

Package: pkgaudit (via r-universe)

October 17, 2026

Title Static Analysis Security Testing (SAST) for R Packages

Version 0.4.0

Description A static analysis tool that flags security-relevant files and code in R source packages for human review without executing anything it scans. Models package lifecycle execution semantics -- it reports not just what a package does, but when it runs, so code that executes on install or load is distinguishable from code that runs only when called. Reads every surface a package can execute, including 'configure' scripts, help-file macros, and vignettes, and states what it could not read, so a clean result can be checked rather than trusted. Detection rules are data, shipped in a versioned and hash-verified 'SQLite' database.

License Apache License (>= 2)

URL <https://github.com/tylerjssmith/pkgaudit>,
<https://tylerjssmith.github.io/pkgaudit/>

BugReports <https://github.com/tylerjssmith/pkgaudit/issues>

Encoding UTF-8

Language en-US

Roxygen list(markdown = TRUE)

Depends R (>= 4.1)

Imports DBI, digest, RSQLite, stats, tools, utils, xml2, xmlparsedata

Suggests covr, jsonlite, knitr, rmarkdown, testthat (>= 3.0.0), yaml

VignetteBuilder knitr

Config/testthat/edition 3

Config/roxygen2/version 8.1.0

Config/Needs/website pkgdown

Config/Needs/coverage covr

Config/pak/sysreqs libxml2-dev

Repository <https://ropensci-staging.r-universe.dev>

Date/Publication 2026-09-17 14:29:24 UTC
RemoteUrl https://github.com/tylerjssmith/pkgaudit
RemoteRef HEAD
RemoteSha 4279f8f5d42300ba9cc860db490392af2ca64b2f

Contents

analyze_segment	2
audit_package	3
audit_tarball	5
emit_sarif	6
export_unscanned	7
extract_segments	9
format.pkgaudit	10
hash_manifest	11
load_rules	12
new_findings	14
new_segment	15
rules_version	16
summary.pkgaudit	17
validate_tar	19
Index	21

analyze_segment	<i>Find code contexts, patterns and matches in one segment</i>
-----------------	--

Description

Dispatches on the segment’s language, which the extractor set. This is a separate axis from the source’s type: a help file and an R script both yield R segments, and one literate file can yield several languages.

Usage

```
analyze_segment(segment, rules)
```

Arguments

- segment A pkgaudit_segment from new_segment().
- rules Named list of rules as returned by load_rules().

Value

A list of patterns, matches, coverage and errors data frames, each with the columns audit_package() documents, less the phase columns.

Method contract

A method must build its return value with `new_findings()`, which holds every analyzer to the same frame shape. `UseMethod()` ends the generic, so this cannot be enforced after dispatch.

A method must not evaluate the code it is given. It reads untrusted text and reports what it finds; nothing in a scan is ever run.

See Also

`extract_segments()`, the other axis of dispatch, `new_findings()`, and `vignette("internals")`.

Examples

```
# Adding a language outside pkgaudit: a method for segments the extractor
# labeled "python", reporting each line that calls eval().
analyze_segment.python <- function(segment, rules) {
  at <- grep("\\beval\\(", segment$lines)
  if (length(at) == 0L) return(new_findings())
  new_findings(matches = data.frame(
    rule = "py_eval", file_context = segment$file_context,
    line_number = at, column_number = NA_integer_,
    preview = trimws(segment$lines[at]),
    message = "eval() in Python code", attck = NA_character_)
  )
  registerS3method("analyze_segment", "python", analyze_segment.python)
```

 audit_package

Audit an R source package

Description

Finds security-relevant file and code contexts, code patterns, and shell and make matches for review before an R source package is trusted.

Usage

```
audit_package(path = ".", rules = load_rules(), .origin = NULL)
```

Arguments

<code>path</code>	Path to an R source package root directory. Defaults to the current directory.
<code>rules</code>	Named list of rules. Defaults to the rules bundled with the package as returned by <code>load_rules()</code> .
<code>.origin</code>	Internal. Used by <code>audit_tarball()</code> to record tarball provenance: a list of path and sha256, each a length-one character, and <code>is_tarball</code> , a length-one logical that is not NA. A malformed one is refused before the scan rather than silently recorded. NULL for a directory scan, which is hashed with <code>hash_manifest()</code> instead.

