Package: shelter (via r-universe)

November 8, 2024

Type Package

Title Support for Secure API Key Management

Version 0.1.1

Maintainer Shawn Garbett <Shawn.Garbett@vumc.org>

Description Secure handling of API keys can be difficult. This package provides secure convenience functions for entering / handling API keys and opening connections via inversion of control on those keys. Works seamlessly between production and developer environments.

Depends R (>= 4.1.0)

License GPL-3

Encoding UTF-8

Imports checkmate, getPass, yaml, filelock, rappdirs, sodium

Suggests testthat (>= 3.0.0), rstudioapi, mockery

URL https://github.com/vubiostat/shelter

RoxygenNote 7.3.2

BugReports https://github.com/vubiostat/shelter/issues

Config/pak/sysreqs libsodium-dev

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

RemoteUrl https://github.com/vubiostat/shelter

RemoteRef HEAD

RemoteSha 716885de8188c14b64a61bcde7f15c881ca84050

Contents

`keyring_create` *Create a new empty keyring.*

Description

Create a new empty keyring with of a given name with the specified password.

Usage

```
keyring_create(keyring, password)
```

Arguments

`keyring`	character(1); Name of keyring
`password`	character(1); Password for keyring

Value

logical(1); Success or failure of operation

Examples

```
## Not run:
keyring_create('mypersonalkeyring', '<PASSWORD>')

## End(Not run)
```

keyring_delete *Delete a given keyring*

Description

Given the name of a keyring, delete it and remove all cached information.

Usage

```
keyring_delete(keyring)
```

Arguments

keyring	character(1); Name of keyring

Value

logical(1); Success or failure of operation

Examples

```
## Not run:
keyring_delete('mypersonalkeyring')

## End(Not run)
```

keyring_exists *Check if a keyring exists.*

Description

Given a keyring name will check if the keyring file exists.

Usage

```
keyring_exists(keyring)
```

Arguments

keyring	character(1); Name of the keyring.

Value

logical(1); Keyring file store existence status.

keyring_list *Provides a 'data.frame' of information on available keyrings.*

Description

Looks in a local directory where keyrings are stored for the current user and returns information about keyrings found. Keyrings are stored in 'rappdirs::user_config_dir("r-shelter")' and end in '.keyring.RDS'

Usage

```
keyring_list()
```

Value

data.frame of (keyring, secrets, locked)

Examples

```
keyring_list()
```

keyring_lock *Locks a given keyring*

Description

Given the name of a keyring lock it.

Usage

```
keyring_lock(keyring)
```

Arguments

keyring	character(1); Name of keyring

Value

logical(1); Success or failure of operation

Examples

```
## Not run:
keyring_lock('mypersonalkeyring')

## End(Not run)
```

keyring_locked *Is a keyring unlocked for key operations and reading*

Description

Query if a keyring is unlocked

Usage

```
keyring_locked(keyring)
```

Arguments

keyring	character(1); Name of keyring

Value

logical(1); Success or failure of operation

Examples

```
## Not run:
keyring_locked('mypersonalkeyring')

## End(Not run)
```

keyring_unlock *Unlock a keyring.*

Description

Unlock a given keyring using the specified password. Secrets exist in plain text in memory while a keyring is unlocked.

Usage

```
keyring_unlock(keyring, password)
```

Arguments

keyring	character(1); Name of keyring
password	character(1); Password for keyring

Value

logical(1); Success or failure of operation

Examples

```
## Not run: keyring_unlock('mypersonalkeyring', '<PASSWORD>')
```

`key_delete` *Delete a key from a keyring*

Description

Delete a key from an unlocked keyring.

Usage

```
key_delete(keyring, key)
```

Arguments

`keyring`	character(1); Name of keyring
`key`	character(1); Name of key

Value

logical(1); Success of operation

Examples

```
## Not run:
key_delete('mypersonalkeyring', 'key1')

## End(Not run)
```

`key_exists` *Does a given key exist in a keyring*

Description

In an unlocked keyring return if a key exists.

Usage

```
key_exists(keyring, key)
```

Arguments

`keyring`	character(1); Name of keyring
`key`	character(1); Name of key

Value

logical(1); Existance of key in keyring

Examples

```
## Not run:
key_exists('mypersonalkeyring', 'key1')

## End(Not run)
```

`key_get` *Get a secret from a keyring.*

Description

Get a secret from an unlocked keyring given it's key.

