dropbox / **zxcvbn**

Low-Budget Password Strength Estimation https://www.usenix.org/conference/use...

379 commits | **2** branches | **30** releases | **32** contributors | MIT

master ▾ **New pull request** **Find file** **Clone or download** ▾

lowe Fix broken demo link in README Latest commit 67c4ece Oct 12, 2017

data-scripts	doc tweak: make usage in data-scripts consistent with filenames in data/	Nov 10, 2015
data	skip non-unicode top passwords in xato. (this only skips one pw curre…	Nov 9, 2015
demo	add password feedback to demo	Oct 29, 2015
dist	latest dist build	Feb 7, 2017
src	make REFERENCE_YEAR dynamic	Feb 5, 2017
test	Add more debug info in test-matching	Nov 30, 2016
.gitignore	re-include src in npm package, exclude lib from git	Aug 12, 2015
.npmignore	README update for browserify/webpack. add .git and demo to .npmignore	Aug 13, 2015
.travis.yml	fix broken pull requests: remove saucelabs from continuous integratio…	Sep 24, 2016
.zuul.yml	fix broken pull requests: remove saucelabs from continuous integratio…	Sep 24, 2016
LICENSE.txt	Update license range	Feb 23, 2016
README.md	Fix broken demo link in README	Oct 13, 2017
bower.json	bower.json packaging bug: lib -> dist. fixes #91	Aug 23, 2015
package.json	4.4.2	Feb 7, 2017

README.md

```
_________________________________________________/\/\___________________
_/\/\/\/\/\__/\/\__/\/\____/\/\/\/\__/\/\__/\/\__/\/\________/\/\/\/\___
_____/\/\______/\/\/\____/\/\________/\/\__/\/\__/\/\/\/\____/\/\__/\/\_
___/\/\________/\/\/\____/\/\__________/\/\/\____/\/\__/\/\__/\/\__/\/\_
_/\/\/\/\/\__/\/\__/\/\____/\/\/\/\______/\______/\/\/\/\____/\/\__/\/\_
________________________________________________________________________
```

build passing

`zxcvbn` is a password strength estimator inspired by password crackers. Through pattern matching and conservative estimation, it recognizes and weighs 30k common passwords, common names and surnames according to US census data, popular English words from Wikipedia and US television and movies, and other common patterns like dates, repeats (`aaa`), sequences (`abcd`), keyboard patterns (`qwertyuiop`), and l33t speak.

Consider using zxcvbn as an algorithmic alternative to password composition policy — it is more secure, flexible, and usable when sites require a minimal complexity score in place of annoying rules like "passwords must contain three of {lower, upper, numbers, symbols}".

- **More secure**: policies often fail both ways, allowing weak passwords (`P@ssword1`) and disallowing strong passwords.
- **More flexible**: zxcvbn allows many password styles to flourish so long as it detects sufficient complexity — passphrases are rated highly given enough uncommon words, keyboard patterns are ranked based on length and number of turns, and capitalization adds more complexity when it's unpredictaBle.
- **More usable**: zxcvbn is designed to power simple, rule-free interfaces that give instant feedback. In addition to strength estimation, zxcvbn includes minimal, targeted verbal feedback that can help guide users towards less guessable passwords.

For further detail and motivation, please refer to the USENIX Security '16 paper and presentation.

At Dropbox we use zxcvbn (Release notes) on our web, desktop, iOS and Android clients. If JavaScript doesn't work for you, others have graciously ported the library to these languages:

- `zxcvbn-python` (Python)
- `zxcvbn-cpp` (C/C++/Python/JS)
- `zxcvbn-c` (C/C++)
- `zxcvbn-rs` (Rust)
- `zxcvbn-go` (Go)
- `zxcvbn4j` (Java)
- `nbvcxz` (Java)
- `zxcvbn-ruby` (Ruby)
- `zxcvbn-js` (Ruby [via ExecJS])
- `zxcvbn-ios` (Objective-C)
- `zxcvbn-cs` (C#/.NET)
- `szxcvbn` (Scala)
- `zxcvbn-php` (PHP)
- `zxcvbn-api` (REST)
- `ocaml-zxcvbn` (OCaml bindings for `zxcvbn-c`)

Integrations with other frameworks:

- `angular-zxcvbn` (AngularJS)

Installation

zxcvbn detects and supports CommonJS (node, browserify) and AMD (RequireJS). In the absence of those, it adds a single function `zxcvbn()` to the global namespace.

Bower

Install `node` and `bower` if you haven't already.

