in Lua logs)
`trace=`	Upstream registry trace ID (`uber-trace-id` or `X-Request-Id` from upstream response), also sent as `X-Trace-Id` response header

Query parameters are stripped from logged URIs to prevent sensitive data leakage.

Access Log Buffering

Control log output buffering with `access_log_buffer`:

YAML

```yaml
nginx:
  access_log_buffer: 64k      # Default — buffer 64k before flushing
  # access_log_buffer: off    # Disable buffering (flush every line)
  # access_log_buffer: 256k   # Larger buffer for high-throughput
```

Value	Behavior
`64k`	Default. Buffers up to 64k before flushing to stdout
`off`	Disables buffering — each log line is written immediately
`256k`	Larger buffer for high-throughput deployments

Set `access_log_buffer: off` when you need real-time log output (e.g., debugging, streaming to log aggregators).

SSL/TLS Certificates

Certificates are stored in `/etc/nginx/ssl` inside the container. Mount from host:

YAML

```yaml
volumes:
  - ./ssl:/etc/nginx/ssl
```

Configuration

YAML

```yaml
ssl:
  cert: /etc/nginx/ssl/fullchain.pem   # Server certificate (default: auto-generated)
  key: /etc/nginx/ssl/privkey.pem      # Server private key (default: auto-generated)
  ca_cert: /etc/nginx/ssl/ca-cert.pem  # Custom CA certificate (optional)
```

Setting	Purpose	Default
`cert`	Server certificate for HTTPS listener	Auto-generated self-signed
`key`	Server private key	Auto-generated self-signed
`ca_cert`	Custom CA certificate — trusted in addition to system root CAs	*(not set)*

Custom CA Certificate (`ca_cert`)

When set, the firewall creates a combined CA bundle at startup that includes:

1. System root CAs (`/etc/ssl/certs/ca-certificates.crt`)
2. Your custom CA certificate
3. Redis CA certificate (if configured)

This bundle is used for **all outbound SSL connections** — upstream registries, Socket API, and Redis.

Use case: Upstream registries (Nexus, Artifactory, etc.) use internal or self-signed certificates.

YAML

```yaml
ssl:
  ca_cert: /etc/nginx/ssl/internal-ca.pem
```

> **Note**: The per-connection overrides `socket.api_ssl_ca_cert` and `socket.upstream_ssl_ca_cert` still work for advanced use cases where different connections need different trust stores.

Required Files

File	Purpose	Permissions
`ssl/fullchain.pem`	Certificate chain (cert + intermediates)	644
`ssl/privkey.pem`	Private key	644

Auto-Generated Certificates

The firewall generates self-signed certs on first run if none exist. Located at `/etc/nginx/ssl/` .

Custom Certificates (Production)

Place your organization's certificates in the `ssl/` directory on the host:

Bash

```bash
mkdir -p ssl
cp /path/to/cert.pem ssl/fullchain.pem
cp /path/to/key.pem ssl/privkey.pem
chmod 644 ssl/fullchain.pem ssl/privkey.pem
```

Generate Self-Signed Certificates

Single domain:

Bash

```bash
mkdir -p ssl
openssl req -x509 -nodes -days 365 -newkey rsa:2048 \
  -keyout ssl/privkey.pem \
  -out ssl/fullchain.pem \
  -subj "/CN=firewall.company.com" \
  -addext "subjectAltName=DNS:firewall.company.com,DNS:localhost"
chmod 644 ssl/fullchain.pem ssl/privkey.pem
```

Wildcard (multiple subdomains):

Bash

```bash
mkdir -p ssl
openssl req -x509 -nodes -days 365 -newkey rsa:2048 \
  -keyout ssl/privkey.pem \
  -out ssl/fullchain.pem \
  -subj "/CN=*.company.com" \
  -addext "subjectAltName=DNS:*.company.com,DNS:company.com,DNS:localhost"
chmod 644 ssl/fullchain.pem ssl/privkey.pem
```

