The Pennsylvania State University

The Graduate School

College of Engineering

PITFALLS IN THE AUTOMATED STRENGTHENING OF PASSWORDS

A Thesis in

Computer Science & Engineering

by

David Schmidt

Submitted in Partial Fulfillment
of the Requirements
for the Degree of

Master of Science

December 2013

The thesis of David Schmidt was reviewed and approved* by the following:

Trent Jaeger
Professor of Computer Science & Engineering
Co-Director: Systems and Internet Infrastructure Security
Thesis Advisor

Patrick McDaniel
Professor of Computer Science & Engineering
Co-Director: Systems and Internet Infrastructure Security

Lee Coraor
Associate Professor of Computer Science & Engineering
Chair of Graduate Program of Department of Computer Science and Engineering

*Signatures are on file in the Graduate School

ABSTRACT

Passwords are the most common form of authentication for computer systems, and with good reason: they are simple, intuitive and require no extra device for their use. Unfortunately, users often choose weak passwords that are easy to guess. Various methods of helping users select strong passwords have been deployed, often in the form of requirements for the minimum length and number of character classes to use. Alternatively, a site could modify a user's password in order to make it more secure; strengthening algorithms have been proposed that extend/modify a user-supplied password until achieving sufficient strength. Researchers have suggested that it may be possible to balance password strength with memorability by limiting automated changes to one or two characters while evaluating the generated passwords' strength against known cracking algorithms. This thesis shows that passwords that were strengthened against the best known cracking algorithms are still susceptible to attack provided the adversary knows the strengthening algorithm. Two attacks are proposed: (1) by strengthening public password sets with the known algorithm, which increases the percentage of recovered passwords by a factor of 2-5, and (2) by a brute-force attack on the initial passwords and space of possible changes, recovering all passwords produced when a sufficiently weak initial password was suggested. As a result, Kerckhoffs's principle is not satisfied with respect to these automated password strengthening systems.

TABLE OF CONTENTS

LIST OF ALGORITHMS

LIST OF TABLES

ACKNOWLEDGEMENTS

I would like to thank the following:

- The Department of Computer Science & Engineering, for agreeing to re-admit me into the Master's program 23 years after I left my graduate studies.
- Gregory Ference and Megan Heysham, with whom I partnered on a preliminary project. Our work on that project – an automated password strengthening system – spawned the ideas contained herein.
- My advisor, Dr. Trent Jaeger, who provided needed and much appreciated advice, guidance and support throughout the process. I am fortunate to have registered for the last spot in his class – without that bit of serendipity I would have been unaware of how interesting and complex a seemingly mundane topic like passwords can be.
- Most of all, my heartfelt thanks go to my wife, Jen, for steadfastly standing by my side and encouraging me to complete my Master's degree. Without her support, this thesis would have been impossible to complete.

Chapter 1

Motivation and Background

Passwords are the overwhelmingly predominant means for user authentication on computer systems, with the choice of password almost always left up to the user. The user is confronted with two opposing goals: first, the password should be easy to remember (functionality) and second, the password should be hard for anyone else to guess (security). Unfortunately, users tend to underestimate the importance of security, which makes their passwords susceptible to guessing algorithms (Bonneau, Herley, van Oorschot, & Stajano, 2012).

To motivate users to consider the security of their passwords, many websites utilize password meters to provide feedback regarding the relative security of their passwords. Other sites suggest alternative passwords that are closely related to the original password, perhaps within 1 or 2 edits, and that are judged to be adequately secure. Whatever mechanism is used, the metric used to gauge password strength is clearly of the utmost importance. If the metric overestimates the number of guesses required for an adversary to learn a password, then the utility of the tool is compromised.

As a result, researchers and adversaries have long studied the effectiveness of password guessing techniques, whose results have subsequently been applied to guide users to choose more secure passwords. A study in the 1970s identified several commonly used guessing techniques, such as dictionary words, common proper nouns, and common numbers (Morris & Thompson, 1979). More recently, researchers have developed guessing algorithms based on Markov chains (Narayanan & Shmatikov, 2005) and probabilistic context free grammars (Weir, Aggarwal, d. Medeiros, & Glodek, 2009) (PCFGs), which take advantage of non-uniformity in character sequences and the predictability of password structures, respectively. Researchers have adopted

these algorithms as metrics for password strength. Using estimated time to crack is arguably the only metric of interest to the user as it directly answers the question of "How secure is my password?".

As a result, researchers have produced automated methods for improving password strength by inserting a small number of characters into a user-selected password until the strength exceeds a limit as measured by the PCFG guessing algorithm (Forget A. , Chiasson, van Oorschot, & Biddle, 2008), (Forget A. , Chiasson, van Oorschot, & Biddle, 2008), (Houshmand & Aggarwal, 2012). For example, simple, user-chosen passwords, such as "life45!", are changed automatically through the addition and/or replacement of characters to passwords such as "life^45!" whose strength against password guessing surpasses a threshold using such metrics.