Get `zxcvbn` :

```
cd /path/to/project/root
bower install zxcvbn
```

Add this script to your `index.html` :

```
<script src="bower_components/zxcvbn/dist/zxcvbn.js">
</script>
```

To make sure it loaded properly, open in a browser and type `zxcvbn('Tr0ub4dour&3')` into the console.

To pull in updates and bug fixes:

```
bower update zxcvbn
```

Node / npm / MeteorJS

zxcvbn works identically on the server.

```
$ npm install zxcvbn
$ node
> var zxcvbn = require('zxcvbn');
> zxcvbn('Tr0ub4dour&3');
```

RequireJS

Add `zxcvbn.js` to your project (using bower, npm or direct download) and import as usual:

```
requirejs(["relpath/to/zxcvbn"], function (zxcvbn) {
    console.log(zxcvbn('Tr0ub4dour&3'));
});
```

Browserify / Webpack

If you're using `npm` and have `require('zxcvbn')` somewhere in your code, browserify and webpack should just work.

```
$ npm install zxcvbn
$ echo "console.log(require('zxcvbn'))" > mymodule.js
$ browserify mymodule.js > browserify_bundle.js
$ webpack mymodule.js webpack_bundle.js
```

But we recommend against bundling zxcvbn via tools like browserify and webpack, for three reasons:

- Minified and gzipped, zxcvbn is still several hundred kilobytes. (Significantly grows bundle size.)
- Most sites will only need zxcvbn on a few pages (registration, password reset).
- Most sites won't need `zxcvbn()` immediately upon page load; since `zxcvbn()` is typically called in response to user events like filling in a password, there's ample time to fetch `zxcvbn.js` after initial html/css/js loads and renders.

See the performance section below for tips on loading zxcvbn stand-alone.

Tangentially, if you want to build your own standalone, consider tweaking the browserify pipeline used to generate `dist/zxcvbn.js`:

```
$ browserify --debug --standalone zxcvbn \
    -t coffeeify --extension='.coffee' \
    -t uglifyify \
    src/main.coffee | exorcist dist/zxcvbn.js.map >| dist/zxcvbn.js
```

- `--debug` adds an inline source map to the bundle. `exorcist` pulls it out into `dist/zxcvbn.js.map`.
- `--standalone zxcvbn` exports a global `zxcvbn` when CommonJS/AMD isn't detected.
- `-t coffeeify --extension='.coffee'` compiles `.coffee` to `.js` before bundling. This is convenient as it allows `.js` modules to import from `.coffee` modules and vice-versa. Instead of this transform, one could also compile everything to `.js` first (`npm run prepublish`) and point `browserify` to `lib` instead of `src`.
- `-t uglifyify` minifies the bundle through UglifyJS, maintaining proper source mapping.

Manual installation

Download zxcvbn.js.

Add to your .html:

```
<script type="text/javascript" src="path/to/zxcvbn.js"></script>
```

Usage

try zxcvbn interactively to see these docs in action.

```
zxcvbn(password, user_inputs=[])
```

`zxcvbn()` takes one required argument, a password, and returns a result object with several properties:

```
result.guesses            # estimated guesses needed to crack password
result.guesses_log10      # order of magnitude of result.guesses

result.crack_times_seconds # dictionary of back-of-the-envelope crack time
                          # estimations, in seconds, based on a few scenarios:
{
  # online attack on a service that ratelimits password auth attempts.
  online_throttling_100_per_hour

  # online attack on a service that doesn't ratelimit,
  # or where an attacker has outsmarted ratelimiting.
  online_no_throttling_10_per_second

  # offline attack. assumes multiple attackers,
  # proper user-unique salting, and a slow hash function
  # w/ moderate work factor, such as bcrypt, scrypt, PBKDF2.
  offline_slow_hashing_1e4_per_second

  # offline attack with user-unique salting but a fast hash
  # function like SHA-1, SHA-256 or MD5. A wide range of
  # reasonable numbers anywhere from one billion - one trillion
  # guesses per second, depending on number of cores and machines.
  # ballparking at 10B/sec.
  offline_fast_hashing_1e10_per_second
}

result.crack_times_display # same keys as result.crack_times_seconds,
                           # with friendlier display string values:
                           # "less than a second", "3 hours", "centuries", etc.

result.score      # Integer from 0-4 (useful for implementing a strength bar)

  0 # too guessable: risky password. (guesses < 10^3)

  1 # very guessable: protection from throttled online attacks. (guesses < 10^6)

  2 # somewhat guessable: protection from unthrottled online attacks. (guesses < 10^8)

  3 # safely unguessable: moderate protection from offline slow-hash scenario. (guesses < 10^10)

  4 # very unguessable: strong protection from offline slow-hash scenario. (guesses >= 10^10)

result.feedback   # verbal feedback to help choose better passwords. set when score <= 2.

  result.feedback.warning     # explains what's wrong, eg. 'this is a top-10 common password'.
                              # not always set -- sometimes an empty string

  result.feedback.suggestions # a possibly-empty list of suggestions to help choose a less
                              # guessable password. eg. 'Add another word or two'

result.sequence   # the list of patterns that zxcvbn based the
                  # guess calculation on.

result.calc_time  # how long it took zxcvbn to calculate an answer,
                  # in milliseconds.
```

