Package: llmshieldr (via r-universe)

May 29, 2026

Title Safety Guardrails for Large Language Model Workflows

Version 0.1.0

Description A model-agnostic safety layer for developers building with large language model (LLM) applications. Maps starter controls to the Open Worldwide Application Security Project Top 10 for Large Language Model Applications 2025 risk categories <https://genai.owasp.org/llm-top-10/> via a modular rule engine. Supports regular-expression rules, lightweight natural language processing (NLP) intent checks, optional scanners, and semantic large language model reviewer checks on prompts, conversations, retrieved context, tool inputs and outputs, streaming chunks, and model outputs. Supports workflows with the 'Ollama' local web service <https://ollama.com/> via 'ellmer', remote reviewer endpoints, and other chat interfaces callable from 'R'. Intended as an experimental guardrail layer that teams should evaluate against their own workflows before relying on it in production.

License Apache License (>= 2)

Encoding UTF-8

Depends R (>= 4.1.0)

Imports jsonlite, cli, rlang, digest, stringi

Suggests ellmer, httr2, processx, filelock, htmltools, covr, lintr, plumber, shiny, testthat (>= 3.0.0), knitr, rmarkdown, withr, dplyr, tokenizers, SnowballC

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.3

VignetteBuilder knitr

Config/testthat/edition 3

URL https://www.indraneelchakraborty.com/llmshieldr/, https://github.com/ineelhere/llmshieldr, https://genai.owasp.org/llm-top-10/

BugReports https://github.com/ineelhere/llmshieldr/issues
Repository https://ineelhere.r-universe.dev
Date/Publication 2026-05-28 18:16:03 UTC
RemoteUrl https://github.com/ineelhere/llmshieldr
RemoteRef HEAD
RemoteSha c1cf25f2bf9b0433e83e683eb4d7ec17d7d5824d

Contents

`add_rule` *Add a rule to a policy*

Description

`add_rule()` appends one validated rule to an existing policy. The function returns the modified policy invisibly so it can be used in assignment or pipes.

Usage

```
add_rule(
  policy,
  id,
  pattern = NULL,
  fn = NULL,
  owasp = NULL,
  severity = "medium",
  action = "redact",
  description = ""
)
```

Arguments

`policy`	A `shieldr_policy` or built-in policy name such as `"comprehensive"`.
`id`	Rule identifier.
`pattern`	Regular expression pattern, or `NULL`.
`fn`	Predicate function, or `NULL`.
`owasp`	Optional OWASP category.
`severity`	One of `"low"`, `"medium"`, `"high"`, or `"critical"`.
`action`	One of `"allow"`, `"redact"`, or `"block"`.
`description`	Rule description.

Details

A rule can be regex-based or function-based, but not both. Regex rules are best when you need span-level redaction. Function rules are useful when the condition is easier to express in R, such as "this text contains both a student reference and a home-address phrase".

Severity determines the numeric contribution to the report score:

- `low`: `0.1`
- `medium`: `0.3`
- `high`: `0.6`
- `critical`: `1.0`

The rule `action` is the rule author's preferred outcome when the rule fires. The final report action also considers total score and policy thresholds.

Value

The modified `shieldr_policy`, invisibly.

Examples

```
policy <- build_policy()
policy <- add_rule(policy, "demo.secret", pattern = "SECRET", owasp = "llm02")
```

`available_policies` *List available built-in policies*

Description

Returns the built-in policy names and their high-level behavior. Use these names directly in scanners and orchestration helpers, for example `scan_prompt("text", policy = "comprehensive")`.

Usage

```
available_policies(selected = NULL)
```

Arguments

`selected`	Optional policy name or `shieldr_policy` to mark in the returned table.

Value

A data frame with policy names, descriptions, rule counts, thresholds, rate-guard availability, and a `selected` column when requested.

Examples

```
available_policies()
available_policies("comprehensive")
scan_prompt("hello", policy = "enterprise_default")
```

build_policy *Build a guardrail policy*

Description

build_policy() combines validated shieldr_rule objects with threshold settings for the scanner layer. OWASP LLM Top 10 references are preserved on each rule; see https://genai.owasp.org/llm-top-10/.

Usage

```
build_policy(
  name = "custom",
  rules = list(),
  thresholds = list(),
  rate_guard = NULL,
  controls = NULL
)
```

Arguments

name	Policy name.
rules	A list of shieldr_rule objects.
thresholds	Threshold overrides. Missing values are filled from redact_at = 0.4 and block_at = 0.75.
rate_guard	Optional shieldr_rate_guard. When present, secure_chat() checks the guard before chat calls and updates it after successful calls.
controls	Optional controls from policy_controls().

Details

A policy is intentionally small and inspectable. It contains a policy name, a list of deterministic rules, threshold values, and an optional rate guard. The scanners do not mutate a policy; they read the rule list, create findings, calculate a risk score from finding severities, and then compare that score with the policy thresholds.

controls configures secure_chat() orchestration behavior after a report has already resolved to block. For example, a policy can refuse blocked prompts with a user-facing message, drop blocked RAG rows, or mark blocked output for human review.

Thresholds are merged over the package defaults:

- redact_at = 0.4
- block_at = 0.75

Lower thresholds make a policy stricter. Higher thresholds make accumulated findings less likely to escalate. Critical findings and explicit block rules still block regardless of threshold.

Value

A `shieldr_policy`.

Examples

```
policy <- build_policy(rules = list(rule_pii_email()))
policy
```

`builtin_rules` *Built-in rule helpers*

Description

Helpers create common OWASP LLM Top 10 guardrail rules for prompts, retrieved context, and model outputs.

Usage

```
rule_injection_basic()

rule_injection_indirect()

rule_nlp_intent()

rule_pii_email()

rule_pii_phone()

rule_pii_ssn()

rule_secrets_api_key()

rule_secrets_bearer()

rule_secrets_aws()

rule_secrets_password()

rule_phi_condition()

rule_agency_language()

rule_system_prompt_leak()

rule_diagnosis_claim()

rule_financial_advice()
```

Details

The helpers are intentionally small wrappers around `shieldr_rule()`. They form the source rule bank used by `policy()`. Each helper encodes one common class of risk, such as prompt injection, NLP intent, PII, secrets, excessive agency, system-prompt extraction, diagnosis claims, or financial advice.