However, the claimed impact of a small number of changes to formerly weak passwords raises concerns that possible vulnerabilities in such methods are being overlooked. This paper illustrates that even when a sophisticated strength metric is used, it is nonetheless possible that algorithmically strengthened passwords are susceptible to attack. This is particularly true if the number of changes made to the original password is limited to one. In this case, the number of possible alternatives to the user's given password will be relatively small and therefore, to some extent, the alternative passwords will have some predictable properties.

For the purposes of this paper, it is assumed that an attacker knows the strengthening algorithm or can determine it from repeated use, analogous to Kerckhoff's principle. Under this assumption, there are two basic ways in which an attacker might proceed. First, the attacker might try to build a library of passwords which are statistically similar to the strengthened passwords; this library could then be used as the basis for a PCFG-based attack. Alternatively, a guided brute force approach could be taken, with the attacker guessing the original, weak password and then trying all strengthened variants. The efficacy of both approaches is

investigated in this paper. It is shown that both approaches can be highly effective in guessing attacks, unless the strengthening algorithm is aware of its limitations and takes some precautions.

This thesis makes the following contributions:

- In Chapter 5, Resistance to PCFG-based Attacks, it is demonstrated that an adversary who trains a password cracking program on passwords generated by applying the strengthening algorithm to a publicly available password list increases the percentage of recoverable passwords by 2 to 5 times.
- Guided brute force guessing is analyzed in the Resistance to Brute Force Attacks section of Chapter 5, and shown to be effective and practical unless the user-suggested passwords are required to meet an initial strength threshold.
- The impact of leaking data used in password strengthening is evaluated in Chapter 6. It is demonstrated that the leakage of data that might typically retained by a strengthening system would enable an adversary to crack 25% of the passwords in less than a day. However, if the retained information about the system's actual passwords is purely statistical, then leakage of the strengthening data provides adversaries no advantage for recovering passwords, other than a minimal impact on the quality of strengthening.

The outline of this thesis is as follows. This chapter continues with an introduction to state-of-the-art password crackers and a discussion of various approaches used in password strengthening; previous work in these areas is discussed in Chapter 3. Chapter 4 discusses the data used, and the algorithms used to both crack and adaptively strengthen passwords. In Chapter 5, the effectiveness of two different methods of attacking strengthened passwords is analyzed, while Chapter 5 looks at the implications of the data within the strengthening system being leaked. Lastly in Chapter 7, the results are summarized and directions for future work are given.

Chapter 2

Current Approaches to Cracking Passwords

It is well known that users tend to create weak passwords (Florencio & Herley, 2007), (Yan, Blackwell, Anderson, & Grant, 2004). Adams and Sasse point out that this is to be expected as users are typically unaware of what makes a password more secure (Adams & Sasse, 1999). The net effect is that the process of systematically guessing a user's password is made much easier than naive statistical analysis would suggest, as the space of the probable passwords is many orders of magnitude smaller than the space of all possible passwords.

Dictionary Attacks

The most basic password cracking algorithm is to gather a dictionary of probable passwords based on leaked password lists and common dictionary words, and to iterate through this list for password guesses. While this approach is quite fast, it is also quite limited in its ability; the addition of a single digit or a difference in capitalization between the password and the dictionary makes the password invulnerable to guessing. To make this approach more effective, the publicly available John the Ripper (JtR) program (Peslyak) operates off of a user-specified dictionary and, optionally, additional "mangling" rules. These rules can take the form of adding a digit to the end of each word in the dictionary, capitalizing the first letter, and/or performing common substitutions such as "0" (zero) for "O" /"o" (oh) or "@" for "A"/"a".

While these rules do make a dictionary attack more effective, the approach is still fairly limited as the basis of the password to guess must be in the list – if "troubadour" is not in the dictionary, then the passwords "Troubador7" or "tr0ub@d0ur" will not be guessed. Further, even

if "troubadour" is in the dictionary, if there is no mangling rule which would test "troubadour#1", then that password is safe from guessing.

Markov Chains

Narayanan and Shmatikov describe a password cracking algorithm based on Markov models (Narayanan & Shmatikov, 2005). This approach exploits the fact that the distribution of character sequences within a language is far from uniform, and the distribution of character sequences within passwords is likely to follow this same distribution. For example, in English, "w" is more likely to be followed by "h" than by another "w"[1]. The concept can be extended to character sequences of arbitrary length, called n-grams; the 3-gram (or tri-gram) "tch" is far more likely to be followed by a vowel than by a "p"[2]. By training the Markov chain on known password lists, dictionaries, or both, these distributions can be estimated and used to generate a list of possible passwords that is significantly more effective than random guessing or JtR. Password composition protocols such as requiring an uppercase letter, a digit and a symbol in the password are meant to reduce the effectiveness of both Markov chains and dictionary attacks. A more in-depth discussion of how Markov chains can be applied to password cracking can be found in (Marechal, 2008).

