Package: CBTF (via r-universe)

October 19, 2026

Type Package

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

Version 0.7.0.9000

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

Roxygen list(markdown = TRUE)

Config/testthat/edition 3

Config/roxygen2/version 8.1.0

Repository https://mcol.r-universe.dev

Date/Publication 2026-10-19 19:59:09 UTC

RemoteUrl https://github.com/mcol/caught-by-the-fuzz

RemoteRef HEAD

RemoteSha d79916dd5b5d71ce3cbf4a084f32524b921aed98

Contents

CBTF-package . . . 2
[[.cbtf . . . 3
fuzz . . . 4
get_exported_functions . . . 8
length.cbtf . . . 9
print.cbtf . . . 10
subset.cbtf . . . 11
summary.cbtf . . . 11
test_inputs . . . 12
whitelist . . . 13

Index **15**

`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, and filtered with subset.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

---

`[[.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 rows from the `$runs` data frame.

Usage

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

Arguments

`x`	An object of class `cbtf`.
`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:

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

For an invalid index, it returns `NULL`.

Examples

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

fuzz *Fuzz-test the specified functions*

Description

The fuzzer calls each function in `funs` 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(
  funs,
  what = test_inputs(),
  args = NULL,
  package = NULL,
  listify_what = FALSE,
  ignore_patterns = "",
  ignore_warnings = FALSE,
  daemons = 2L,
  timeout = 2
)
```

Arguments

`funs`	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.
`what`	A list of objects; each is used, in turn, to modify the list of arguments in `args` before calling each of the functions in `funs`. If no inputs are provided, a default set of inputs generated by test_inputs will be used. If set to `NULL`, then `args` must be specified, and all functions will be called with that exact list of arguments with no fuzzing occurring.
`args`	A list of arguments to pass to the functions being fuzzed. Each element in the list is in turn replaced by each object in `what`, then each modified argument list is used to fuzz the functions in `funs`. If arguments are named, they will be passed with their names to the fuzzed functions. Any argument name may be prefixed by `..` to fix its value to the one provided by the user. If `args = NULL` (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 `list(x = x)`, the function will operate as if it were `list(x = x, "list(x)" = list(x))`, 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 "SKIP".

Details

Multiple arguments:

A list of arguments to be passed to the functions being fuzzed can be provided via the `args` argument. If this is left at its default of `NULL`, only the first argument of each function is fuzzed.

Internally, `args` is expanded into multiple lists, each differing by one element: each element in the `args` 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()`. Thus, for example, if `args` has length 3 and `what` has length 2, in general a total of 6 input lists will be generated and tested.

If arguments are named, they will be passed with their names to the fuzzed functions. This allows to fuzz arguments out of order or arguments that are passed via `...`. 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".

If more arguments are given than the number of formal arguments accepted by a function, that function will produce a "SKIP" result.

Fixed arguments:

It is possible to define arguments that should remain unchanged during 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. A similar behaviour can be obtained by specifying no fixed arguments in `args` while setting `what = NULL`.

Parallel execution:

The implementation uses `mirai` as a backend to execute tasks asynchronously in parallel worker processes. There are two main approaches to control parallel execution:

1. Setting the `daemons` argument to an integer value (2 by default): this will start as many daemons as specified (and shut them down automatically at the end of the fuzz run). Note that there is no benefit in starting more daemons than the number of available cores.
2. Manually setting up the daemons before the start of the function: this can be accomplished via mirai::daemons, which allows to specify remote daemons as well as local ones. This

also avoids the cost of starting and closing daemons if `fuzz()` were to be called multiple times. It remains responsibility of the user to close the daemons when no longer in use. When active daemons are found, the `daemons` argument will be ignored. Refer to the original `mirai` documentation for a complete description of its arguments and behaviour.

Note that by default, the daemons load the package being fuzzed as installed when they start. This means that after any corrections are made to the package, this must be reinstalled before the fuzzer can notice the changes. However, this may not be the expected behaviour when the package is being built with `devtools`. In this setting, it is recommended to start the daemons manually, then run `mirai::everywhere(devtools::load_all())` after any changes that need to be tested.

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 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.

If `options(CBTF.whitelist.function.names = TRUE)` is set, the following rule is also applied:

- 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.

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

Additionally, if a `.cbtf` file is present in the current working directory, its contents are used as an additional source of whitelist patterns. Each non-empty line is treated as a regular expression pattern and appended to the `ignore_patterns` argument; lines starting with # are ignored. This is useful for sharing whitelist patterns across a project or for automatically applying known false positives without having to pass them to every `fuzz()` call.

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

Timeouts:

Long-running functions can slow down the progress of `fuzz()`, so by default if a function does not produce an error within 2 seconds, it will be stopped.

A timed out function returns a `"SKIP"` result, with the corresponding `$msg` field recording that a timeout was applied. All skipped tests (either timed out or not executed for other reasons) can be shown using `print(res, show = "skip")`.

However, the default timeout may be too short (or perhaps too long) in some applications. If desired, the maximum running time of a job (in seconds) can be controlled via the `timeout` argument of `fuzz()`.

Sandboxing:

Running the fuzzer can have the same side effects as the functions being called (writing files, opening plot devices, making network calls, and so on). To prevent those side effects from polluting the current working directory, tests are run within a temporary directory.

Value

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

`runs` a data frame containing the following columns:
- `idx`: The index of the input tested.
- `res`: The result of the fuzz test, see below for the possible values.
- `msg`: The error or warning message returned by the function, if any.
- `fun`: The name of the function tested.
- `what`: The character representation of the input tested.

`funs` a vector of names of the functions tested.

`args` a named list of arguments, with names generated by deparsing the `args` argument if not already provided.

`package` a character string specifying the package name where function names were searched, or `NA` if none was provided.

`ignore_patterns` The value of the `ignore_patterns` argument, including the patterns found in the `.cbtf` file, if present.

`ignore_warnings` The value of the `ignore_warnings` argument.

The `res` column in the `$runs` data frame 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`).
- **SKIP**: no test was run to completion, 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, or the function execution timed out, or the test could not be completed because the worker process died or was otherwise interrupted; 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")

## specify multiple arguments to fuzz
fuzz("matrix", what = list(NA, NULL),
     args = list(data = 1:4, nrow = 2, dimnames = NULL))

## keep the `data` argument fixed
fuzz("matrix", what = list(NA, NULL),
     args = list(..data = 1:4, nrow = 2, dimnames = NULL))

## 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

`package`	Name of the package where functions are searched.
`ignore_names`	Names of functions to ignore: these are removed from the names returned. This can be helpful, for example, to discard function aliases.
`ignore_deprecated`	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 *Compute the number of tests performed*

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` *Print the results from a fuzz run*

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

`x`	An object of class `cbtf`.
`show`	A character vector representing the subset of results to be printed, any of "fail", "warn", "skip", "ok", "all" and "none".
`group`	Either `"input"` to show results grouped by test input (default), or `"function"` to show them grouped by function name.
`...`	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")
```