The rules are conservative defaults, not exhaustive detectors. They are designed to be readable, testable, and easy to replace with organization- specific rules when needed.

Value

A `shieldr_rule`.

Examples

```
rule_injection_basic()
rule_pii_email()
```

`evaluate_security_cases` *Evaluate scanner behavior on a labeled corpus*

Description

`evaluate_security_cases()` runs llmshieldr scanners over a small labeled corpus and returns action-level metrics. It is designed for repeatable local evaluation, release notes, and adoption reviews; it is not a substitute for a full red-team benchmark.

Usage

```
evaluate_security_cases(
  cases = NULL,
  policy = "comprehensive",
  reviewer = NULL,
  checks = "rules",
  redaction = NULL,
  scanners = scanner_options()
)
```

Arguments

`cases`	Optional data frame. If `NULL`, the packaged `inst/extdata/security_eval_cases.csv` corpus is loaded.
`policy`	A `shieldr_policy` or built-in policy name.
`reviewer`	Optional reviewer function or object with `$chat()`.
`checks`	One of `"rules"`, `"nlp"`, `"llm"`, or `"both"`.
`redaction`	Optional redaction strategy from `redaction_strategy()`.
`scanners`	Optional scanner configuration from `scanner_options()`.

Details

The input corpus should contain at least `stage`, `text`, and `expected_action` columns. If `stage` is `"output"`, rows are scanned with `scan_output()`. If `stage` is `"context"`, each row is scanned as a one-row context data frame with `scan_context()`. All other stages are scanned with `scan_prompt()`.

The returned data frame includes per-case latency in milliseconds and a Boolean `matched` column. Use the summary columns to calculate detection rate, false-positive rate, action accuracy, and latency percentiles in vignettes or release notes.

Value

A data frame with case metadata, expected and actual actions, `matched`, `latency_ms`, and `n_findings`.

Examples

```
## Not run:
results <- evaluate_security_cases(policy = "comprehensive")
mean(results$matched)

## End(Not run)
```

---

example_prompts *Example prompts*

Description

Returns example prompts spanning clean, injection, PII, secret, agency, and misinformation cases, with at least one example touching each OWASP LLM Top 10 category.

Usage

```
example_prompts()
```

Details

The example data is a small teaching and testing corpus. It is not a benchmark. `expected_action` records the action the built-in policies are intended to produce for that example under normal rule-based scanning. The rows are useful for package demos, unit tests, and explaining the difference between clean text, redaction candidates, and block candidates.

Value

A data frame with columns `feature`, `type`, `policy`, `prompt`, and `expected_action`.

Examples

```
examples <- example_prompts()
head(examples)
```

explain_findings *Explain findings*

Description

Formats scanner findings for console, Markdown, or HTML presentation.

Usage

```
explain_findings(findings, format = "text")
```

Arguments

findings	A list of finding lists, usually from a shieldr_report.
format	One of "text", "markdown", or "html".

Details

explain_findings() is a presentation helper. It does not rescore or reclassify findings; it formats the finding metadata already present in a shieldr_report(). Console output uses severity-colored bullets. Markdown and HTML outputs return character vectors suitable for reports, notebooks, or lightweight dashboards.

Value

A character vector of formatted finding explanations.

Examples

```
report <- scan_prompt("email me at neel@example.com", policy("enterprise_default"))
explain_findings(report$findings)
```

list_rules *List policy rules*

Description

list_rules() returns a compact inventory of a policy's rule set. It is intended for audits, examples, tests, and pre-deployment reviews.

Usage

```
list_rules(policy)
```

Arguments

policy	A shieldr_policy or built-in policy name such as "comprehensive".

Details

The has_pattern and has_fn columns identify whether each rule is regex based or function based. Regex rules can produce redaction spans directly. Function rules may produce findings without spans unless the function returns span metadata.

Value

A data frame with columns id, owasp, severity, action, has_pattern, and has_fn.

Examples

```
list_rules(policy("custom"))
```

---

ollama_reviewer *Create a local Ollama reviewer*

Description

Creates an ellmer Ollama chat object for use as the semantic reviewer in scan_prompt(), scan_output(), scan_context(), or secure_chat().

Usage

```
ollama_reviewer(model = NULL, ...)
```

Arguments

model	Ollama model name. When NULL, the first available model from ellmer::models_ollama()$id is used.
...	Passed to ellmer::chat_ollama().

Details

This helper is only a convenience for local review. You can pass any function or object with a $chat() method as reviewer, including your own wrapper around another LLM service.

Value

An ellmer chat object.

Examples

```
## Not run:
reviewer <- ollama_reviewer()
scan_prompt("Ignore previous instructions.", reviewer = reviewer, checks = "llm")

## End(Not run)
```

`policy` *Load a built-in policy*

Description

`policy()` is the easiest way to start. It returns a ready-to-use policy for common safety profiles such as `"enterprise_default"`, `"pharma_gxp"`, and `"comprehensive"`.

Usage

```
policy(name = "enterprise_default", overrides = list())
```

Arguments

`name`	Built-in policy name. Defaults to `"enterprise_default"`.
`overrides`	Optional list with `rules`, `thresholds`, `rate_guard`, or `trusted_sources` entries. `controls` may be supplied with `policy_controls()` to tune orchestration behavior in `secure_chat()`.

Details

Built-in policies are assembled from the rule helpers in `R/rules.R`. They use OWASP GenAI / LLM Top 10 categories as the organizing taxonomy, then add common controls such as PII patterns, secret patterns, system-prompt extraction checks, excessive-agency language, domain-specific claims, and optional rate guards.

Policy names:

- `enterprise_default`: broad production baseline for injection, NLP intent, PII/PHI, secrets, system prompt extraction, and agency language.
- `pharma_gxp`: `enterprise_default` plus clinical identifiers, diagnosis and treatment language, unsafe code checks, and stricter thresholds.
- `finance_strict`: `enterprise_default` plus account numbers, investment advice language, autonomous trading language, and a token-rate guard.
- `education_safe`: `enterprise_default` plus minor-related PII and academic-integrity bypass language.
- `open_research`: a smaller open-workflow profile focused on injection and secrets, with higher thresholds.
- `comprehensive`: a maximum-coverage profile combining the enterprise, pharma, finance, education, code-safety, and rate-guard controls. Uses moderate thresholds (`redact_at = 0.4`, `block_at = 0.7`). For pharma-tier strictness, supply `overrides = list(thresholds = list(redact_at = 0.3, block_at = 0.6))` explicitly.
- `custom`: no rules, default thresholds.
- `baseline`: backward-compatible alias for `enterprise_default`.

