EntroPass

Release 0.1

Jochem Stevense

Jun 09, 2021

CONTENTS:

CHAPTER

ONE

USER MANUAL

1.1 Configuration

The configuration file for EntroPass is found in the config/ directory with the name entropass_conf.toml. TOML is used for its readable format and ease of use with Python3. Following is an explanation of the configuration file, starting from the top and working to the end of the file.

1.1.1 words

list

This is the list of seed words for the program. It should be filled with personal information on the target. This is recommended to be a combination of usernames, names of close relations, pets, nicknames and other names in general. Next to this, hobbies, animals, keyboard strokes and erotic (and mostly offensive) phrases are popular in common passwords. Also, the most popular characters are lower case alphabetical characters and digits. By default, the seed words are all converted to lowercase, since uppercase is generated by the password generator. Once the information is filled in, the result should look like the following:

```
[words]
list = [
 'johann',
 '01-12-1890',
 '2021',
 'jack'
 ]
```

1.1.2 results

max_words

If there should be a limit to the number of passwords in the final list, this can be set to any value. This will not reduce processing time, since it will first generate the passwords, then score and sort them, after which the top N words are written into the output file. Setting this to 0 will result in no limit.

```
max_words = 0 # no maximum
```

1.1.3 print

shell
Setting this to true will print all passwords to the shell. Default is false.

```
shell = false # do not print to shell
```

file
Setting this to true will print the passwords to the configured file.

```
file = true # print results to file
```

dir
Directory where the results should be written into (should end with a '/').

```
dir = '../results/' # print results to file in the specified directory
```

fd
File name to write results into. The file will be created or overwritten in the previously specified directory.

```
fd = 'results.txt' # print results to results.txt
```

sort
Set to true to sort the generated passwords.

```
sort = true
```

sortby
Set to 'DESC' or 'ASC' to sort by score, starting with the highest score and working down to the lower scores. This is recommended, since a tool like Hydra will start using passwords, starting at the top of the file (as will most other tools most likely).

```
sortby = 'DESC' # Sort by score, starting with top scoring password
```

1.1.4 categories

This holds the directories to the category lists that are used for scoring.

names
List of names to be used for scoring

```
names = '../categories/names/names.txt'
```

foods
List of foods to be used for scoring

```
names = '../categories/food/food.txt'
```

1.1.5 patterns

Regular expression patterns to check for matches. If it matches, the password score will be incremented.

birthdate
Birthdate pattern. Matches birthdates such as 12-12-2012, 12-12-12, 12/12-2021, 12.12.2021, etc.

```
birthdate = '(.*)[\d]+(\.|\/|\-|\_)[\d]+(\.|\/|\-|\_)[\d]+(.*)'
```

1.1.6 processed-data

dir
Directory where the processed formats and characters are stored.

```
dir = '../processed/'
```

format
File with formats and the number of occurrences.

```
format = 'format.txt'
```

char
File with chars and the number of occurrences.

```
char = 'char.txt'
```

1.1.7 generating-rules

Set specific rules to filter the generated passwords and reduce processing time, by not scoring all passwords.

use-rules
Set to true to use custom rules. Not implemented yet.

```
use-rules = false
```

use-formats
Set to true to use custom specified formats to filter output.

```
use-formats = false
```

formats
The custom formats that the generated password should match with to be used. Only use if use-formats is set to true.

```
formats = ['cccccc']
```

use-top-formats
Set to true to use the top N password formats to match the generated passwords against.

```
use-top-formats = true
```

top-formats
The number of formats to use to filter the generated passwords. Only used if use-top-formats is set to true.

```
top-formats = 1000 # use top 1000 most used formats to match passwords against
```

1.1.8 common-separators

list
List of characters that are often used to separate words.

```
list = [
 '.', '-', '/', '*'
 ]
```

1.1.9 common-end

list
List of characters that are often used to end passwords.

```
list = [
 '!', '?', '.', '123', '12345'
 ]
```

1.1.10 common-replacements

This is a defined 'library' of characters that are often used to replace others. It is loosely based on the '1337' principle. Each of the characters defined will be replaced by the others. For example, if the word is 'word' and the replacements for 'w', 'o', 'r' and 'd' are defined to be the following:

```
'w' = []
'o' = ['0', '']
'r' = ['12']
'd' = [')']
```

This will result in the generation of variations of 'word' like 'w0rd', 'ord', 'wol2d', 'wor)', 'wr)', etc. The characters on the left side of the = will only be processed correctly if they are single characters, while the characters on the right can be of arbitrary length.

