Package 'safeconduct'

November 20, 2026

Type Package

Title Fetch Files Through a Real Browser When a Plain Request is Refused

Version 0.1.0

Description Some servers refuse any request that does not come from a web browser, with a browser check or a block page instead of the file, 'Cloudflare' being the common case. This package opens a real 'Chrome' with a throwaway profile, lets any check run, and fetches the file or page: through 'curl' carrying the clearance the browser was given where the server accepts that, and through the browser itself where it does not. Nothing is simulated; where a check asks a person to click, a window waits for them. Built on 'chromote'.

License MIT + file LICENSE

Encoding UTF-8

Language en-CA

SystemRequirements Google Chrome or another Chromium-based browser (<https://www.google.com/chrome/>).

Depends R (>= 4.1)

Imports chromote (>= 0.3.0), curl (>= 5.0), jsonlite, processx

Suggests knitr, rmarkdown, testthat (>= 3.0.0), withr, xml2

VignetteBuilder knitr

URL https://mountainmath.github.io/safeconduct/, https://github.com/mountainMath/safeconduct

BugReports https://github.com/mountainMath/safeconduct/issues

Config/testthat/edition 3

Config/roxygen2/version 8.1.0

NeedsCompilation no

Author Jens von Bergmann [aut, cre]

Maintainer Jens von Bergmann <jens@mountainmath.ca>

Repository CRAN

Date/Publication 2026-11-20 11:50:02 UTC

Contents

browser_available . . . 2
browser_session . . . 3
clearance . . . 4
clearance_handle . . . 5
download_in_browser . . . 5
fetch . . . 6
is_block . . . 8
is_challenge . . . 9
page_text . . . 9
probe . . . 10
read_page . . . 11

Index **13**

`browser_available` *Can this package open a browser here?*

Description

Can this package open a browser here?

Usage

```
browser_available()
```

Value

`TRUE`, or `FALSE` with a `reason` attribute: the `safeconduct.browser` option is `FALSE`, or no Chrome or Chromium was found (the `CHROMOTE_CHROME` environment variable points at one).

Examples

```
browser_available()
attr(browser_available(), "reason")
```

`browser_session` *Keep a browser open on a host*

Description

Opens a Chrome with a throwaway profile on the host's root, lets any check run, and keeps it open for `read_page()` and `fetch()` through their `session` argument. Close it with `close()`; otherwise it lives until the R session ends. A browser that has gone away, its window closed, is reopened the same way at the next call, which says so.

Usage

```
browser_session(
  url,
  quiet = FALSE,
  wait = NULL,
  headed = "auto",
  useragent = TRUE
)
```

Arguments

`url`	A URL on the host. `headed` and `useragent` hold for the session's life.
`quiet`	Suppress progress messages.
`wait`	Seconds to wait for the check to pass: by default two minutes with a window, so that a person can click, and thirty seconds headless.
`headed`	How to run the browser when one is needed. `"auto"`, the default, runs headless and opens a window only when the headless run ends on a check or a block and there is a display to show it on; `TRUE` opens a window at once; `FALSE` never does. A second check, at the file's URL, takes headless Chrome about two minutes to pass unclicked, where a window is through as soon as its box is clicked: `"auto"` waits those minutes before opening a window, `TRUE` skips the wait.
`useragent`	The browser's user-agent string. `TRUE`, the default, drops the word `Headless` from Chrome's own string, which some hosts refuse on that word alone, and changes nothing else; `NULL` leaves Chrome's own; a string is used as given.

Value

A `safeconduct_session`, whose `clearance` is what `clearance()` would return. Errors as `fetch()`'s.

Examples

```
## Not run:
s <- browser_session("https://results.vancouver.ca/")
r <- read_page("https://results.vancouver.ca/data/arearesults.json",
               session = s)
```

```
close(s)

## End(Not run)
```

`clearance` *Get a clearance for a host from a browser*

Description

Opens a Chrome on the host, lets any check run, and returns the browser's user-agent string and cookies, which `clearance_handle()` puts on a curl request. Whether the server then serves curl is the server's call: after a block page it usually does not, since the rule is about the client, and `fetch()` then downloads in the browser instead. Clearances are cached per host for the R session.