These policies are starting points. They are transparent, testable, and can be extended with `add_rule()` for application-specific requirements.

Value

A shieldr_policy.

Examples

```
policy()
policy("open_research", overrides = list(thresholds = list(redact_at = 0.7)))
```

---

policy_controls *Configure policy-level control actions*

Description

policy_controls() defines how secure_chat() should respond after a scanner has already resolved a prompt, context row, or output as blocked. Scanner reports still use the core actions allow, redact, and block; controls decide whether the orchestration layer should drop context, return a refusal message, or mark a run for human review.

Usage

```
policy_controls(
  on_prompt_block = "block",
  on_context_block = "drop",
  on_output_block = "block",
  refusal_message = "I can't safely complete that request.",
  escalation_message = "Human review requested by llmshieldr policy."
)
```

Arguments

on_prompt_block
: One of "block", "refuse", or "escalate".

on_context_block
: One of "drop", "keep_redacted", "block", "refuse", or "escalate".

on_output_block
: One of "block", "refuse", or "escalate".

refusal_message
: Message returned as result$output when a control maps a block to refuse.

escalation_message
: Optional human-readable reason stored in policy metadata when a control maps a block to escalate.

Details

Control fields:

- `on_prompt_block`: applied when the user prompt is blocked before the chat call.
- `on_context_block`: applied when one or more retrieved context rows are blocked. `"drop"` excludes blocked rows and continues. `"keep_redacted"` includes their redacted text. `"block"`, `"refuse"`, and `"escalate"` stop before the chat call.
- `on_output_block`: applied when model output is blocked after the chat call.

`refuse` returns `refusal_message` as the result output. `escalate` returns no output and records the final action as `"escalate"` for downstream routing.

Value

A list of policy controls.

Examples

```
guardrails <- policy(
  "enterprise_default",
  overrides = list(
    controls = policy_controls(
      on_prompt_block = "refuse",
      on_context_block = "drop"
    )
  )
)
```

---

preflight_check *Preflight-check a prompt*

Description

Backward-compatible alias for `scan_prompt()`.

Usage

```
preflight_check(
  text,
  policy = "enterprise_default",
  reviewer = NULL,
  checks = "rules",
  redact = TRUE,
  redaction = NULL,
  scanners = scanner_options(),
  show_tokens = FALSE
)
```

Arguments

text	Prompt text.
policy	A shieldr_policy or built-in policy name such as "comprehensive".
reviewer	Optional reviewer function or object with $chat().
checks	One of "rules", "nlp", "llm", or "both".
redact	Whether to redact matched spans in text_clean.
redaction	Optional redaction strategy from redaction_strategy(). Ignored when redact = FALSE.
scanners	Optional scanner configuration from scanner_options().
show_tokens	Whether to attach token counts when ellmer is available.

Value

A shieldr_report.

Examples

```
preflight_check("hello")
preflight_check("hello", show_tokens = TRUE)
```

---

print.shieldr_policy *Print a* shieldr_policy

Description

Print a shieldr_policy

Usage

```
## S3 method for class 'shieldr_policy'
print(x, ...)
```

Arguments

x	A shieldr_policy.
...	Unused.

Value

The policy, invisibly.

print.shieldr_report *Print a* shieldr_report

Description

Print a shieldr_report

Usage

```
## S3 method for class 'shieldr_report'
print(x, ...)
```

Arguments

x	A shieldr_report.
...	Unused.

Value

The report, invisibly.

rate_guard *Create or check a rate guard*

Description

Rate guards are explicit stateful environments used to cap token and request budgets for LLM workflows. Resource exhaustion is covered by OWASP LLM10; see https://genai.owasp.org/llm-top-10/.

Usage

```
rate_guard(
  max_tokens = NULL,
  max_requests = NULL,
  window_seconds = 3600L,
  strict = FALSE,
  concurrent = FALSE
)
```

Arguments

max_tokens	Maximum tokens per window, NULL, or an existing shieldr_rate_guard when checking a guard with rate_guard(guard).
max_requests	Maximum requests per window, or NULL.
window_seconds	Window length in seconds.
strict	Whether secure_chat() should reserve estimated prompt tokens before calling the model.
concurrent	Whether to protect $usage() and $update() with a file-based lock from the suggested filelock package.

Details

Calling rate_guard() with limits creates a new shieldr_rate_guard environment. The environment stores counters for the current window and exposes two methods:

- $usage(): returns current counters and configured limits.
- $reserve(tokens, requests): atomically checks projected usage and then increments counters when the reservation stays within limits.
- $update(tokens, requests): backward-compatible alias for $reserve().
- $rollback(tokens, requests): subtracts a previous reservation after a guarded operation fails before completion.

Calling rate_guard(guard) checks an existing environment and returns TRUE if all counters are within limits. Reservation methods fail before projected usage exceeds the configured token or request limit. Limits set to NULL are disabled for that dimension.

Windows reset automatically when window_seconds has elapsed. This object is intentionally stateful; it is the one place where llmshieldr expects mutable state, because rate limiting is inherently session-based.

Value

When creating a guard, a shieldr_rate_guard environment. When checking a guard, TRUE if usage is within limits.

Concurrency

The rate guard is not safe for concurrent use by default. Parallel or async R code (future, parallel, callr) that shares a single guard environment will produce inaccurate counts. Use concurrent = TRUE and install the filelock package to make each $usage(), $reserve(), $update(), and $rollback() call acquire a file-based lock within a single machine. Cross-machine coordination is not supported.

Pre-call Reservation

With strict = TRUE, secure_chat() reserves an estimated prompt token cost and one request before the model call, then records only the positive difference between the actual token estimate and the reserved amount after the call. If the chat call or output scan fails, the pre-call reservation is rolled back. This makes shared guards more useful under bursty load, but estimated tokens may differ from actual usage. Strict mode is recommended when multiple callers share one guard.

Examples

```
guard <- rate_guard(max_tokens = 100)
guard$reserve(tokens = 10)
rate_guard(guard)
```

`redaction_strategy` *Configure redaction behavior*

Description

`redaction_strategy()` controls how scanner spans are rewritten in `text_clean`. The default strategy preserves the original package behavior: every matched span is replaced by `[REDACTED]`.