Probabilistic Context-Free Grammars

A context-free grammar $G = \{V, \Sigma, S, P\}$, consists of a finite set of variables V, a finite set of terminal characters Σ, a start variable S, and a finite set of productions P. Each production is of the form A→ B, where A is a single variable and B is a string containing any combination of

[1] Double w's occur in compound words such as "glowworm".
[2] "Catchphrase" is an example of the latter.

variables and terminals. The language of the grammar is the set of strings that can be derived by starting with S and using any number of productions until only terminals remain.

A probabilistic context-free grammar (PCFG) is a context free grammar with a probability assigned to each production so that for every variable, the productions for which that rule is on the left-hand side have probabilities that sum to 1. The probability of a derivation is the product of the probabilities of the productions used in the derivation.

Weir, et al. proposed using a probabilistic context-free grammar (PCFG) for password cracking (Weir, Aggarwal, d. Medeiros, & Glodek, 2009). This algorithm is based on the observation that passwords tend to have predictable "structures". The structure of a password is defined as the way in which the password can be broken into strings (or tokens) of letters, digits, and symbols (e.g. $\mathbf{S_1U_1L_6D_2}$ represents a special character followed by an uppercase letter, then 6 lowercase letters, and ending with two digits). Attacks using PCFGs are highly effective because when a password composition policy requires that a digit or symbol be included in a password, users are far more likely to append the required character to an existing password rather than place it in the middle. Similarly, uppercase letters are predominantly used at the beginning of an alphabetic string. By tabulating the number of occurrences of each distinct structure found in a training set, an attacker can gain valuable insight into the likely distribution of structures of the passwords. Thus, a PCFG-based attack is a counter-measure to password composition policies, and such an attack can be markedly more effective than an attack based solely on a Markov chain.

Importantly, the effectiveness of a PCFG-based attack is dependent upon the statistical distribution of structures within the password database. The more passwords are concentrated into a relatively small number of structures, the more effective a PCFG is, because its guessing is concentrated in the structures most likely to be correct. Similarly, the closer the structures are to equal probability, the less effective a PCFG will be.

Dell'Amico, Michiardi and Roudier use a variety of guessing attacks to determine which type of attack would be most effective for a given composition policy (Dell'Amico, Michiardi, & Roudier, 2010). They used JtR, PCFGs, and Markov chains using 1- to 5-grams. They found that for cracking weak passwords, a simple dictionary search such as JtR is the most efficient. After the dictionary attack had been exhausted, PCFGs were the most efficient for finding the next set of passwords. For the toughest passwords, Markov chain techniques were the most powerful. The use of longer chains proved most effective in the early going, but the use of longer n-grams was more efficient only to a point. Past a certain number of guesses, the longer n-grams were unable to efficiently crack new passwords, and a chain using a shorter n-gram was needed. In the end, using a chain based on single characters was the most efficient tool for the toughest passwords.

Fighting Back: Strengthening Algorithms

A strengthening algorithm requires a measuring system to rate how hard a password is to crack. Password meters, commonly used to graphically depict to a user how strong or weak their password is, employ this same concept. As defined by Castelluccia, et al., a password strength metric is a function that takes a string (password) and outputs a positive real number score *s*, such that the higher the score, the stronger the password (i.e., harder to crack) (Castelluccia, Duermuth, & Perito, 2012). Extending this concept, the authors define an adaptive strength metric as a function that takes a list of previously received passwords and the user's password as inputs and outputs a score *s*. Each password presented to the adaptive password meter would be included in the list of previously received passwords on the next invocation so that the scoring is truly adaptive.

Since both Markov chains and PCFG-based algorithms use known password lists to fine-tune the distribution of character sequences and structures, it makes sense that a password scoring system should also use this information when calculating the work required to successfully guess a given password. Since password distributions are different from website to website, due to differences in password composition policies and other factors, a training database that works well for one website may not be suitable for another website (Castelluccia, Duermuth, & Perito, 2012). Adaptive scoring algorithms, because they can see the current passwords, or at least the distribution of relevant statistical properties of the passwords, allow the algorithm to give more accurate scores for different websites.

Processes for automatically strengthening passwords use an adaptive approach as described above. If a password does not generate a sufficiently high score from the scoring algorithm, one or more changes to the user's password are made and the new, altered password is scored. If the process is successful, one or more altered (strengthened) passwords are presented to the user for selection. In the event the user's password cannot be made sufficiently strong given the editing rules, the user would have to start over. The initial password and any strengthened passwords are added to whatever database the scoring algorithm requires for its use. This keeps the process adaptive, and helps to reduce the prevalence of common constructs, which is discussed by Bonneau, et al. as a method of improving security (Bonneau, Just, & Matthews, What's in a Name? Evaluating Statistical Attack on Personal Knowledge Questions, 2010).