Usage

```
clearance(
  url,
  refresh = FALSE,
  quiet = FALSE,
  wait = NULL,
  headed = "auto",
  useragent = TRUE
)
```

Arguments

`url`	The file's URL.
`refresh`	Get a new clearance even when one is cached.
`quiet`	Suppress progress messages.
`wait`	Seconds to wait for the check to pass: by default two minutes with a window, so that a person can click, and thirty seconds headless.
`headed`	How to run the browser when one is needed. `"auto"`, the default, runs headless and opens a window only when the headless run ends on a check or a block and there is a display to show it on; `TRUE` opens a window at once; `FALSE` never does. A second check, at the file's URL, takes headless Chrome about two minutes to pass unclicked, where a window is through as soon as its box is clicked: `"auto"` waits those minutes before opening a window, `TRUE` skips the wait.
`useragent`	The browser's user-agent string. `TRUE`, the default, drops the word `Headless` from Chrome's own string, which some hosts refuse on that word alone, and changes nothing else; `NULL` leaves Chrome's own; a string is used as given.

Value

A `safeconduct_clearance`: a list with `host`, `useragent`, `cookie` (one `Cookie:` header value) and `time`. Errors as `fetch()`'s.

Examples

```
## Not run:
cl <- clearance("https://www12.statcan.gc.ca/")
curl::curl_fetch_memory("https://www12.statcan.gc.ca/", clearance_handle(cl))

## End(Not run)
```

`clearance_handle` *A curl handle that carries a clearance*

Description

The user-agent string and the cookies go on a request together: Cloudflare ties the clearance cookie to the user agent that earned it, and the cookie alone is answered with the check again.

Usage

```
clearance_handle(clearance = NULL, ...)
```

Arguments

`clearance`	A clearance from `clearance()`, or `NULL` for a plain handle.
`...`	Further options for `curl::new_handle()`.

Value

A curl handle with `followlocation` set, and the clearance's `useragent` and `cookie` when one was given.

Examples

```
h <- clearance_handle(list(useragent = "Mozilla/5.0", cookie = "cf_clearance=x"))
```

`download_in_browser` *Download a file in a browser*

Description

Opens a Chrome with a throwaway profile on the host, lets any check run, and downloads `url` in it, with no plain request first. `fetch()` does this only when it has to.

Usage

```
download_in_browser(
  url,
  destfile,
  quiet = FALSE,
  timeout = 3600,
  headed = "auto",
  useragent = TRUE
)
```

Arguments

`url`	The file's URL.
`destfile`	Where to write it. A failed transfer leaves nothing there.
`quiet`	Suppress progress messages.
`timeout`	Seconds a transfer in the browser may take.
`headed`	How to run the browser when one is needed. `"auto"`, the default, runs headless and opens a window only when the headless run ends on a check or a block and there is a display to show it on; `TRUE` opens a window at once; `FALSE` never does. A second check, at the file's URL, takes headless Chrome about two minutes to pass unclicked, where a window is through as soon as its box is clicked: `"auto"` waits those minutes before opening a window, `TRUE` skips the wait.
`useragent`	The browser's user-agent string. `TRUE`, the default, drops the word `Headless` from Chrome's own string, which some hosts refuse on that word alone, and changes nothing else; `NULL` leaves Chrome's own; a string is used as given.

Value

`destfile`, invisibly. Errors as `fetch()`'s.

Examples

```
## Not run:
download_in_browser("https://example.org/large.zip", "large.zip")

## End(Not run)
```

`fetch` *Download a file, through a browser if the server insists on one*

Description

Fetches `url` to `destfile`. A plain request is made first, and if the file comes no browser is involved. If the server answers with a browser check (`is_challenge()`) or a block page (`is_block()`), a Chrome with a throwaway profile is opened on the host, any check runs, and the file is downloaded: through curl carrying the browser's clearance where the server accepts that, and in the browser itself where it does not. A clearance curl can use is kept for the rest of the R session, so later files from the host need no browser.

Usage

```
fetch(
  url,
  destfile,
  quiet = FALSE,
  timeout = 3600,
  session = NULL,
  headed = "auto",
  useragent = TRUE
)
```

Arguments

`url`	The file's URL.
`destfile`	Where to write it. A failed transfer leaves nothing there.
`quiet`	Suppress progress messages.
`timeout`	Seconds a transfer in the browser may take.
`session`	A `browser_session()`. The file is then downloaded in that browser without probing, with the session's `headed` and `useragent`, and the browser stays open; one that has gone away is reopened first.
`headed`	How to run the browser when one is needed. `"auto"`, the default, runs headless and opens a window only when the headless run ends on a check or a block and there is a display to show it on; `TRUE` opens a window at once; `FALSE` never does. A second check, at the file's URL, takes headless Chrome about two minutes to pass unclicked, where a window is through as soon as its box is clicked: `"auto"` waits those minutes before opening a window, `TRUE` skips the wait.
`useragent`	The browser's user-agent string. `TRUE`, the default, drops the word `Headless` from Chrome's own string, which some hosts refuse on that word alone, and changes nothing else; `NULL` leaves Chrome's own; a string is used as given.