Usage

```
redaction_strategy(
  operator = c("replace", "mask", "hash", "drop", "keep"),
  replacement = "[REDACTED]",
  mask = "*",
  hash_algo = "sha256",
  hash_prefix = 12L
)
```

Arguments

`operator`	One of `"replace"`, `"mask"`, `"hash"`, `"drop"`, or `"keep"`.
`replacement`	Replacement text used by `operator = "replace"`.
`mask`	Single-character mask used by `operator = "mask"`.
`hash_algo`	Digest algorithm passed to `digest::digest()` for `operator = "hash"`.
`hash_prefix`	Number of digest characters to keep in hash labels.

Details

Redaction only applies to findings that include `start` and `end` character offsets. Regex rules and some semantic reviewer schemas can provide spans. Function rules and synthetic findings can still change score and action, but they do not rewrite text unless they include span metadata.

Supported operators:

- `"replace"`: replace the span with `replacement`.
- `"mask"`: replace each character in the span with `mask`.
- `"hash"`: replace the span with a short deterministic digest label.
- `"drop"`: remove the span.
- `"keep"`: leave the span unchanged while still returning findings.

Hash redaction is deterministic for the same matched string and algorithm, which can help link repeated values without storing the original secret. It is not anonymization and should still be treated as sensitive metadata.

Value

A shieldr_redaction_strategy object.

Examples

```
scan_prompt(
  "Email neel@example.com",
  redaction = redaction_strategy("mask", mask = "*")
)

scan_prompt(
  "Email neel@example.com",
  redaction = redaction_strategy("hash")
)
```

---

remote_reviewer *Create a remote semantic reviewer*

Description

remote_reviewer() creates a reviewer function backed by an HTTP endpoint. It is intended for teams that already have a policy service, review gateway, or hosted model wrapper that returns llmshieldr-compatible JSON findings.

Usage

```
remote_reviewer(
  url,
  headers = list(),
  body_field = "prompt",
  response_path = NULL,
  timeout = 30
)
```

Arguments

url	Endpoint URL.
headers	Named character vector or list of HTTP headers.
body_field	JSON field name used for the reviewer prompt.
response_path	Optional character vector path to extract from a JSON response, such as c("data", "findings").
timeout	Request timeout in seconds.

Details

The returned function accepts the reviewer prompt generated by llmshieldr and sends a JSON body to `url`. By default, the body is:

```
{"prompt": "..."}
```

The endpoint may return either a JSON response body or plain text containing JSON. If the response body is JSON and `response_path` is supplied, the value at that path is extracted before returning to the scanner. Otherwise the response body is returned as text and parsed by llmshieldr's semantic-review JSON extraction logic.

This helper requires the optional `httr2` package and performs no network calls until the reviewer function is invoked.

Value

A reviewer function suitable for `reviewer =` in `scan_prompt()`, `scan_output()`, `scan_context()`, or `secure_chat()`.

Examples

```
## Not run:
reviewer <- remote_reviewer(
  "https://policy.example.com/review",
  headers = c(Authorization = "Bearer <token>")
)

scan_prompt("Review me", reviewer = reviewer, checks = "llm")

## End(Not run)
```

`remove_rule` *Remove a rule from a policy*

Description

Remove a rule from a policy

Usage

```
remove_rule(policy, id)
```

Arguments

`policy`	A `shieldr_policy` or built-in policy name such as `"comprehensive"`.
`id`	Rule identifier to remove.

Value

The modified `shieldr_policy`, invisibly.

Examples

```
policy <- build_policy(rules = list(rule_pii_email()))
policy <- remove_rule(policy, "llm02.pii.email")
```

`reviewer_prompt` *Return the default semantic reviewer prompt*

Description

Returns the internal prompt used by the semantic reviewer path in `scan_prompt()`, `scan_context()`, `scan_output()`, and `secure_chat()`. This helper is for inspection and auditability; the returned value is not a package option. Users who need custom reviewer instructions should wrap their reviewer function or chat object and prepend additive organization-specific context before delegating to the model, while preserving llmshieldr's JSON finding schema.

Usage

```
reviewer_prompt()
```

Value

A single prompt string.

Examples

```
reviewer_prompt()

base_reviewer <- function(prompt) "[]"
reviewer <- function(prompt) {
  custom_context <- paste(
    "Additional reviewer policy:",
    "- Treat PHI leakage as high severity.",
    "- Return [] when there are no findings.",
    sep = "\n"
  )
  base_reviewer(paste(custom_context, prompt, sep = "\n\n"))
}
```

scan_context *Scan retrieved context rows*

Description

Scans data-frame context chunks and adds OWASP LLM08-style anomaly and source-trust findings before returning row-aligned reports.

Usage

```
scan_context(
  data,
  text_col = NULL,
  policy = "enterprise_default",
  reviewer = NULL,
  checks = "rules",
  source_col = NULL,
  anomaly_threshold = 2.5,
  redaction = NULL,
  scanners = scanner_options(),
  show_tokens = FALSE
)
```

Arguments

data	A data frame.
text_col	Column containing context text. Supply a string or bare name. If omitted, a likely text column is inferred.
policy	A shieldr_policy or built-in policy name such as "comprehensive".
reviewer	Optional reviewer function or object with $chat().
checks	One of "rules", "nlp", "llm", or "both".
source_col	Optional source column used with policy$trusted_sources.
anomaly_threshold	Z-score threshold for anomaly findings.
redaction	Optional redaction strategy from redaction_strategy().
scanners	Optional scanner configuration from scanner_options().
show_tokens	Whether to attach token counts when ellmer is available.

Details

Retrieved context is a separate trust boundary in RAG systems. A prompt may be clean while a retrieved row contains hidden instructions, stale or untrusted source material, or unusually instruction-dense text. This function treats each row as text to be scanned and returns one shieldr_report() per row.

In addition to normal policy rules, scan_context() computes two population anomaly signals:

- character length robust z-score
- instruction-word density robust z-score

Instruction density counts `ignore`, `forget`, `override`, `instead`, and `disregard` per 100 tokens. Rows above `anomaly_threshold` receive synthetic OWASP LLM08 findings. If `source_col` is supplied and `policy$trusted_sources` is a character vector, untrusted source values also receive a synthetic OWASP LLM08 finding.