Trust Self-Signed Certificates

macOS:

```Bash
sudo security add-trusted-cert -d -r trustRoot \
  -k /Library/Keychains/System.keychain ssl/fullchain.pem
```

Linux:

```Bash
sudo cp ssl/fullchain.pem /usr/local/share/ca-certificates/socket-firewall.crt
sudo update-ca-certificates
```

Windows:

```PowerShell
Import-Certificate -FilePath ssl\fullchain.pem -CertStoreLocation Cert:\LocalMachine\Root
```

Environment Variables Reference

All configuration can be overridden via environment variables. Useful for Docker/Kubernetes deployments.

Core Settings

```Bash
# Required
SOCKET_SECURITY_API_TOKEN=${SOCKET_SECURITY_API_TOKEN} # Socket.dev API key

# Socket API
SOCKET_API_URL=https://api.socket.dev         # Default
SOCKET_FAIL_OPEN=true                         # Allow on API error (default: true)
SOCKET_FAIL_OPEN_UNSCANNED=true               # Allow unscanned packages (default: true)
SOCKET_API_READ_TIMEOUT=300                   # Socket API response body timeout (defaults to
PROXY_READ_TIMEOUT)
SOCKET_API_CONNECT_TIMEOUT=5                  # Socket API connect timeout (seconds)
SOCKET_API_SEND_TIMEOUT=10                    # Socket API send timeout (seconds)
SOCKET_CACHE_TTL=600                          # Freshness window (seconds)
CACHE_REVALIDATION_LOCK_LEASE_SECONDS=0       # 0 = auto (api_read_timeout + 30s)
CACHE_REVALIDATION_JITTER_SECONDS=120         # Freshness-boundary jitter (0 = disabled)
CACHE_REVALIDATION_ASYNC=false                # true = serve stale + background refresh
SOCKET_PURL_CONFIRM_ALLOW_MODE=wait           # Confirm-stale-ALLOW mode for PURL cache
CACHE_WARM_ENABLED=false                      # Proactive cache warming (requires Redis)
CACHE_WARM_INTERVAL=60                        # Seconds between warm cycles

# Circuit breaker
SOCKET_BREAKER_ENABLED=true                   # Kill-switch for Socket API circuit breaker
SOCKET_BREAKER_LATENCY_EWMA_THRESHOLD_MS=10000
SOCKET_BREAKER_EWMA_ALPHA=0.3
SOCKET_BREAKER_ERROR_RATE_THRESHOLD=0.5
SOCKET_BREAKER_ERROR_MIN_REQUESTS=20
SOCKET_BREAKER_ERROR_WINDOW_S=10
SOCKET_BREAKER_OPEN_WINDOW_S=30
SOCKET_BREAKER_MAX_OPEN_WINDOW_S=300
SOCKET_BREAKER_BACKOFF_FACTOR=2
SOCKET_BREAKER_HALFOPEN_PROBES=3
SOCKET_BREAKER_HALFOPEN_PROBE_SPACING_S=1
SOCKET_BREAKER_PROBE_TIMEOUT_MS=10000

# Ports
HTTP_PORT=8080                                # HTTP port
HTTPS_PORT=8443                               # HTTPS port

# Deployment mode
CONFIG_MODE=upstream                          # 'upstream' or 'middle'

# SSL verification
SOCKET_API_SSL_VERIFY=true                    # Verify Socket API SSL (default: true)
```

```
SOCKET_API_SSL_CA_CERT=/path/to/ca.crt       # Custom Socket API CA
SOCKET_UPSTREAM_SSL_VERIFY=true              # Verify upstream registry SSL (default: true,
inherits api_ssl_verify)
SOCKET_UPSTREAM_SSL_CA_CERT=/path/to/ca.crt  # Custom upstream CA

# Corporate proxy
SOCKET_OUTBOUND_PROXY=http://proxy:3128      # Egress proxy
SOCKET_NO_PROXY=localhost,127.0.0.1          # No-proxy exceptions

# Request tracking
SOCKET_REQUEST_ID_HEADER=X-Socket-Request-ID # Request ID header name (default)

# Log level
SOCKET_LOG_LEVEL=info                        # Console log level (error/warn/info/debug)

# Debug logging
SOCKET_DEBUG_LOGGING_ENABLED=false           # Enable debug logging
SOCKET_DEBUG_USER_AGENT_FILTER="*pattern*"   # Glob filter for user-agent
```

