Practical Passphrase Cracking

Luc Gommans
Radically Open Security

January 10, 2018

1 Introduction

Most of user data is password protected, but the choice of these passwords is largely left to users. Password recommendations propose using hard-to-remember patterns in order to make it harder to guess, while in reality many of those 'random' tweaks (such as a number or exclamation mark at the end) are easily iterable by a computer.

Using passphrases, users can approach the problem differently: form a string of random, common words. The most famous example is 'correct horse battery staple': four words with no coherence, but many people in the field can recall this phrase without ever trying to memorize it[1]. Studies also show that users have fewer difficulties when memorizing a passphrase than when memorizing equally-strong random passwords[2][6].

Labrande[3] showed that at least 4 million in a set of 139 million passwords (2.9%) consist of phrases from publicly available sources. The exact percentage of passphrases used as passwords is not known, but this shows they are used by a lof of users while they have been studied relatively little.

Password cracking tools are widely available, which help us understand the strength of them. For passphrases, there is no commonly used approach using a single tool or dataset, probably because there are very few of them.

In this work, we will attempt to create a program which generates likely passphrases. It can be used in combination with traditional tools such as Hashcat or John the Ripper. We will use a probabilistic method to create realistic sentences and use rulesets of the traditional tool to do additional manipulation such as uppercasing the first word, removing spaces, and other operations that can be done using the optimized code of that tool.

2 Related Work

Sparell and Simovits[4] created a passphrase cracking program by using Markov chains on a word level. Markov chains are used in many applications, such as next word prediction in keyboards on smartphones. Similarly, one can use them to approximate the next most likely word in a passphrase given a few starting words (which could be a dictionary walk). In Sparell and Simovits' work, the LinkedIn hashdump of 2012 was used as one of the test sets. This contains mostly passwords (not phrases), but is still frequently used due to its large size of 64 million hashes. Of these, 21 000 phrases were cracked of between 10 and 20 characters in length.

In previous work, we[1] generated passphrases from quotes of various well-known persons and song lyrics. Using this method on the LinkedIn set, we recovered 92 000 passphrases between 15 and 67 characters. A relatively large amount of time was spent on collecting this data, which we think could be optimized in future work. We also looked into whether users incorporate personal data, such as lyrics specifically from a band they enjoy, but we have not found this to be the common case.

Labrande[3] attempted to crack hashes from the Korelogic dump (139 million entries) used a compilation of different public sources. Around 4.3 million passwords were cracked, of which 200 000 are 16 characters or longer. Public sources include quotes from WikiQuote and text written in bold and italics on Wikipedia.

To summarize, the problem of cracking passphrases is generally approached by either using a large set of phrases directly, or by training a model on a large set and using those to derive phrases which are likely to be logical.

3 Research Questions

Our main research question is: *How can software efficiently generate likely passphrases, to be used in passphrase cracking?*

To achieve this, we will use a probabilistic method which generates phrases in decreasing likelihood. An important factor in this design is the speed at which it can do this: most methods are not designed to iteratively list all possibilities in decreasing likelihood, which makes it difficult to do so using methods such as context-free grammar (as described by Weir[5], p. 84–85).

From this follow more questions to answer our main question:

1. Which probabilistic methods can be used to generate output in decreasing order of likelihood?
2. Which probabilistic method produces the most likely passphrases?
3. How much data, both in terms of storage and random access memory, does this method require?

Our questions focus on probabilistic methods because those do not require users to download or even compile large lists of passphrases to use directly. Instead, a model can be made available which contains probabilities for a certain language.

4 Scope

The software to be created will interface with at least one common password cracking tool, such as Hashcat or John the Ripper, and should be easy to use in concert. It is considered out of scope to implement cracking in the software itself, since that functionality can well be provided by other software.

The software will work for a general population and cannot be primed with facts such as birth dates of specific users. In our previous work[1], we found this to be not as effective as expected. It seems better to focus on the general method in this limited-time project.

5 Requirements

For this project, a software prototype will be built which might use a medium-sized dataset to operate. We estimate that a system with 4GiB RAM and 10GiB persistent storage is required. This is supplied by Radically Open Security.

For software, we expect to use only free and open source products.

6 Ethical Considerations

In this investigation, we will need to evaluate the prototype. We will do this using passwords from publicly available datasets. In all cases, the usernames will be removed and the only information kept is the password (or a hash thereof) and how frequently it occurred in the dataset.

7 Planning

Week 1	Write proposal, choose probabilistic method
Week 2	Early prototype, interfacing with password cracking software
Week 3	Improving and evaluating prototype
Week 4	Finalise prototype and paper, prepare presentation

References

[1] Luc Gommans, Dirk Gaastra, and Lennart van Gijtenbeek. *Practical Passphrase Cracking*. 2017. URL: https://github.com/lgommans/passphrase-cracking.

[2] Mark Keith, Benjamin Shao, and Paul Steinbart. *A Behavioral Analysis of Passphrase Design and Effectiveness*. 2014. URL: https://www.researchgate.net/profile/Mark_Keith2/publication/220580400_A_Behavioral_Analysis_of_Passphrase_Design_and_Effectiveness/links/0c960522109fa48244000000.pdf.

[3] Hugo Labrande. *Crack Me I'm Famous: cracking weak passphrases using publicly-available sources*. URL: https://www.sstic.org/media/SSTIC2015/SSTIC-actes/crack_me_im_famous_cracking_weak_passphrases_using/SSTIC2015-Article-crack_me_im_famous_cracking_weak_passphrases_using_freely_available_sources-labrande.pdf.

[4] Peder Sparell and Mikael Simovits. *Linguistic Cracking of Passphrases using Markov Chains*. 2015. URL: http://www.simovits.com/sites/default/files/files/PederSparell_Linguistic_Cracking_of_Passphrases_using_Markov_Chains.pdf.

[5] Charles Matthew Weir. *Using Probabilistic Techniques to Aid in Password Cracking Attacks*. 2010. URL: http://diginole.lib.fsu.edu/islandora/object/fsu:175769/datastream/PDF/view.

[6] Jeff Yan et al. *Password Memorability and Security: Empirical Results*. 2004. URL: http://www.lancaster.ac.uk/people/yanj2/jyan_ieee_pwd.pdf.