Chapter 3

Related Work

There is a large and growing body of literature on the insecure password choices that users tend to make: (Adams & Sasse, 1999), (Florencio & Herley, 2007), (Bonneau, The science of guessing: analyzing an anonymized corpus of 70 million passwords, 2012), (Yan, Blackwell, Anderson, & Grant, 2004), (Garrison, 2006). This tendency is understandable as users must balance the competing goals of security and memorability (Forget, Chiasson, & Biddle, Helping users create better passwords: is this the right approach?, 2007), (Yan, Blackwell, Anderson, & Grant, 2004). Passwords that are more random (through composition policies or system generation) or contain more characters are harder for password crackers to guess (Kelley, et al., 2012). However, users have trouble remembering random or complex passwords (Forget & Biddle, Memorability of persuasive passwords, 2008), (Leonhard & Venkatakrishnan, 2007) and resort to insecure workarounds, such as writing down the password or following predictable patterns to meet password requirements (Inglesant & Sasse, 2010), (Komanduri, et al., 2011), (Shay, et al., 2010), (Weir, Aggarwal, Collins, & Stern, 2010), (Bonneau, The science of guessing: analyzing an anonymized corpus of 70 million passwords, 2012), (Shay & Bertino, A comprehensive simulation tool for the analysis of password policies, 2009). Garrison shows that educating users on good password choices can be helpful in increasing security (Garrison, 2006).

Mnemonic passwords, such as those based on the first letters of words in the sentence or phrase, can increase the number of character classes, while maintaining memorability (Vu, Proctor, Bhargav-Spantzel, Tai, Cook, & Eugene Schultz, 2007). Thus, “Sunday, bloody Sunday. How long? How long must we sing this song?” might become “S,bS.Hl?Hlmwsts?”. While the approach does generate strings that appear “more random”, users have a tendency to choose song lyrics or other phrases readily found on the internet as their mnemonic. Consequently, a

dictionary of common phrase-based passwords and corresponding mnemonics can be built, limiting the security gain (Kuo, Romanosky, & Cranor, 2006).

Similar to mnemonic passphrases, Shay, et al. investigated the use of simple passphrases – sequences of 3-5 words, chosen at random by the website, and presented to the user for acceptance or selection of another passphrase (Shay, et al., 2012). In this study, the authors found that even when the generated passphrase was grammatically correct – for instance, by following simple structures such as [noun, verb, adjective, noun] – users had trouble remembering the phrase and resorted to writing it down. As a result, the authors found that when compared to computer-assigned passwords of equivalent strength, there were no usability benefits to passphrases.

Password composition policies such as requiring a digit, or a symbol and an uppercase letter are very commonly used techniques to try and increase the security of user's passwords. Komanduri, et al. found that (a) to alleviate user frustration, the verification system needs to explain why a password was rejected and (b) a large percentage of users made incremental changes to their password in order to satisfy the composition requirements (Komanduri, et al., 2011). The latter finding was echoed in work by Weir, et al., and Shay, et al., where both sets of authors document that users tend to modify old passwords in order to meet composition policies, rather than creating new passwords (Weir, Aggarwal, Collins, & Stern, 2010), (Shay, et al., 2010). Kelley, et al. report that the composition policies of `basic16` and `comprehensive8` exhibit the best resistance to cracking attacks (Kelley, et al., 2012). The first policy allows any password of length 16 or longer. The second policy mandates that the password be at least 8 characters long, contain all four character classes, and, after stripping out the digits and symbols, the remaining letters cannot spell a word.

However, composition policies tend to be ineffective due to the strong tendency of users to select memorable passwords. Wier, et al., find that policy mechanisms are hampered by

circumvention strategies employed by users (Weir, Aggarwal, Collins, & Stern, 2010). For example, if the policy stipulates that passwords must contain three digits, users will tend to simply add "123" to the end of their existing, insecure password. Similarly, when examining a publicly available set of leaked passwords, the authors find that of the passwords that were seven characters long and contained at least one digit, 64% of the sample had the digit(s) at the end. Testing symbols rather than the digits, the results were better, but 28% of the sample was a string of alpha characters followed by a single symbol. Knowledge of these patterns can aid an attacker, muting the effectiveness of the policy to protect passwords from automated guessing attacks.

Another effort to thwart online guessing attacks is with the use of a Reverse Turing Test (RTT), a CAPTCHA or similar mechanism (Pinkas & Sander, 2002), (Xu, Reynaga, Chiasson, Frahm, Monrose, & Van Oorschot, 2012). The RTT would be presented when the password is initially created and then only on a small fraction of a user's login attempts or after an incorrect password was entered. This limits the inconvenience to the user of CAPTCHA while simultaneously slowing down a guessing attack. This approach may not be scalable, however, as the authors acknowledge.

Password strength meters are also commonly used by websites to assist users with creating more secure passwords. Typically, the strength meter either follows static rules such rewarding users for simply adding a digit or symbol, or uses a measure of entropy for strength. As documented in multiple studies, NIST measures of entropy are deeply flawed as a resistance-to-guessing measure, significantly overstating the security of some passwords and significantly understating the security of others (Weir, Aggarwal, Collins, & Stern, 2010), (Ma, Campbell, Tran, & Kleeman, 2010) and (Bonneau, The science of guessing: analyzing an anonymized corpus of 70 million passwords, 2012). Ur, et al. state that strength meters are effective in getting user to create longer passwords, but significant increases in resistance to guessing were only made with very stringent scoring, which users found very frustrating in practice (Ur, et al., 2012).

