Email Policies

- Disclaimers
- External Banner
- Link Guard
- Organizational Signatures

Disclaimers

Disclaimers

Pro Edition feature. Maps to **Email Policies > Disclaimers** (`view_disclaimers.cfm`, `edit_disclaimer.cfm`, `disclaimer_delete.cfm`).

Hermes appends a configurable disclaimer to outbound mail at the gateway, with two scopes:

Scope	Sender match	Use case
Domain	All senders in `@example.com`	Default org-wide compliance/legal language
Relay Recipient	Specific full address (e.g. `vendor@example.com`)	Per-relay-user override for tenants with extra regulatory language

Most-specific match wins: a relay-recipient match is used before the domain default.

Pipeline placement

Disclaimers are applied at SMTP receive time by the **`hermes_body_milter`** container, which Postfix consults as a milter alongside OpenDKIM and OpenDMARC.

```
External MTA / MUA submission
        │
        ▼
Postfix smtpd
   ├─ smtpd_milters chain (in order):
   │   1. OpenDKIM            (signs/verifies)
   │   2. OpenDMARC           (DMARC policy)
   │   3. hermes_body_milter  (THIS — disclaimers, signatures, banners)
   ▼
content_filter → Amavis    (unmodified path; sees the body milter's output)
   ▼
Ciphermail              (server-side S/MIME or PGP, if configured)
   ▼
```

```
Postfix :10026          (OpenDKIM signs the final composed body here)
   ▼
external
```

Body modification happens at smtpd time, **before** content_filter routes to Amavis. By the time Amavis sees the message, the disclaimer is already baked in. Amavis processes a normal-looking message; no internal-state coupling, no temp-file races.

OpenDKIM's outbound signing fires at the `:10026` re-injection — after both the body milter and Ciphermail. Hermes' own DKIM therefore always covers whatever the recipient ultimately receives. Ciphermail's server-side crypto also covers the disclaimer because Ciphermail runs after the milter.

Behavior with S/MIME, PGP, and DKIM-signed mail

The behavior depends on **who** signed/encrypted the message and **when** in the pipeline.

Server-side: signed/encrypted by Ciphermail — disclaimer is applied

Ciphermail runs **after** the body milter. Mail arrives at the milter as plaintext, the disclaimer is appended, then Ciphermail signs or encrypts the modified body. The recipient sees a valid signature **and** the disclaimer. No conflict.

Client-side: signed/encrypted by the user's MUA — disclaimer is skipped

Mail signed in Outlook (S/MIME) or Thunderbird+Enigmail (PGP) arrives at the gateway with the cryptographic envelope already sealed. Modifying the body would either invalidate the signature or mangle the ciphertext.

The body milter detects the following patterns in the headers (or first 32 KB of the body) and exits unchanged when any matches:

Pattern matched	Meaning

`Content-Type: multipart/signed; protocol="application/pkcs7-signature"`	S/MIME detached signature
`Content-Type: application/pkcs7-mime`	S/MIME opaque-signed or enveloped
`Content-Type: multipart/signed; protocol="application/pgp-signature"`	PGP/MIME detached signature
`Content-Type: multipart/encrypted; protocol="application/pgp-encrypted"`	PGP/MIME encrypted
`-----BEGIN PGP SIGNED MESSAGE-----` in body	PGP inline-signed
`-----BEGIN PGP MESSAGE-----` in body	PGP inline-encrypted

When any of those match, the body is left untouched, the signature stays valid, the user's legal-text expectations are preserved (their MUA template is already in the body), and the gateway gets out of the way.

> **Operational consequence.** A site whose users sign client-side will not get gateway disclaimers on those specific signed messages — by design. If org-wide legal text on **all** outbound is mandatory, the only safe pattern is server-side signing in Ciphermail with the disclaimer applied first.

DKIM: Hermes-signed mail is fine; upstream-signed mail is skipped

OpenDKIM signs at the Postfix `:10026` re-injection step — **after** the body milter. So Hermes' own DKIM signature always covers the recipient's view of the message (with disclaimer baked in). No conflict.

The risk is mail that arrives at Hermes **already DKIM-signed by an upstream MTA** — typically a relay user whose own mail server signs before forwarding through us. Modifying that body would invalidate the upstream signature at the recipient.

The body milter treats a pre-existing `DKIM-Signature:` header the same way as a sealed S/MIME or PGP envelope and skips the disclaimer. Since Hermes' own DKIM signs at `:10026` (downstream of this milter), any DKIM-Signature header present at the milter's point in the pipeline came from somewhere upstream of Hermes.

Reply-chain handling — no dedup, by design

The milter does **not** detect or skip messages that already carry a previous disclaimer in their quoted history. Every outbound message gets a fresh disclaimer applied — including replies inside a long thread.

This matches industry norm: commercial server-side disclaimer / signature platforms (Exclaimer, Crossware, CodeTwo, Microsoft 365 transport rules) all stamp every outbound without dedup. The reasoning:

- **Compliance.** Many regulatory regimes (HIPAA email confidentiality, GDPR data-controller notices, financial-services disclosure) treat each transmission as requiring its own disclaimer. Stamping only the first message in a thread arguably leaves later replies non-compliant.
- **Self-contained messages.** If a recipient forwards a reply (with quoted history) to a third party, the disclaimer is preserved per-message in the forwarded text.
- **Predictable behavior.** Operators don't have to explain "sometimes the disclaimer shows, sometimes it doesn't."
- **Cosmetic concern is weak.** Modern MUAs (Gmail, Outlook, Apple Mail) collapse quoted history by default, so stacked disclaimers in long threads are rarely visible to readers.

Earlier iterations of #214 included a sentinel-marker dedup mechanism (`[HD]` / `<!-- HERMES_DISCLAIMER_V1 -->`). That was removed during DEV testing in favor of the industry-norm pattern.

Position: append vs prepend

The schema and UI both expose `position = append | prepend`, but **v1 honors append only.** Prepend is tracked as a v2 enhancement.

Failure semantics

The body milter is **graceful-degradation** by design. Postfix's `milter_default_action = accept` means:

- Milter container down or unreachable → mail flows **unmodified** (missed disclaimer, but no delivery outage)

- Map file unreadable → no entries match → all mail flows unmodified
- Modifier raises an exception → caught and logged → mail flows unmodified
- altermime / parse errors → caught and logged → mail flows unmodified

In every failure case, mail keeps flowing. Worst case is a missed disclaimer, never lost mail. Compare the legacy "modify in amavis hook" approach (#214 Phase 3 v1, retired) which silently dropped messages when the in-place body modification desynced amavis's internal state.

Files generated on save/delete

The CFML include `inc/disclaimer_write_and_reload.cfm` runs after every save or delete and rewrites the entire on-disk state from the `disclaimers` table:

```
/etc/hermes/body_milter/disclaimers/disclaimer_by_sender   sender → option map
/etc/hermes/body_milter/disclaimers/files/<option>/
    body.txt          plain-text disclaimer
    body.html         html disclaimer (may have <img src="cid:..."> refs)
    images/
        1.png         per-disclaimer inline images (#230)
        2.jpg
        ...
```

Where `<option>` is `domain_<safe>` or `relay_<safe>` (non-alphanumeric chars in the source key are replaced with `_`).

Each disclaimer gets its own subdirectory. The files directory is wiped (per-option subdirectories deleted recursively, but the parent `files/` directory and its `.gitkeep` are preserved) and rewritten on every save. There is no incremental update — this guarantees deleted rows and renamed scope keys never leave stale files (or stale image binaries) behind.

No reload step needed. The body milter mtime-watches each map file on every message and reloads when it changes. The CFML `cffile` write to the map file is enough to make the change take effect on the next message processed by the milter.

Inline images (#230)

Admins can paste or upload images directly into the Quill editor when authoring a disclaimer. Supported formats: **PNG, JPEG, GIF**. SVG and WebP are explicitly rejected (security and recipient-compatibility reasons). Limits enforced at save time:

- **5 images max** per disclaimer
- **200 KB per image** (after base64 decode)
- **1 MB total** across all images in a single disclaimer

If any limit is exceeded, the save is rejected with a specific error explaining what failed. Admins can reduce image count or size and re-save.

