passwordmetrics Documentation

Release 1.0b1

Lennart Regebro

April 29, 2015

Contents

Contents:

CHAPTER 1

passwordmetrics

Checking the entropy of a password. Useful in password checkers.

- Free software: BSD license
- Documentation: http://passwordmetrics.readthedocs.org.

1.1 Features

passwordmetrics will return information about a password, such as length, words found, what types of characters that are used, and a metric called "entropy" which will give an approximate measure of how hard the password is to guess.

CHAPTER 2

Installation

At the command line:

```
$ pip install passwordmetrics
```

CHAPTER 3

Password strength

Password strength is a measure of the effectiveness of a password in resisting guessing and brute-force attacks.[1]_ Since users like to have passwords that are easy to remember, they tend to pick weak passwords that are easy to guess.

Many administrators of systems and sites try to force their users to have more secure passwords by forcing a length or requiring that you use both uppercase and lowercase, or add punctuation, etc. This XKCD cartoon illustrates why this is a bad idea:

In addition, many users will just take their weak password and add ones and exclemation marks to it. So instead of "banana" they now have "Banana1!". That is harder to guess, but not a lot harder.

The most secure passwords are long passwords with randomly generated characters, such as "/JGu-Jzxbm<K%n3d^7#,)". This is however hard to remember and type, so it's a good password for when you have a password manager, like in your web browser, but not a good password for an admin password on a computer.

3.1 How to get your users to have strong passwords

The passwordmetrics package enables you to get a measurement of how hard the entered password is to guess. The 'entropy' is a measurement that is approximately how many bits of information is contained in the password, and therefore how hard it is to guess. The higher, the better.

You can then tell your users if that is a weak, average or strong password. Google's user management is a good example of a UI that does this correctly. You can also set a minimum strength.

To determine what is "good", run pass wordmetrics on a couple of passwords you would think are good, and use the resulting number as a measurement.

For reference, "banana", a password that is unacceptably easy, has an entropy of approximately, 6. The above 20 character random string has an entropy of around 124, and XKCD's "correcthorsebatterystaple" has an entropy of around 45.

CHAPTER 4

Usage

4.1 Basic usage

To use passwordmetrics you import it and run the configure method.

```
>>> import passwordmetrics
>>> passwordmetrics.configure()
```

You can then get metrics for passwords:

```
>>> passwordmetrics.metrics('correcthorseb!wdbatterystaplerWd6t')
{'unused_groups': set(['punctuation', 'non-printable', 'other', 'whitespace']),
 'word_entropy': 53.48690135737497,
 'entropy': 89.21207921969622,
 'words': set(['battery', 'i', 'horse', 'stapler', 'correct']),
 'unknown_chars': set([]),
 'used_groups': set(['digits', 'uppercase', 'lowercase']),
 'length': 34,
 'character_entropy': 35.72517786232125}
```

4.2 Advanced usage

The configure method makes it possibly to configure passwordmetrics for your usecase. This is particularily useful if you want to have language-specific wordlists or character groups:

```
>>> import passwordmetrics

>>> groups = {'lowercase': set(u'abcdefghijklmnopqrstuvxyzääö'),
              'uppercase': set(u'ABCDEFGHIJKLMNOPQRSTUVXYZÅÄÖ'),
              'digits': set(string.digits),
              'punctuation': set(string.punctuation),
              'whitespace': set(string.whitespace),
              }

>>> with open('ordlista_sv.txt', 'rt', encoding='latin-1') as wordlist:
...     for line in wordlist.readlines():
...         word, entropy = line.strip().split(' ')
...         words[word] = float(entropy)
```

```
>>> substitutions = {'0': 'o', '1': 'i', '2': 'z', '3': 'e', '4': 'a',
                     '5': 's', '6': 'b', '7': 't', '8': 'b', '9': 'g',
                     '!': 'i', '#': 3, '$': 's', '&': 'g', '@': 'a',
                     '[': 'c', '(': 'c', '+': 't', '{': 'ä', '|': ö,
                     '}': 'å', '[': 'Ä', '\': 'Ö', ']': 'Å'}

>>> passwordmetrics.configure(grops=groups, words=words, substitutions=substitutions)
>>> passwordmetrics.metrics('korrekthästbatter1häftapparat')
{'unused_groups': set(['punctuation', 'whitespace']),
 'word_entropy': 62.24279530886037,
 'entropy': 85.04455418142474,
 'words': set(['korrekt', 'häst', 'batteri', 'häftapparat']),
 'unknown_chars': set([]),
 'used_groups': set(['uppercase', 'lowercase']),
 'length': 31,
 'character_entropy': 22.80175887256437}
```