Automated methods for improving a user-selected password have also been suggested. One such approach is Persuasive Text Passwords (PTP), as described in (Forget A. , Chiasson, van Oorschot, & Biddle, 2008) and (Forget A. , Chiasson, van Oorschot, & Biddle, 2008). The basic idea here is for the user to enter their desired password, and PTP inserts 1-4 random characters at randomly selected positions. The random selection forces a break from the clustering of patterns seen in (Weir, Aggarwal, Collins, & Stern, 2010). However, PTP always inserts the same number of random characters into the user's password, irrespective of the strength of the original password. As a result, users who were subjected to more random inserts into their password chose less secure beginning passwords in order to compensate for the memory load. In turn, this reduces the effectiveness of the technique, with the authors finding that two randomly inserted characters was the best tradeoff between increases in security versus memorability.

Castelluccia, et al., use a Markov chain to estimate the strength of a user-supplied password and use that estimate of strength in an adaptive strength meter shown to the user (Castelluccia, Duermuth, & Perito, 2012). By building and maintaining the Markov chain with data from the actual passwords at the site, the meter automatically adjusts to any cultural or site-specific tendencies that might cause passwords to cluster. For instance, passwords from a religiously-oriented site and a dating site could be expected to have different tendencies and themes, which an adaptive meter would detect to steer users towards more differentiated passwords.

Houshmand and Aggarwal combine these ideas and describe a method to automatically strengthen user passwords by applying a set of editing rules to the original passwords and measuring strength with a PCFG database obtained by training on a previously leaked set of passwords (Houshmand & Aggarwal, 2012). They state that a minimal edit of only one character can be effective in terms of strengthening a password.

The work in this thesis extends the above research in the following ways. A password-scoring mechanism such as that described in (Castelluccia, Duermuth, & Perito, 2012) or (Houshmand & Aggarwal, 2012) is assumed to be in use. This mechanism would adaptively score a newly presented password in the context of the other passwords already seen at a site and, using that information, guide users to select strong passwords. Following the approaches of (Forget A. , Chiasson, van Oorschot, & Biddle, 2008), (Forget A. , Chiasson, van Oorschot, & Biddle, 2008) and (Houshmand & Aggarwal, 2012), an automated approach to strengthening passwords is taken, with a well-defined set of editing rules that may be applied to a password. However, previous work has not examined the ability of strengthened passwords to be guessed by either an alternative algorithm or by attempting to build a PCFG training set with statistical properties similar to the strengthened passwords. Results are presented which show that a large percentage of passwords strengthened and scored using a sophisticated algorithm like PCFG can be guessed by precisely these means. Thus, the apparent success of some strengthening algorithms may be illusory. The analysis continues with a look at the security implications of an accidental leak of the data used in the strengthening process and the thesis concludes with some suggested guidelines for strengthening algorithms and directions for future work.

Chapter 4

Algorithms for Guessing and Strengthening Passwords

This chapter describes the data used in the analyses presented within this thesis, as well as the formula used to compute password strength (GP). The algorithm used to guess passwords is presented and analyzed with respect to several implementation decisions. The strengthening algorithm is also presented, and the chapter concludes with a description of the strengthened password sets which are used in the subsequent analyses.

Data Used

Multiple leaked password lists exist in the public domain. Using actual passwords lists such as those found at http://www.skullsecurity.org/wiki/index.php/passwords (Skull security) is the best, if not only, way to study password distributions and cracking algorithms in a large scale manner. This paper uses the passwords from the rockyou website which is the largest set available. Rockyou imposed no composition policy on the passwords – passwords which contained only lowercase letters or only digits were allowed, for example.

In order to keep the password set large, the only constraint imposed by this work was that a password needed to be at least eight characters long, which reduced the size from 14M to 9.5M. Since the assumption is that a strengthening system is in place, stringent initial constraints on the user's password should be unnecessary. Either the system can successfully strengthen the user's password or it cannot, and it informs the user of this fact. For this study, passwords which cannot be successfully strengthened are simply discarded and removed from the analysis pool.

In order to permit out-of-band testing, the rockyou set was randomly divided into two sets, rockyou-1 and rockyou-2. In turn, both sets were divided into subsets A, B and C in order to

bootstrap the strengthening algorithm, as discussed in the section Making Passwords Stronger – the Strengthening Algorithm.

Measuring Password Strength

To measure password strength, a program to calculate the "guess probability" (GP) of a password was implemented. Following the approach of Weir, et al., a PCFG was utilized, and the GP was computed as the product of the base structure (i.e., $\mathbf{L_6D_2}$) probability and the probabilities of the strings which fill each variable (i.e., $\mathbf{L_6}$ and $\mathbf{D_2}$), with the strings selected from an input dictionary (Weir, Aggarwal, d. Medeiros, & Glodek, 2009). If the dictionary were the only source of strings for guessing, then obviously only strings which are present in the dictionary could be guessed.

