diceware Documentation

Release 0.9.3

Uli Fouquet

Sep 14, 2017

Contents

Version: 0.9.3

CHAPTER 1

diceware

Passphrases to remember...

| documentation | sources | issues

diceware is a passphrase generator following the proposals of Arnold G. Reinhold on http://diceware.com . It generates passphrases by concatenating words randomly picked from wordlists. For instance:

```
$ diceware
MyraPend93rdSixthEagleAid
```

The passphrase contains by default six words (with first char capitalized) without any separator chars. Optionally you can let *diceware* insert special chars into the passphrase.

diceware supports several sources of randomness (including real life dice) and different wordlists (including cryptographically signed ones).

Contents

Install

This Python package can be installed via pip:

```
$ pip install diceware
```

The exact way depends on your operating system.

Usage

Once installed, use `--help` to list all available options:

```
$ diceware --help
Create a passphrase

positional arguments:
  INFILE                Input wordlist. `-' will read from stdin.

optional arguments:
  -h, --help            show this help message and exit
  -n NUM, --num NUM     number of words to concatenate. Default: 6
  -c, --caps            Capitalize words. This is the default.
  --no-caps             Turn off capitalization.
  -s NUM, --specials NUM
                        Insert NUM special chars into generated word.
  -d DELIMITER, --delimiter DELIMITER
                        Separate words by DELIMITER. Empty string by default.
  -r SOURCE, --randomsource SOURCE
                        Get randomness from this source. Possible values:
                        `realdice', `system'. Default: system
  -w NAME, --wordlist NAME
                        Use words from this wordlist. Possible values: `en',
                        `en_eff', `en_orig', `en_securedrop'. Wordlists are
                        stored in the folder displayed below. Default: en_eff
  -v, --verbose         Be verbose. Use several times for increased verbosity.
  --version             output version information and exit.

Arguments related to `realdice' randomsource:
  --dice-sides N        Number of sides of dice. Default: 6

Wordlists are stored in <WORDLISTS-DIR>
```

With `-n` you can tell how many words are supposed to be picked for your new passphrase:

```
$ diceware -n 1
Thud
```

```
$ diceware -n 2
KnitMargo
```

You can *diceware* additionally let generate special chars to replace characters in the 'normal' passphrase. The number of special chars generated can be determined with the `-s` option (*default is zero*):

```
$ diceware -s 2
Heroic%unkLon#DmLewJohns
```

Here `"%"` and `"#"` are the special chars.

Special chars are taken from the following list:

```
~!#$%^&*()-=+[]\{}:;\"'<>?/0123456789
```

Please note that several special chars might replace the same original char, resulting in a passphrase with less special chars than requested.

With `-d` you can advise *diceware* to put a delimiter string between the words generated:

```
$ diceware -d "_"
Wavy_Baden_400_Whelp_Quest_Macon
```

By default we use the empty string as delimiter, which is good for copying via double click on Linux systems. But other delimiters might make your passphrases more readable (and more secure, see *Security Traps* below).

By default the single phrase words are capitalized, i.e. the first char of each word is made uppercase. This does not neccessarily give better entropy (but protects against entropy loss due to non prefix code, see *Security Traps* below), and it might improve phrase readability.

You can nevertheless disable caps with the `--no-caps` option:

```
$ diceware --no-caps
oceanblendbaronferrylistenvalet
```

This will leave the input words untouched (upper-case stays upper-case, lower-case stays lower-case). It does *not* mean, that all output words will be lower-case (except if all words of your wordlist are lowercase).

As the default lists of *diceware* contain only lower-case terms, here `--no-caps` means in fact lower-case only output, which might be easier to type on smart phones and similar.

diceware supports also different sources of randomness, which can be chosen with the `-r <SOURCENAME>` or `--randomsource <SOURCENAME>` option. Use the `--help` option to list all valid values for this option.

By default we use the random.SystemRandom class of standard Python lib but you can also bring your own dice to create randomness:

```
$ diceware -r realdice --dice-sides 6
Please roll 5 dice (or a single dice 5 times).
What number shows dice number 1? 2
What number shows dice number 2? 3
...
DogmaAnyShrikeSageSableHoar
```

Normally dice have six sides. And this is also the default in *diceware* if you do not use `--dice-sides`. But if you do, you can tell how many sides (all) your dice have. More sides will lead to less rolls required.

We support even sources of randomness from other packages. See the documentation for more details.

diceware comes with an English wordlist provided by the EFF, which will be used by default and contains 7776 (=6^5) different words. This list is registered as `en_eff`.

