tokenGen: Generating High-Entropy Passphrases by Approximating an Individual's Vocabulary

Mark Xue

***Abstract*— By generating suitably large vocabulary lists, we can make it feasible for a person to memorize a high-entropy password of drawn from a space of 75-128 bits.**

I. Motivation

The idea of passphrase passwords has been proposed for several decades [1], and recently was revitalized by its presentation in the R Munroe's web comic xkcd [2]. Yet there are few existing tools to generate such passwords, and the existing tools fail to take full advantage of the average human vocabulary. I proposed an improved implementation by generating custom vocabulary lists to approximate an individual's personal vocabulary, thereby improving our ability to recall a chosen passphrase.

II. Are high-entropy passwords necessary?

Despite ample evidence that people do not choose strong passwords, passwords remain in use as a primary means of identity authentication on the internet. So although passwords are imperfect, they remain good enough for many commercial applications. J Bonneau explained in [3] that most websites mitigate poor choices of password by combining the password with other signals about the user's identity to build a holistic risk model for user authentication.

Moreover, all major web browsers will offer, by default, to store, manage, and suggest strong passwords. While these tools reduce the scope of challenge of password management, they don't completely eliminate it. The security of your password manager is still dependent on your master password - whether that is to your device, your cloud account, or a stand-alone software package.

Commercial biometric authentication have been widely adopted in recent years. To take Apple's Touch ID as an example, it's best viewed as a caching mechanism for the device password. Once you have authenticated with your password, the fingerprint sensor grants you the convenience of not having to reenter it for the next 24 hours. But as is clear when you reboot the device, or upgrade the software, the password is still the primary means of authentication.

In the past few years, there has been a movement by journalists to adopt better cybersecurity measures to protect their sources and reporting, spurred by the extraordinary measures taken by those reporting on the documents released by Edward Snowden. Against an active adversary, the requirement for high-entropy passwords becomes even more important. It is not unreasonable to want a passphrase as strong as your encryption key - which would justify passwords of complexity 2^{128} or higher. At the end of the day, it's hard to avoid holding a highly complex secret in your mind.

III. Alternate Schemes

What would such passwords look like? For comparison, we can look at what Apple and Google recommend in their browsers for internet passwords. The Safari browser suggests passwords with 12 tokens, delimited in blocks of 3, where each token is a letter or number. This gives a total password space of 2^{72}. Both Apple and Google, under their two-factor authentication schemes, provide application passwords of 16 letters, for a password space of 2^{75}.

Even if we generously grant the use of all 95 printable ASCII characters, it would take 20 characters to exceed a password space of 2^{128}. These far exceed the ability of the average person to remember, relegating us to writing down, shortening, or otherwise compromising our passwords.

Several sources, including Bruce Schneier, suggest mimicking such high-entropy passwords by applying a series of transformations to a known phrase - for example, taking the first letters of a sentence and applying some "personally memorable tricks" to turn it into a password. [4]

While this scheme is successful in creating a password that is not a dictionary word and is less vulnerable to a dictionary attack - for example in the unfortunately common case when password hashes are compromised and exhaustively searched offline, it is still not truly random. The human brain is notoriously bad at creating entropy. In violation of Kerckoff's principle, the security of the scheme draws heavily on the secrecy of the method of generation the phrase and the "personally memorable tricks" used to perturb the result. It would be possible for an attacker with access to previously generated passwords and biographical information about the target, to make inferences about the password scheme and shrink the search space.

The literature on usability of passphrases has some mixed results. M Ghazvininejad and K. Knight [5] compared several password schemes, including Randall Munroe's implementation of passphrase (4 tokens from a dictionary of size 2048), and a simplified version of the Schneier scheme that used the first letters of a phrase without any further perturbation. On a small sample size (62 participants) they found the latter to be the least memorable scheme, with almost half the retention of passphrases.

R. Shay el al conducted a 1,476 participant study comparing system-assigned random character passwords and random word passphrases. They found little difference in performance between the two, in practical measures of retention, the rates at which people chose to store their password rather than remember them and in user perception of annoyance, difficulty, and fun.[6]

However, their study chose extremely small dictionaries ranging in size from 181 - 1024, resulting in comparable token counts for their selected entropy setting of 30 bits. They compared passphrases of 4 words to passwords of 5-6 characters, or 3-4 syllables. Based on their choice of dictionaries, they concluded that larger dictionaries didn't offer much benefit.