Details

Nothing is simulated. A check that shows a "Verify you are human" box waits in a window, for up to two minutes, for a person to click it. The window is brought to the front when a check shows in it, and wants to be left there until it is through: a check run with the window behind something else is reissued on every click. A file downloaded in the browser lands under `tempdir()` and is moved to `destfile` when complete.

Value

`destfile`, invisibly. Errors are classed under `safeconduct_error`: `safeconduct_unavailable` when no browser can be opened here, `safeconduct_not_passed` when the check was not passed in time or the server blocks the browser too, with a `reason` field, and `safeconduct_download_error` when the transfer failed.

Requirements

Chrome or Chromium, found by `chromote::find_chrome()`; the `CHROMOTE_CHROME` environment variable points at one somewhere unusual, and `browser_available()` says whether one can be opened here. `options(safeconduct.browser = FALSE)` never opens a browser; the window's size and the other options are described in safeconduct-package.

Tips

At the console, once a host has been found to need a window or the browser for the transfer, a tip says how to skip the finding out next time: `headed = TRUE`, or a `browser_session()`. Tips come once per host and R session, not when `quiet`, and not for calls made from another package. They are messages of class `safeconduct_hint`.

Examples

```
## Not run:
url <- paste0("https://www12.statcan.gc.ca/census-recensement/2021/",
              "geo/sip-pis/rnf-frr/files-fichiers/lrnf000r21a_e.zip")
fetch(url, "lrnf000r21a_e.zip")

## End(Not run)
```

`is_block` *Is a response a block page that a browser would not be shown?*

Description

Cloudflare's firewall can refuse a client that does not look like a browser outright, with a `403` and its own block page, while a real browser is served.

Usage

```
is_block(status, headers, body = NULL)
```

Arguments

`status`	The HTTP status code.
`headers`	The response headers, as the raw header block (one string, lines separated by `\r\n`) or as a named list such as `curl::parse_headers_list()` makes of it.
`body`	The response body, as a string.

Value

`TRUE` for that page: a `403` from `server: cloudflare` whose body says "Sorry, you have been blocked".

Examples

```
is_block(403, "HTTP/2 403\r\nserver: cloudflare\r\n\r\n",
         "<html><title>Attention Required! | Cloudflare</title>...")
is_block(403, "HTTP/2 403\r\nserver: nginx\r\n\r\n", "Forbidden")
```

`is_challenge` *Does a response carry a browser check instead of the resource?*

Description

Does a response carry a browser check instead of the resource?

Usage

```
is_challenge(headers)
```

Arguments

`headers`	The response headers, as the raw header block (one string, lines separated by `\r\n`) or as a named list such as `curl::parse_headers_list()` makes of it.

Value

`TRUE` for a check a real browser would pass: Cloudflare's, by its `cf-mitigated: challenge` header.

See Also

`is_block()` for the other refusal a browser gets past.

Examples

```
is_challenge("HTTP/2 403\r\ncf-mitigated: challenge\r\n\r\n")
is_challenge("HTTP/2 200\r\ncontent-type: application/zip\r\n\r\n")
```

`page_text` *Parse a response from read_page()*

Description

`page_text()` decodes the body to a UTF-8 string, `page_json()` parses it with `jsonlite::fromJSON()`, and `page_html()` with `xml2::read_html()`.

Usage

```
page_text(resp, encoding = NULL, check = TRUE)

page_json(resp, ..., check = TRUE)

page_html(resp, encoding = NULL, check = TRUE)
```

Arguments

resp	A response from read_page(), or a URL, which is then read with read_page()'s defaults, so that page_html(url) stands in for rvest::read_html(url).
encoding	The body's charset, overriding the response's.
check	Whether an HTTP status outside 200–299 is an error.
...	Passed on to jsonlite::fromJSON().