Details

Every findings frame carries one logical column per phase – `at_autoconf`, `at_build`, `at_check`, `at_install_src`, `at_install_bin`, `at_load`, `at_attach`, `at_unload` and `at_detach`. Value is `TRUE` when that finding's code runs then. A finding can belong to several phases.

A file in a language pkgaudit cannot read is left unscanned rather than scanned badly, so a file's absence from patterns and matches is not evidence that it is clean; see coverage and errors.

`preview` is a display-only excerpt of the line, whitespace collapsed and a long line windowed on the match, so `column_number` does not index into it.

Value

A `new_pkgaudit()` object: a named list with class `pkgaudit` holding five data frames and a metadata list. Every findings frame carries the nine phase columns described in Details and joins to the others on `file_context`. Paths are relative to the package root.

file_contexts rule, file_context, message.

patterns rule, file_context, line_number, column_number, code_context, guarded, indirect, preview, message, attck. `code_context` is where the code sits within its file: `top_level`, `in_function`, an enclosing lifecycle hook, or the part of a help file it came from.

matches rule, file_context, line_number, column_number, preview, message, attck: regular-expression matches in the shell and Make-like file contexts.

coverage file_context, language, status, reason, first_line, last_line, lines, bytes, rule. One row per file the package carries that is, or could be, code, plus one per span of an unanalyzed language inside a literate file – a python chunk in a vignette is its own row. status is `parsed`, `matched`, `exportable`, `unexamined` or `error`; reason says what stood in the way, and is `NA`, `no_analyzer`, `no_extractor`, `no_rule`, `serialized`, `binary`, `symlink`, `too_large`, `unreadable` or `unparseable`.

errors step, file_context, rule, message. Recoverable failures, collected rather than aborting the scan.

metadata pkg_name, pkg_version, pkg_path, pkg_is_tarball, pkg_sha256, pkgaudit_version, pkgaudit_rules_version, pkgaudit_rules_sha256, scanned. The two rules fields are `NA` for a rules list that did not come from `load_rules()`.

See Also

`vignette("pkgaudit")` for a worked scan, and `vignette("rules")` for the rules, phases and contexts behind a finding.

Examples

```
# untrustedpkg is a small package shipped with pkgaudit to be scanned.
tarball <- system.file(
  "extdata", "untrustedpkg", "untrustedpkg-0.1.0.tar.gz",
  package = "pkgaudit"
)
exdir <- file.path(tempdir(), "untrustedpkg-example")
utils::untar(tarball, exdir = exdir)
```

```

rules <- load_rules()
result <- audit_package(file.path(exdir, "untrustedpkg"), rules = rules)
result$file_contexts
result$patterns
print(result)

```

audit_tarball

Audit an R source package tarball

Description

Finds security-relevant file and code contexts, code patterns, and shell and make matches for review before an R source package tarball is trusted.

Usage

```

audit_tarball(
  path,
  rules = load_rules(),
  temp_dir = tempdir(),
  max_entries = 100000L,
  max_bytes = 2 * 1024^3,
  max_ratio = 256
)

```

Arguments

path	Path to a gzip-compressed or uncompressed R source package tarball (.tar.gz, .tgz, or .tar). A bzip2-, xz-, zstd- or compress-compressed archive is refused by its magic bytes, whatever its filename says.
rules	Named list of rules. Defaults to the rules bundled with the package as returned by <code>load_rules()</code> .
temp_dir	Directory used for extraction. A unique subdirectory is created here and removed after auditing regardless of success or failure. Defaults to <code>base::tempdir()</code> .
max_entries	Maximum number of entries to read before failing closed. Default 100,000.
max_bytes	Maximum uncompressed bytes to read before failing closed. Default 2 GB. Raise for ecosystems with larger artifacts, e.g. Bioconductor annotation and experiment-data packages.
max_ratio	Maximum uncompressed:compressed ratio before failing closed, or Inf to disable. Targets decompression bombs, which are characterised by extreme ratios rather than absolute size. Default 256, well under the ~1032:1 ceiling of a single gzip layer.

Details

Extracts a source package tarball to a temporary directory, applies `audit_package()`, removes the temporary directory, and returns the result.