Additionally *diceware* comes with an English wordlist provided by @heartsucker, which contains 8192 different words. This list is based off the original diceware list written by Arnold G. Reinhold.

Both the original and 8k diceware wordlists by Mr. Reinhold are provided. You can enable a certain (installed) wordlist with the `-w` option:

```
$ diceware --wordlist en_orig
YorkNodePrickEchoToriNiobe
```

See `diceware --help` for a list of all installed wordlists.

If you do not like the wordlists provided, you can use your own one. Any *INFILE* provided will be parsed line by line and each line considered a possible word. For instance:

```
$ echo -e "hi\nhello\n" > mywordlist.txt
$ diceware mywordlist.txt
HelloHelloHiHiHiHello
```

With dash (-) as filename you can pipe in wordlists:

```
$ echo -e "hi\nhello\n" > mywordlist.txt
$ cat mywordlist.txt | diceware -
HiHiHelloHiHiHello
```

In custom wordlists we take each line for a valid word and ignore empty lines (i.e. lines containing whitespace characters only). Oh, and we handle even PGP-signed wordlists.

You can set customized default values in a configuration file `.diceware.ini` (note the leading dot) placed in your home directory. This file could look like this:

```
[diceware]
num = 7
caps = off
specials = 2
delimiter = "MYDELIMITER"
randomsource = "system"
wordlist = "en_securedrop"
```

The options names have to match long argument names, as output by `--help`. The values set must meet the requirements valid for commandline usage. All options must be set within a section `[diceware]`.

What is it good for?

Normally, *diceware* passphrases are easier to remember than shorter passwords constructed in more or less bizarre ways. But at the same time *diceware* passphrases provide more entropy as xkcd can show with the famous '936' proof: The standard english wordlist of this *diceware* implementation contains 7776 = 6^5 different english words. It is the official EFF wordlist. compiled by Joseph Bonneau. Therefore, picking a random word from this list gives an entropy of nearly 12.9 bits. Picking six words means an entropy of 6 x 12.9 = 77.54 bits.

The special chars replacing chars of the originally created passphrase give some more entropy (the more chars you have, the more additional entropy), but not much. For instance, for a sixteen chars phrase you have sixteen possibilities to place one of the 36 special chars. That makes 36 x 16 possibilitities or an entropy of about 9.17 you can add. To get an entropy increase of at least 10 bits, you have to put a special char in a phrase with at least 29 chars (while at the

same time an additional word would give you 13 bits of extra entropy). Therefore you might think again about using special chars in your passphrase.

Is it secure?

The security level provided by Diceware depends heavily on your source of random. If the delivered randomness is good, then your passphrases will be very strong. If instead someone can foresee the numbers generated by a random number generator, your passphrases will be surprisingly weak.

This Python implementation uses (by default) the random.SystemRandom source provided by Python. On Un*x systems it accesses */dev/urandom*. You might want to follow reports about manipulated random number generators in operating systems closely.

The Python API of this package allows usage of other sources of randomness when generating passphrases. This includes real dice. See the `-r` option.

Security Traps

There are issues that might reduce the entropy of the passphrase generated. One of them is the prefix code problem:

Prefix Code

If the wordlist contains, for example, the words:

```
"air", "airport", "portable", "able"
```

and we switched off caps *and* delimiter chars, then *diceware* might generate a passphrase containing:

```
"airportable"
```

which could come from `air-portable` or `airport-able`. We cannot tell and an attacker would have less combinations to guess.

To avoid that, you can leave caps enabled (the default), use any word delimiter except the empty string or use the `en_eff` wordlist, which was checked to be a prefix code (i.e. it does not contain words that start with other words in the list).

Each of these measures is sufficient to protect you against the prefix code problem.

Reduced Entropy

Overall, *diceware* is a kind of mapping input values, dice throws for instance, onto wordlist entries. We normally want each of the words in the wordlist to be picked for passphrases with the same probability.

This, however, is not possible, if the number of wordlist entries is not a power of dice sides. In that case we cut some words of the wordlist and inform the user about the matter. Reducing the number of words this way makes it easier for attackers to guess the phrase picked.

You can fix that problem by using longer wordlists.

Developer Install

Developers want to fork me on github:

```
$ git clone https://github.com/ulif/diceware.git
```

We recommend to create and activate a virtualenv first:

```
$ cd diceware/
$ virtualenv -p /usr/bin/python3.4 py34
$ source py34/bin/activate
(py34) $
```

We support Python versions 2.6, 2.7, 3.3 to 3.6, and pypy.