To avoid unnecessarily limiting what strings can be assigned probabilities and/or guessed, the GP calculation and guessing algorithm used in this thesis employ both a dictionary and a Markov chain. Conceptually, the Markov chain is a comprehensive dictionary as the chain can produce any string from the characters within its domain. For example, if the Markov chain is built only from digit strings, it can produce all 10^n digit strings of length *n*, together with their associated probabilities.

Specifically, four Markov chains are built from the training data: a 1-gram chain for symbols only, a 1-gram chain for digits only, a tri-gram chain for alphabetic strings of length 3 or longer and a 1-gram chain for alphabetic strings shorter than 3[3]. During training, the starting and transition probabilities for each chain are built. In addition to building these probability tables, the training algorithm also builds a dictionary by keeping track of every string seen, and tabulating

[3] Tri-grams were used for alphabetic strings throughout the analysis as they appear to be the most effective. However, a different size n-gram could have been used. Similarly, no advantage to the use of larger n-grams was found for the digit and symbol Markov chains.

the frequencies with which they occur. More precisely, when a token *T* of length *k* is encountered during training, Algorithm 1processes the token into the dictionary and Markov chain's probability tables. Within the algorithm, $T_{i,j}$ indicates the substring from characters *i* to *j*, inclusive.

Data: token T of length k, dictionary D, n-gram Markov chain
Result: updates the dictionary and Markov chain probability tables
Increment the total number of tokens seen; // step 1
Increment the number of tokens of length k seen; // step 2
if *T is not in D* **then**
  add T to D; // step 3
end
Increment the tally for T in D; // step 4
Increment count of $T_{1,n}$ beginning a token; // step 5
for *j from 1 to k-n* **do**
  Increment the transition count from $T_{j,j+n-1}$ to $T_{j+1,j+n}$; // step 6[4]
end

Algorithm 1: Token Processing

As will be shown, the dictionaries significantly increase the efficiency of the guessing algorithm. The purpose of the dictionaries is to identify those strings that appear more frequently than the product of the corresponding transition probabilities would indicate; for instance, "love", "123", and its shifted counterpart "!@#" are all very common sequences.

The alphabetic Markov chains are case-insensitive as the case information is already contained within the password structures. Thus, the alphabetic Markov chains compute the same probability for any particular word, independent of the pattern of upper- and lower-case letters. The pattern of upper- and lower-case letters does impact the probability of the structure – for example, $\mathbf{U_1L_7}$ and $\mathbf{L_4U_2L_2}$ will have different probabilities even though both represent 8-letter words – and so the password's GP is impacted by the case of the letters (cf. Equation 2).

[4] The number of occurrences of all starting n-grams and all transitions are initialized to 1, not 0, so that all computed probabilities are non-zero.

With that background, the GP for a token T of length L is computed as:

Equation 1: Guess Probability of a token T

$$GP_{MC}(T) = \max\{ObservedFrequency(T), BP(T_{1,n}) * \prod_{n}^{L-1} TP(T_{i-n+1,i}, T_{i-n+2,i+1})\}$$

where $T_{i,j}$ is again the substring from characters *i* to *j* (inclusive), *BP* is the probability a token begins with a given *n*-gram (cf. step 5, Algorithm 1) and *TP* is the transition probability from one *n*-gram to another (cf. step 6). For common tokens, GP_{MC} evaluates to *ObservedFrequency(T)*, which is the ratio of the number of occurrences of *T* to the total number of strings seen of that same length (cf. steps 4 and 2, respectively). For uncommon or previously unseen strings, GP_{MC} is computed using the probabilities within the Markov chain. For example, given the token "troubador", the algorithm multiplies the probability that a string starts with "tro", the probability of transitioning from "tro" to "rou", and so on. As shown, all of the probabilities referenced are computed from the training set[5].

The strength calculator combines the probabilities from the dictionaries, Markov chains and PCFG to compute the GP of password *PW* as:

Equation 2: Guess Probability of a Password

$$GP(PW) = SP(PW) * \prod_{s \in SS(PW)} GP_{MC}(s)$$

where *SP* is the observed probability of the password structure (e.g., $\mathbf{L_4S_2D_1L_4}$) within the PCFG, and *SS* returns substrings from the password based off the structure (i.e. "pass**1word" would return {"pass", "**", "1", "word"}). As noted previously, the Markov chains are case-insensitive. Consequently, *SS* is also case-insensitive. Thus, if the password were "Pass=+=wORd414", the structure would be $\mathbf{U_1L_3S_3L_1U_2L_1D_3}$ and SS would return {"pass", "=+=", "word", "414"}.

[5] Since the GP_{MC}s do not sum to 1, they are technically not probabilities. However, "probability" captures the intuition of what is being measured.