Usage

```
key_get(keyring, key)
```

Arguments

`keyring`	character(1); Name of keyring
`key`	character(1); Name of key

Value

character(1); The requested secret

Examples

```
## Not run:
key_get('mypersonalkeyring', 'key1')

## End(Not run)
```

key_list *Returns vector of keys in a keyring.*

Description

Return vector key names in a keyring that is unlocked.

Usage

```
key_list(keyring)
```

Arguments

keyring	character(1); Name of keyring

Value

character; Key names

Examples

```
## Not run:
key_list('mypersonalkeyring')

## End(Not run)
```

key_set *Set a key secret in a keyring*

Description

Sets a key secret in a keyring

Usage

```
key_set(keyring, key, secret)
```

Arguments

keyring	character(1); Name of keyring
key	character(1); Name of key to store in keyring
secret	character(1); The secret to store in keyring

Value

logical(1); Status of operation

Examples

```
## Not run:
key_set('mypersonalkeyring','key1','a secret')

## End(Not run)
```

---

unlockKeys *Open an API key and use it build a connection.*

Description

Opens a set of connections from API keys stored in an encrypted keyring. If the keyring does not exist, it will ask for password to this keyring to use on later requests. Next it will ask for the API keyss specified in 'connections'. If an API key does not work, it will request again. On later executions it will use an open keyring to retrieve all API_KEYs or for a password if the keyring is currently locked.

Usage

```
unlockKeys(
  connections,
  keyring,
  connectFUN = NULL,
  envir = NULL,
  passwordFUN = .default_pass(),
  service = "shelter",
  ...
)
```

Arguments

connections	character vector. A list of strings that define the connections with associated API_KEYs to load into environment. Each name should correspond to a REDCap project for traceability, but it can be named anything one desires. The name in the returned list is this name.
keyring	character. Potential keyring, not used by default.
connectFUN	function or list(function). A function that takes a key and returns a connection. the function should call 'stop' if the key is invalid in some manner. The first argument of the function is the API key. The validation of the key via a connection test is important for the full user interaction algorithm to work properly. If one wished to just retrieve an API key and not test the connection this would work 'function(x, ...) x', but be aware that if the key is invalid it will not query the user as the validity is not tested.
envir	environment. The target environment for the connections. Defaults to NULL which returns the keys as a list. Use [globalenv()] to assign in the global environment. Will accept a number such a '1' for global as well.

passwordFUN	function. Function to get the password for the keyring. Usually defaults 'getPass::getPass'. On MacOS it will use rstudioapi::askForPassword if available.
service	character(1). The keyring service. Defaults to package name.
...	Additional arguments passed to 'connectFUN()'.

Details

If one forgets the password to this keyring, or wishes to start over: 'keyring_delete("<NAME_OF_KEY_RING_HERE>")'

For production servers where the password must be stored in a readable plain text file, it will search for '../<basename>.yml'. DO NOT USE this unless one is a sysadmin on a production hardened system, as this defeats the security and purpose of a local encrypted file. The expected structure of this yaml file is as follows:

```
other-config-stuff1: blah blah
shelter:
  keys:
    intake: THIS_IS_THE_INTAKE_DATABASE_APIKEY
    details: THIS_IS_THE_DETAILS_DATABASE_APIKEY
other-config-stuff2: blah blah
other-config-stuff3: blah blah
```

For production servers the use of ENV variables is also supported. The connection string is converted to upper case for the search of ENV. If a YAML file and ENV definitions both exist, the YAML will take precedence.

IMPORTANT: Make sure that R is set to NEVER save workspace to .RData as this *is* writing the API_KEY to a local file in clear text because connection objects contain the unlocked key in memory. One can use the following in .Rprofile, 'usethis::edit_r_profile()':

```
newfun <- function (save = "no", status = 0, runLast = TRUE)
  .Internal(quit(save, status, runLast))
pkg <- 'base'
oldfun <- 'q'
pkgenv <- as.environment(paste0("package:", pkg))
unlockBinding(oldfun, pkgenv)
utils::assignInNamespace(oldfun, newfun, ns = pkg, envir = pkgenv)
assign(oldfun, newfun, pkgenv)
lockBinding(oldfun, pkgenv)
```

Value

If 'envir' is NULL returns a list of opened connections. Otherwise connections are assigned into the specified 'envir'.

Examples

```
## Not run:
unlockKeys(c(test_conn    = 'Testshelter',
             sandbox_conn = 'SandboxAPI'),
```

```
                keyring      = '<NAME_OF_KEY_RING_HERE>',
                envir        = globalenv(),
                passwordFUN  = function(x, ...) x)

    ## End(Not run)
```

Index