Now you can create the devel environment:

```
(py34) $ python setup.py dev
```

This will fetch test packages (py.test). You should be able to run tests now:

```
(py34) $ py.test
```

If you have also different Python versions installed you can use tox for using them all for testing:

```
(py34) $ pip install tox   # only once
(py34) $ tox
```

Should run tests in all supported Python versions.

Documentation Install

The docs can be generated with Sphinx. The needed packages are installed via:

```
(py34) $ python setup.py docs
```

To create HTML you have to go to the `docs/` directory and use the prepared `Makefile`:

```
(py34) $ cd docs/
(py34) $ make
```

This should generate the docs in `docs/_build/html/`.

Creating the Man Page

We provide a ReStructuredTexT template to create a man page. When the documentation engine is installed (Sphinx, see above), then you can create a manpage doing:

```
(py34) $ rst2man.py docs/manpage.rst > diceware.1
```

The template is mainly provided to ease the job of Debian maintainers. Currently, it is not automatically updated. Dates, authors, synopsis, etc. have to be updated manually. Information in the manpage may therefore be wrong, outdated, or simply misleading.

Credits

Arnold G. Reinhold deserves all merits for the working parts of Diceware. The non-working parts are certainly my fault.

People that helped spotting bugs, providing solutions, etc.:

- Conor Schaefer (conorsch)
- Rodolfo Gouveia suggested to activate the `--delimiter` option.
- @drebs provided patches and discussion for different sources of randomness. @drebs also initiated and performed the packaging of *diceware* for the Debian platform. Many kudos for this work! @drebs is also the official Debian maintainer of the *diceware* package.
- @heartsucker hand-compiled and added a new english wordlist.
- dwcoder revealed and fixed bugs #19, #21, #23. Also showed sound knowledge of (theoretical) entropy. A pleasure to work with.
- George V. Reilly pointed to new EFF wordlists.
- lieryan brought up the prefix code problem.
- LogosOfJ discovered and fixed serious *realdice* source of randomness problem.

Many thanks to all of them!

Links

- The Diceware home page. Reading definitely recommended!
- fork me on github

Wordlists:

_ Diceware standard list by Arnold G. Reinhold. - Diceware8k list by Arnold G. Reinhold. - Diceware SecureDrop list by @heartsucker. - EFF large list provided by EFF.

License

CHAPTER 2

Sources of Randomness

The security of your passphrase depends naturally heavily on the source of randomness you use. If the source is good, it is really hard to predict your passphrase. If it is bad, your passphrase might be surprisingly easy to guess. *diceware* does not provide own pseudo-random number generators or similar. Instead we let you choose yourself the source of randomness you trust.

diceware supports different sources of randomness, which can be chosen with the `-r <SOURCENAME>` or `--randomsource <SOURCENAME>` option.

Use the `--help` option to list all valid values for the `--randomsource` option.

Python-developers can provide their own source of randomness. If their package is installed together with *diceware* (and their source is registered correctly), *diceware* will offer their source as valid option.

System Random

By default *diceware* uses the Python standard lib `random.SystemRandom` class to retrieve randomness. This class calls an OS-specific source of randomness that returns data normally unpredictable enough for our purposes. The quality of randomness therefore depends on the quality of your OS implementation.

As a user you can enforce the use of this source of randomness with the `-r system` option.

Please note that the Raspberry Pi is said to provide a hardware random number generator that delivers "real randomness". One has to enable it system-wide to make it the active source of randomness on a Raspberry Pi. If done properly, also `randomSystemRandom` (and hence *diceware*) should use good quality random numbers.

Real Dice

diceware also supports real dice as source of randomness. You can pick this source of randomness with the `-r realdice` option.:

```
$ diceware -r realdice
Warning: entropy is reduced!
Please roll 5 dice (or a single dice 5 times).
What number shows dice number 1? 1
What number shows dice number 2? 2
What number shows dice number 3? 3
What number shows dice number 4? 4
What number shows dice number 5? 5
Warning: entropy is reduced!
Please roll 5 dice (or a single dice 5 times).
What number shows dice number 1? 2
What number shows dice number 2? 3
What number shows dice number 3? 3
What number shows dice number 4? 5
What number shows dice number 5? 1

...

What number shows dice number 5? 3
AnyDogmaShrikeSageSableHoar
```

If you see a warning "entropy is reduced!", this means that not the whole range of the wordlist you use can be put to account. Instead we use (in case of 5 rolls) the first 6^5 words only. If you use a wordlist with 6^n elements (for instance the original list with 7776 elements of Mr. Rheinhold), you will not get this warning.

Currently we support only 6-sided dice.