How it works:

1. Quill embeds pasted/uploaded images as base64 inline `<img src="...">` in the HTML body. The base64 representation is what's stored in the `disclaimers.body_html` column.
2. At save time, the regenerator parses `body_html` for `data:` URLs, decodes each base64 blob, writes the binary as `<option>/images/<N>.<ext>`, and rewrites the HTML in `<option>/body.html` to use `<img src="cid:disclaimer_<option>_img_<N>">` references.
3. At message-send time, the body milter reads `body.html`, walks `<img src="cid:...">` references, and attaches each referenced image as an `image/<format>` MIME part with `Content-ID: <disclaimer_<option>_img_<N>>` and `Content-Disposition: inline`.
4. The milter wraps the message as `multipart/related` so the recipient MUA resolves cid references against the inline parts.

MIME structure transformation (representative example):

```
Original outbound:
  multipart/alternative
    text/plain
    text/html (no images)

After milter (with disclaimer including 1 image):
  multipart/related
    multipart/alternative
      text/plain  (with text disclaimer appended; images omitted from text)
      text/html   (with html disclaimer + <img src="cid:...">)
    image/png
      Content-ID: <disclaimer_..._img_1>
      Content-Disposition: inline
```

This structure renders inline in all major MUAs (Gmail, Outlook, Apple Mail, Thunderbird, mobile clients).

The plain-text version of the disclaimer omits images entirely — base64 inline images don't translate to text, and recipients viewing the message in plain-text mode see the disclaimer text without any image markers.

Hermes' own DKIM signature covers the modified body (including the multipart/related wrap and image parts), because OpenDKIM signs at the postfix `:10026` re-injection step — downstream of the body milter. The signature validates against what the recipient receives.

Auto-derive of plain-text part

The Quill editor on `edit_disclaimer.cfm` drives `body_html`. By default the plain-text part shipped to recipients with a non-HTML MUA is auto-derived from the HTML on save: `<br>`, `</p>`, `</li>` become newlines, all other tags are stripped, runs of 3+ newlines collapse to 2.

Admins who need character-perfect plain text different from the auto-strip (e.g. for regulated industries) can toggle **Edit plain-text version separately** to expose a second editor. When set, `body_text` is shipped verbatim instead of derived.

Disabled rows

Rows with `enabled = 0` are skipped entirely on regen — no files written, no map entry. The milter never matches that scope until the row is re-enabled.

Internal-only mail

v1 does **not** suppress disclaimers for internal-only mail (sender + all recipients in `@local_domains`). Domain disclaimers will be applied to internal mail in the same domain. If this is a problem for your install, file a feature request to add an internal-only bypass.

Why a separate milter and not an amavis hook

Earlier #214 iterations attempted to dispatch the disclaimer from inside an amavisd-new `Custom.pm` `before_send` hook, calling altermime via `system()` on the temp file amavis was managing. amavisd-new 2.13 caused two problems: the legacy `@disclaimer_options_bysender_maps` dispatch path was removed (variables still parse but no code reads them), and the `before_send` hook documentation says "may modify mail" but in practice in-place body modification desynchronizes amavis's internal MIME state and silently loses mail.

The body milter approach moves the body-modification step out of amavis entirely. amavis's role is unchanged from before #214 ever existed; the milter sits in postfix's smtpd_milters chain alongside OpenDKIM and OpenDMARC, the same architectural pattern Hermes already uses for body-touching policy enforcement. amavis is fully decoupled from the disclaimer feature, which means amavis upgrades and the disclaimer feature evolve independently.

This same milter container is intended to host:

- **#226 User Signatures** (per-mailbox personal text from LDAP attributes or user-portal editor)
- **#228 External Sender Banner** (warning banner on inbound external mail)
- **Future Link Guard** (URL rewriting through a click-through endpoint)

Each is a `Modifier` subclass in `/usr/local/bin/hermes-body-milter` registered in the `MODIFIERS` list. The dispatcher is unchanged.

External Banner

External Banner

Maps to **Email Policies > External Banner** (`view_external_banners.cfm`, `edit_external_banner.cfm`, `external_banner_delete.cfm`). Available on both **Community** and **Pro** editions — phishing protection is a baseline security feature, not a Pro upsell.

Hermes prepends (or optionally appends) a warning banner to **inbound** mail from external senders destined for a local recipient. The banner is injected into the message body itself, so every MUA — webmail, Outlook, Apple Mail, mobile clients — renders it without relying on transport rules or recipient-side configuration. Tracked as #228.

Scope

Scope	Recipient match	Use case
System default	All recipient domains (no override)	Single banner used everywhere; recommended starting point
Per-recipient-domain	Specific local mailbox domain (e.g. `legal.example.com`)	Different copy or compliance language for one domain

Resolution at message time, in the body milter's `ExternalBannerModifier`:

1. Look up the first **local recipient**'s domain in `/etc/hermes/body_milter/banners/banner_by_recipient_domain`.
2. If a matching row exists, use it.
3. Otherwise fall back to the `_default` system-wide entry.
4. Otherwise no banner is applied.

Only the first local recipient is consulted — mixed-domain envelopes get the banner of the first local recipient encountered. This keeps the modification deterministic regardless of envelope ordering.

The `recipient_domain` field is **locked after creation**. Delete and re-create the row to change scope.

What counts as "external"

The body milter uses Postfix's `/etc/postfix/relay_domains` file as the source of truth for "local". A message is considered **inbound from an external sender** when:

- The `MAIL FROM` sender domain is **not** in `relay_domains`, AND
- At least one `RCPT TO` recipient domain **is** in `relay_domains`.

Internal-to-internal mail (sender + all recipients local) is classified as `direction = internal` and the banner is **not** applied. There is no separate allowlist of "trusted partner" external senders today — every external sender to a local recipient triggers the banner if one is configured for that recipient's domain.

Pipeline placement

The banner is injected at SMTP receive time by the **`hermes_body_milter`** container, the same container that emits outbound disclaimers (disclaimers.md) and organizational signatures (organizational-signatures.md). The milter listens on `inet:hermes_body_milter:8893` and Postfix consults it as part of `smtpd_milters`.

```
Inbound external MTA
        |
        v
Postfix smtpd
   +- smtpd_milters chain (in order):
   |    1. OpenDKIM            (verifies upstream DKIM signature)
   |    2. OpenDMARC           (DMARC policy + ARC verification)
   |    3. hermes_body_milter  (THIS -- banner prepended here)
   |       --> Authentication-Results header has already been written
   |           by OpenDKIM/OpenDMARC BEFORE the banner touches the body
   v
content_filter --> Amavis    (sees the banner-prepended body)
   v
Ciphermail              (server-side S/MIME or PGP, if configured)
   v
Postfix :10026          (multi-instance OpenDKIM re-signs the final body)
   v
Local delivery (Dovecot LMTP)
```

Key ordering points:

- **OpenDKIM verifies first.** The upstream sender's DKIM verdict is captured in `Authentication-Results:` headers **before** the banner is injected. The header is preserved on the message; the banner does not retroactively change what OpenDKIM saw at smtpd time.
- **Amavis sees the modified body.** Spam scoring runs against the banner-prepended message. This is intentional — the banner content is short and stable and does not skew SpamAssassin scores in practice.
- **Hermes' downstream re-sign covers the modified body.** The multi-instance OpenDKIM at `:10026` (#232) signs after Ciphermail rebuild, so the final outgoing-to-Dovecot body is covered by Hermes' own signature.

Behavior with signed and encrypted mail

The modifier inherits the same skip rules as Disclaimers for sealed envelopes:

Pattern matched	Meaning	Banner action
`Content-Type: multipart/signed; protocol="application/pkcs7-signature"`	S/MIME detached	**Skip**
`Content-Type: application/pkcs7-mime`	S/MIME opaque/enveloped	**Skip**
`Content-Type: multipart/signed; protocol="application/pgp-signature"`	PGP/MIME detached	**Skip**
`Content-Type: multipart/encrypted; protocol="application/pgp-encrypted"`	PGP/MIME encrypted	**Skip**
`-----BEGIN PGP SIGNED MESSAGE-----` in body	PGP inline-signed	**Skip**
`-----BEGIN PGP MESSAGE-----` in body	PGP inline-encrypted	**Skip**
Pre-existing `DKIM-Signature:` header on inbound mail	Upstream DKIM signed	**Modify anyway** (see below)

The corresponding flags on `ExternalBannerModifier` are `skip_on_signed = True`, `skip_on_pgp_inline = True`, `skip_on_dkim = False`.