IV. Vocabulary

I hypothesize that by using dictionaries that mimic the size of a person's vocabulary, we can significantly improve the passphrase scheme by achieving a practical reduction in token size. An oft-cited online study [7] reported that the majority of native English speakers have a vocabulary of 20,000 - 35,000 words. Previous academic studies have pointed to 15,000 as an average English vocabulary.

I noted earlier that it would take 20 printable ASCII characters to represent a password space of size 2^{128}. With a dictionary size of 2^11, we can reduce that to 12 words. With a dictionary of 19112, we can express 128 bits of entropy in only 9 words - just barely within George Miller's estimate of the size of our working memory at 7 ± 2.

There are diminishing returns to expanding the dictionary size. A naive implementation using the OS X system library of size 235,886 is almost large enough to shrink the token count to 7, but at the cost of having only a 1 in 10 chance of recognizing each token.

I believe one of the motivations behind the use of small dictionaries is that while our individual vocabularies are very large, their overlap is much smaller. Diceware, a scheme for generating passphrases by rolling dice to select from a list of 6^5 hand-chosen words, remarks that their list is heavily slanted to American English, leading to the creation of alternate lists. Why not create a personalized list for every individual? If we assume that our dictionary is public knowledge, and that revealing it does not harm the security of our passphrase scheme, it is to our advantage to personalize it as much as possible.

V. Method

I propose building this vocabulary by parsing corpuses of text - mainly things that the user has written, but also corpuses that the user commonly reads - for example professional journals or genres of literature. Because passphrases need only be recognizable to the user, and not to a broader audience, this expands the scope of possible words beyond those that are found in dictionaries - proper names, for example.

This idea is closely related to lexical parsing - obtaining a set of tokens or lexemes from text. The linguistic notion of lexeme is actually too powerful for this - the goal of parsing lexemes is to collect words with a shared meaning - so that "run, runs, running, ran" are all different forms of one lexeme. For our purposes, we want as many distinct forms as possible.

For that matter, why not consider a very broad category of what we can consider a word? Source code and the operators and variable names could produce an ample body of vocabulary. As could romanizations of foreign languages.

My goal for this project was to determine if it would be possible to build a usable dictionary of 20-30,000 words by analyzing small corpuses of text specific to an individual (as opposed to large shared corpuses such as Google N-gram, the British National Corpus, or the Brown Corpus of American English. Moreover, I want to compare different types of input against a reference to see how much of an improvement this method would be, against selecting the 30,000 most common words in a shared corpus.

VI. Design

For this project, I created a program that crawls a given directory and runs a parser against any text files to generate a histogram of tokens and their frequency. I perform some comparison against a fixed dictionary and pad the vocabulary list if desired. I generate passphrases using the Crypto.Random.DRBG package, using the recommended generator using AES-128 in counter mode, seeded with system entropy.

For the reference corpus, I used the 2^{16} most common words in the British National Corpus The corpuses I compared against were:

- The Enron Email Dataset
- National University of Singapore SMS corpus
- A folder of source code from previous Stanford classes, measured with SLOCCount at 27,445 lines of code. This was primarily in C and C++, along with some Haskell, LaTeXand Python code.
- Approximately 20mb of long English novels - Moby Dick, War and Peace and Middlemarch for example.
- A dictionary of Chinese vocabulary from the standard Hanyu Shuiping Kaoshi (HSK) exams in pinyin form.