The optional `user_inputs` argument is an array of strings that zxcvbn will treat as an extra dictionary. This can be whatever list of strings you like, but is meant for user inputs from other fields of the form, like name and email. That way a password that includes a user's personal information can be heavily penalized. This list is also good for site-specific vocabulary — Acme Brick Co. might want to include ['acme', 'brick', 'acmebrick', etc].

Performance

runtime latency

zxcvbn operates below human perception of delay for most input: ~5-20ms for ~25 char passwords on modern browsers/CPUs, ~100ms for passwords around 100 characters. To bound runtime latency for really long passwords, consider sending `zxcvbn()` only the first 100 characters or so of user input.

script load latency

`zxcvbn.js` bundled and minified is about 400kB gzipped or 820kB uncompressed, most of which is dictionaries. Consider these tips if you're noticing page load latency on your site.

- Make sure your server is configured to compress static assets for browsers that support it. (nginx tutorial, Apache/IIS tutorial.)

Then try one of these alternatives:

1. Put your `<script src="zxcvbn.js">` tag at the end of your html, just before the closing `</body>` tag. This ensures your page loads and renders before the browser fetches and loads `zxcvbn.js`. The downside with this approach is `zxcvbn()` becomes available later than had it been included in `<head>` — not an issue on most signup pages where users are filling out other fields first.

2. If you're using RequireJS, try loading `zxcvbn.js` separately from your main bundle. Something to watch out for: if `zxcvbn.js` is required inside a keyboard handler waiting for user input, the entire script might be loaded only after the user presses their first key, creating nasty latency. Avoid this by calling your handler once upon page load, independent of user input, such that the `requirejs()` call runs earlier.

3. Use the HTML5 `async` script attribute. Downside: doesn't work in IE7-9 or Opera Mini.

4. Include an inline `<script>` in `<head>` that asynchronously loads `zxcvbn.js` in the background. Advantage over (3): it works in older browsers.

```
// cross-browser asynchronous script loading for zxcvbn.
// adapted from http://friendlybit.com/js/lazy-loading-asyncronous-javascript/

(function() {

  var ZXCVBN_SRC = 'path/to/zxcvbn.js';

  var async_load = function() {
    var first, s;
    s = document.createElement('script');
    s.src = ZXCVBN_SRC;
    s.type = 'text/javascript';
    s.async = true;
    first = document.getElementsByTagName('script')[0];
    return first.parentNode.insertBefore(s, first);
  };

  if (window.attachEvent != null) {
    window.attachEvent('onload', async_load);
  } else {
    window.addEventListener('load', async_load, false);
  }

}).call(this);
```

Development

Bug reports and pull requests welcome!

```
git clone https://github.com/dropbox/zxcvbn.git
```

zxcvbn is built with CoffeeScript, browserify, and uglify-js. CoffeeScript source lives in `src`, which gets compiled, bundled and minified into `dist/zxcvbn.js`.

```
npm run build    # builds dist/zxcvbn.js
npm run watch    # same, but quickly rebuilds as changes are made in src.
```

For debugging, both `build` and `watch` output an external source map `dist/zxcvbn.js.map` that points back to the original CoffeeScript code.

Two source files, `adjacency_graphs.coffee` and `frequency_lists.coffee`, are generated by python scripts in `data-scripts` that read raw data from the `data` directory.

For node developers, in addition to `dist`, the zxcvbn `npm` module includes a `lib` directory (hidden from git) that includes one compiled `.js` and `.js.map` file for every `.coffee` in `src`. See `prepublish` in `package.json` to learn more.

Acknowledgments

Dropbox for supporting open source!

Mark Burnett for releasing his 10M password corpus and for his 2005 book, Perfect Passwords: Selection, Protection, Authentication.

Wiktionary contributors for building a frequency list of English words as used in television and movies.

Researchers at Concordia University for studying password estimation rigorously and recommending zxcvbn.

And xkcd for the inspiration ❤