1.2 Usage

1.2.1 Options

To get the options for EntroPass, simply run it as follows:

```
./entropass.py --help
```

This will output the optional arguments and how to use them. At the time of writing, the output is the following:

```
usage: entropass.py [-h] [-d] [-u] [-c COMPARE]

optional arguments:
-h, --help            show this help message and exit
-d, --debug           Turn on debugging mode. NOT IMPLEMENTED YET
-u, --update          Update the formats and characters with the passwords
                      in the passwords directory. WARNING! This can take a
                      long time depending on the number of passwords in the
                      passwowords directory.
-c COMPARE, --compare COMPARE
                      Compare password to results.
```

1.2.2 Basic Running

To run the program, first make sure the configuration is set to your likings. After this, the program can be run by using the following:

```
./entropass.py
```

1.2.3 Comparing with existing password

To compare the generated passwords with an existing password, the following option can be used:

```
./entropass.py -c COMPARE
```

For example, for a password like johann12345 the command would be like the following:

```
./entropass.py -c johann12345
```

This will run the program as usual, but it checks if the given password exists in the list of generated passwords. After this, the passwords get scored and the number in the list of passwords will be given to demonstrate how many guesses it would take to get this password, using the generated password list. An example output is the following:

```
[*] Loaded configuration from ../config/entropass_conf.toml
[*] Expanding seed words:  4 -> 22
[*] Generating permutations
[*] Replacing characters
[*] Resulting passwords: 339722
[*] Removing duplicates
[*] Remaining: 314160
[*] Filtering passwords with formats: 314160 ...
[*] Filtered passwords with formats: 1088
[!] Found the password in list of 1088 passwords
[*] Scoring password 1088 of 1088
[*] Sorting passwords
[!] Password is #692 in the scored list.
[*] Wrote 1088 passwords to test_sort.txt
```

This is meant to demonstrate the vulnerability of someones password, but can also be used to make sure a certain password is in the list, to check if more seed words should be added or not.

1.3 What to do next

Once you've generated your wordlist, you can simply use tools such as John the Ripper or Hydra to perform penetration tests on systems. Remember, this tool is not meant to be used for illegal purposes. Consent from the target is required!

CHAPTER

TWO

DEVELOPMENT

2.1 Introduction

As you have probably realised by now, EntroPass has been written in Python3. It has been developed, using a virtual environment to keep the development system from getting cluttered and to keep dependency management relatively easy. The dependencies are stored in a text file called "requirements.txt". The project has been developed in a Object Oriented style and has Numpy style Sphinx comments embedded in the code for this documentation and to allow developers to easily identify the purpose of the bits of code. The code was written in a text editor (Emacs, for those interested), and should remain possible to develop on outside of IDE's.

2.2 Guidelines

Python version
Python3 is used and the project is not compatible with Python2. The exact version at the time of writing is 3.8.6

Indentation
The indentation is set to the Emacs Python default, which is 4 spaces.

Dependencies
The dependencies are stored in "requirements.txt" in the root directory of the repository. These can be installed in the virtual environment using the following command:

```
pip install -r requirements.txt
```

Outside of the virtual environment, it is possible to install all dependencies in a similar manner, using the following command:

```
python3 -m pip install -r requirements.txt
```

If you installed new dependencies using pip, make sure to update the requirements file:

```
pip freeze > requirements.txt
```

Virtual Environment
The virtual environment is not included in the repository, since it is system specific. To create a virtual environment, navigate to the root directory of the repository and use the following command:

```
python3 -m venv venv
```

After this, activate the virtual environment:

```
source venv/bin/activate
```

Next, you can install the dependencies, as mentioned before:

```
pip install -r requirements.txt
```

2.3 To Do

This to-do list includes a list of all possible improvements and/or fixes.

- Add more password generating possibilities. This may be a never-ending item on the to-do list. Add more possibilities to add generate more passwords based on the seed words.
- Improve iteration in character replacement in password generating. Iteration does not always seem fully functional when replacing characters. Iteration is used to replace more than 1 type of character in a word. For example, a word 'word' can turn into 'w0rd' during the first iteration, and can turn into 'w0r)' during the second iteration. This works, but it seems there is little to no difference between 3 and 10 iterations, when it comes to the number of results. This requires some research.
- Add generating rules. The configuration holds some variables that can be used for password generating rules, to limit processing time, by limiting the number of passwords actually created, and allowing users to specify the type of passwords they expect. More rules should be added to improve this functionality.
- Add category support. There are a number of categories in a dedicated directory. These categories all hold a list of words that relate to these categories. More words should be added to all these categories and the scoring should parse all these word lists to see if any of these words exist in a given password, incrementing the score if so.
- Add pattern scoring. Pattern scoring can be useful for patterns such as birthdates, where it is not feasible to have a list of all possible birthdates to check against. The pattern can be used to check if the password (probably) has a birthdate embedded into it. If so, the score can be incremented.