Why the banner does NOT skip on upstream DKIM

About 95% of inbound mail today carries a `DKIM-Signature:` header. If the banner skipped on DKIM, the feature would be effectively inert — the warning would only land on the unsigned minority that needs it least.

Hermes already records the upstream DKIM verdict in `Authentication-Results:` before modifying the body. Recipients overwhelmingly read mail through Dovecot/IMAP and the recipient MUA does not re-verify upstream DKIM. The banner is therefore safe in the common case.

The narrower edge case — a recipient who forwards Hermes-banner'd mail to a downstream MX that **does** re-verify upstream DKIM — is addressed by ARC sealing (#229). Hermes' ARC seal at `:10026` records `cv=fail` for the upstream chain (because we modified the body), but the seal itself is mathematically valid and the downstream MX can trust Hermes' ARC verdict if Hermes is on its allowlist. See ARC Settings for the full discussion of the `cv=fail`-by-design pattern.

> **Operational consequence.** Banner injection breaks the original sender's DKIM body hash and any upstream ARC body hash. **This is by design.** Hermes is the authoritative auth boundary for the domains it relays; customer downstream MX servers must allowlist Hermes and accept its delivered mail without re-running DKIM/SPF/DMARC/ARC. A downstream MX that re-verifies upstream auth on mail Hermes forwards is misconfigured — cross-ref ARC Settings, DKIM Settings, and DMARC Settings.

Position: prepend vs append

Position	Behavior	Recommendation
Top (`prepend`)	Banner becomes the first child of the message body (above any quoted history)	**Industry standard** — users see the warning before reading any content
Bottom (`append`)	Banner is appended after the user-visible body	Available for sites that prefer it; rarely used

Both positions are implemented end to end (unlike Disclaimers, where only `append` is honored in v1). HTML prepend is done with BeautifulSoup: the banner fragment is inserted as the first child of `<body>` when present, otherwise prepended to the root.

Templates

Banners use a **server-side template gallery**, not a free-form WYSIWYG editor. Quill 2.x's HTML normalization strips inline styles that Gmail and Outlook need (the same problem hit on Organizational Signatures #226 Phase 2 and on this feature), so admins pick a template and fill in form fields; the server renders pixel-perfect HTML at save time.

Bundled templates (each `inc/external_banner_templates/<key>.cfm`):

Template key	Display name	When to pick it
`warning_yellow`	**Warning Yellow**	Default. Yellow background with orange accent. Matches Microsoft 365 / Mimecast banner style most users recognize
`critical_red`	**Critical Red**	Red background, white text. Phishing-prone industries or post-incident periods where alert level needs to be raised
`subtle_info`	**Subtle Info**	Light gray with blue accent. Less alarming for high-volume inbound (support/sales) where alert fatigue is a concern
`plain_text`	**Plain Text**	Bold prefix + text, no background or border. Maximum cross-MUA compatibility, including text-only clients

All four templates expose the same field set:

Field	Type	Default	Notes
`prefix`	text	`[EXTERNAL]`	Short tag rendered bold at the start. Plain ASCII recommended for Outlook
`headline`	text	"This message originated from outside your organization."	First line, regular weight
`body`	text	"Do not click links or open attachments unless you recognize the sender..."	Second line, smaller text
`show_learn_more`	checkbox	`false`	Reveals the next two fields
`learn_more_url`	url	*empty*	Optional link to internal phishing-awareness training or wiki
`learn_more_label`	text	"Learn more about phishing"	Visible label for the learn-more link

All templates emit **table-based HTML with `bgcolor=` attributes** so Outlook (which strips inline CSS but honors deprecated HTML attributes) renders the banner correctly. Inline styles are belt-

and-suspenders for Gmail, Apple Mail, and mobile clients.

The edit page renders a live preview in an iframe via `inc/render_external_banner_preview.cfm` so the admin sees exactly what `save_external_banner_action.cfm` will store.

Files generated on save/delete

`inc/external_banner_write_and_reload.cfm` runs after every save or delete and rewrites the entire on-disk state from the `external_banners` table:

```
/etc/hermes/body_milter/banners/banner_by_recipient_domain
    <recipient_domain>\t<option>
    _default\t<option>          special key, system-wide fallback

/etc/hermes/body_milter/banners/files/<option>/
    body.txt          plain-text banner (auto-derived at save)
    body.html         pre-rendered html banner
    position          "prepend" or "append" sidecar file
    images/           per-banner inline images (#230 cid pattern)
        1.png
        2.jpg
        ...
```

Where `<option>` is:

- `banner_default` for the system-wide row (NULL `recipient_domain`)
- `banner_<sanitized_recipient_domain>` for per-domain overrides (non-alphanumeric characters replaced with `_`)

The `files/` subdirectory is wiped on every regen (per-banner subdirs deleted recursively; the `.gitkeep` is preserved). This guarantees deleted rows and renamed scopes never leave stale files behind.

No reload step needed. The body milter mtime-stats each map file on every message and reloads automatically when its mtime changes. The CFML `cffile` write to the map file is enough to make the change take effect on the next message.

Plain-text part

The HTML body stored in `external_banners.body_html` is rendered server-side from the chosen template. The plain-text counterpart in `body_text` is auto-derived at save time:

- `<br>` becomes a newline
- `</p>`, `</li>`, `</tr>`, `</td>`, `</div>` become newlines
- All remaining tags are stripped
- Runs of 3+ newlines collapse to 2

The plain-text version is shipped to recipients viewing the message as `text/plain`. Inline images are omitted from the plain-text part — data URLs don't translate to text and recipients in text mode see the banner copy without image markers.

Inline images (#230)

The banner modifier inherits the `#230` cid inline-image pattern from Disclaimers. If a template's HTML contains `<img src="cid:banner_<option>_img_<N>">` references, the body milter:

1. Loads matching `images/<N>.<ext>` files from the option directory.
2. Attaches each as an `image/<format>` MIME part with `Content-ID: <banner_..._img_N>` and `Content-Disposition: inline`.
3. Wraps the message as `multipart/related` so MUAs resolve cid references against the inline parts.

The cid prefix is `banner_` so banner images cannot collide with `disclaimer_` or `signature_` cids inside the same composed message (the three modifiers can all add images to the same outbound; namespacing keeps them separate).

The bundled templates do not currently use inline images — banners are pure text. The infrastructure is present for future template additions (logo, warning icon, etc.).

Failure semantics

The body milter is **graceful-degradation** by design. Postfix's `milter_default_action = accept` means:

- Milter container down or unreachable -> mail flows **unmodified** (missed banner, no delivery outage)
- Map file unreadable -> no entries match -> all mail flows unmodified
- Per-option files missing -> log + skip the modify -> mail flows unmodified
- MIME parse exception -> caught and logged -> mail flows unmodified
- Modifier raises any other exception -> caught and logged -> mail flows unmodified

In every failure case, mail keeps flowing. Worst case is a missed banner, never lost mail. Compare the legacy "modify in amavis hook" approach (#214 Phase 3 v1, retired) which silently dropped messages when the in-place body modification desynced amavis's internal state.

Disabled rows

Rows with `enabled = 0` are skipped entirely during regen — no files written, no map entry. The milter never matches that scope until the row is re-enabled. Useful for staging copy changes before going live (build the new row disabled, preview it on `edit_external_banner.cfm`, flip the switch when ready).

Schema

```
CREATE TABLE IF NOT EXISTS external_banners (
  id               int(11)         NOT NULL AUTO_INCREMENT,
  recipient_domain varchar(255)    DEFAULT NULL,                -- NULL = system default
  template_key     varchar(64)     NOT NULL DEFAULT 'warning_yellow',
  fields_json      longtext        DEFAULT NULL,                 -- form values for re-edit
  body_text        longtext        DEFAULT NULL,                 -- auto-derived plain text
  body_html        longtext        NOT NULL,                     -- pre-rendered html
  position         enum('prepend','append') NOT NULL DEFAULT 'prepend',
  enabled          tinyint(3)      NOT NULL DEFAULT 1,
  updated_at       timestamp       NULL DEFAULT current_timestamp() ON UPDATE
current_timestamp(),
  PRIMARY KEY (id),
  UNIQUE KEY uk_recipient_domain (recipient_domain)
);
```

The `UNIQUE KEY` on `recipient_domain` ensures only one row per recipient domain (and at most one system-default row where `recipient_domain IS NULL`). The `fields_json` blob stores the original form values so reopening the editor restores exactly what the admin typed; `body_html` is the rendered output the milter actually ships.