Details

The charset comes from the response's content-type; without one, page_text() and page_json() assume UTF-8 and page_html() leaves it to the page's own <meta charset>. ISO-8859-1 is read as windows-1252, as browsers do. A body that is not valid in its charset is an error. A "not found" page served with status 200 passes the status check; page_json() refuses an HTML body, and for a page, compare resp$url with the URL asked for.

Value

page_text(): a UTF-8 string. page_json(): what jsonlite::fromJSON() returns. page_html(): an xml_document whose relative links resolve against resp$url. Errors are classed under safeconduct_error: safeconduct_http_error for a status outside 200–299, carrying status and url, and safeconduct_parse_error for a body that cannot be decoded, or HTML where JSON was asked for.

Examples

```
## Not run:
page_json("https://results.vancouver.ca/data/arearesults.json")
page_html("https://www12.statcan.gc.ca/census-recensement/index-eng.cfm")

## End(Not run)
```

---

probe *Ask a server what it serves at a URL*

Description

Sends a GET for the first byte, so that the answer carries the resource's headers without the resource; a HEAD is not used because some servers answer it differently.

Usage

```
probe(url, clearance = NULL)
```

Arguments

url	The URL.
clearance	A clearance from clearance(), to probe as the browser that earned it; NULL probes as a plain client.

Value

A list: reachable, status (NA when unreachable), challenged (is_challenge()), blocked (is_block()), type (the content-type, lower case), size (bytes, from content-range or content-length, NA when unknown) and headers (a named list). A browser is called for when challenged or blocked is TRUE.

Examples

```
## Not run:
probe("https://www12.statcan.gc.ca/census-recensement/index-eng.cfm")

## End(Not run)
```

---

read_page *Read a page or a feed into memory, through a browser if the server insists*

Description

fetch() for a page or a feed: the response is returned rather than written to disk, shaped like curl::curl_fetch_memory()'s, by the same route; where only the browser is served, the page the browser holds on the host reads it. For a feed read every few minutes, pass a browser_session() instead of opening a browser per call; a check the host asks for again meanwhile is passed and the read repeated.

Usage

```
read_page(
  url,
  session = NULL,
  quiet = FALSE,
  timeout = 120,
  headed = "auto",
  useragent = TRUE
)
```

Arguments

`url`	The file's URL.
`session`	A `browser_session()` on the host. The page is then read in that browser without probing.
`quiet`	Suppress progress messages.
`timeout`	Seconds a read in the browser may take.
`headed`	How to run the browser when one is needed. `"auto"`, the default, runs headless and opens a window only when the headless run ends on a check or a block and there is a display to show it on; `TRUE` opens a window at once; `FALSE` never does. A second check, at the file's URL, takes headless Chrome about two minutes to pass unclicked, where a window is through as soon as its box is clicked: `"auto"` waits those minutes before opening a window, `TRUE` skips the wait.
`useragent`	The browser's user-agent string. `TRUE`, the default, drops the word `Headless` from Chrome's own string, which some hosts refuse on that word alone, and changes nothing else; `NULL` leaves Chrome's own; a string is used as given.

Value

A list: `url` (after redirects), `status_code`, `type` (the `content-type`, lower case), `headers` (a named list) and `content` (raw). An HTTP error status is returned, not raised. Parse it with `page_text()`, `page_json()` or `page_html()`. Errors as `fetch()`'s.

Examples

```
## Not run:
r <- read_page("https://results.vancouver.ca/data/arearesults.json")
page_json(r)

## End(Not run)
```

Index

browser_available, 2
browser_available(), *8*
browser_session, 3
browser_session(), *7, 8*, *11, 12*

chromote::find_chrome(), *8*
clearance, 4
clearance(), *3*, *5*, *11*
clearance_handle, 5
clearance_handle(), *4*
close(), *3*
curl::curl_fetch_memory(), *11*
curl::new_handle(), *5*
curl::parse_headers_list(), *8, 9*

download_in_browser, 5

fetch, 6
fetch(), *3–6*, *11, 12*

is_block, 8
is_block(), *6*, *9*, *11*
is_challenge, 9
is_challenge(), *6*, *11*

jsonlite::fromJSON(), *9, 10*

page_html *(*page_text*)*, 9
page_html(), *12*
page_json *(*page_text*)*, 9
page_json(), *12*
page_text, 9
page_text(), *12*
probe, 10

read_page, 11
read_page(), *3*, *10*

safeconduct-package, *8*

tempdir(), *7*

xml2::read_html(), *9*