VII. Results

Corpus	Words	Overlap with BNC	Avg length
Source code	22,401	31%	6.5
Books	37,322	42%	7.4
Chinese	2016	-	-
sms	17554	28.8%	5.5
50MB email	35695	30.28%	6.8
20MB email	19454	33.23%	6.7
10MB email	15856	27.07%	6.5
2MB email	5970	24.3%	6.4

Each vocabulary list was compared with a comparable sized list drawn from head of the BNC words, sorted in decreasing order of frequency. The email database was separated by user, allowing me to isolate emails in a single user's mailbox, and examine mailboxes of varying size. By combining several portions of these corpuses, it was easy to generate a vocabulary list of 2^{15}. This is sufficient to create a 2^{75} set of 6-word passphrases or a 2^{128} set of 9-word passphrases

I think the most surprising result is the low level of overlap of all datasets with a similarly sized dictionary drawn from the BNC. I would suspect that the program is too accepting of garbage input, but inspection hasn't shown that to be the case. Truncating the tail of the frequency distribution, both per-file and across the whole corpus, seems to filter out outliers well.

A better indication that we aren't simply reading 70% malformed words is that even the books, which we know are composed of well-formed English words, only had 42% commonality with a equal size dictionary drawn from the BNC.

This would indicate that an individual's effective vocabulary as measured by this textual analysis is very different from word frequency as measured by an aggregate examination of the population at large. Moreover, the low level of overlap indicates that we can effectively pad our locally generated list with words from a shared dictionary.

Source code provided a surprising wealth of words, although I was unsuccessful in using language-specific lexers to inject operators into the vocabulary. Although it might be interesting to use

an applicative functor in your passphrase, there just aren't enough language-specific operators to significantly shift the size of this vocabulary. However, method and variable names provided a wealth of non-dictionary words. In retrospect, this makes sense, as it is common to stitch together variable names out of abbreviations, prefixes, and suffixes while trying to keep them short yet memorable.

I expected that romanization of Chinese would provide a large body of alternate tokens, but this proved not to be the case. The highest level of HSK proficiency tests for 5000 words, composed from only 2663 characters. Since there are some collisions when anglicizing Chinese into pinyin, this resulted in only 2016 distinct tokens. Since each of these tokens is 2-5 characters long, composing them back into words would result in tokens longer than the average English word. It should not be that surprising that a non-native representation of a foreign language in a Roman alphabet would not be as dense as English (or another latin language), but this could be an avenue for further exploration.

VIII. Further Directions

Having demonstrated that it is feasible, with a generated vocabulary list of approximately $33,000$ words to create a 2^{75} set of 6-word passphrases or a 2^{128} set of 9-word passphrases, I think a good next step would be to recreate the user studies of [5,6] with these passphrases.

As for this method and the software, a good next step would be to expand the files and vocabularies it can recognize. Assuming that passwords are constrained to ASCII, the biggest barrier to parsing foreign languages is formalizing the unicode to ASCII translation. This ranges from as simple as stripping accents from latin languages, to the difficult task of converting unicode logograms to their corresponding Romanization. For this project, I converted a dictionary that expressed Chinese characters in the form $[a-z]^*[1-5]$, but this ASCII-friendly format is uncommon. The software package is setup to read unicode, but I was unable, within the scope of this project, to develop a functional translation of Chinese logograms or accented pinyin into an ASCII format.

References are important to the reader; therefore, each citation must be complete and correct. If at all possible, references should be commonly available publications.

References

[1] S. N. Porter. A password extension for improved human factors. Computers and Security, 1(1), 1982.

[2] R. Munroe. xkcd: Password strength. https://www.xkcd.com/936/, 2012.

[3] J. Bonneau, C. Herley, P. C. van Oorschot, F. Stajano. Passwords and the Evolution of Imperfect Authentication. Communications of the ACM vol. 58 no. 7, July 2015 pp. 7887 H. Poor, An Introduction to Signal Detection and Estimation. New York: Springer-Verlag, 1985, ch. 4.

[4] B. Schneier. Schneier on security blog. https://www.schneier.com/blog/archives/2014/03/choosing_secure_1.html

[5] Ghazvininejad, Marjan, and Kevin Knight. "How to Memorize a Random 60-Bit String." Parking 11.70.8: 58-83.

[6] Shay, Richard, et al. "Correct horse battery staple: Exploring the usability of system-assigned passphrases." Proceedings of the eighth symposium on usable privacy and security. ACM, 2012.

[7] Test Your Vocabulary http://testyourvocab.com/blog/2013-05-10-Summary-of-results