Verifying it works

The banner appears in the message body, so the easiest verification is to send an inbound message from an external account to a local mailbox and view the result in any MUA (webmail,

Outlook, Apple Mail). Beyond that:

- **Body milter logs** — the modifier logs `external_banner applied: option=<name> position=<prepend|append> plain=<n> html=<n>` per modified message. Surface with `docker logs hermes_body_milter` or via System Logs.
- **`Authentication-Results:` header** is preserved from upstream and visible in the recipient's "view source"; this confirms OpenDKIM ran **before** the banner.
- **`ARC-Seal: ... cv=fail`** in the outgoing message confirms the body was modified after the upstream chain — expected behavior, cross-ref ARC Settings.

Related

- Disclaimers — the outbound counterpart; same `hermes_body_milter` container, parallel design (sender-keyed instead of recipient-keyed)
- Organizational Signatures — second outbound modifier in the same container, with per-recipient resolution
- ARC Settings — full explanation of `cv=fail` after body modification and the Hermes-as-auth-boundary model
- DKIM Settings, DMARC Settings — upstream-verdict context preserved in `Authentication-Results`
- Domains — local mailbox-hosting domains drive the per-domain dropdown on `edit_external_banner.cfm`
- System Logs — surface the body-milter log stream for troubleshooting

Link Guard

Link Guard

Pro Edition feature. Maps to **Email Policies > Link Guard** (`view_linkguard.cfm`, `inc/linkguard_write_and_reload.cfm`).

Link Guard provides **time-of-click protection** for inbound mail. Links in delivered messages are rewritten to point at a Hermes redirect endpoint; when a recipient clicks, Hermes evaluates the destination's reputation **at that moment** and decides — instantly — whether to allow, warn, or block. This closes the gap that delivery-time scanning misses: a link that is clean when the message arrives but is weaponized hours or days later.

Protection travels with the link. It works in the inbox, after a forward to a colleague, or when the message is opened days later on a phone — because the verdict is computed on click, not on delivery.

Components

Link Guard spans three containers plus the admin console:

Component	Role
`hermes_body_milter`	Rewrites inbound links at SMTP receive time (`LinkGuardModifier`) and restores original links on outbound replies/forwards (`LinkGuardRestoreModifier`).
`hermes_linkguard`	The verdict + redirect engine. Serves the public `/lg/` click endpoint and a console-only management API. Holds the operational SQLite store (verdict cache, feeds, click log).
`hermes_nginx`	Reverse-proxies `/lg/` from the public console host to the Link Guard container's public port.
Admin console	`view_linkguard.cfm` page; on save, `inc/linkguard_write_and_reload.cfm` pushes settings, scope, URL rules, and HMAC keys to the engine and reloads the milter maps.

The Link Guard container exposes a **hardened two-port split**:

Port	Surface	Exposure
8894 (public)	`GET /lg/?t=<token>`, `POST /lg/proceed`, `GET /healthz`	nginx-proxied; reachable by recipients clicking links
8895 (mgmt)	`POST /api/config`, `POST /api/keys`, `POST /api/feed-refresh`, `GET /api/stats`	console-only; never exposed publicly

The public surface can never push config or read keys; the management surface is never reachable from the internet.

Pipeline placement

Link Guard's two body-modification steps run in the **`hermes_body_milter`** container, the same milter Postfix consults for disclaimers, signatures, and banners.

```
INBOUND (rewrite)                          OUTBOUND (restore)
External MTA ──► Postfix smtpd             User reply/forward ──► Postfix smtpd
        │                                          │
        ▼                                          ▼
   smtpd_milters chain:                       smtpd_milters chain:
     1. OpenDKIM                                1. OpenDKIM
     2. OpenDMARC                               2. OpenDMARC
     3. hermes_body_milter                      3. hermes_body_milter
          └─ LinkGuardModifier:                      └─ LinkGuardRestoreModifier:
             rewrite links ──► /lg/ token               unwrap /lg/ tokens ──► original URLs
        │                                          │
        ▼                                          ▼
   Amavis ──► Ciphermail ──► deliver          Amavis ──► deliver to external
```

Rewriting happens at smtpd time, before content filtering. Hermes' own DKIM signs at the Postfix `:10026` re-injection (downstream of the milter), so the signature always covers the rewritten body the recipient receives. Inbound mail that arrives already DKIM-signed, S/MIME-signed, or PGP-sealed is **skipped** — the same envelope-detection logic the disclaimer feature uses — so Link Guard never breaks an existing signature.

The click flow

```
Recipient clicks rewritten link
        |
        ▼
GET https://<console-host>/lg/?t=<token>   (nginx ──► linkguard :8894)
        |
        ├─ token invalid / expired ──► block page
        ▼
   resolve original URL from token
        |
        ▼
   verdict pipeline (see below) ──► {clean | suspicious | malicious}
        |
        ▼
   admin action for that tier:
     clean      ──► 302 redirect to the real URL   (default)
     suspicious ──► warning interstitial (resolved host shown; user may proceed)
     malicious  ──► block page (hard block, or block_override allowing proceed)
```

Every click is logged (recipient domain, URL hash, resolved host, verdict, source, action taken, client IP) for the reporting dashboard.

Verdict pipeline

`verdict.resolve(url, recipient_domain)` evaluates layers in **precedence order** and returns the first match:

#	Layer	Result	Notes
1	**Admin blocklist**	malicious	Operator-curated, console-managed
2	**Admin allowlist**	clean	Operator-curated; trumps feeds and heuristics
3	**Verdict cache**	cached result	Avoids re-running heuristics / re-hitting external APIs
4	**Open-redirect extraction**	escalate	Embedded target in an open-redirect param / nested URL is resolved and inherits its verdict (no fetch)

#	Layer	Result	Notes
5	**Local feeds**	malicious	URLhaus / OpenPhish, stored in SQLite; only ever escalate
6	**Heuristics**	suspicious	Lookalike/punycode, IP-literal host, `@` in authority, known shorteners, excessive subdomains, **abused cloud-storage/redirector hosts**
7	**GSB / VirusTotal**	malicious	Optional, admin-supplied keys; string-reputation lookups, cached. Also escalates a warn-tier (step 6) link to a **hard block** when flagged
8	**Guarded redirect-follow**	escalate	**Optional** (admin toggle): follow the `30x` chain under SSRF guards, verdict the final destination; also runs proactively on a step-6 warn so a malicious destination becomes a hard block
9	**Default**	clean	Nothing flagged it

Steps 1–7 never fetch the target URL — every reputation check sends the URL only as a *string* (Google Safe Browsing, VirusTotal). The **only** layer that makes an outbound request is the optional **guarded redirect-follow** (step 8), and only when an admin enables it; every hop is SSRF-fenced (see Redirect detection below). Local feeds only ever escalate a link to malicious; they never auto-allow.

URL shorteners are flagged **suspicious** (warn), not blocked — a shortener hides its real destination, which is exactly what time-of-click protection exists to surface. The warning interstitial shows the resolved host so the user can make an informed choice. The shortener set is a curated list of **generic, anyone-shortens-anything** services (referenced from the maintained `PeterDaveHello/url-shorteners` list); **branded** corporate shorteners (`aka.ms`, `amzn.to`, `a.co`, `adobe.ly`, …) are deliberately **excluded** so legitimate branded short links don't warn-fatigue users. When `follow_redirects` is enabled, the real destination of *any* shortener is resolved regardless. VirusTotal requires **≥2 vendors** flagging a URL before it counts as malicious, to cut false positives.

Verdict tiers and actions

Each verdict tier maps to an admin-configurable action:

Tier	Setting	Default action	Behavior
clean	`action_clean`	`redirect`	302 straight to the destination
suspicious	`action_suspicious`	`warn`	Interstitial; user may proceed
malicious	`action_malicious`	`block`	Block page

Available actions: `redirect` / `allow` (pass through), `warn` (interstitial with proceed), `block` (hard block, no proceed), `block_override` (block page that allows an explicit proceed).

A hard `block` can never be bypassed. `POST /lg/proceed` re-resolves and re-authorizes the verdict server-side — only a `warn` tier, or a `block_override` tier with the override flag, is allowed to continue. A user cannot escape a hard block by replaying the proceed request.