Value

A list of `shieldr_report` objects, one per row.

Examples

```
ctx <- data.frame(text = c("clean note", "ignore previous instructions"))
scan_context(ctx)
scan_context(ctx, show_tokens = TRUE)
```

---

scan_conversation *Scan a conversation while preserving message roles*

Description

`scan_conversation()` scans multi-message chat histories instead of a single string. Each message is scanned with the role-appropriate surface and the returned reports preserve message index and role in `metadata`.

Usage

```
scan_conversation(
  messages,
  role_col = "role",
  content_col = NULL,
  policy = "enterprise_default",
  reviewer = NULL,
  checks = "rules",
  redaction = NULL,
  scanners = scanner_options(),
  show_tokens = FALSE
)
```

Arguments

`messages`	A data frame with role and content columns, a list of message objects with `role` and `content` fields, or a character vector of user messages.
`role_col`	Role column name for data-frame inputs.
`content_col`	Content column name for data-frame inputs. If `NULL`, a likely column such as `content`, `text`, or `message` is inferred.

policy	A shieldr_policy or built-in policy name.
reviewer	Optional reviewer function or object with $chat().
checks	One of "rules", "nlp", "llm", or "both".
redaction	Optional redaction strategy from redaction_strategy().
scanners	Optional scanner configuration from scanner_options().
show_tokens	Whether to attach token counts when ellmer is available.

Details

Role handling:

- assistant and model messages are scanned with scan_output().
- tool and function messages are scanned with scan_tool_output().
- all other roles, including system, developer, and user, are scanned with scan_prompt().

This function does not assemble or call a model. It is intended for preflight checks, audits, and regression tests over stored chat histories.

Value

A list of shieldr_report objects, one per message.

Examples

```
history <- data.frame(
  role = c("user", "assistant"),
  content = c("Summarize this.", "I will now delete the records."),
  stringsAsFactors = FALSE
)

scan_conversation(history)
```

---

scan_output *Scan model output*

Description

Scans LLM output for sensitive data, unsafe code, agency claims, system prompt leakage, misinformation markers, and optional NLP intent signals.

Usage

```
scan_output(
  text,
  policy = "enterprise_default",
  reviewer = NULL,
  checks = "rules",
  redaction = NULL,
  scanners = scanner_options(),
  show_tokens = FALSE
)
```

Arguments

text	Model output text.
policy	A shieldr_policy or built-in policy name such as "comprehensive".
reviewer	Optional reviewer function or object with $chat().
checks	One of "rules", "nlp", "llm", or "both".
redaction	Optional redaction strategy from redaction_strategy().
scanners	Optional scanner configuration from scanner_options().
show_tokens	Whether to attach token counts when ellmer is available.

Details

Output scanning is the last guardrail before model text is displayed, stored, or passed to another tool. It runs the policy rule set over the full output and adds output-specific checks for common failure modes:

- fenced code blocks are scanned for unsafe code and command patterns
- excessive-agency language such as "I will now" or "I have deleted"
- system-prompt structural markers such as "# System" or role declarations
- high-confidence medical or financial claim markers

Use checks = "nlp" when you want a lightweight local NLP-only pass over model output. The return value is a shieldr_report() with the same scoring and action semantics as scan_prompt().

Value

A shieldr_report.

Examples

```
scan_output("A concise answer.")
scan_output("A concise answer.", show_tokens = TRUE)
```

scan_prompt *Scan a prompt*

Description

Scans user prompt text with rule-based, NLP, and optional semantic reviewer checks. Findings retain OWASP LLM Top 10 categories when known; see https://genai.owasp.org/llm-top-10/.

Usage

```
scan_prompt(
  text,
  policy = "enterprise_default",
  reviewer = NULL,
  checks = "rules",
  redact = TRUE,
  redaction = NULL,
  scanners = scanner_options(),
  show_tokens = FALSE
)
```

Arguments

text	Prompt text.
policy	A shieldr_policy or built-in policy name such as "comprehensive".
reviewer	Optional reviewer function or object with $chat().
checks	One of "rules", "nlp", "llm", or "both".
redact	Whether to redact matched spans in text_clean.
redaction	Optional redaction strategy from redaction_strategy(). Ignored when redact = FALSE.
scanners	Optional scanner configuration from scanner_options().
show_tokens	Whether to attach token counts when ellmer is available.

Details

scan_prompt() is usually the first guardrail in a workflow. It normalizes text with Unicode NFKC normalization, collapses whitespace, applies policy rules, optionally applies the NLP intent rule, optionally asks a semantic reviewer for JSON findings, calculates a risk_score, resolves an action, and returns a shieldr_report().

checks = "rules" uses deterministic policy rules. Built-in policies include regular expressions and an NLP intent rule. checks = "nlp" runs only NLP intent checks, using tokenizers for word tokenization and SnowballC for stemming when those optional packages are installed. checks = "llm" uses only the semantic reviewer when one is supplied. checks = "both" combines policy rules with semantic review. If LLM review returns malformed JSON, the function warns and continues with the findings it already has.

Redaction replaces matched spans with [REDACTED]. Function-based findings can influence score and action even when they do not provide exact spans.

Value

A shieldr_report.

Examples

```
scan_prompt("hello")
scan_prompt("patient has cancer password ak$1234567890", policy = "comprehensive")
scan_prompt("email neel@example.com", redaction = redaction_strategy("hash"))
scan_prompt("hello", show_tokens = TRUE)
```

---

scan_stream *Scan streamed output chunks with rolling context*

Description

scan_stream() scans character chunks as they arrive from a streaming model API. Each scan uses the current chunk plus a configurable overlap from the previous text so rules can catch phrases split across chunk boundaries.

Usage

```
scan_stream(
  chunks,
  policy = "enterprise_default",
  reviewer = NULL,
  checks = "rules",
  chunk_size = 1000L,
  overlap = 200L,
  on_block = c("stop", "return"),
  redaction = NULL,
  scanners = scanner_options(),
  show_tokens = FALSE
)
```

Arguments

chunks	Character vector of streamed text chunks, or one long string.
policy	A shieldr_policy or built-in policy name.
reviewer	Optional reviewer function or object with $chat().
checks	One of "rules", "nlp", "llm", or "both".
chunk_size	Maximum size used to split a single long string.
overlap	Number of trailing characters from prior output to include when scanning the next chunk.

on_block	One of "stop" or "return".
redaction	Optional redaction strategy from redaction_strategy().
scanners	Optional scanner configuration from scanner_options().
show_tokens	Whether to attach token counts when ellmer is available.