Before extracting anything, the tarball is validated with `validate_tar()`, which fails closed. After extraction, the extracted directory is re-checked and rejected if it contains any symlink, as defense in depth against a `validate_tar()/untar()` disagreement.

After extraction, the tarball filename must be consistent with the top-level directory it produced (e.g. `foo_0.1.0.tar.gz` must extract to `foo/`); otherwise the tarball is rejected.

The audited DESCRIPTION is then checked against the tarball filename: if the Package name or Version disagrees with the name and version implied by the filename (`foo` and `0.1.0` for `foo_0.1.0.tar.gz`), a `pkgaudit_provenance_mismatch` warning is issued, which a caller can catch by class. The DESCRIPTION values are authoritative and are what the returned object reports.

Value

The same `pkgaudit` object as `audit_package()`: a named list with class `pkgaudit` containing five data frames and a metadata list.

Examples

```
# untrustedpkg is a small package shipped with pkgaudit to be scanned.
tarball <- system.file(
  "extdata", "untrustedpkg", "untrustedpkg-0.1.0.tar.gz",
  package = "pkgaudit"
)

result <- audit_tarball(tarball)
print(result)
```

emit_sarif

Render a scan as SARIF

Description

Renders a `pkgaudit` object as a SARIF 2.1.0 document, the format code scanning tools publish results in, so a scan can be read in an editor or loaded by any SARIF consumer.

Usage

```
emit_sarif(object, pretty = TRUE)
```

Arguments

<code>object</code>	A <code>pkgaudit</code> object.
<code>pretty</code>	Logical; if TRUE (default) indent the JSON for reading.

Details

Every finding in `file_contexts`, `patterns` and `matches` becomes a result, located by the path and, where there is one, the line and column. Rule ids are labeled by the kind of rule – `pattern/curl`, `match/curl`, `file/configure` – because a rule name is unique only within its kind.

A result's message is a one-line title – the rule, the code context it sits in, and the phases it runs in – because a consumer displays it as the heading of a finding. A rule's own text is the same for every finding the rule produces, so it describes the rule rather than the finding and is written as the descriptor's `fullDescription` and `help`.

`level` is note for every result: `pkgaudit` does not rank findings, so nothing is mapped onto SARIF's severity field. When a finding's code executes is carried in the title and in `properties.phases`, and a note is never a claim that a finding is minor.

`partialFingerprints` identifies a finding by its rule, its file, the code context it sits in, and the text of the line – not by line number, which shifts whenever anything above it is edited. Two occurrences a consumer could not otherwise tell apart are numbered, since a fingerprint repeated within a run makes several findings read as one. The coverage frame becomes the `artifacts` array, so a consumer can see which files were never read, and errors become execution notifications on the invocation.

Requires `jsonlite`, a suggested dependency.

Value

A length-one character vector holding the SARIF document. Nothing is written; `writeLines()` it where you want it.

Examples

```
# untrustedpkg is a small package shipped with pkgaudit to be scanned.
tarball <- system.file(
  "extdata", "untrustedpkg", "untrustedpkg_0.1.0.tar.gz",
  package = "pkgaudit"
)
result <- audit_tarball(tarball)

writeLines(emit_sarif(result), file.path(tempdir(), "pkgaudit.sarif"))
```

export_unscanned

Export the code pkgaudit could not read

Description

Writes the parts of a package that `pkgaudit` does not analyze – C, C++, Fortran, Rust, Python and JavaScript among them, and the vignette chunks written in those languages – into a directory another static analyzer can be pointed at.

Usage

```
export_unscanned(
  object,
  dir,
  source = NULL,
  overwrite = FALSE,
  max_bytes = .max_export_bytes
)
```

Arguments

object	A pkgaudit object.
dir	Directory to write into. Required, and created if absent. Naming it is how the caller consents to being written to, so there is no default.
source	The package to read from. Defaults to <code>object\$metadata\$pkg_path</code> , which is right for a directory scan. A tarball scan must supply this: the directory it was extracted into is gone by the time the scan returns.
overwrite	Logical; if FALSE (default), <code>dir</code> must be absent or empty and an existing file is left alone. TRUE allows writing into a directory that already holds something and replaces a previous export of the same file. Nothing is ever deleted either way.
max_bytes	Largest file to copy, in bytes; defaults to 64 MB. A larger one is recorded in the manifest and left behind. This is a separate limit from the 10 MB one audit_package() scans under: copying a file costs far less than parsing it.

