

Package ‘CBTF’

July 3, 2026

Type Package

Title Caught by the Fuzz! - A Minimalistic Fuzz-Test Runner

Version 0.7.0

Date 2026-07-03

Maintainer Marco Colombo <mar.colombo13@gmail.com>

Description A simple runner for fuzz-testing functions in an R package's public interface. Fuzz testing helps identify functions lacking sufficient argument validation, and uncovers problematic inputs that, while valid by function signature, may cause issues within the function body.

URL <https://mcol.github.io/caught-by-the-fuzz/>

BugReports <https://github.com/mcol/caught-by-the-fuzz/issues>

Imports mirai (>= 2.5.2), rlang (>= 1.1.7), cli (>= 3.6.5)

Suggests testthat (>= 3.2.3), withr (>= 3.0.2)

License GPL-3

Encoding UTF-8

Config/testthat/edition 3

Config/roxygen2/version 8.0.0

NeedsCompilation no

Author Marco Colombo [aut, cre] (ORCID:
<<https://orcid.org/0000-0001-6672-0623>>)

Repository CRAN

Date/Publication 2026-07-03 13:40:02 UTC

Contents

CBTF-package	2
fuzz	3
get_exported_functions	6
length.cbtf	7

<code>print.cbtf</code>	8
<code>summary.cbtf</code>	9
<code>test_inputs</code>	10
<code>whitelist</code>	11
<code>[].cbtf</code>	11

Index	13
--------------	-----------

CBTF-package

CBTF: Caught by the Fuzz! A minimalistic fuzz-test runner

Description

This package implements a very simple mechanism for fuzz-testing functions in the public interface of an R package.

Details

Fuzz testing helps identify functions that lack sufficient argument validation, and uncovers sets of inputs that, while valid by function signature, may cause issues within the function body.

The core functionality of the package is in `fuzz`, which calls each provided function with a certain input and records the output produced. If an error or a warning is generated, this is captured and reported to the user, unless it matches a pattern of whitelisted messages. The objects returned by `fuzz` can be inspected with `summary.cbtf` and `print.cbtf`.

Whitelisting can also be done after a fuzz run has been completed via the `whitelist` function, so that only messages that need to be acted upon are actually shown. Using `whitelist` has the advantage of not requiring the completion of a fuzz run of all functions over all inputs again.

Note that `fuzz` relies on the `mirai` package for asynchronous operations and parallelisation, and execution occurs on persistent background processes. These can be started automatically by specifying the `daemons` option; alternatively, they can be set up manually with the `mirai::daemons()` function; refer to the original `mirai` documentation for a complete description of its arguments and behaviour.

The helper function `get_exported_functions` identifies the functions in the public interface of a given package, facilitating the generation of the list of functions to be fuzzed.

The helper function `test_inputs` is invoked by `fuzz` if the user doesn't specify the set of inputs to be tested. By default it generates a large set of potentially problematic inputs, but these can be limited just to the desired classes of inputs.

Author(s)

Maintainer: Marco Colombo <mar.colombo13@gmail.com> ([ORCID](#))

Authors:

- Marco Colombo <mar.colombo13@gmail.com> ([ORCID](#))

See Also

Useful links:

- <https://mcol.github.io/caught-by-the-fuzz/>
- Report bugs at <https://github.com/mcol/caught-by-the-fuzz/issues>

fuzz

Fuzz-test the specified functions

Description

The fuzzer calls each function in `funcs` with the argument list provided in `args` (each of its elements in turn modified by each object in `what`) and records any errors or warnings that are thrown. If no error occurs within the first timeout seconds, the execution of the function being fuzzed is interrupted and the next one is started.

Usage

```
fuzz(
  funcs,
  what = test_inputs(),
  args = NULL,
  package = NULL,
  listify_what = FALSE,
  ignore_patterns = "",
  ignore_warnings = FALSE,
  daemons = 2L,
  timeout = 2
)
```