Redis

Bash

```
REDIS_ENABLED=true                           # Enable Redis
REDIS_HOST=redis.company.com                 # Redis hostname
REDIS_PORT=6379                              # Redis port
REDIS_PASSWORD=${REDIS_PASSWORD}             # Redis password
REDIS_DB=0                                   # Redis database number
REDIS_TTL=86400                              # Stale window (seconds)

# Redis SSL
REDIS_SSL=true                               # Enable SSL
REDIS_SSL_VERIFY=true                        # Verify Redis SSL
REDIS_SSL_CA_CERT=/path/to/redis-ca.pem      # Redis CA cert
REDIS_SSL_SERVER_NAME=redis.company.com      # SNI hostname
```

Nginx Performance

Bash

```
WORKER_PROCESSES=2                           # nginx worker processes
WORKER_CONNECTIONS=4096                      # Connections per worker
# HTTP/2 is configured via nginx.http2 in socket.yml (default: true)
```

Proxy Timeouts

Bash

```
PROXY_CONNECT_TIMEOUT=60                     # Connection timeout (seconds)
PROXY_SEND_TIMEOUT=60                        # Send timeout
PROXY_READ_TIMEOUT=60                        # Read timeout
```

Auto-Discovery

Auto-discovery is configured via `socket.yml` under `path_routing.private_registry` (see above). The `api_key` can also be provided via the `PRIVATE_REGISTRY_KEY` environment variable.

Metadata Filtering

Bash

```
METADATA_FILTERING_ENABLED=true              # Enable filtering (v1.1.108+)
METADATA_FILTER_BLOCKED=true                 # Filter blocked packages
METADATA_FILTER_WARN=false                   # Filter warned packages
METADATA_INCLUDE_UNCHECKED_VERSIONS=true     # Keep unchecked versions
METADATA_MAX_VERSIONS=100                    # Max versions to check per package
METADATA_CACHE_TTL=3600                      # Cache TTL for metadata lookups (seconds)
METADATA_FILTER_BATCH_SIZE=4000              # Max PURLs per batch
METADATA_MAX_BODY_SIZE=524288000             # Max body size for filtering (500MB default)
METADATA_PACKAGE_FILTER_TIMEOUT=60           # Time budget per filter operation
METADATA_SEMAPHORE_WAIT_TIMEOUT=60           # Max wait for a filter slot
```

```
METADATA_MAX_CONCURRENT=3                    # Concurrent filter operations per worker
METADATA_PREFETCH_ENABLED=true               # Enable/disable background prefetch
(true/false)
METADATA_PREFETCH_TTL=600                    # Prefetch refresh interval in seconds
PREFETCH_MAX_CONCURRENT=2                    # Max concurrent prefetch operations across
workers
PREFETCH_BATCH_CONCURRENCY=4                 # Max concurrent PURL batch API calls per
filter
METADATA_RESPONSE_CACHE_ENABLED=false        # Filtered response body cache (opt-in)
METADATA_RESPONSE_CACHE_TTL=600              # Max age before full re-filter
METADATA_RESPONSE_CACHE_FRESH=5              # Serve-without-upstream window
METADATA_CACHE_AUTHENTICATED=false           # Cache authenticated upstream responses
METADATA_RESPONSE_CACHE_EXCLUDED_ECOSYSTEMS=  # Comma-separated ecosystems to skip
METADATA_CONFIRM_ALLOW_MODE=wait             # Confirm-stale-ALLOW for response cache
```