Details

A whole file is copied verbatim. A span – a vignette chunk in a language with no analyzer – is written into a file of its own, blank-padded so its code sits at the same line numbers it occupies in the source. A finding another tool reports at line 40 of `intro.python.py` is therefore at line 40 of `intro.Rmd`, with no offset to apply. All spans of one language in one source share a file, since that is how they run.

What gets exported is read from the coverage frame, so the scan’s own account of what it could not read is the single source of truth.

Value

Invisibly, a manifest data frame with one row per exportable span: `path` (relative to `dir`, NA if not written), `file_context`, `language`, `first_line`, `last_line`, `written`, and `note`.

Security considerations

This is the only function that writes where a caller sends it, and both the content and the file names come from an untrusted package. Therefore:

- every target path is resolved and must lie under `dir`;
- a path component that is `.`, `..`, or that contains a separator or a control character is refused;

- symlinks are never followed, and content is read and rewritten rather than copied, so a link pointing out of the package cannot pull a file in;
- nothing is written executable, and nothing is removed.

A refused span is recorded in the manifest rather than dropped, since a file that cannot be exported safely is one worth knowing about. Exporting still executes nothing.

Examples

```
# untrustedpkg is a small package shipped with pkgaudit to be scanned. It is
# R and shell throughout, so there is nothing to hand on and the manifest
# comes back empty; a package carrying compiled code yields one row per
# exported file.
tarball <- system.file(
  "extdata", "untrustedpkg", "untrustedpkg-0.1.0.tar.gz",
  package = "pkgaudit"
)
exdir <- file.path(tempdir(), "untrustedpkg-example")
utils::untar(tarball, exdir = exdir)

result <- audit_package(file.path(exdir, "untrustedpkg"))
manifest <- export_unscanned(result, file.path(tempdir(), "for-semgrep"))
nrow(manifest)
```

extract_segments	<i>Read one source file into the code segments it contains</i>
------------------	--

Description

Dispatches on the file-context rule's type, which is the only place a new variety of file is named.

Usage

```
extract_segments(source)
```

Arguments

source	A <code>pkgaudit_source</code> : a list with the file's path, its package-root-relative <code>file_context</code> , the claiming rule in <code>file_rule</code> , the applicable <code>code_contexts</code> , and macros. Sources are built inside <code>audit_package()</code> , one per claimed file; a method receives them and never constructs one.
--------	--

Value

A list of two elements:

segments List of segments from `new_segment()`, each holding lines aligned to the lines of the source file.

errors Data frame with columns `step`, `file_context`, `rule`, `message`.

Security considerations

The scanning size limit is enforced here, before dispatch, so no method can skip it. A method must not re-implement it.

A method reads untrusted bytes and must never evaluate them. Return the code as text in a segment; deciding what it means is `analyze_segment()`'s job.

See Also

`analyze_segment()`, the other axis of dispatch, and `vignette("internals")`.

Examples

```
# Adding a file format outside pkgaudit: a method for a new rule `type`,
# here one that treats a .txt file as if it held R code.
extract_segments.txt <- function(source) {
  list(segments = list(new_segment(language = "R",
                                   lines = readLines(source$path),
                                   file_context = source$file_context)),
       errors = NULL)
}
registerS3method("extract_segments", "txt", extract_segments.txt)
```

format.pkgaudit

Format or print a pkgaudit result

Description

`format.pkgaudit()` renders the scan metadata and finding counts as a character vector of lines; `print.pkgaudit()` writes those lines and returns the object invisibly.

Usage

```
## S3 method for class 'pkgaudit'
format(x, path = TRUE, ...)

## S3 method for class 'pkgaudit'
print(x, path = TRUE, ...)
```

Arguments

<code>x</code>	A <code>pkgaudit</code> object.
<code>path</code>	Logical; if <code>TRUE</code> (default) include the Path: line showing the local filesystem location scanned, with the home directory written as <code>~</code> . Set <code>FALSE</code> to omit the line.
<code>...</code>	Ignored, for S3 compatibility.

Value

`format.pkgaudit()` returns a character vector of lines. `print.pkgaudit()` returns `x` invisibly.