4.2.1 groups

`groups` are character groups, such as digits, lowercase and uppercase. You pass in sets of characters, and passwordmetrics will tell you which ones were used. This is so that you, if desired, can disallow certain character groups. You can also, of course, force certain character groups, but that is a bad idea (more on that elsewhere).

4.2.2 words

`words` is a mapping of words to word frequency values. It is used to find words in the password and base a calculation of the entropy based on that. For example, a password containing common words should have a lower entropy than a password containing uncommon words, as it is easier to guess.

4.2.3 substitutions

`substitutions` are character mappings for "leet" character substitutions. For example, the words "alive" can be written "al1v3" in a way to make it harder to guess. A mapping of substitutions is used to try and find words. This is because a word with substitutions is easier to guess than a random character string, although it is harder to guess than the same word without substitutions.

4.3 The metrics

The returned metrics are:

- entropy: A relative measure of how hard a password is to guess.
- length: The total length of the password.
- words: The words found in the password.
- word_entropy: The entropy of the words.
- character_entropy: The entropy of any characters not found in any words.
- used_groups: The character groups used.
- unused_groups: Any character groups that were not used.

- unknown_chars: Characters in the password that were not found in any groups.

Most of the metrics returned are returned only for completeness, not because they are very useful.

`length` is useful if you have a maximum password length your system can handle and `used_groups` and `unknown_chars` is useful to make sure there are no characters your system can't handle.

The most important metric is the "entropy". The higher the harder it is to guess the passwords, and the harder it will be to forcibly crack the password. Giving the user feedback on how safe/unsafe the password is based on this is a good way to ensure that you have safe passwords.

CHAPTER 5

Contributing

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

You can contribute in many ways:

5.1 Types of Contributions

5.1.1 Report Bugs

Report bugs at https://github.com/regebro/passwordmetrics/issues.

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

5.1.2 Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with "bug" is open to whoever wants to implement it.

5.1.3 Implement Features

Look through the GitHub issues for features. Anything tagged with "feature" is open to whoever wants to implement it.

5.1.4 Write Documentation

passwordmetrics could always use more documentation, whether as part of the official passwordmetrics docs, in docstrings, or even on the web in blog posts, articles, and such.

5.1.5 Submit Feedback

The best way to send feedback is to file an issue at https://github.com/regebro/passwordmetrics/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

5.2 Get Started!

Ready to contribute? Here's how to set up *passwordmetrics* for local development.

1. Fork the *passwordmetrics* repo on GitHub.
2. Clone your fork locally:

   ```
   $ git clone git@github.com:your_name_here/passwordmetrics.git
   ```

3. Install spiny for running tests:

   ```
   $ pip install spiny
   ```

4. Create a branch for local development:

   ```
   $ git checkout -b name-of-your-bugfix-or-feature
   ```

   Now you can make your changes locally.

5. When you're done making changes, check that your changes pass flake8 and the tests, including testing other Python versions with spiny:

   ```
   $ python setup.py test  # Tests with one Python version
   $ spiny  # Tests with all supported Python versions
   ```

6. Commit your changes and push your branch to GitHub:

   ```
   $ git add .
   $ git commit -m "Your detailed description of your changes."
   $ git push origin name-of-your-bugfix-or-feature
   ```

7. Submit a pull request through the GitHub website.

5.3 Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests, and they should pass for all supported Python versions.
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring, and add the feature to the list in README.rst.

CHAPTER 6

Credits

6.1 Development

- Lennart Regebro <regebro@gmail.com>

6.2 Obvious inspiration

- http://xkcd.com/936/

CHAPTER 7

History

CHAPTER 8

1.0b1 (2015-04-29)

- First release on PyPI.

CHAPTER 9

Indices and tables

- *genindex*
- *modindex*
- *search*