Guessing Passwords

In order to determine real-world limits on what level of GP might be considered secure, the algorithm to compute GPs was used to create a password-cracking program. Algorithm 2 was used to generate all passwords at or below a threshold GP (*minGP*) using password structures, dictionaries and Markov chains as described in the section Measuring Password Strength.

Data: minGP, cracking data
Result: outputs all passwords successfully guessed
foreach *password structure SS* **do**
  use Algorithm 3 to try all passwords with structure SS and GP ≥ minGP;
end

Algorithm 2: Cracking Program

Algorithm 2 invokes the guess generating function (Algorithm 3) for each password structure. This initial call to Algorithm 3 uses an empty *currentGuess*, *SP(SS)* for *cumulativeGP* and the structure *SS*.

Data: currentGuess, cumulativeGP, password structure SS,minGP,list of valid passwords
Result: outputs all passwords successfully guessed
if *SS is empty* **then** // All components of the passwords are filled
  if *currentGuess is a valid password* **then**
    output currentGuess;
  end
else // Fill the next component of the password
  S = first element of SS; // If SS = $L_8D_1S_1$ then S = L_8
  MC = the Markov chain used to guess S; // digit, symbol or alphabetic chain
  foreach *word W ∈ MC's dictionary* **do** // try all the words in the dictionary
    if $cumulativeGP * GP_{MC}(W) \geq minGP$ **then** // if resultant GP not too small
      recursive call with $currentGuess + W, cumulativeGP * GP_{MC}(W), SS - S$;
    end
  end
  L = list of words W built from MC such that $GP_{MC}(W) \geq minGP/cumulativeGP$;
  foreach word W ∈ L **do**
    recursive call with $currentGuess + W, cumulativeGP * GP_{MC}(W), SS - S$;
  end
end

Algorithm 3: Guess Generating Algorithm

Whenever the guess generator (Algorithm 3) is called for a particular password structure (e.g., $\mathbf{L_8D_1S_1}$), it fills in the first component of the structure ($\mathbf{L_8}$) and makes a recursive call to fill the remaining parts of the structure ($\mathbf{D_1S_1}$). At each stage of both algorithms, selections are made in descending probability order: the most common password structures are tried first, and similarly for dictionary words and strings built from the Markov chain's probability tables. This guides the program to more likely passwords early on. Modifying Algorithm 2 to utilize multiple processors can be accomplished by splitting the password structures into P groups, where P is the number of processors to use. This approach to sharing the work load is both simple and effective and can easily be implemented on a distributed basis. Assuming an efficient method of splitting and distributing the password structures to the processors, if P is increased by a factor of K, run time would be reduced by that same factor. This is true only to a point, however: if there were as many processors as structures, the run time from processor to processor would vary greatly, depending on the structure assigned. Nonetheless, an attacker with a large network or botnet of PCs would be formidable.

Algorithm 2 and Algorithm 3 operate differently than the PCFG algorithm described by Weir, et al. (Weir, Aggarwal, d. Medeiros, & Glodek, 2009). Algorithm 2 guesses all passwords which can be built from a given structure *SS* and minimum GP (*minGP*) before moving on to the next password structure. In contrast, the algorithm described by Weir, et al. guesses passwords in descending probability order, using different password structures as the probabilities dictate. The difference is roughly analogous to depth-first search (Algorithm 2) versus breadth-first search (PCFG).

The number of guesses actually made and the run time for a range of GPs are shown in Table 4-1[6]. The PC used for these results has a 12-core Intel i7 CPU running at 3.20GHz. The run time reflects the elapsed clock time when all of the machine's 12 cores were deployed. Based

[6] Estimated times are based on a log-log regression.

on these results, a GP of 10^{-15} can be considered fairly secure. Consequently, in the remainder of this thesis, passwords will be strengthened to a GP of 10^{-16} to allow for an additional margin of safety.

Table 4-1. Run Times for Algorithm 2

Minimum GP	*# Guesses Made*	*Run Time, 12 cores*
10^{-9}	132M	15 seconds
10^{-10}	1.6B	2 minutes
10^{-11}	15.9B	16 minutes
10^{-12}	136.3B	2.2 hours
10^{-13}	1092.7B	17.3 hours
10^{-14}	---	6 days (est)
10^{-15}	---	1.5 months (est)
10^{-16}	---	1 year (est)

Impact of Implementation Decisions

As presented, Algorithm 2 and Algorithm 3 incorporate several important implementation decisions – in particular, how are strings to be generated when guessing and how case is handled when dealing with alphabetic strings. In the remainder of this section, the impact on efficiency resulting from these decisions is analyzed.