Bring Your Own Source (for developers)

diceware uses Python entry-points for looking up sources of randomness. That means you can write your own source of randomness in Python, register it in your own package and once both, your package and *diceware* are installed together on a system, your source of randomness will be offered and used by *diceware* (if the user selects it).

To build your own source of randomness you have to provide a class with a constructor that accepts a single *options* object. Furthermore a source of randomness has to provide a *choice(sequence)* method. It comes down to something like that:

```
class MySourceOfRandomness(object):
    "Tell about your source..."
    def __init__(self, options):
        # initialize, etc.

    def choice(sequence):
        # return one of the elements in `sequence`
```

The *choice()* method will be called for each word of the passphrase and for each special char. Please do not make assumptions about the *sequence* passed to choice. It will be a list of "somethings" and be indexable.

If your source is ready, you can register it in the `setup.py` of your package like this:

```
# setup.py

...

setup(
```

```
    ...

    entry_points={
        'diceware_random_sources': [
          'mysrc = mypkg.sources:MySourceOfRandomness',
          # add more sources of randomness here...
        ],
    }
)
```

Here we assume that you defined *MySourceOfRandomness* in a package *mypkg* and a module called *sources*.

Once this package is installed, you can run *diceware* like this:

```
$ diceware -r mysrc
```

and your source of randomness will be used.

CHAPTER 3

Configuration Files

You can use configuration files to persistently override built-in defaults and make your custom settings the default.

diceware configuration files follow simple `.ini`-style and look like this:

```
[diceware]
num = 3
caps = off
specials = 2
delimiter = "MYDELIMITER"
randomsource = system
wordlist = "en"
dice_sides = 6
```

These settings would mean that by default phrases with three words (instead six) would be created. Commandline options, however, override config file settings. So, with the settings above:

```
$ diceware
Duma7YDELIMITER56MYDE^IMITERJock
```

we will get three-word phrases while with:

```
$ diceware --delimiter=FOO
AmuseFOO]us(FOO18th
```

we will override the config file setting for `delimiter`. Other settings from config file are still valid.

Option Names

The options names have to match long argument names, as output with `--help`. The values set must meet the requirements valid for commandline usage.

You can use all or only some (or none) of the above options. Please note that other entries, providing unknown option names, are ignored. That means that also typos might lead to ignored entries.

Please note, that all options must be set within a section `[diceware]`.

Config File Name and Path

Currently, we look for configuration files only in the calling users' home directory. The file must be called:

```
.diceware.ini
```

(please note the leading dot). If such a file is missing, build-in defaults apply.

Option Values

The option values set can be strings, integers, or boolean values.

diceware accepts `yes`, `no`, `1`, `0`, `true`, `false`, `on`, and `off` as boolean values.

Some options require their setting to be taken from a fixed set of names/values, for instance the `randomsource` option. You can normally get the allowed values from calling `diceware --help`.

String-based options (like *delimiter*) accept values enclosed in quotes to allow whitespace-only values.

If some value cannot be parsed, an exception is raised.

CHAPTER 4

Wordlists

The passphrases generated by *diceware* naturally depend on the set of words used, the wordlists.

diceware comes with some wordlists out-of-the-box, that might be a good choice for usual private use.

> **Warning:** We do – by default – *not* use the diceware standard wordlist, but the long EFF wordlist (see below), because it is more secure and more comfortable to use.
>
> But the "original" list is included in diceware as well and you can pick it with the `-w en_orig` option. You *should* pick it when you use real dice as source of randomness.

Currently we provide the following lists:

- *en_securedrop* (8192 words)

  We provide a hand-crafted *en_securedrop* wordlist provided by @Heartsucker. It contains 8,192 english words and phrases. This list is based on the diceware standard wordlist and extended to offer better memorizable words. Please see https://github.com/heartsucker/diceware for details. The name *en_securedrop* refers to the securedrop project.

- *en* (8192 words)

  Apart from it we also provide the so-called 8k wordlist from Mr. Reinhold as published on http://diceware.com/. It also contains 8,192 english words and phrases and is something like the canonical wordlist for use with binary-geared entities like computers or nerds.

- *en_eff* (7776 words, default)

  This is the long EFF wordlist as published by the Electronic Frontier Foundation in mid-2016 and used by default. They put real scientific effort into the creation of this list which might considerably ease the use of passphrases generated with it. When using real dice (or other six-based randomness generators) use is definitely recommended!

  Please note, that this is currently the only list, that provides the prefix property. That means it contains no word which is a prefix of another word. Lists without this property might provide a slightly decreased entropy.