CHAPTER

THREE

ENTROPASS

3.1 EntroPass package

3.1.1 Submodules

3.1.2 EntroPass.entropass module

class EntroPass.entropass.**EntroPass**

Bases: object

Main class that loads the configuration and controls all the other classes.

comp_pwd = None

config_fd = '/home/jochemste/Projects/python_projects/entropass/EntroPass/../config/entropass_conf.toml'

debug = False

get_count()

Gets the number of passwords.

pass_w_scores = []

passwords = []

print_passwords()

Prints the passwords, separated by a comma.

run()

Main function to run the needed functions. Will start by expanding the number of seed words, followed by permutations, replacement of characters, removing duplicates, filtering to match formats, searching for the given password and finally scoring the passwords and limiting number of passwords to a specified number.

run_filter()

Filter the found passwords to those that match the configured formats.

run_permutations()

Generates permutations of the seed words and adds them to the password lists.

run_replace()

Replaces characters in all words with common replacements. Does not overwrite any words, but simply adds the new ones.

run_seeding_expanse()

Expand seeding words by generating all possible combinations of upper and lower case characters of the words.

run_update()
Calls the update script to process information in the password directory and generate formats and such in the processed directory.

scored_pwds = {}

write_t_file()
Writes the passwords to the configured file. Creates the specified directory if it does not exist.

3.1.3 EntroPass.print_utils module

EntroPass.print_utils.**cprint**(**args*, *fore=''*, *back=''*, *style=''*, *start='\x1b[32m\x1b[1m[*]\x1b[0m'*, *sep=' '*, *end='\n'*, ***kwargs*)
Print messages with a colour. Resets all colour settings after printing.

Parameters

- **args** (*list*) – The text arguments. Can be any type of variable.
- **fore** (*str*) – Code to set the colour of the text.
- **back** (*str*) – Code to set the colour of the background of the text.
- **style** (*str*) – Code to set the style of the text.
- **start** (*str, default = START_SUCCESS*) – Starting string of the text.
- **sep** (*str, default = ' '*) – Separator for in between arguments.
- **end** (*str, default =*) – End of the text.
- **kwargs** (*dict*) – Unused

EntroPass.print_utils.**error**(**args*, *sep=' '*, *end='\n'*, ***kwargs*)
Print a error message with the accompanying colour.

Parameters

- **args** (*list*) – The text arguments. Can be any type of variable.
- **sep** (*str, default = ' '*) – Separator for in between arguments.
- **end** (*str, default =*) – End of the text.
- **kwargs** (*dict*) – Unused

EntroPass.print_utils.**success**(**args*, *sep=' '*, *end='\n'*, ***kwargs*)
Print a success message with the accompanying colour.

Parameters

- **args** (*list*) – The text arguments. Can be any type of variable.
- **sep** (*str, default = ' '*) – Separator for in between arguments.
- **end** (*str, default =*) – End of the text.
- **kwargs** (*dict*) – Unused

EntroPass.print_utils.**warning**(**args*, *sep=' '*, *end='\n'*, ***kwargs*)
Print a warning message with the accompanying colour.

Parameters

- **args** (*list*) – The text arguments. Can be any type of variable.
- **sep** (*str, default = ' '*) – Separator for in between arguments.

- **end** (*str, default =*) – End of the text.
- **kwargs** (*dict*) – Unused

3.1.4 EntroPass.pwd_gen module

class EntroPass.pwd_gen.**Pwd_gen**(*config_fd*)
Bases: object

A class to handle generating passwords, by using permutations, upper/lower character combinations, replacement of characters, and more.

filter(*results*)
Filters the given results to only match the formats or regexes, specified in the configuration. The configuration allows users to specify formats or regexes, or use the top formats from the processed passwords, or not filter at all (not recommended).

Parameters **results** (*list*) – Results to be checked and filtered.

Returns **results** – Results from the filtering.