Per-Ecosystem recentlyPublished Downgrade

Bash

```
RECENTLY_PUBLISHED_ENABLED_ECOSYSTEMS=npm,pypi  # Downgrade recentlyPublished to warn (allow
through) for these ecosystems (v1.1.134+)
```

Splunk

Bash

```
SPLUNK_ENABLED=true                          # Enable Splunk
SPLUNK_HEC_URL=https://splunk.company.com:8088/services/collector/event
SPLUNK_HEC_TOKEN=${SPLUNK_HEC_TOKEN}         # Splunk HEC token
SPLUNK_INDEX=security                        # Splunk index (optional)
SPLUNK_SOURCE=socket-firewall                # Splunk source
SPLUNK_SOURCETYPE=socket:firewall:event      # Splunk sourcetype (default:
socket:firewall:event)
SPLUNK_SSL_VERIFY=true                       # Verify Splunk SSL
SPLUNK_BATCH_SIZE=1                          # Events per batch (default: 1)
```

Webhook

Bash

```
WEBHOOK_ENABLED=true                         # Enable webhook
WEBHOOK_URL=https://siem.company.com/api/events  # Webhook endpoint URL
WEBHOOK_AUTH_HEADER="Bearer ${WEBHOOK_AUTH_TOKEN}" # Authorization header (optional)
WEBHOOK_SSL_VERIFY=false                     # Verify TLS (default: false)
WEBHOOK_TIMEOUT=5000                         # Timeout in ms (default: 5000)
WEBHOOK_ON_BLOCK=true                        # Fire on block (default: true)
WEBHOOK_ON_WARN=true                         # Fire on warn (default: true)
WEBHOOK_ON_MONITOR=true                      # Fire on monitor (default: true)
WEBHOOK_ON_IGNORE=true                       # Fire on ignore (default: true)
WEBHOOK_BATCH_ENABLED=false                  # Accumulate events as NDJSON batches
WEBHOOK_BATCH_SIZE=524288                    # Max NDJSON bytes before flush
WEBHOOK_BATCH_PERIOD=5                       # Max seconds before periodic flush
```

Docker Compose Examples

Minimal Configuration

YAML

```yaml
services:
  socket-firewall:
    image: socketdev/socket-registry-firewall:latest
    ports:
      - "8080:8080"
      - "8443:8443"
    environment:
      - SOCKET_SECURITY_API_TOKEN=${SOCKET_SECURITY_API_TOKEN}
    volumes:
      - ./socket.yml:/app/socket.yml:ro
```

```
      - ./ssl:/etc/nginx/ssl
    restart: unless-stopped
```

Full Configuration with Redis

YAML

```yaml
services:
  socket-firewall:
    image: socketdev/socket-registry-firewall:latest
    ports:
      - "8080:8080"
      - "8443:8443"
    environment:
      # Core
      - SOCKET_SECURITY_API_TOKEN=${SOCKET_SECURITY_API_TOKEN}
      - SOCKET_FAIL_OPEN=true
      - SOCKET_FAIL_OPEN_UNSCANNED=true
      - SOCKET_CACHE_TTL=600

      # Redis
      - REDIS_ENABLED=true
      - REDIS_HOST=redis
      - REDIS_PORT=6379
      - REDIS_PASSWORD=${REDIS_PASSWORD}
      - REDIS_TTL=86400

      # Performance
      - WORKER_PROCESSES=4
      - WORKER_CONNECTIONS=8192

      # Corporate proxy
      - SOCKET_OUTBOUND_PROXY=http://proxy.company.com:3128
      - SOCKET_NO_PROXY=localhost,127.0.0.1

    volumes:
      - ./socket.yml:/app/socket.yml:ro
      - ./ssl:/etc/nginx/ssl
    restart: unless-stopped
    healthcheck:
      test: ["CMD", "curl", "-fk", "https://localhost:8443/health"]
      interval: 30s
      timeout: 10s
      retries: 3
    depends_on:
      - redis

  redis:
    image: redis:7-alpine
    ports:
      - "6379:6379"
    command: redis-server --requirepass ${REDIS_PASSWORD}
    volumes:
      - redis-data:/data
    restart: unless-stopped

volumes:
  redis-data:
```