See Also

[summary.pkgaudit\(\)](#) for a sectioned report of the findings themselves rather than their counts.

Examples

```
# untrustedpkg is a small package shipped with pkgaudit to be scanned.
tarball <- system.file(
  "extdata", "untrustedpkg", "untrustedpkg-0.1.0.tar.gz",
  package = "pkgaudit"
)
result <- audit_tarball(tarball)

print(result)
print(result, path = FALSE) # omit the local Path: line for sharing
```

hash_manifest

Compute a manifest hash for a directory

Description

Hashes a package manifest, a canonically ordered list of every included file's relative path and SHA-256 content hash.

Usage

```
hash_manifest(path, exclude = c("^\\.git/", "^\\.Rproj\\.user/"))
```

Arguments

<code>path</code>	Path to the directory to hash.
<code>exclude</code>	Character vector of regular expressions matched against paths relative to <code>path</code> . Defaults exclude version-control and IDE scratch state, which are not part of the package and (for <code>.git/</code>) are large and volatile. Pass <code>character(0)</code> to hash everything present.

Details

A directory, unlike a tarball, has no single canonical byte stream to hash. The manifest is returned alongside the hash so two disagreeing scans can be diffed to find which file differs.

A directory hash is a weaker provenance claim than a tarball hash: its scope depends on `exclude`, and empty directories and symlinks are not represented, so trees differing only in those hash identically. For vetting untrusted code, prefer hashing the tarball.

Value

A list with:

hash SHA-256 of the manifest text (character scalar).

manifest The manifest text (character scalar), one line per file, formatted as "<hash> <relative path>".

n_files Number of files included (integer).

exclude The exclusion patterns applied, for recording in metadata.

symlinks Relative paths of symlinks and files under symlinked directories, excluded from the manifest (character).

unreadable Relative paths that could not be hashed, excluded from the manifest (character).

Examples

```
# untrustedpkg is a small package shipped with pkgaudit to be scanned.
tarball <- system.file(
  "extdata", "untrustedpkg", "untrustedpkg_0.1.0.tar.gz",
  package = "pkgaudit"
)
exdir <- file.path(tempdir(), "untrustedpkg-example")
utils::untar(tarball, exdir = exdir)

hash_manifest(file.path(exdir, "untrustedpkg"))$hash
```

load_rules

Load security rules from the pkgaudit rules database

Description

Loads the file-context, code-context, pattern, and match rules, and the lifecycle phases of every context, from the bundled SQLite database as a named list suitable for passing to [audit_package\(\)](#) or [audit_tarball\(\)](#).

Usage

```
load_rules(db_path = .db_path())
```

Arguments

db_path Path to the rules database. Defaults to the database bundled with the installed package.

Details

Before reading, the database is verified against its SHA-256 sidecar so that a tampered or corrupted rules file is rejected rather than silently trusted.

Value

A named list of five data frames. In the four rule frames each row is a rule, the common columns are name and version, and message is what a finding reports; phases is keyed by context instead.

file_contexts name, version, type, message, path, recursive, report, filename, code_context, assume_called. type selects how a matched file is read. report is TRUE for a rule whose matches are findings in their own right. code_context names the code-context rules that can apply inside these files, space-separated – NA where none can, "computed" where only top_level and in_function can. assume_called says whether code inside a function definition is taken to run when the code around it runs, and is NA wherever code_context is.

code_contexts name, version, language, message, kind, xpath, segment. kind is "xpath" for a rule matched against the parse tree and "segment" for one matched against a label the extractor stamped; exactly one of the two columns is set accordingly.

patterns name, version, language, message, attck, functions, xpath. functions is the space-separated names the rule matches as a bare call, which is how `find_indirect()` attributes an indirect call back to it; empty for a rule matching on more than the callee.

matches name, version, language, message, attck, regex. A match rule is evaluated only against code in its language.

phases context, version, and one logical column per lifecycle phase. One row per file-context or code-context rule – pattern and match rules carry no phases of their own. The computed contexts top_level and in_function have none, and need none: they inherit.

The list carries a "provenance" attribute – db_path, version and sha256 – which `audit_package()` reports in a scan's metadata. The sha256 is the hash computed during verification, not re-read from the sidecar afterwards, so it records what this call measured.