Redirect detection

Attackers increasingly chain through **reputable hosts** — `storage.googleapis.com`, `firebasestorage.googleapis.com`, `*.web.app`, Azure `*.blob.core.windows.net`, Cloudflare `*.r2.dev` / `*.pages.dev`, and classic open redirectors like `google.com/url?q=…` — because the host reputation is clean, so a string-only check passes the link through. Link Guard adds three layers to catch this "living off trusted services" pattern.

Open-redirect extraction (always on, no fetch)

The engine scans a link's query and fragment (raw, once-, and twice-URL-decoded) for an **embedded** `http(s)://` target on a different host than the redirector — the real destination of `…/url?q=https://evil.example`. That embedded target is re-run through the verdict pipeline (string-only); if it is suspicious or malicious, the original link inherits the verdict (`source = redirect`, detail `open-redirect -> <host>`). A benign embedded URL is a no-op. No outbound request is made.

Abused-host heuristic (always on, no fetch)

Cloud-storage and app-hosting hosts commonly abused to host or bounce to phishing are flagged **suspicious** (warn) in the heuristics layer, so the user gets the interstitial and the resolved host instead of a silent redirect. Gated by `flag_cloud_storage` (default on). The match is suffix-based (`host == suffix` or `host` ends with `.suffix`).

The list is an **operator-managed table** (`linkguard_abused_hosts`), **shipped pre-seeded** with a curated baseline — there is no authoritative machine-readable feed of "abused hosting platforms" (the actual-bad *URLs* are what the URLhaus / OpenPhish feeds cover), so the seed is hand-curated from public abuse research (Trustwave SpiderLabs, Proofpoint, Netskope, Phishing.Database). It covers object storage (`storage.googleapis.com`, `firebasestorage.googleapis.com`, `firebaseapp.com`, `web.app`, `appspot.com`, `blob.core.windows.net`, `s3.amazonaws.com`), edge/static-site hosting (`r2.dev`, `pages.dev`, `workers.dev`, `github.io`, `netlify.app`, `vercel.app`, `herokuapp.com`, `onrender.com`, `surge.sh`), free site builders (`weebly.com`, `wixsite.com`, `000webhostapp.com`), and tunneling services (`trycloudflare.com`, `ngrok.io`, `ngrok-free.app`).

Manage it from the **Abused / redirector hosts** card on the Link Guard page:

- **Add** any host (covers its subdomains) — react to a new abuse pattern immediately, no rebuild.
- **Remove** a baseline entry you don't want (select → Delete). The seed is **one-time** (only when the table is empty), so your add/deletes survive upgrades.
- Or **suppress** without deleting by adding the host/path to the admin **URL allowlist** — the allowlist wins over the heuristic (e.g. a company that legitimately serves files from `storage.googleapis.com/<your-bucket>`).

Add/delete in this card **applies immediately** — the handler regenerates the engine's `config.json` on each change (same as the URL-rules and domains cards), so there's no separate Save for table edits. The `flag_cloud_storage` master switch and the other settings live in the settings card and take effect on its **Save & reload settings** button.

Guarded redirect-follow (optional, the one fetch layer)

When `follow_redirects` is enabled, the engine — at click time, after the string layers — follows the HTTP `30x` redirect chain and verdicts the **real destination**, catching a trusted-host link that issues a server-side redirect to phishing (the `storage.googleapis.com` → phishing case). This is the **only** layer that makes an outbound request, so every hop is fenced (`_safe_to_fetch`):

- **http/https only**, and only a **standard web port** (80/443).
- Each hop's host must resolve to **public IPs only** — any answer in a loopback / RFC1918 / link-local / reserved / multicast range aborts the follow. This stops the follower being used as an SSRF pivot into the internal Docker network.
- **HEAD only**, no body downloaded; bounded by `follow_max_hops` (default 5), a short per-hop timeout, and a loop guard.
- No cookies/credentials sent; neutral User-Agent; `Referrer-Policy: no-referrer`.

Each followed hop is verdicted string-only (a follow never recurses into another follow). If the chain reaches a suspicious/malicious destination, the link inherits that verdict (`source = redirect`, detail `redirect chain -> <final host>`). A follow failure (timeout, guard stop, HEAD not allowed) fails

closed for the follow — the link keeps whatever verdict the string layers produced; a click is never blocked by a follow error.

Proactive on warn-tier links. A shortener or abused-host link is flagged *suspicious* by the heuristics layer (step 6), which is *before* the follow layer in precedence. When `follow_redirects` is on, those warn-tier links are **also followed at that point** — proactively, during the verdict, not on "Continue":

- destination resolves **malicious** → the link is escalated to a **hard block** (the user never gets a *Continue* into a known-bad site);
- destination resolves **clean or suspicious**, or the chain can't be followed → the link **stays a warn**, now annotated with the resolved final host (`<reason> (resolves to <final host>)`).

So with redirect-following enabled, a `storage.googleapis.com` / shortener link that actually bounces to phishing becomes a **block**, while one that resolves somewhere benign stays an informative **warn**. The verdict (block or warn) is cached, so the follow runs at most once per link per cache-TTL.

> **SSRF posture.** Steps 1–7 never fetch the target. Enabling `follow_redirects` is a deliberate trade: it resolves redirect chains a string check cannot, at the cost of one guarded outbound request per uncached click (latency) and a controlled egress surface. The residual DNS-rebinding window (resolve-then-connect) is accepted for this release. Leave it **off** to preserve the zero-fetch guarantee; the two no-fetch layers above still run.

Out of scope for this release: following **JavaScript** or `<meta>` **refresh** redirects, which require fetching and parsing the page body — tracked as a later enhancement.

Tokens — stateful v2 (default) with stateless v1 fallback

The rewritten link carries a token in the `t` query parameter. Two formats exist:

v2 — stateful (default). The token is just `2.<128-bit opaque id>`. The milter writes the mapping `id → {original_url, recipient_domain, expiry}` to a shared SQLite store (`url_map.db`) on the `linkguard_data` volume; the Link Guard container reads it. Because the token itself is tiny, **there is no link-length limit** — every link is protected regardless of how long the original URL is. This closes the v1 over-length fail-open gap (see below).

v1 — stateless (fallback + in-flight). The token is a self-contained HMAC signature: `1.<recipient_domain>.<url>.<expiry>.<signature>`. The milter mints a v1 token if the shared store is

unavailable (e.g. off-box deployment, or transient DB contention), so **mail flow never depends on the store**. v1 tokens already in delivered mailboxes continue to verify until they age out via the token TTL.

The milter's mint/verify logic is a **byte-for-byte mirror** of the container's `lg_token.py`, so the container verifies exactly what the milter mints. The `url_map.db` store uses a rollback journal (not WAL), so the container can read it cross-container without a `-shm` file.

> **Why v2 exists.** Under v1, a URL longer than the inline cap was left *unprotected* (the original link was passed through unrewritten). An attacker could pad a URL past the cap to dodge Link Guard entirely. v2's short opaque id removes the length dependency, so nothing is ever skipped. The `max_inline_url` setting is now a **fallback-only** bound for the v1 path.

Outbound link restoration

`restore_outbound` (default **on**) unwraps Link Guard tokens back to the original URLs on **outbound** mail — when a recipient replies to or forwards a protected message, the quoted history shows the real links again, not `/lg/?t=...` redirects. This keeps conversations readable and prevents Hermes redirect URLs from leaking to external parties. (Microsoft 365 was verified not to strip the tokens on manual replies, so restoration is the correct default.)

HMAC key rotation

The signing key for v1 tokens is rotatable from the console. Rotation keeps a **current + previous** overlap: newly minted tokens use the current key, while tokens signed with the previous key still verify until they age out. The teardown on a Pro license lapse blanks only the dispatch maps and **never** the keys, so in-flight links keep resolving and a renew resumes minting with the same key.

Settings reference

Settings live in the `parameters2` table under `module = 'linkguard'` (not `system_settings`). On save they are pushed to the engine via `POST /api/config`.