- *en_orig* (7776 words)

  This is the diceware standard wordlist as provided by Mr. Reinhold. Something like the canonical list in former times, there are now considerable alternatives.

You can pick another list with the `-w` or `--wordlist` option.

Add Own Wordlists

You can use any wordlist you like. Simply give the filename and it will be used:

```
$ diceware mywordlist.txt
HiHelloHelloHiHiHi
```

You can even pipe-in dynamic wordlists. Just use the dash – as filename:

```
$ cat mywordgenerator.sh | diceware -
HiHiHelloHiHiHello
```

for instance.

Of course you have to give the filenames of your files with each call to *diceware*.

But, if you want to store a wordlist persistently, you can do so too.

The wordlists we offer for use with *diceware* are all stored in a single folder. The exact location is output by `--help` at the very end:

```
$ diceware --help
...
Wordlists are stored in /some/path/to/folder
```

Just put your own wordlists into this folder (here: `/some/path/to/folder`) and rename the file to something like `wordlist_MY_SPECIAL_NAME.txt`. Afterwards you can pick your wordlist by running:

```
$ diceware -w MY_SPECIAL_NAME
```

diceware will use this file of yours then to create a passphrase. Please note that *diceware* only accepts files that are named like:

```
wordlist_NAME.txt
```

or:

```
wordlist_OTHER_NAME.asc
```

I.e. we expect `wordlist_` at the beginning and some filename extension like `.txt` at the end. Furthermore names must not contain funny characters. In fact we accept regular letters, dashes, numbers, and underscores only. Files that do not follow these naming convention are ignored.

A list of all available wordlist names can also be retrieved with `--help`. See the `--wordlist` explanation.

Plain Wordlists

Out of the box, *diceware* supports plain wordlists, PGP-signed wordlists, and numbered wordlists. Plain wordlists look like this:

```
termone
termtwo
anotherterm
```

Each line in such a file is considered a word of the wordlist. Empty lines are ignored.

Whitespaces are allowed if they are not at the beginning or end of a line, stripped off otherwise.

Numbered Wordlists

Numbered wordlists contain numbers in each line, telling a sequence of dice rolls like so:

```
11111    aterm
11112    anotherterm
...
```

diceware detects such lines and in this case extracts `aterm` and `anotherterm` as wordlist entries.

Apart from simple digits written next to each other, *diceware* also accepts numbers separated by dashes like this:

```
1-1-1-1-1   aterm
1-1-1-1-2   anotherterm
```

which is handy when working with wordlists for dice with more than 9 sides.

PGP-signed Wordlists

PGP-signed wordlists are wordlists (ordinary or numbered ones), that have been cryptographically signed with PGP or GPG. They look like this:

```
-----BEGIN PGP SIGNED MESSAGE-----
Hash: SHA512

foo
bar
baz

-----BEGIN PGP SIGNATURE-----
Version: GnuPG v1

iJwEAQEKAAYFAlW00GEACgkQ+5ktCoLaPzSutwP8DVgdjBFqRXNKaZlvd8pR+P3k
8xx5XLC0OFwZQFx4Ls8xl3+/xfvCNxCGSZjD6BGPzNZCK7bmQQYWcrsoEyX5jAC3
dXjAPj0nct/PkJQlrUjUI2qrO0dFfU7sRj0Gn9TOlQQkKoQVwy7pY/6HaScGNepL
J8BNUPYdOWeVgxY1jSY=
=WXfu
-----END PGP SIGNATURE-----
```

and are normally stored with the `.asc` filename extension. Signed wordlists can be verified to detect changes, although this is not automatically done by *diceware*.

Warning: Diceware does *not* automatically verify PGP-signed files.

CHAPTER 5

API

diceware code is geared towards commandline usage. You can, however, use it from Python. The API docs are here to assist you with that.

For using *diceware* in your own, *setuptools*-based Python project, you can add it as an install requirement in `setup.py` of your project:

```
from setuptools import setup
# ...
setup(
    name="myproject",
    # ...
    install_requires=[
        #  packages we depend on...
        'setuptools',
        'diceware',
        # ...
    ],
    # ...
)
```

Of course there are other ways to make *diceware* available.

diceware main module

diceware – rememberable passphrases

diceware.**SPECIAL_CHARS = '~!#$%^&*()-=+[]\\{};:'"\'<>?/0123456789'**
Special chars inserted on demand

diceware.**get_passphrase**(*options=None*)
Get a diceware passphrase.

options is a set of arguments as provided by *argparse.OptionParser.parse_args()*.