Return type list

fraction(*word*)
Returns fractions of the given word. For example, for "word" it will return ['w', 'wo', 'wor']

rep_cmmn_chars(*word: str*, *count=None*, *iterations=4*)
Replace commonly replaced characters. Iterate the results a specified number of times and replace more characters in results. For example: 'start' becomes '5tart' and is added to results. After second iteration '5tart' becomes '57ar7', which is added to the results as well. In iterations, all previous results are used, exponentially increasing the nr of results. Results are filtered from duplicates at the end, making that 5 iterations can provide the same nr of results as 100. However, this will make a significant difference in the processing time.

Parameters

- **word** (*str*) – Word to replace common characters from.
- **count** (*int, default = None*) – Nr of characters to replace every time. For example, if the word is 'all' and 'l' would be replaced by '|', having count=None would result in 'a||', while having count=1 would result in 'a|l'.
- **iterations** (*int, default = 4*) – Number of times to parse previous results. This can allow words like 'all' to become words like '@||' for example.

Returns **results** – List of results

Return type list

reverse(*word*)
Reverse a word. For example, "word" turns into "drow".

Parameters **word** (*str*) – Word to reverse

upper_perms(*word*)
Change characters in word to upper characters. Changes single characters first: Test, tEst, teSt. Followed by all chars left to right: test, Test, TEst, TESt Followed by the inverse: TEST, tEST, teST, tesT. Removes duplicates before returning. Runs in three threads for speed.

Parameters **word** (*str*) – Word to base the new results on.

Returns **results** – Results from the three threads.

Return type list

word_perms(*words*, *nr_in_result=2*, *add_cmmn_sep=True*, *add_cmmn_end=True*)
Permutations of given words, using a given amount of words per result. For example: words=['hello', 'world', '!'], nr_in_result=2 would give results like: 'hello!', 'helloworld', '!world', etc. It also parses over itself, incrementing the number of words in the results until it reaches the nr_in_result.

Parameters

- **words** (*list*) – List of words to create permutations of.
- **nr_in_result** (*int*) – Number of words in the final results.
- **add_cmmn_sep** (*bool, default = True*) – Adds a commonly used separator to be used in the permutations.
- **add_cmmn_end** (*bool, default = True*) – Adds a commonly used end to be used in the permutations.

Returns **results** – Resulting permutations.

Return type list

3.1.5 EntroPass.pwd_score module

class EntroPass.pwd_score.**Pwd_score**(*config_fd*)
Bases: object

char_fd = ''

chars = {}

form_fd = ''

forms = {}

isalphabet(*char*)
Checks if a character is in the alphabet.

Parameters **char** (*str*) – Character to check.

Returns True if in alphabet, False otherwise

Return type bool

score_pwd(*pwd*)
Scores the given password, based on format, used chars, presence in categories and known patterns. Returns the score.

Parameters **pwd** (*str*) – The password to score.

Returns **score** – The score of the password.

Return type int

score_pwd_chars(*pwd*, *char*, *occurrences*, *chars_weight=0.1*)
Scores the given password based on used characters.

Parameters

- **pwd** (*str*) – The password to score
- **char** (*str*) – The character to check the password for.
- **occurrences** (*int*) – The number of times the character occurred.

- **chars_weight** (*float, 0.1*) – The weight that character score have in the scoring process.

Returns The rounded score as an integer.

Return type int

score_pwd_format(*pwd*, *format_*, *occurrences*, *format_weight=100.0*)
Scores the given password on the given format.

Parameters

- **pwd** (*str*) – The password to score
- **format** (*str*) – The format to compare the password against
- **occurrences** (*int*) – The number of times the format occurred. This is used to give a weight to the score.
- **format_weight** (*float, 100.0*) – The weight that formats receive in the entire scoring process.

Returns The rounded score as an integer.

Return type int

score_pwd_name(*pwd*, *name*, *name_weight=10000.0*)
Scores the password based on a name being present.

Parameters

- **pwd** (*str*) – The password to score
- **name** (*str*) – The name to check the password for.
- **occurrences** (*int*) – The number of times the name occurred.
- **chars_weight** (*float, 10000.0*) – The weight that name scores have in the scoring process.

Returns The rounded score as an integer.

Return type int

score_pwd_pattern(*pwd*, *pattern*, *patt_weight=100.0*)
Scores a pwd for a specific pattern.

Parameters

- **pwd** (*str*) – The password to score
- **pattern** (*str*) – The regex pattern to check the password for.
- **patt_weight** (*float, 10000.0*) – The weight that pattern scores have in the scoring process.

Returns The rounded score as an integer.

Return type int

3.1.6 Module contents

CHAPTER

FOUR

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

e

INDEX

S

U

W