subset.cbtf *Filter a* cbtf *object*

Description

This filters the rows of the $runs data frame by applying a regular expression to the messages or to the function names, then prints the filtered result and returns it invisibly.

Usage

```
## S3 method for class 'cbtf'
subset(x, msg_patterns = NULL, fun_patterns = NULL, ...)
```

Arguments

x	An object of class cbtf.
msg_patterns	Regular expression pattern to be applied to the msg column. If NULL (default), all messages are reported.
fun_patterns	Regular expression pattern to be applied to the fun column. If NULL (default), all functions are reported.
...	Arguments passed to print.cbtf.

Value

The filtered cbtf object is printed and returned invisibly.

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 *Value* 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

`use`	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".
`skip`	Names of input classes to skip. Overrides any classes selected via the `use` 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")
```

Index

[[.cbtf, 3

CBTF *(*CBTF-package*)*, 2
CBTF-package, 2

fuzz, *2*, *3*, 4, *9*, *11–13*

get_exported_functions, *2*, *7*, 8
getNamespaceExports, *8*

length.cbtf, 9

mirai::daemons, *5*

print.cbtf, *2*, *7*, 10, *11*, *12*

subset.cbtf, *2*, 11
summary.cbtf, *2*, *7*, *10*, 11

test_inputs, *2*, *4*, *7*, 12

whitelist, *2*, *6*, *7*, 13