The passphrase returned will contain *options.num* words deliimted by *options.delimiter* and *options.specials* special chars.

For the passphrase generation we will use the random source registered under the name *options.randomsource* (something like "system" or "dice").

If *options.caps* is `True`, all words will be caps.

If *options.infile*, a file descriptor, is given, it will be used instead of a 'built-in' wordlist. *options.infile* must be open for reading.

`diceware.`**`get_random_sources`**`()`
Get a dictionary of all entry points called diceware_random_source.

Returns a dictionary with names mapped to callables registered as *entry_point's for the ''diceware_randomsource'* group.

Callables should accept *options* when called and return something that provides a *choice(sequence)* method that works like the respective method in the standard Python lib *random* module.

`diceware.`**`handle_options`**`(`*args*`)`
Handle commandline options.

`diceware.`**`insert_special_char`**`(`*word*, *specials='~!#$%^&*()-=+[]\\{};:"\'<>?/0123456789'*, *rnd=None*`)`
Insert a char out of *specials* into *word*.

rnd, if passed in, will be used as a (pseudo) random number generator. We use *.choice()* only.

Returns the modified word.

`diceware.`**`main`**`(`*args=None*`)`
Main programme.

Called when *diceware* script is called.

args is a list of command line arguments to process. If no such args are given, we use *sys.argv*.

`diceware.`**`print_version`**`()`
Output current version and other infos.

diceware.logger

logging – output status and other data.

The *logger* provided in this module is meant to be used by other components for messages to users.

It is named *"ulif.openoffice"* and can, as a singleton, be retrieved by calling standard lib *logging.getLogger("ulif.diceware")*.

By default it provides a *logging.NullHandler* as libraries normally do. Other components might add other handlers.

`diceware.logger.`**`configure`**`(`*verbosity=None*`)`
Configure global diceware logger.

verbosity sets the diceware logger verbosity. `0` enables info mode, while all numbers > 2 enable debug mode.

If no *verbosity* is given, we leave the logging level untouched.

`diceware.logger.`**`logger = <logging.Logger object>`**
Logger that can be used for all diceware related messages.

diceware.config

config – diceware configuration

diceware is configurable via commandline, configuration files and direct API calls.

diceware.config.**get_config_dict**(*path_list=None*, *defaults_dict={'wordlist': 'en_eff', 'verbose': 0, 'dice_sides': 6, 'randomsource': 'system', 'caps': True, 'specials': 0, 'delimiter': '', 'num': 6}*, *section='diceware'*)

Get config values found in files from *path_list*.

Read files in *path_list* config files and return option values from section *section* as regular dictonary.

We only accept values for which a default exists in *defaults_dict*. If *defaults_dict* is `None` we use `OPTIONS_DEFAULTS`.

Values are interpolated to have same value type as same-named values from *defaults_dict* if they are integers or boolean.

String/text values are stripped from preceding/trailing quotes (single and double).

diceware.config.**get_configparser**(*path_list=None*)

Parse *path_list* for config values.

If no list is given we use *valid_locations()*.

Return a list of paths read and a config parser instance.

diceware.config.**valid_locations**()

The list of valid paths we look up for config files.

diceware.wordlist

wordlist.py – special handling of wordlists.

diceware.wordlist.**MAX_IN_MEM_SIZE = 20971520**

Maximum in-memory file size in bytes (20 MB).

This value is used when creating temporary files replacing unseekable input streams. If an input file is larger, we write to disk.

diceware.wordlist.**RE_NUMBERED_WORDLIST_ENTRY = <_sre.SRE_Pattern object>**

A regular expression matching numbered entries in wordlists.

diceware.wordlist.**RE_VALID_WORDLIST_FILENAME = <_sre.SRE_Pattern object>**

A regular expression describing valid wordlist file names.

diceware.wordlist.**RE_WORDLIST_NAME = <_sre.SRE_Pattern object>**

A regular expression matching allowed wordlist names. We allow names that cannot easily mess up filesystems.

class diceware.wordlist.**WordList**(*path*)

A word list contains words for building passphrases.

path is the path of the wordlist file. With single dash (−) as path, we read from *sys.stdin*.

In case input comes from stdin, we write the input stream into a file if the content length is larger than *MAX_IN_MEM_SIZE*. Otherwise, the wordlist is kept in memory.

Wordlist files are expected to contain words, one word per line. Empty lines are ignored, also whitespaces before or trailing a line are stripped. If a "word" contains inner whitespaces, then these are preserved.

The input file can be a signed wordlist. Signed wordlists are expected to be ordinary lists of words but with ASCII armored signatures (as described in RFC 4880).