Security considerations

The sidecar ships beside the database, so anyone who can rewrite one can rewrite the other. It is an integrity check, not an authenticity check: it catches a corrupted download or a database swapped for another, and it cannot catch a deliberate edit of an installed copy. The out-of-band anchor is the hash published in the README, which CI compares against the shipped database on every push and pull request; checking an installed copy against that detects a tampered install, and `rules_version()` reports what a scan actually used.

Verification is time-of-check to time-of-use: the database is hashed and then re-opened by path to query it, so a file swapped in between is not detected. That does not weaken the bundled default, since an attacker able to win the race already has write access to the installed package. When loading rules from a path other parties can write, treat the check as protection against corruption or a substituted database given an authentic sidecar, not against an attacker modifying the file concurrently.

Examples

```
rules <- load_rules()
vapply(rules, nrow, integer(1))
```

new_findings

*Build the return value of an [analyze_segment\(\)](#) method***Description**

Each frame is reduced to its canonical columns and an omitted one is empty. `rbind()` accepts a stray column silently, so conforming here is what keeps a malformed frame out of the result.

Usage

```
new_findings(
  patterns = NULL,
  matches = NULL,
  coverage = NULL,
  errors = .empty_errors()
)
```

Arguments

`patterns, matches, coverage` Data frames with the columns [audit_package\(\)](#) documents, less the phase columns, or NULL for none. Extra columns are dropped; the phases are resolved later, from context.

`errors` Data frame with columns `step`, `file_context`, `rule` and `message`.

Value

A list of the four frames, each conformed to its canonical columns.

See Also

[analyze_segment\(\)](#), whose methods return this.

Examples

```
# An analyzer that found nothing still returns all four frames.
str(new_findings(), max.level = 1)
```

new_segment	<i>Build one code segment</i>
-------------	-------------------------------

Description

A contiguous, line-aligned run of code in one language, classed by that language so [analyze_segment\(\)](#) can dispatch on it. An [extract_segments\(\)](#) method returns these.

Usage

```
new_segment(
  language,
  lines,
  file_context,
  context = NA_character_,
  file_rule = NA_character_,
  code_contexts = NULL,
  guarded_lines = integer(0L)
)
```

Arguments

language	The language of the code, which selects the analyzer.
lines	Character vector of the segment's lines, blank-padded to the line numbers they occupy in the file so a finding's line is the real one.
file_context	Package-root-relative path of the file it came from.
context	The segment's label, matched by a kind: segment code-context rule, or NA where it has none. It is the only record of which part of a help file the code came from.
file_rule	The file-context rule that claimed the file.
code_contexts	The code-context rules that can apply, from the source.
guarded_lines	Integer vector of line numbers whose code ships but the lifecycle does not run – a <code>\dontrun{}</code> block, or a chunk marked <code>eval=FALSE</code> . Phases still come from the context, so they are an upper bound.

Value

A `pkgaudit_segment` object.

See Also

[extract_segments\(\)](#), whose methods return these.

Examples

```
seg <- new_segment(language = "R", lines = c("", "system('id')"),
                  file_context = "R/f.R")
class(seg)
```

rules_version	<i>Return the current rules database version</i>
---------------	--

Description

Returns the version string of the rules database currently shipped with the package. Findings reports should always record the rules version to ensure reproducibility across audit cycles.

Usage

```
rules_version(db_path = .db_path())
```

Arguments

db_path	Path to the SQLite rules database. Defaults to the database bundled with the installed package.
---------	---

Details

Like [load_rules\(\)](#), this verifies the database against its SHA-256 sidecar before reading; see its Security considerations.

Value

A character string giving the rules database version (e.g., "0.1.0").

Examples

```
rules_version()
```

summary.pkgaudit	<i>Summarize a pkgaudit result</i>
------------------	------------------------------------

Description

Takes a pkgaudit object produced by `audit_package()` or `audit_tarball()` and returns summaries of it: how often each rule fired and in which lifecycle phase, how much of the package was read, and the errors, if any. `print.summary.pkgaudit()` writes them as a sectioned report.