Arguments

<code>funcs</code>	A character vector of function names to test. If the vector has a "package" attribute and no package argument is given, functions are loaded from the namespace specified in that attribute.
<code>what</code>	A list of objects; each is used, in turn, to modify the list of arguments in <code>args</code> before calling each of the functions in <code>funcs</code> . If no inputs are provided, a default set of inputs generated by test_inputs will be used. If set to <code>NULL</code> , then <code>args</code> must be specified, and all functions will be called with that exact list of arguments with no fuzzing occurring.
<code>args</code>	A list of arguments to pass to the functions being fuzzed. Each element in the list is in turn replaced by each object in <code>what</code> , then each modified argument list is used to fuzz the functions in <code>funcs</code> . Argument names are preserved and used as named arguments in the fuzzed functions. If <code>NULL</code> (default), only the first argument of each function is fuzzed.

package	Name of the package where functions are searched. If NULL (default), the function first checks the "package" attribute of funs, and if that is not set, names are searched in the global namespace.
listify_what	Whether each input in what should also be tested in its listified version (FALSE by default). When set to TRUE, if what is <code>list(x = x)</code> , the function will operate as if it were <code>list(x = x, "list(x)" = list(x))</code> , for any input object x.
ignore_patterns	One or more strings containing regular expressions to match the errors to ignore. The strings "unused argument" and "is missing, with no default" are always ignored.
ignore_warnings	Whether warnings should be ignored (FALSE by default).
daemons	Number of daemons to use (2 by default). As many mirai daemons as specified will be started when entering the function and closed at the end, unless active daemons are already available, in which case the argument is ignored and the active daemons are used.
timeout	Number of seconds (2 by default) after which the function being fuzzed is interrupted with result status set to "OK".

Details

Multiple arguments:

A list of arguments to be passed to the functions being fuzzed can be provided via the `args` argument. Each element in that list is modified in turn by each object in `what` and the resulting list of arguments is then passed to each function via `do.call()`. If more arguments are given than the number of formal arguments accepted by a function, that function will produce a "SKIP" result.

If arguments are named, they will be passed with their names to the fuzzed functions. If a function doesn't have a formal argument of that name and doesn't accept `...`, then the standard R behaviour is to return an "unused argument" error. This is whitelisted by default in `fuzz()`, and the corresponding result status is set to "OK".

It is possible to define arguments that should remain unchanged while fuzzing by prefixing their names with `...`. These arguments will use the values assigned in `args` without modification. For example, to ensure that the argument `na.rm` is always set to TRUE, it should be specified as `..na.rm = TRUE` in `args`. If all elements in `args` are fixed, what is ignored and all functions in `funs` will be called with the provided `args` list.

Parallel execution:

The implementation uses `mirai` as a backend to execute tasks asynchronously in parallel worker processes. The function can start a pool of persistent background processes (daemons) of size given by the `daemons` argument (note that starting more daemons than available cores yields no benefit). Alternatively, the function can also make use of already active daemons started with the [mirai::daemons](#) function: this allows to control in greater detail the number of processes to use, which can also be remote.

Whitelisting:

In order to reduce the number of false positive results produced, this function applies the following set of rules, to establish if an error or warning condition should be ignored (whitelisting):

- If the name of the function appears in the error or warning message, as it is considered that the condition has been handled by the developer.
- If the error or warning message contains the text "is missing, with no default", which is produced when a missing argument is used without a value being assigned to it.
- If the error or warning message contains any of the patterns specified in `ignore_patterns`. In case of multiple warnings being captured, the result is whitelisted only if *all* warnings match any of the patterns.
- If a warning is thrown but `ignore_warnings = TRUE` is set.

In all whitelisted cases, the result is "OK", and the message that was received is stored in the `$msg` field (see the *Value* section).

Note: Whitelisting can also be applied post-hoc on the results of a fuzz run using the [whitelist](#) function.

Value

An object of class `cbtf` that stores the results obtained for each of the functions tested. This contains the following fields:

<code>runs</code>	a list of data frames, each containing the results of fuzzing all the functions in <code>funcs</code> with one of the inputs in <code>what</code> . The data frame contains the following columns and attributes: - <code>res</code>: The result of the fuzz test, see below for the possible values. - <code>msg</code>: The error or warning message returned by the function, if any. - <code>attr(*, "what")</code>: The character representation of the input tested.
<code>funcs</code>	a vector of names of the functions tested.
<code>args</code>	a named list of arguments, with names generated by deparsing the <code>args</code> argument if not already provided.
<code>package</code>	a character string specifying the package name where function names were searched, or NA if none was provided.
<code>ignore_patterns</code>	The value of the <code>ignore_patterns</code> argument.
<code>ignore_warnings</code>	The value of the <code>ignore_warnings</code> argument.