In case of signed wordlists the signature headers/footers are stripped and the contained list of words is read.

WordList are generators. That means, that you can retrieve the words of a wordlist by iterating over an instance of *WordList*.

is_signed()
check, whether this file is cryptographically signed.

This operation is expensive and resets the file descriptor to the beginning of file.

refine_entry(*entry*)
Apply modifications to form a proper wordlist entry.

Refining means: strip() *entry* remove escape-dashes (if this is a signed wordlist) and extract the term if it is preceded by numbers.

diceware.wordlist.**get_wordlist_names**()
Get a all names of wordlists stored locally.

diceware.wordlist.**get_wordlist_path**(*name*)
Get path to a wordlist file for a wordlist named *name*.

The *name* string must not contain special chars beside -, _, regular chars A-Z (upper or lower case) or numbers. Invalid names raise a ValueError.

If a path with the given name (names are not filenames here) does not exist, *None* is returned.

diceware.wordlist.**get_wordlists_dir**()
Get the directory in which wordlsts are stored.

diceware.random_sources

Sources of randomness.

Please register all sources as entry point in `setup.py`. Look out for "SystemRandomSource" for an example.

For developers of interfaces to other sources of randomness: Currently, you can extend *diceware* random sources by registering a class, that provides a suitable *__init__(self, options)* and a *choice(self, sequence)* method. Optionally, you can also provide a *classmethod* called `update_arparse` that will get the possibility to update the *argparser.ArgumentParser* used by *diceware*.

The *__init__* method of your class will be called with *options*, a set of options as parsed from the commandline. The initialization code can use the options to determine further actions or ignore it. The *__init__* method is also the right place to ask users for one-time infos you need. This includes infos like the number of sides of a dice, an API key for random.org or other infos that should not change between generating different words (but might change from one *diceware* call to the next).

The *choice* method then, will get a sequence of chars, strings, or numbers and should pick one of them based on the source of randomness intended to be utilized by your code. If further user interaction is required, *choice* might also ask users for input or similar. Typically, *choice* is called once for each word and once for each special char to generate.

If you want to manage own commandline options with your plugin, you can implement a *classmethod* called `update_argparser(parser)` which gets an *argparse.ArgumentParser* instance as argument (no pun intended).

Finally, to register the source, add some stanza in *setup.py* of your project that looks like:

```
# ...
setup(
    # ...
    entry_points={
        # console scripts and other entry points...
        'diceware_random_sources': [
            'myrandom = mypkg.mymodule:MyRandomSource',
            'myothersrc = mypkg.mymodule:MyOtherSource',
        ],
    },
    # ...
)
# ...
```

Here the *myrandom* and *myothersrc* lines register random sources that (if installed) *diceware* will find on startup and offer to users under the name given. In the described case, users could do for instance:

```
diceware -r myrandom
```

and the random source defined in the given class would be used for generating a passphrase.

class diceware.random_sources.**RealDiceRandomSource**(*options*)
A source of randomness working with real dice.

choice(*sequence*)
Pick one item out of *sequence*.

get_num_rolls(*seq_len*)
Compute how many dice rolls we need to pick a value from a sequence

pre_check(*num_rolls*, *sequence*)
Checks performed before picking an item of a sequence.

We make sure that *num_rolls*, the number of rolls, is in an acceptable range and issue an hint about the procedure.

class diceware.random_sources.**SystemRandomSource**(*options*)
A Random Source utilizing the standard Python *SystemRandom* call.

As time of writing, SystemRandom makes use of `/dev/urandom` to get fairly useable random numbers.

This source is registered as entry_point in setup.py under the name 'system' in the `diceware_random_sources` group.

The constructor will be called with options at beginning of a programme run if the user has chosen the respective source of random.

The SystemRandomSource is the default source.

choice(*sequence*)
Pick one item out of *sequence*.

The *sequence* will normally be a sequence of strings (wordlist), special chars, or numbers.

Sequences can be (at least) lists, tuples and other types that have a *len*. Generators do not have to be supported (and are in fact not supported by this source).

This method should return one item of the *sequence* picked based on the underlying source of randomness.

In the long run, the choice should return each *sequence* item (i.e.: no items should be 'unreachable').

It should also cope with any length > 0 of *sequence* and not break if a sequence is "too short" or "too long". Empty sequences, however, might raise exceptions.

CHAPTER 6

Changes

0.9.3 (2017-09-14)

- Fix broken test.