Setting	Default	Meaning
`enabled`	`0`	Master on/off for Link Guard

Setting	Default	Meaning
`redirect_base_url`	(console host)	Public base URL for `/lg/` links
`action_clean`	`redirect`	Action for clean verdicts
`action_suspicious`	`warn`	Action for suspicious verdicts
`action_malicious`	`block`	Action for malicious verdicts
`restore_outbound`	`1`	Unwrap tokens on outbound replies/forwards
`token_ttl_days`	`14`	How long a rewritten link stays valid
`max_inline_url`	`4000`	Fallback-only length bound for v1 stateless tokens
`rate_limit_per_min`	`120`	Per-client-IP rate limit on `/lg/`
`flag_cloud_storage`	`1`	Flag abused cloud-storage/redirector hosts as suspicious (warn)
`follow_redirects`	`0`	Follow `30x` redirect chains at click time (guarded outbound fetch)
`follow_max_hops`	`5`	Max hops to follow when `follow_redirects` is on
`cache_ttl_clean_hours`	`24`	Verdict cache lifetime — clean
`cache_ttl_suspicious_hours`	`6`	Verdict cache lifetime — suspicious
`cache_ttl_malicious_hours`	`168`	Verdict cache lifetime — malicious
`feed_urlhaus_enabled`	`1`	Pull the URLhaus blocklist feed
`feed_openphish_enabled`	`1`	Pull the OpenPhish blocklist feed
`feed_refresh_minutes`	`60`	Feed refresh interval
`gsb_enabled` / `gsb_api_key`	`0` / —	Google Safe Browsing lookups (optional key)
`vt_enabled` / `vt_api_key`	`0` / —	VirusTotal lookups (optional key)
`clicks_retention_days`	`90`	Click-log retention for reporting

Two additional console-managed lists drive the verdict pipeline:

- **Protected recipient domains** (`linkguard_domains`) — which recipient domains have their inbound links rewritten. A `_default` catch-all entry protects all domains.
- **URL allow / block rules** (`linkguard_url_rules`) — operator allow/block patterns that take precedence over feeds and heuristics (layers 1–2 above).
- **Abused / redirector hosts** (`linkguard_abused_hosts`) — the seeded, operator-managed warn list for the abused-host heuristic (layer 6); see Redirect detection.

Reputation feeds and optional API lookups

- **URLhaus** and **OpenPhish** are pulled on the `feed_refresh_minutes` interval into the container's SQLite store and matched as exact URL-hash lookups (a phishing URL on a shared host blocks only that URL, not the whole host).
- **Google Safe Browsing** and **VirusTotal** are off by default; enable each and supply an API key to add a string-reputation layer. Results are cached per the cache-TTL settings to limit API calls.

Setting up VirusTotal and Google Safe Browsing

Both are **optional** — Link Guard works without them (admin lists + feeds + heuristics + redirect-following). They add a malicious-verdict layer that checks the URL **as a string** against the provider (the target is never fetched). Each is enabled the moment you save its key (no Save & Reload), and a malicious result will **escalate even a warn-tier link to a hard block**.

Where to enter keys: Link Guard page → **Reputation sources** → the provider's **Edit key** button. Entering a key auto-enables the provider; clearing it disables and wipes the stored key.

VirusTotal (recommended):

1. Create a free account at virustotal.com and sign in.
2. Open your profile menu → **API key**, and copy it.
3. Paste it into Link Guard → VirusTotal → Edit key.
4. **Quota:** the **free Public API** is rate-limited (≈ **4 lookups/minute, 500/day, ~15.5k/month**) and is for non-commercial use; high-volume or commercial sites need a paid **Premium API** key. The verdict cache (and the per-tier cache-TTL settings) is what keeps you under quota — which is why **Clear verdict cache** warns before re-querying. A verdict counts as malicious only when **≥ 2 vendors** flag the URL.

Google Safe Browsing:

1. In the Google Cloud Console, create (or pick) a project.
2. **APIs & Services → Library →** search **"Safe Browsing API" → Enable**.
3. **APIs & Services → Credentials → Create credentials → API key**, and copy it.
4. Paste it into Link Guard → Google Safe Browsing → Edit key.
5. ⚠ **Commercial-use caveat:** the Safe Browsing **Lookup API v4** that Link Guard uses is **free but designated non-commercial only by Google, and is deprecated** in favour

of the paid **Web Risk API** for commercial use. Review Google's usage terms for your deployment. If you only want one reputation provider, **VirusTotal is the simpler choice**; Web Risk support may be added later.

Privacy: with either provider enabled, the clicked URL string is sent to that provider's API for the lookup. Nothing else leaves Link Guard.

Branded interstitials

The warning and block pages are served by the container (`templates.py`) and carry Hermes SEG branding — an inline logo, a "Hermes SEG Link Guard" header, and a footer link to hermesseg.io — rather than a generic browser error. The warning page shows the **resolved host** so a user can judge a shortened or suspicious link before proceeding.

Reporting and diagnostics

The admin page includes:

- **Check a URL** — enter any URL to see the live verdict, which pipeline layer decided it, and the resolved host. This is side-effect-free (`verdict.resolve(cache_write=False)`) so it does not pollute the cache.
- **Recent activity** — a table of recent clicks (Time, Recipient domain, Host, Verdict, Decided by, **Detail**, Action). The **Host** column is the link's original host; the **Detail** column carries the verdict detail per click — e.g. `redirect chain -> evil.example`, `Safe Browsing: SOCIAL_ENGINEERING`, `open-redirect -> …` — so a follow-decided click shows its **final/destination host** right in the table (host-only, never the full URL). The `redirect-follow` container log below additionally captures follows that came back **clean** (which still produce a click row, now with the chain in Detail).
- **Redirect-follow log** — when `follow_redirects` is on, every actual follow is logged (hosts only, never full URLs):

  ```
  docker logs hermes_linkguard | grep "redirect-follow:"
  # redirect-follow: bit.ly -> evil.example (1 hop) verdict=malicious
  ```

  This shows *what* got followed, the hop chain, the hop count, and the resulting verdict — including follows that came back clean (which leave no row in Recent activity).
- **Troubleshooting commands** — a collapsible card of `docker exec` one-liners for inspecting the scope map, store, and feeds.
- **Clear verdict cache** — a button at the **top of the page** that flushes the **entire** verdict cache (every source, including GSB / VirusTotal and feed-derived results) via the mgmt endpoint `POST /api/cache-clear`, forcing a complete re-evaluation on next click. Its confirm

modal warns that this **re-queries Google Safe Browsing / VirusTotal**, consuming additional lookups against your API quota. You rarely need it: a config edit already auto-purges the *config-dependent* cache (see below).

Verdict cache invalidation

The engine caches each URL's verdict (per-source TTL — clean 24h, suspicious 6h, malicious 168h) to avoid re-running heuristics / re-hitting APIs on every click. Two things keep it from going stale against admin edits:

- **Automatic, scoped:** when `config.json` reloads after any console change (remove an abused host, add/remove a URL rule, toggle follow), `config.py` drops the **config-dependent** cached verdicts (`admin` / `heuristic` / `redirect` / `none`) so the edit takes effect on the **next click** — no TTL wait, no button. Feed (`local`) and external (`gsb`/`vt`) caches are kept (they don't depend on console config). This is why removing a host stops the warning immediately.
- **Manual, full:** the **Clear verdict cache** button above flushes *everything*, for the rare "force a complete re-check / I think an external result is stale" case (it re-queries GSB/VirusTotal, so it's confirm-gated).

Deployment — in-stack or separate host

Link Guard runs **inside the Hermes SEG stack** (the default; `hermes_linkguard` service on the compose network) or on a **separate host** for isolation and scale. The nginx `/lg/` location lives in the **vhost template**; it is delivered by regenerating the per-domain vhosts from the template (via the headless `schedule/regen_nginx_config.cfm`), not by hand-editing a generated vhost.

When Link Guard runs off-box, the milter cannot reach the shared `url_map.db`, so it mints **v1 stateless** tokens — the same fallback path described above. Mail flow is never affected by the container's location or availability.

Failure semantics

Link Guard is **graceful-degradation** by design, consistent with the rest of the body milter:

- **Link Guard container down / unreachable** → the milter falls back to v1 stateless tokens (rewrite still happens); already-delivered links cannot be resolved until the container returns, but **mail keeps flowing** (`milter_default_action = accept`).

- **Shared store unavailable** → milter mints v1 tokens; mail flow never depends on the store.
- **Scope map empty** (e.g. after a Pro license-lapse teardown) → no inbound links are rewritten, but mail flows unmodified. Re-enabling / re-saving Link Guard repopulates the map.
- **External API (GSB/VT) error or timeout** → that layer is skipped; the verdict falls through to the next layer (worst case `clean`).

In every failure case the worst outcome is a missed rewrite or a fall-through verdict — never lost mail.