With Splunk Integration

YAML

```yaml
services:
  socket-firewall:
    image: socketdev/socket-registry-firewall:latest
    ports:
      - "8080:8080"
      - "8443:8443"
    environment:
      - SOCKET_SECURITY_API_TOKEN=${SOCKET_SECURITY_API_TOKEN}

      # Splunk
```

```
      - SPLUNK_ENABLED=true
      - SPLUNK_HEC_URL=https://splunk.company.com:8088/services/collector/event
      - SPLUNK_HEC_TOKEN=${SPLUNK_HEC_TOKEN}
      - SPLUNK_INDEX=security
      - SPLUNK_SOURCE=socket-firewall

    volumes:
      - ./socket.yml:/app/socket.yml:ro
      - ./ssl:/etc/nginx/ssl
    restart: unless-stopped
```

Health Checks

The firewall exposes a health endpoint at `/health`:

Bash

```
curl -k https://localhost:8443/health
```

Response:

```
SocketFirewall/1.1.94 - Health OK - npm (registry.npmjs.org)
```

The response is plain text (`Content-Type: text/plain`) and includes the firewall version, registry name, and domain.

HTTP status codes:

- `200 OK` - Firewall is healthy
- `503 Service Unavailable` - Firewall is unhealthy (configuration error)

Docker healthcheck:

YAML

```yaml
healthcheck:
  test: ["CMD", "curl", "-fk", "https://localhost:8443/health"]
  interval: 30s
  timeout: 10s
  retries: 3
  start_period: 10s
```

Complete Configuration Example

socket.yml:

YAML

```yaml
# Core Socket settings
socket:
  api_url: https://api.socket.dev
  fail_open: true
  fail_open_unscanned: true
  outbound_proxy: http://proxy.company.com:3128
  no_proxy: localhost,127.0.0.1,internal.company.com
  api_ssl_verify: true
  api_ssl_ca_cert: /etc/ssl/certs/corporate-ca.crt
  upstream_ssl_verify: true

# Ports
ports:
  http: 8080
  https: 8443

# Deployment mode
config_mode: upstream

# Path-based routing with auto-discovery
path_routing:
```

```
  enabled: true
  domain: socket-firewall.company.com
  mode: artifactory

  private_registry:
    api_url: https://artifactory.company.com/artifactory
    api_key: ${ARTIFACTORY_API_KEY}
    interval: 5m
    exclude_pattern: "(tmp|test|snapshot)-.*"

# Caching
cache:
  ttl: 600

redis:
  enabled: true
  host: redis.company.com
  port: 6380
  password: ${REDIS_PASSWORD}
  ttl: 86400
  ssl: true
  ssl_verify: true
  ssl_ca_cert: /etc/redis/ssl/ca-cert.pem

# Performance
nginx:
  worker_processes: 8
  worker_connections: 16384

proxy:
  connect_timeout: 120
  send_timeout: 300
  read_timeout: 300

# Advanced features (v1.1.108+)
metadata_filtering:
  enabled: true
  filter_blocked: true
  filter_warn: false
  include_unchecked_versions: true
  max_versions: 100
  cache_ttl: 3600
  batch_size: 100
  max_body_size: 500m
  package_filter_timeout: 60   # raise for very large package indexes (v1.1.323+)

# Per-ecosystem recentlyPublished override (v1.1.134+)
# recently_published_enabled_ecosystems:
#   - npm
#   - pypi

splunk:
  enabled: true
  hec_url: https://splunk.company.com:8088/services/collector/event
  hec_token: ${SPLUNK_HEC_TOKEN}
  index: security
  source: socket-firewall
  sourcetype: socket:firewall:event
  ssl_verify: true
```

Updated 4 days ago

← AWS CodeArtifact Configuration

Installation Guide →

Did this page help you? Yes No