Details

This helper is intentionally transport-agnostic: pass the text chunks you receive from an SDK, callback, or websocket handler. It returns per-window reports and a combined action. If on_block = "stop", the function aborts as soon as a window resolves to block; use on_block = "return" when you want a full report object instead.

chunk_size is used only when chunks is a single long string. Character vectors with more than one element are treated as already chunked.

Value

A shieldr_stream_result list with action, text, and reports.

Examples

```
scan_stream(
  c("I will now ", "delete the records."),
  on_block = "return"
)
```

---

scan_tool_call *Scan a tool call before execution*

Description

scan_tool_call() validates tool-call intent and arguments before an application executes the tool. It serializes the tool name and arguments, scans that text with scan_prompt(), and adds an explicit finding when the tool is outside an allowlist.

Usage

```
scan_tool_call(
  tool_name,
  arguments = list(),
  allowed_tools = NULL,
  policy = "enterprise_default",
  reviewer = NULL,
  checks = "rules",
  redaction = NULL,
  scanners = scanner_options(),
  show_tokens = FALSE
)
```

Arguments

tool_name	Tool name requested by a model or orchestrator.
arguments	Tool arguments as a list, data frame, character string, or other JSON-serializable value.
allowed_tools	Optional character vector of approved tool names.
policy	A shieldr_policy or built-in policy name.
reviewer	Optional reviewer function or object with $chat().
checks	One of "rules", "nlp", "llm", or "both".
redaction	Optional redaction strategy from redaction_strategy().
scanners	Optional scanner configuration from scanner_options().
show_tokens	Whether to attach token counts when ellmer is available.

Details

This helper does not execute tools. It is designed to sit immediately before an application-level dispatcher. Use allowed_tools for a simple allowlist, and use normal policy rules or custom rules to validate argument content.

The returned shieldr_report() stores stage = "tool_call" and tool_name in metadata, so audit logs can distinguish tool input checks from prompt, context, and output checks.

Value

A shieldr_report.

Examples

```
report <- scan_tool_call(
  "send_email",
  list(to = "neel@example.com", body = "hello"),
  allowed_tools = c("search_docs", "send_email")
)

report$action
```

---

scan_tool_output *Scan tool output before it re-enters model context*

Description

scan_tool_output() checks text returned by tools before that text is shown to a user, stored, or appended back into model context.

Usage

```
scan_tool_output(
  tool_name,
  output,
  policy = "enterprise_default",
  reviewer = NULL,
  checks = "rules",
  redaction = NULL,
  scanners = scanner_options(),
  show_tokens = FALSE
)
```

Arguments

tool_name	Tool name requested by a model or orchestrator.
output	Tool output text or object coercible to text.
policy	A shieldr_policy or built-in policy name.
reviewer	Optional reviewer function or object with $chat().
checks	One of "rules", "nlp", "llm", or "both".
redaction	Optional redaction strategy from redaction_strategy().
scanners	Optional scanner configuration from scanner_options().
show_tokens	Whether to attach token counts when ellmer is available.

Details

Tool outputs are scanned with scan_output() because they are untrusted downstream content. The returned report stores stage = "tool_output" and tool_name in metadata.

Value

A shieldr_report.

Examples

```
scan_tool_output("search_docs", "Result includes neel@example.com")
```

---

scanner_options *Configure optional text scanners*

Description

scanner_options() enables optional checks that sit beside deterministic policy rules. These scanners are intentionally lightweight and local. They are useful for catching common wrappers around risky text, such as invisible Unicode format characters, encoded payloads, disallowed URLs, simple token budget violations, language allowlists, and topic bans.

Usage

```
scanner_options(
  invisible_text = TRUE,
  encoded_payloads = TRUE,
  urls = FALSE,
  malicious_urls = TRUE,
  max_tokens = NULL,
  allowed_languages = NULL,
  language_fn = NULL,
  blocked_topics = NULL,
  blocked_url_hosts = NULL,
  allowed_url_hosts = NULL
)
```

Arguments

`invisible_text`	Whether to flag Unicode format characters such as zero-width spaces. Normalization removes these characters before rule matching, but a finding records that evasive formatting was present.
`encoded_payloads`	Whether to inspect URL-encoded and base64-like payloads by decoding candidates and scanning the decoded text.
`urls`	Whether to create low-severity inventory findings for URLs.
`malicious_urls`	Whether to flag URLs whose hosts are explicitly blocked or fall outside `allowed_url_hosts`.
`max_tokens`	Optional maximum estimated tokens for a single scanned text. Exceeding the limit creates an OWASP LLM10 block finding.
`allowed_languages`	Optional language allowlist. Uses `language_fn` when supplied, otherwise a minimal ASCII/non-Latin heuristic.
`language_fn`	Optional function that receives text and returns a single language label.
`blocked_topics`	Optional character vector of regular expressions, or a named character vector. Matches create topic-ban findings.
`blocked_url_hosts`	Optional character vector of blocked URL hosts.
`allowed_url_hosts`	Optional character vector of allowed URL hosts. When supplied, URL hosts outside the allowlist are flagged.

Details

Scanner findings use the same finding schema as rule findings and therefore contribute to `risk_score`, `action`, audit logs, and explanations.

The encoded-payload scanner tries URL decoding and base64 decoding on candidate substrings, then runs the active policy rules over decoded text. It does not execute decoded content. The language scanner is deliberately basic unless `language_fn` is supplied; a custom function should accept a single string and return a language label such as `"en"`, `"es"`, or `"non_latin"`.

Value

A shieldr_scanner_options object.

Examples

```
scanners <- scanner_options(
  max_tokens = 500,
  blocked_topics = c("internal layoffs", "unreleased earnings")
)

scan_prompt("Summarize this public note.", scanners = scanners)
```

---

secure_chat *Run a guarded chat call*

Description

Orchestrates prompt scanning, optional context scanning, chat execution, output scanning, rate guarding, and audit creation.