0.9.2 (2017-09-14)

- Fixed #33. Make *en_eff* the new default wordlist. This results in slightly decreased entropy per word (12.92 bits instead of 13.0), but provides prefix code and better memorizable words. Thanks to @anarcat for the suggestion.
- Fixed #35. Make *realdice* source of randomness provide an equal distribution of roll numbers even for sequences shorter than number of dice sides.
- Added a man page.
- Support Python 3.6.
- Import *ConfigParser* instead of *SafeConfigParser* if the latter is an alias of the former.
- Fixed #37. Ensure file descriptors are closed properly.
- Fixed #38. Get wordlists dir by function (instead of const) to allow reproducible builds. Kudos go to @drebs, again.

0.9.1 (2016-12-24)

- Fixed #32, in docs tell that `--no-caps` option does not generate lower-case terms.
- Fixed #31, broken *realdice* source of randomness. *argparse* related bug, Bug was discovered and fixed by @LogosOfJ, thanks a lot!
- Fixed #29. Tell about code prefix problem in README.

- Activated logging. Using *verbose* will result in additional output.

0.9 (2016-09-14)

- Added *–dice-sides* option to tell how many sides used dices provide.
- Changed API interface of *get_config_dict()* to allow more flexible handling of config files.
- Support different verbosity levels.
- Added new wordlist `en_eff`. It is a 7776-terms list provided by the Electronic Frontier Foundation. See https://www.eff.org/deeplinks/2016/07/new-wordlists-random-passphrases for details. Thanks to George V. Reilly for hinting!
- Fixed #27. Allow dashes in numbered wordlists. Yet, these looked like `1234 myterm`. We now also accept `1-2-3-4 myterm`.

0.8 (2016-05-07)

- Closed #23. @dwcoder provided a fix that allows use of whitespace-only values in diceware confg files if they are enclosed in quotes.
- Fixed #21. @dwcoder revealed and fixed (again!). This time *–caps* and *–no-caps* settings did not work properly when set in CLI or in *.diceware.ini* config file.
- Shortened real-dice randomness source.
- Added logger as common interface to send messages to users.
- New dependency: *sphinx_rtd_theme* for generating docs. This theme was formerly a dependency of *Sphinx*.

0.7.1 (2016-04-21)

- Fixed #19. @dwcoder revealed and fixed a nasty bug in the real-dice randomness-source. Thanks a lot!

0.7 (2016-04-17)

- Added sample `.diceware.ini`.
- Added new english wordlist `en_securedrop`. This is the new default list. Thanks to heartsucker who compiled and added the list.
- Remove support for Python 3.2. Several packages we depend on for testing and sandboxing stopped Python 3.2 support. We follow them.

0.6.1 (2015-12-15)

- Minor doc changes: add separate config file docs.
- Fix docs: the default wordlist is named `en`. Some docs were not up-to-date in that regard.

0.6 (2015-12-15)

- Officially support Pyhthon 3.5.
- Tests do not depend on *pytest-cov*, *pytest-xdist* anymore.
- Support configuration files. You can set different defaults in a file called `.diceware.ini` in your home directory.
- Renamed wordlist `en_8k` to `en` as it serves as the default for english passphrases.

0.5 (2015-08-05)

- New option `-r`, `--randomsource`. We support a pluggable system to define alternative sources of randomness. Currently supported sources: `"system"` (to retrieve randomness from standard library, default) and `realdice`, which allows use of real dice.
- New option `-w`, `--wordlist`. We now provide several wordlists for users to choose from. Own wordlists could already be fed to *diceware* before. By default we still use the 8192 words list from http://diceware.com.
- Rename *SRC_DIR* to *WORDLISTS_DIR* (reflecting what it stands for).
- Use also flake8 with tox.
- Pass *options* to *get_passphrase()* instead of a bunch of single args.
- Output wordlists dir in help output.

0.4 (2015-03-30)

- Add –delimiter option (thanks to Rodolfo Gouveia).

0.3.1 (2015-03-29)

- Turned former *diceware* module into a Python package. This is to fix bug #1 Wordlists aren't included during installation, this time really. Wordlists will from now on be stored inside the *diceware* package. Again many thanks to conorsch who digged deep into the matter and also came up with a very considerable solution.
- Use readthedocs theme in docs.

0.3 (2015-03-28)

- Fix bug #1 Wordlists aren't included during installation . Thanks to conorsch
- Add –version option.

0.2 (2015-03-27)

- Minor documentation changes.

- Updated copyright infos.
- Add support for custom wordlists.

0.1 (2015-02-18)

- Initial release.

CHAPTER 7

Indices and tables

- genindex
- modindex
- search

Python Module Index

d

Index