The `res` column in each of the data frames in the `$runs` field can contain the following values:

- **OK**: either no error or warning was produced (in which case, the `msg` entry is left blank), or it was whitelisted (in which case, the message received is stored in `msg`), or it was timed out (in which case, `msg` records that a timeout was applied).
- **SKIP**: no test was run, either because the given name cannot be found, or it doesn't correspond to a function, or the function accepts no arguments, or more arguments were provided than the function accepts; the exact reason is given in `msg`.
- **WARN**: a warning was thrown for which no whitelisting occurred and `ignore_warnings = FALSE`; its message is stored in `msg`.
- **FAIL**: an error was thrown for which no whitelisting occurred; its message is stored in `msg`.

See Also

[get_exported_functions](#), [test_inputs](#), [whitelist](#), [summary.cbtf](#), [print.cbtf](#)

Examples

```
## set up persistent background processes
mirai::daemons(2L)

## this should produce no errors
res <- fuzz(funs = c("list", "matrix", "mean"),
           what = test_inputs(c("numeric", "raw")))
summary(res)

## display all results even for successful tests
print(res, show = "all")

## this will catch an error (false positive)
fuzz(funs = "matrix", what = test_inputs("scalar"))

## apply a whitelist pattern to remove the false positive
fuzz(funs = "matrix", what = test_inputs("scalar"),
     ignore_patterns = "'data' must be of a vector type")

## close the background processes
mirai::daemons(0L)
```

`get_exported_functions`

Get the names of the exported functions of a package

Description

This function extracts the exports from the namespace of the given package via [getNamespaceExports](#) and discards non-fuzzable objects (non-functions and functions with no arguments). The set of names returned can be further restricted via the `ignore_names` and `ignore_deprecated` arguments.

Usage

```
get_exported_functions(package, ignore_names = "", ignore_deprecated = TRUE)
```

Arguments

<code>package</code>	Name of the package where functions are searched.
<code>ignore_names</code>	Names of functions to ignore: these are removed from the names returned. This can be helpful, for example, to discard function aliases.
<code>ignore_deprecated</code>	Whether deprecated functions should be ignored (TRUE by default).

Value

A character vector of the names of the fuzzable functions exported from the given package, with the "package" attribute set. This can be used directly as the `funs` argument of [fuzz](#) without the need to specify the package argument.

See Also

[fuzz](#)

Examples

```
## get the fuzzable functions in the public interface of this package
funs <- get_exported_functions("CBTF")
```

length.cbtf	<i>Compute the number of tests performed</i>
-------------	--

Description

This returns the total number of tests as the product of the number of inputs used and the number of functions tested.

Usage

```
## S3 method for class 'cbtf'
length(x)
```

Arguments

`x` An object of class `cbtf`.

Value

An integer corresponding to the number of tests performed in a run.

Examples

```
res <- fuzz(funs = c("list", "matrix", "mean"),
            what = test_inputs(c("numeric", "raw")))
length(res)
```

print.cbtf	<i>Print the results from a fuzz run</i>
------------	--

Description

This formats the results from a fuzz run with colours and prints them to the terminal. By default, only tests labelled as "FAIL" or "WARN" are displayed, but this can be controlled via the `show` argument.

Usage

```
## S3 method for class 'cbtf'
print(x, show = c("fail", "warn"), group = "input", ...)
```

Arguments

<code>x</code>	An object of class <code>cbtf</code> .
<code>show</code>	A character vector representing the subset of results to be printed, any of "fail", "warn", "skip", "ok", "all" and "none".
<code>group</code>	Either "input" to show results grouped by test input (default), or "function" to show them grouped by function name.
<code>...</code>	Further arguments passed to or from other methods. These are currently ignored.

Details

The use of unicode icons in the output messages can be disabled by setting `options(cli.unicode = FALSE)`.

Value

No return value, called for side effects.

See Also

[summary.cbtf](#)

Examples

```
res <- fuzz(funs = c("list", "matrix", "mean"),
            what = test_inputs(c("numeric", "raw")))
print(res)
print(res, show = "all")
print(res, show = "none")
print(res, group = "function")
```

summary.cbtf

*Results summary from a fuzz run***Description**

Reports some summary statistics from the results of a run of [fuzz](#).