Usage

```
secure_chat(
  prompt,
  chat = NULL,
  policy = "enterprise_default",
  reviewer = NULL,
  checks = "rules",
  context = NULL,
  redaction = NULL,
  scanners = scanner_options(),
  show_tokens = FALSE,
  ...
)
```

Arguments

prompt	User prompt.
chat	An ellmer chat object, an object with $chat(), or a function.
policy	A shieldr_policy or built-in policy name such as "comprehensive".
reviewer	Optional reviewer function or object with $chat().
checks	One of "rules", "nlp", "llm", or "both".
context	Optional data frame of retrieved context.
redaction	Optional redaction strategy from redaction_strategy().
scanners	Optional scanner configuration from scanner_options().
show_tokens	Whether to attach token counts when ellmer is available.
...	Reserved for backwards-compatible aliases.

Details

`secure_chat()` is the main end-to-end workflow when you already have an `ellmer` chat object or another object with a `$chat()` method. Plain functions are also accepted for small tests. The function executes these steps:

1. Scan the prompt with `scan_prompt()`.
2. If the prompt is blocked, return a `shieldr_result()` without calling the chat.
3. If context is supplied, scan it with `scan_context()` and append only allowed context rows to the cleaned prompt, using row IDs, source labels, and separators.
4. Reserve request and token budget with the policy rate guard, if present.
5. Call the chat object.
6. Scan model output with `scan_output()`.
7. Resolve the final action, update the rate guard, and build an audit.

The returned `risk_summary` aggregates finding severity scores by OWASP category across prompt, context, and output reports. The final action is the most conservative action across input and output: `block` beats `redact`, and `redact` beats `allow`. Policy controls can map blocked prompt or output reports to final actions of `refuse` or `escalate`.

Value

A `shieldr_result`.

Examples

```
## Not run:
model <- ellmer::models_ollama()$id[1]
if (is.na(model)) {
  stop(
    "Check if you have any Ollama models available, ",
    "or enter a specific name as a string for the model argument."
  )
}
chat <- ellmer::chat_ollama(model = model)
secure_chat("hello", chat, show_tokens = TRUE)

## End(Not run)
```

`shield_ollama` *Guard an Ollama chat workflow*

Description

Convenience wrapper that creates separate `ellmer` Ollama chats for the assistant and semantic reviewer, then delegates to `secure_chat()`.

Usage

```
shield_ollama(
  prompt,
  policy = "enterprise_default",
  checks = "both",
  model = NULL,
  context = NULL,
  redaction = NULL,
  scanners = scanner_options(),
  show_tokens = FALSE
)
```

Arguments

prompt	User prompt.
policy	A shieldr_policy or built-in policy name such as "comprehensive".
checks	One of "rules", "nlp", "llm", or "both".
model	Ollama model name. When NULL, the first available model from ellmer::models_ollama()$id is used.
context	Optional data frame of retrieved context.
redaction	Optional redaction strategy from redaction_strategy().
scanners	Optional scanner configuration from scanner_options().
show_tokens	Whether to attach token counts when ellmer is available.

Details

This is an optional local-model path. It requires the suggested ellmer package and a running Ollama installation. Two chats are created: one for the assistant response and one for reviewer checks. Keeping them separate avoids mixing safety-review instructions into the assistant's conversation state.

For an existing chat object, use secure_chat() directly.

Value

A shieldr_result.

Examples

```
## Not run:
shield_ollama("Summarise this safely.")

## End(Not run)
```

shieldr_audit *Construct a* shieldr_audit

Description

Audits collect the scanner reports and operational metadata from a guarded run.

Usage

```
shieldr_audit(
  input_report = NULL,
  output_report = NULL,
  context_reports = NULL,
  prompt_clean,
  output_raw = NULL,
  elapsed_ms,
  token_estimate,
  action
)
```

Arguments

input_report	A shieldr_report, or NULL.
output_report	A shieldr_report, or NULL.
context_reports	A list of shieldr_report objects, or NULL.
prompt_clean	Cleaned prompt.
output_raw	Raw model output, or NULL.
elapsed_ms	Elapsed time in milliseconds.
token_estimate	Integer token estimate.
action	Final action. secure_chat() may return "refuse" or "escalate" when policy controls map a blocked report to those outcomes.

Details

secure_chat() builds this object automatically. elapsed_ms captures wall-clock elapsed time for the guarded workflow. token_estimate uses an ellmer token counter when one is available and falls back to ceiling(nchar(text) / 4) over prompt and output text. It is intended for guardrails and trend monitoring, not billing reconciliation.

Value

A shieldr_audit S3 object.

Examples

```
shieldr_audit(NULL, NULL, NULL, "hello", NULL, 0, 1L, "allow")
```

shieldr_policy *Construct a* shieldr_policy

Description

Creates a validated policy from a list of shieldr_rule objects.

Usage

```
shieldr_policy(
  name,
  rules,
  thresholds,
  rate_guard = NULL,
  trusted_sources = NULL,
  controls = NULL
)
```

Arguments

name	Policy name.
rules	A list of shieldr_rule objects.
thresholds	A list containing numeric redact_at and block_at values between 0 and 1.
rate_guard	A shieldr_rate_guard environment, or NULL.
trusted_sources	Optional character vector of trusted context sources.
controls	Optional list from policy_controls().

Details

This is the low-level constructor. Most users should start with policy(), which returns a ready-to-use built-in policy. shieldr_policy() is exported so advanced users and tests can construct exact policy objects.

trusted_sources is used by scan_context() only. If it is NULL, all sources are treated as trusted. If it is a character vector and source_col is supplied to scan_context(), rows with source values outside the allowlist receive an OWASP LLM08 finding.

controls is used by secure_chat() after scanner reports have already resolved to allow, redact, or block. Use policy_controls() to decide whether blocked prompts or outputs should return block, refuse, or escalate, and whether blocked context rows should be dropped, kept in redacted form, or stop the chat call.

Value

A shieldr_policy S3 object.

Examples

```
shieldr_policy("empty", list(), list(redact_at = 0.4, block_at = 0.75))
```

`shieldr_report` *Construct a* `shieldr_report`

Description

A `shieldr_report` is the scanner result returned by `scan_prompt()`, `scan_context()`, and `scan_output()`.

Usage

```
shieldr_report(
  action,
  text_clean,
  findings,
  risk_score,
  policy,
  checks,
  timestamp = .now_iso(),
  tokens = NULL,
  metadata = list()
)
```

Arguments

`action`	Resolved action: `"allow"`, `"redact"`, or `"block"`.
`text_clean`	Cleaned or redacted text.
`findings`	A list of finding lists.
`risk_score`	Numeric risk score between 0 and 1.
`policy`	Policy name.
`checks`	Check mode used.
`timestamp`	ISO8601 timestamp.
`tokens`	Optional token count for the original text.
`metadata`	Optional list of operational metadata.

