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.

Revision #17
Created 2026-06-17 18:36:34 UTC by Dino Edwards
Updated 2026-06-20 13:32:41 UTC by Dino Edwards