Usage

```
## S3 method for class 'cbtf'
summary(object, tabulate = TRUE, ...)
```

Arguments

object	An object of class cbtf.
tabulate	Whether a tabulation of results should be printed out (TRUE by default). The tabulation can always be retrieved from the "summary_table" attribute of the returned object also when tabulate = FALSE.
...	Further arguments passed to or from other methods. These are currently ignored.

Details

The use of unicode icons in the output messages can be disabled by setting `options(cli.unicode = FALSE)`.

Value

A data frame containing the following columns and attributes is returned invisibly:

fun	The names of the function tested.
what	The inputs tested.
res	One of "OK", "FAIL", "WARN" or "SKIP" for each combination of function and input tested (see the <i>Value</i> section in fuzz).
msg	The message received in case of error, warning or skip, or an empty string if no failure occurred.
attr(*, "summary_table")	The tabulation of results that was printed out.

See Also

[print.cbtf](#)

Examples

```
res <- fuzz(funs = c("list", "matrix", "mean"),
           what = test_inputs(c("numeric", "raw")))
summary(res)
```

`test_inputs`*Default input tests*

Description

This function provides a selection of potentially problematic inputs by class. List inputs are very limited by design, as they can be automatically generated by setting `listify_what = TRUE` in [fuzz](#).

Usage

```
test_inputs(use = "all", skip = "")
```

Arguments

<code>use</code>	Names of input classes to use. Valid names are "all" (default), "scalar", "numeric", "integer", "logical", "character", "factor", "data.frame", "matrix", "array", "date", "time", "raw", "na" and "list". A vector of valid classes can be retrieved programmatically by setting this argument to "help".
<code>skip</code>	Names of input classes to skip. Overrides any classes selected via the <code>use</code> argument.

Value

A named list of inputs corresponding to the input classes selected, or a character vector of valid input classes if `use = "help"`.

See Also

[fuzz](#)

Examples

```
## only the scalar and numeric tests
inputs1 <- test_inputs(use = c("scalar", "numeric"))

## everything but the data, raw and list tests
inputs2 <- test_inputs(skip = c("date", "raw", "list"))

## print the valid input classes
test_inputs("help")
```

whitelist

Apply additional whitelist patterns to the results of a fuzz run

Description

This allows for post-hoc whitelisting of results according to the patterns specified.

Usage

```
whitelist(object, patterns)
```

Arguments

object	An object of class cbtf.
patterns	One or more strings containing regular expressions to match the errors to whitelist.

Value

An object of class cbtf with the additional whitelist patterns applied.

See Also

[fuzz](#)

Examples

```
## this reports a false positive result
(res <- fuzz(funs = "matrix", what = test_inputs("scalar"))))

## with whitelisting, we can remove that
whitelist(res, "must be of a vector type")
```

[[.cbtf

Extract the results for a specific test input

Description

This extracts the raw results for a single test input by its position, returning the corresponding data frame from the \$runs list.

Usage

```
## S3 method for class 'cbtf'
x[[i]]
```

Arguments

x	An object of class <code>cbtf</code> .
i	An index between 1 and the number of test inputs used.

Value

If the index is valid, a data frame containing the following columns and attributes:

res	One of "OK", "FAIL", "WARN" or "SKIP" for each combination of function and input tested (see the <i>Value</i> section in fuzz).
msg	The message received in case of error, warning or skip, or an empty string if no failure occurred.
attr(*, "what")	The character representation of the input tested.
	Otherwise, FALSE.

Examples

```
res <- fuzz(funs = c("list", "matrix", "mean"),
           what = test_inputs(c("numeric", "raw")))
res[[6]]
```

Index

`[].cbtf`, [11](#)

CBTF (CBTF-package), [2](#)

CBTF-package, [2](#)

fuzz, [2](#), [3](#), [7](#), [9–12](#)

get_exported_functions, [2](#), [6](#), [6](#)

getNamespaceExports, [6](#)

length.cbtf, [7](#)

mirai::daemons, [4](#)

print.cbtf, [2](#), [6](#), [8](#), [9](#)

summary.cbtf, [2](#), [6](#), [8](#), [9](#)

test_inputs, [2](#), [3](#), [6](#), [10](#)

whitelist, [2](#), [5](#), [6](#), [11](#)