Details

Reports separate the cleaned text from the findings that explain why the text was allowed, redacted, or blocked. `risk_score` is a deterministic severity index from `0` to `1`; it is not a probability. The `checks` field records whether the report came from deterministic rules, NLP checks, an LLM reviewer, or a combined mode. `metadata` carries optional operational details such as semantic reviewer parse errors, scanner settings, context row indexes, source labels, tool names, or conversation roles.

Value

A `shieldr_report` S3 object.

Examples

```
shieldr_report("allow", "hello", list(), 0, "custom", "rules")
```

`shieldr_result` *Construct a* `shieldr_result`

Description

A `shieldr_result` is the high-level return value from `secure_chat()` and `shield_ollama()`.

Usage

```
shieldr_result(output = NULL, audit, risk_summary, action)
```

Arguments

`output`	Cleaned model output, or `NULL`.
`audit`	A `shieldr_audit` object.
`risk_summary`	Named numeric vector keyed by OWASP category.
`action`	Final action. May be `"allow"`, `"redact"`, `"block"`, `"refuse"`, or `"escalate"`.

Details

`output` is `NULL` when the final action is `block`; otherwise it contains the cleaned model output. `risk_summary` aggregates finding severity scores by OWASP category across prompt, context, and output reports, capping each category at `1.0`. This gives dashboards and audit logs a compact view of which OWASP categories were triggered.

Value

A `shieldr_result` S3 object.

Examples

```
aud <- shieldr_audit(NULL, NULL, NULL, "hello", NULL, 0, 1L, "allow")
shieldr_result(NULL, aud, numeric(), "allow")
```

`shieldr_rule` *Construct a* `shieldr_rule`

Description

Creates a validated rule for the llmshieldr rule engine. Rules map to OWASP LLM Top 10 categories where possible; see https://genai.owasp.org/llm-top-10/.

Usage

```
shieldr_rule(
  id,
  pattern = NULL,
  fn = NULL,
  owasp = NULL,
  severity = "medium",
  action = "redact",
  description = ""
)
```

Arguments

`id`	A unique rule identifier.
`pattern`	A regular expression pattern, or `NULL`.
`fn`	A predicate function, or `NULL`.
`owasp`	Optional OWASP LLM category such as `"llm01"`.
`severity`	One of `"low"`, `"medium"`, `"high"`, or `"critical"`.
`action`	One of `"allow"`, `"redact"`, or `"block"`.
`description`	Human-readable rule description.

Details

A rule is the atomic unit of a policy. Each rule either supplies a regular expression pattern or an R function. Regex rules are applied with `gregexpr(..., perl = TRUE)` and can produce character spans for redaction. Function rules receive the full text and can return `TRUE`, `FALSE`, a finding list, a list of finding lists, or a data frame of findings.

`severity` is converted to a numeric score by the scanner:

- `low`: `0.1`
- `medium`: `0.3`
- `high`: `0.6`
- `critical`: `1.0`

The scanner caps the summed report score at `1.0`. Critical findings and rules with `action = "block"` force the resolved report action to `block`.

Value

A `shieldr_rule` S3 object.

Examples

```
shieldr_rule(
  id = "demo.email",
  pattern = "\\\\b[^@]+@example\\\\.com\\\\b",
  owasp = "llm02",
  description = "Example-domain email address"
)
```

`trust_boundary` *Wrap a chat object in a trust boundary*

Description

Validates chat identity before calls cross into an LLM service. This covers supply-chain and model-integrity concerns related to OWASP LLM03; see https://genai.owasp.org/llm-top-10/.

Usage

```
trust_boundary(
  chat = NULL,
  allowed_models = NULL,
  allowed_hosts = NULL,
  require_hash = NULL,
  ...
)
```

Arguments

`chat`	An `ellmer` chat object, an object with `$chat()`, or a function.
`allowed_models`	Optional character vector of allowed model names.
`allowed_hosts`	Optional character vector of allowed hosts or base URLs.
`require_hash`	Optional expected SHA-256 hash for an Ollama modelfile manifest.
`...`	Reserved for backwards-compatible aliases.

Details

`trust_boundary()` returns a chat wrapper. The wrapper validates the chat on creation and again on each call when `require_hash` is supplied. Plain functions are passed through without model or host checks because a function has no standard model metadata. Chat objects with a `$chat()` method may expose model and host fields through common ellmer-style internals or attributes.

`allowed_models` and `allowed_hosts` are allowlists. If the chat exposes a model or host and it is outside the allowlist, the wrapper raises an OWASP LLM03 error. `require_hash` is intended for local Ollama workflows where the model manifest can be checked with `ollama show --modelfile`.

This function is not a network firewall. It is an application-level assertion that the chat object being called is the chat object you intended to allow.

Value

A callable chat wrapper.

Examples

```
chat <- function(prompt) paste("ok:", prompt)
safe_chat <- trust_boundary(chat)
safe_chat("hello")
```

---

write_audit_log *Write an audit log*

Description

Persists a `shieldr_audit` object as JSON Lines, CSV, or RDS for operational auditability across LLM workflows.

Usage

```
write_audit_log(audit, path, format = "jsonl")
```

Arguments

`audit`	A `shieldr_audit` object.
`path`	Output file path.
`format`	One of `"jsonl"`, `"csv"`, or `"rds"`.

Details

JSON Lines is the preferred append-only format for production logs because each call writes one complete audit object as one line. CSV flattens findings into one row per finding, which is convenient for spreadsheets and simple dashboards but loses some nested structure. The CSV path preserves common metadata such as context row index, context source, tool name, conversation role, and reviewer error counts. RDS preserves the R object exactly and overwrites the target path.

Audit logs may contain sensitive source text, raw model output, or redacted findings depending on the workflow. Treat audit paths as sensitive storage in regulated or internal environments.

Value

The path, invisibly.

Examples

```
audit <- shieldr_audit(NULL, NULL, NULL, "hello", NULL, 0, 1L, "allow")
path <- tempfile(fileext = ".jsonl")
write_audit_log(audit, path)
```

Index