Files and data locations

Path	Container	Contents
`/etc/hermes/body_milter/linkguard/linkguard_by_recipient_domain`	body_milter	Scope map: protected recipient domains (`_default` = all)
`/var/lib/linkguard/url_map.db`	body_milter (writer) / linkguard (reader)	v2 token id → original URL store, on the shared `linkguard_data` volume
`/opt/linkguard/app/`	linkguard	Engine code (server, verdict, feeds, store, token, templates)
Operational SQLite store	linkguard	Verdict cache, feed entries, click log

The scope map is mtime-watched by the milter and reloaded on the next message when it changes — no explicit milter reload step is needed after a console save.

Security properties (summary)

- **SSRF-fenced** — steps 1–7 never fetch the target; the optional redirect-follow (step 8) is the only fetch, and every hop is restricted to http/https + standard ports + hosts that resolve to public IPs only.
- **Hard blocks are unbypassable** — `/lg/proceed` re-authorizes server-side.
- **Hardened port split** — public click surface cannot push config or read keys.
- **Rate-limited** public surface (`rate_limit_per_min`, per client IP).
- **Signature-safe** — inbound S/MIME, PGP, and upstream-DKIM-signed mail is skipped, never re-bodied.
- **Mail-flow-safe** — the container being down, off-box, or torn down on a license lapse never blocks delivery.

Organizational Signatures

Organizational Signatures

Pro Edition feature. Maps to **Email Policies > Org Signatures** (`view_org_signatures.cfm`, `edit_org_signature.cfm`, `org_signature_delete.cfm`).

Hermes attaches a centrally-managed signature to outbound mail at the gateway. Admins design the signature once per domain (and optionally per department); every user on that domain gets a personalized version of it on every outbound message — no per-user setup required.

Two signature types, one pipeline

Hermes ships two distinct signature concepts that run through the same body milter and the same resolver:

Type	Tier	Owner	Storage	Per-domain control
Personal Signature	Community + Pro	The user (in `/users/2/view_signature.cfm`)	`user_signatures` table, one row per user	Toggled via `domains.allow_user_signatures`
Organizational Signature	Pro only	The admin (in `Email Policies > Org Signatures`)	`org_signatures` table, one row per `(domain_id, department_label)`	One default per domain + optional per-department variants

The milter never decides which one to apply at message time. The CFML resolver picks a winner per mailbox at admin-action time and writes a precomputed `sender → option` map; the milter just looks up the option and applies whatever it finds.

Department names — single source of truth

Departments are defined once on the **mailbox edit form** (Email Server > Mailboxes > Edit Options > Personal Information > Department), as free-text values typed by the admin. There is no

separate "Departments" table; a department exists as soon as one mailbox is in it.

The Org Sig form's Department field is a **strict dropdown sourced from the distinct `mailboxes.department` values for the selected domain.** This means:

- You cannot create an Org Sig for a dept that has no mailboxes — the dept won't appear in the dropdown.
- The dept name on both sides is guaranteed to match exactly. No typo-class drift.
- Workflow: assign at least one mailbox to the new dept first, then come back and create the Org Sig targeting it.
- Changing the domain in the Org Sig form repopulates the dropdown with that domain's depts via JavaScript (no AJAX round-trip; the per-domain map is dumped into a JS const at page load).

The mailbox edit form's Department field is a free-text input with a `<datalist>` **typeahead** showing the same per-domain dept list. Admins can pick an existing dept (auto-completes) or type a brand-new dept name (which then appears in the dropdown next time).

If you edit an existing Org Sig whose `department_label` no longer matches any current mailbox (the dept was renamed elsewhere, or all mailboxes in it were reassigned), the orphan value is preserved in the dropdown with a `(no mailboxes)` suffix so you can still see and edit/delete the row instead of silently losing the value.

The resolver at send time does a **case-insensitive trimmed match** against `mailboxes.department`, so casing or whitespace drift across edits is forgiving even in the rare cases the dropdown is bypassed (e.g. direct SQL changes).

Resolution order

For every enabled mailbox, the resolver walks this priority chain top-down and stops at the first match:

```
1. Personal Signature
   └─ if domains.allow_user_signatures = 1
      AND user_signatures has an enabled, non-empty row for this mailbox
   -> wins. option = user_<sanitized_email>

2. Department Organizational Signature
   └─ else if mailboxes.department is non-empty
      AND org_signatures has enabled = 1 row matching
          (domain_id, department_label)
   -> wins. option = org_<row_id>
```

```
3. Domain Default Organizational Signature
   └─ else if org_signatures has enabled = 1 row matching
          (domain_id, department_label IS NULL)
   -> wins. option = org_<row_id>

4. None
   └─ no map entry; the milter applies no signature to this sender's mail.
```

The chain is **per-mailbox**, not per-message. The resolver runs at admin-action time (see Triggers), serializes the winning option for every mailbox into one map file, and the milter consults that map at send time. There is no per-message DB query and no per-message resolution logic in the milter.

Pipeline placement

Same chain as Disclaimers (#214) — see disclaimers.md for the full Postfix / OpenDKIM / Ciphermail picture. The summarized order:

```
External MTA / MUA submission
        |
        ▼
Postfix smtpd
   ├─ smtpd_milters chain (in order):
   |   1. OpenDKIM            (signs/verifies)
   |   2. OpenDMARC           (DMARC policy)
   |   3. hermes_body_milter  (THIS — signatures, then disclaimers)
   ▼
Amavis  →  Ciphermail  →  Postfix :10026 (DKIM sign) → external
```

Inside the body milter, **SignatureModifier runs before DisclaimerModifier**, so the layered output on the outbound message is:

```
[user body]
[signature]      ← Personal or Organizational, picked by resolver
[disclaimer]     ← if a disclaimer is configured for this sender
```

OpenDKIM signs at the `:10026` re-injection — after both modifiers — so Hermes' own DKIM signature always covers the recipient's view of the message (with signature and disclaimer baked in).

Templates

Phase 2A ships **six bundled templates** under `/admin/2/inc/org_signature_templates/`:

Template key	Layout
`modern_card`	Logo left, accent bar, contact stack right
`two_column_pro`	Left contact, right org block + CTA button
`with_social_bar`	Vertical contact + horizontal social-icon row
`banner_with_logo`	Full-width banner with logo top, contact below
`promo_footer`	Contact + bottom promotional image with link
`compact_text`	Minimal text-only, no images, no styling

Each template is a `.cfm` file that declares its **field schema** (text / email / url / color / checkbox / image fields with optional `showIf` gating) and renders pixel-perfect HTML when the renderer is invoked. Admins fill in the form on `edit_org_signature.cfm`; the gallery thumbnail + live sandboxed-iframe preview show the result before save.

All the auto-filled fields — Name, Title, Phone, Mobile, Email (`{{user.*}}` from the mailbox row) plus Website and Address (`{{org.*}}` from the domain row) — are **collapsed under an "Override auto-filled fields" toggle** by default. The admin doesn't see or edit them in the common case; the placeholders flow through to the rendered HTML unchanged and the milter fills in the recipient's data at send time. Toggling the override on exposes the fields for the rare cases that need literal text instead of substitution (shared mailboxes without personal info, seasonal URL overrides, etc.).

The genuinely admin-supplied fields stay always visible — Logo, accent color, show/hide toggles for each line, CTA text and URL, social URLs, disclaimer text, and any template-specific extras (banner height, promo image, etc.). These are the admin's actual editing surface.

Net workflow: pick a template, upload a logo, set the accent color, save. Done. Every mailbox on the domain gets a fully personalized signature on its next outbound message without any per-user form input.

Templates are version-controlled in the repo, not in the database. To add a new template, drop a new `.cfm` file in `org_signature_templates/`, add its key to the registry in `inc/org_signature_template_loader.cfm`, and produce a 240×140 thumbnail PNG. No schema migration needed.

Placeholder substitution at send time

The signature HTML stored in `org_signatures.rendered_html` (and on disk in `body.html`) keeps `{{namespace.field}}` tokens **literal**. The body milter substitutes them per recipient at message-send time against the sender's row in `sender_data.json`.