Usage

```
## S3 method for class 'pkgaudit'
summary(object, path = TRUE, phase = NULL, ...)

## S3 method for class 'summary.pkgaudit'
print(x, path = x$path, ...)
```

Arguments

object	A pkgaudit object.
path	Logical; if TRUE (default) include the Path: line showing the local filesystem location scanned, with the home directory written as ~. Set FALSE to omit the line. <code>summary.pkgaudit()</code> records the choice in the object it returns; <code>print.summary.pkgaudit()</code> uses that recorded value unless given its own.
phase	Character vector of lifecycle phases to report, e.g. "at_load", or "none" for occurrences that execute in no phase. NULL (default) reports every phase. An unrecognized name is an error.
...	Ignored, for S3 compatibility.
x	A summary.pkgaudit object.

Details

The report opens with the same metadata block as `print.pkgaudit()`, then gives the R Patterns, Shell / Make Matches, Coverage and Errors sections; a section with nothing to report says so. The `file_contexts` summary is returned for programmatic use but is not part of the report.

Both findings tables are grouped by phase and rule alone. Where a finding sits is on the object's own frames; the report answers what runs, and when. An occurrence executes in every phase its context does, so it contributes one row per phase and n can sum to more than the number of occurrences. Those executing in no phase are gathered under none.

Coverage is counts by status and location, and deliberately no percentage: coverage never reaches 100%. Which files went unexamined is in the object's coverage frame.

validate_tar

Validate a source package tarball before extraction

Description

Fail closed: refuse the whole archive rather than extracting a filtered subset, so a partially-validated archive is never written to disk. This is the check `audit_tarball()` runs before extracting an untrusted tarball.

Usage

```
validate_tar(
  tarfile,
  max_entries = 100000L,
  max_bytes = 2 * 1024^3,
  max_ratio = 256
)
```

Arguments

tarfile	Path to a .tar or .tar.gz archive.
max_entries	Maximum number of entries to read before failing closed. Default 100,000.
max_bytes	Maximum uncompressed bytes to read before failing closed. Default 2 GB. Raise for ecosystems with larger artifacts, e.g. Bioconductor annotation and experiment-data packages.
max_ratio	Maximum uncompressed:compressed ratio before failing closed, or Inf to disable. Targets decompression bombs, which are characterised by extreme ratios rather than absolute size. Default 256, well under the ~1032:1 ceiling of a single gzip layer.

Details

Rejects link entries (symlink, hard link), non-standard typeflags (GNU long-name, PAX), path traversal, absolute and drive-qualified paths, paths containing backslashes or control characters, empty paths, unparseable size fields, and archives that do not extract to exactly one top-level directory. It also enforces the entry-count, uncompressed-size, and expansion-ratio caps applied while reading (`max_entries`, `max_bytes`, `max_ratio`). Reads gzip and uncompressed tar only, judged by the file's magic bytes rather than its name: a bzip2, xz, zstd or compress stream is refused whatever it is called.

A refusal is signaled as a `pkgaudit_invalid_tarball` condition (a subclass of error), so it stops by default but can be caught by class.

Value

Invisibly, a data frame of the archive's entries with columns `name`, `type`, `linkname`, and `size`.

Examples

```
# Returns the archive's entries, or stops with a "Refusing archive" error if
# the tarball is malicious or malformed.
tarball <- system.file(
  "extdata", "untrustedpkg", "untrustedpkg_0.1.0.tar.gz",
  package = "pkgaudit"
)

entries <- validate_tar(tarball)
entries
```

Index

analyze_segment, 2
analyze_segment(), 10, 14, 15
audit_package, 3
audit_package(), 2, 6, 8, 9, 12–14, 17
audit_tarball, 5
audit_tarball(), 3, 12, 17, 19

base::tempdir(), 5

emit_sarif, 6
export_unscanned, 7
extract_segments, 9
extract_segments(), 3, 15

find_indirect(), 13
format.pkgaudit, 10

hash_manifest, 11
hash_manifest(), 3

load_rules, 12
load_rules(), 2–5, 16

new_findings, 14
new_findings(), 3
new_pkgaudit(), 4
new_segment, 15

print.pkgaudit(format.pkgaudit), 10
print.pkgaudit(), 17, 18
print.summary.pkgaudit
 (summary.pkgaudit), 17

rules_version, 16
rules_version(), 13

summary.pkgaudit, 17
summary.pkgaudit(), 11

validate_tar, 19
validate_tar(), 6