Available placeholders (Phase 2B):

Token	Source column
`{{user.first_name}}`	`mailboxes.first_name`
`{{user.last_name}}`	`mailboxes.last_name`
`{{user.title}}`	`mailboxes.title`
`{{user.phone}}`	`mailboxes.phone`
`{{user.mobile}}`	`mailboxes.mobile`
`{{user.department}}`	`mailboxes.department`
`{{user.email}}`	`mailboxes.username`
`{{org.name}}`	`domains.org_name`
`{{org.phone}}`	`domains.org_phone`
`{{org.address}}`	`domains.org_address`
`{{org.website}}`	`domains.org_website`
`{{org.logo_url}}`	`domains.org_logo_path` (raw URL — not cid: extracted)
`{{dept.name}}`	`mailboxes.department`

Tokens whose corresponding field is empty resolve to **empty string**, not literal `{{...}}`. So if `mailboxes.title` is blank for a given user, the `{{user.title}}` token disappears cleanly from delivered mail. Unknown namespaces (anything outside `user`, `org`, `dept`) are also substituted to empty.

The substitution is a single regex pass on the body html and body text inside `SignatureModifier.modify()` — well under a millisecond per message. The map and `sender_data.json` both live in process memory, refreshed only when their file mtime changes.

No DB connection from the milter, ever. All resolution and substitution data is precomputed by CFML and dropped on disk; the milter consumes the file artifacts.

Triggers — when the resolver re-runs

Both `signature_by_sender` and `sender_data.json` are rewritten in full by `inc/signature_regen_map.cfm` on every event that could affect a winner or a substitution value:

Event	Why it matters
Admin saves an Org Sig	New / edited row may win for senders that previously had no match or a lower-tier match
Admin deletes an Org Sig	Losers fall back to the next tier (or none)
Admin edits a domain (`allow_user_signatures` or `org_*` columns)	Per-domain toggle flips the Personal-vs-Org winner; org_* values feed `{{org.*}}` substitution
Admin edits a mailbox (Pro fields: first_name, title, dept, etc.)	`{{user.*}}` and `{{dept.name}}` substitution data changes; a department change can flip Default → Department winner
Admin adds a mailbox	New sender enters resolution and may pick up a domain-default Org Sig
Admin deletes a mailbox	Sender must drop from the map
User saves their Personal Signature	May now win over the previously-resolved Org Sig (or vice versa if disabling)

Each trigger runs the same shared resolver. **Full rebuild every time, not incremental.** With low-thousands of mailboxes the rebuild is well under a second, and the simplicity rules out drift bugs ("did we forget to update X for sender Y" can't happen).

The body milter mtime-watches both files and reloads in process memory on the next message after the file changes. **No SIGHUP, no IPC, no container restart.**

Files generated on save

The CFML resolver writes:

```
/etc/hermes/body_milter/signatures/signature_by_sender   sender → option map
/etc/hermes/body_milter/signatures/sender_data.json       sender → {{token}} dict
/etc/hermes/body_milter/signatures/files/<option>/
    body.txt         plain-text signature (auto-derived from html)
    body.html        html signature with cid: refs (placeholders intact)
```

```
    images/
        1.png        per-option inline images (#230 pattern)
        2.jpg
        ...
```

Where `<option>` is `user_<sanitized_email>` (Personal Sig) or `org_<row_id>` (Org Sig). `<sanitized_email>` is `bob.smith@example.com` → `bob_smith_at_example_com` (`@` → `_at_`, non-alphanumerics → `_`).

`signature_by_sender` example:

```
alice@example.com   org_42
bob@example.com     user_bob_at_example_com
carol@example.com   org_43
```

`sender_data.json` example (post-Lucee uppercasing — the milter normalizes to lowercase on load):

```
{
  "alice@example.com": {
    "USER.FIRST_NAME": "Alice",
    "USER.TITLE": "Sales Manager",
    "ORG.NAME": "Acme",
    "ORG.PHONE": "555-0100",
    "DEPT.NAME": "Sales"
  }
}
```

The `files/<option>/` dir is wiped before re-render on every save of that row, so deleted images and renamed scope keys never leave stale binaries behind.

Inline images (cid: extraction)

Same pattern as Disclaimers (#230) — see disclaimers.md for the MIME / multipart-related details.

For Org Signatures, the cid: namespace is `signature_org_<row_id>_img_<N>` (Personal Signatures use `signature_user_<sanitized_email>_img_<N>`). Both share the milter regex `cid:(signature_[\w.-]+_img_\d+)`, so cid: refs from either signature type are queued for inline attachment alongside any cid: refs from a domain disclaimer on the same message — no namespace collisions.

Per-template image fields use the **same data: → cid: pipeline** as user-pasted Personal Sig images. At admin-save time:

1. Admin uploads the file in the form (or pastes a URL — both are handled).
2. Browser converts the file to a `...` URI via `FileReader`, capped at 1 MB per image.
3. CFML renders the template; the resulting HTML carries the data: URI inline.
4. `inc/org_signature_write_files.cfm` extracts each `data:` URI, decodes the base64 into a binary file under `images/`, and rewrites the html to reference `<img src="cid:signature_org_<id>_img_<N>">`.
5. At message-send time the milter walks the cid: refs, attaches each image as an `image/<format>` MIME part with `Content-ID` and `Content-Disposition: inline`, and wraps the message as `multipart/related`.

`{{org.logo_url}}` is **not** cid: extracted — it's a raw URL substituted into the html as-is. Use it for hosted-elsewhere logos (your CDN, your website). Use the per-template **image** field for cid:-attached inline logos when you want them to render even in MUAs that block external images.

Behavior with S/MIME, PGP, DKIM-signed mail

Identical to Disclaimers — same skip rules in the same Modifier base class. Pre-signed envelopes, PGP inline, and pre-existing DKIM-Signature headers all cause the body milter to leave the message untouched.

See disclaimers.md "Behavior with S/MIME, PGP, and DKIM-signed mail" for the table of patterns and the operational consequences.

Reply-chain handling

No dedup — every outbound gets a fresh signature, including replies inside a long thread. Same industry-norm pattern Disclaimers uses; same rationale (compliance, self-contained messages, predictability). See disclaimers.md "Reply-chain handling".

Failure semantics

Same graceful-degradation contract as Disclaimers (`milter_default_action = accept`). If the milter container is down, if the map file is unreadable, if the modifier raises an exception, if substitution blows up — **mail flows unmodified**. Worst case is a missed signature; mail never gets dropped.

See disclaimers.md "Failure semantics".

Disabled rows

`org_signatures.enabled = 0` causes the resolver to skip the row entirely:

- The on-disk `org_<id>/` dir is wiped clean
- No mailboxes resolve to that option
- Mailboxes that previously resolved to that option fall back to the next tier (department → default → none)

Re-enabling rebuilds the on-disk files and re-points the affected mailboxes' map entries on the next regen.

Interaction with `domains.allow_user_signatures`

This per-domain toggle is the single switch that controls whether Personal Signatures can win over Organizational Signatures.

`allow_user_signatures`	Personal Sig present?	Result
`0`	yes	Personal Sig **ignored**; resolver falls to Department / Default Org Sig
`0`	no	Resolver falls to Department / Default Org Sig
`1`	yes	Personal Sig **wins** (top of resolution chain)
`1`	no	Resolver falls to Department / Default Org Sig

Toggle this off when you need to **lock everyone into Organizational Signatures** for branding/compliance — useful when a marketing or legal team wants centrally-controlled output and doesn't want individual users overriding it.

A previously-saved Personal Signature is **not deleted** when the toggle goes off — it just stops being resolved. Toggling back on re-activates it on the next regen.

Pro license behavior

The Org Signatures admin page is gated by `session.edition EQ "Pro"`:

- **Pro license valid:** Full UI access; admins can create / edit / delete / enable / disable Org Signatures and toggle `allow_user_signatures`.
- **Pro license missing or expired:** Sidebar entry shows a `PRO` badge; the list and edit pages reject the load with an upsell flash. The save action handler ALSO rejects on the server side (defense in depth — UI-level gating alone isn't a security control).
- **Existing Org Signatures on a downgrade:** Rows persist in the database. The resolver still runs and the milter continues applying them at send time. Personal Signatures continue working as well. The downgrade is a **UI restriction**, not a runtime feature kill.

This is the same "feature stays on, admin UI locks" pattern as Disclaimers and other Pro features. If a customer wants the Pro feature actually disabled at runtime on downgrade, the path is to delete the rows or set them all to `enabled = 0`.

Why a separate milter and not an amavis hook

Same reasoning as Disclaimers (#214 Phase 3). amavisd-new 2.13's `before_send` hook silently desynchronizes amavis's internal MIME state during in-place body modification, which can drop mail. The body milter approach moves body modification out of amavis entirely; amavis is fully decoupled.

See disclaimers.md "Why a separate milter and not an amavis hook".