Dictionaries versus Markov Chains

As described, Algorithm 3 uses both dictionaries and Markov chains. In contrast, Weir, et al., used only dictionaries when introducing the use of PCFGs as a password cracking mechanism (Weir, Aggarwal, d. Medeiros, & Glodek, 2009). Houshmand and Aggarwal extend the original work on PCFGs with the use of probability smoothing techniques to assign non-zero probabilities to strings of digits and symbols. This allows digit strings and symbol strings which are not part of the dictionary to be used in cracking attempts; however, the alphabetic strings are selected solely from a static dictionary, which is not dynamically updated when new alphabetic strings are encountered (Houshmand & Aggarwal, 2012).

The hybrid approach of using both dictionaries and Markov chains within a PCFG-based algorithm is new, to this author's knowledge. In order to ascertain the merits of combining Markov chains into the guessing algorithm, it needs to be determined whether or not this hybrid approach increases the number of passwords which can be successfully guessed in the same period of time, when compared to other approaches. To this end, the rockyou-1 set was used as a training set for a cracking attack which tried to guess the passwords in rockyou-2. Table 4-2 shows the run times for a range of GPs together with the percent of passwords which were successfully guessed using four different cracking attacks:

- Dictionaries and Markov chains were used as described in Algorithm 3.
- Only dictionaries were used; no Markov chains.
- Only Markov chains were used; no dictionaries.
- Dictionaries were used and Markov chains were used for digit and symbol strings only. Alphabetic strings could only be guessed from the dictionary. This roughly approximates the probability smoothing technique of Houshmand and Aggarwal.

Table 4-2. Impact of the use of Dictionaries and Markov Chains on Guess Rates

	Dictionaries and Markov chains		Dictionaries only		Markov chains only		Dictionaries; Digit, Symbol chains	
Minimum GP	Percent	Run Time	Percent	Run Time	Percent	Run Time	Percent	Run Time
10^{-9}	25.4	15 sec	20.0	7 sec	8.2	7 sec	22.7	8 sec
10^{-10}	40.0	2 min	26.9	30 sec	23.2	1 min	32.4	1 min
10^{-11}	56.5	16 min	31.4	3 min	43.0	8 min	42.5	8 min
10^{-12}	69.0	2.2 hrs	34.5	21 min	59.0	1 hr	48.7	1 hr
10^{-13}	77.5	17.3 hrs	36.4	2.5 hrs	70.4	7.6 hrs	52.0	6.3 hrs
10^{-14} (est)	83.7	*6 days*	37.7	*13 hrs*	78.6	*3 days*	54.3	*2 days*
10^{-15} (est)	88.4	*1.5 mon*	38.7	*4 days*	84.6	*3 wks*	56.0	*2 wks*
10^{-16} (est)	91.8	*1 year*	39.4	*3 wks*	88.9	*6 mon*	57.2	*3 mon*

The use of dictionaries is clearly beneficial at the very lowest level of GP; roughly 2.5x as many passwords were recovered at a GP of 10^{-9} when using only a dictionary as compared to using only Markov chains. At a GP of 10^{-10}, the relative advantage is down to about a 16% premium, but the run time is only half as long. However, at tougher GP levels, the advantage goes to the Markov chain only approach. Using only dictionaries, less than 40% of the passwords could be successfully recovered in any time frame shown. In comparison, using only Markov chains, over 40% of the passwords were successfully cracked at a GP of 10^{-11} and in only 8 minutes.

Not surprisingly, compared to a dictionary-only attack, adding Markov chains for digit and symbol strings only is also clearly beneficial. At every GP level, more passwords can be

recovered when using digit and symbol Markov chains. Here too, however, beginning at a GP of 10^{-11}, more passwords can be recovered using only Markov chains than can be recovered using a dictionary and Markov chains for digits and symbols only, albeit at a somewhat longer run time.

The clear winner, however, is the hybrid approach of Algorithm 3. The use of a Markov chain for all types of strings, in addition to a dictionary, results in a higher number of passwords recovered at every GP level. While it is true that at any given GP level, the hybrid approach takes at least twice as long as any of the other strategies, the most salient comparison is the time to crack a fixed percentage of the passwords. With this metric, the hybrid approach is the superior strategy: in 16 minutes, Algorithm 3 can crack 56.5% of the passwords, a level that is far beyond what can be reached using only a dictionary. Using only Markov chains, recovering that many passwords would take close to an hour and it would take weeks if only digit and symbol Markov chains were used in conjunction with dictionaries.

Handling Upper and Lower Case Letters

Wier, et al., in their paper introducing PCFGs as a password cracking tool, only handled lowercase letters, and left the handling of mixed case strings to future research (Weir, Aggarwal, d. Medeiros, & Glodek, 2009). Thus, their original work had three variables which could occur within a structure: **L** for letters, **D** for digits and **S** for symbols. To avoid confusion with the use of **L** for a lowercase letter, a generic letter will be denoted by **A** (for alphabetic) instead.

Houshmand and Aggarwal incorporate case into the grammar with the use of case masks. A case mask is a string of *n* **U**s and **L**s which determines the pattern of upper- and lower-case letters within an alphabetic string of length *n*; for example, **ULLLL** is the case mask for a 5-letter word with only the first letter capitalized. During training of the grammar, the use of each case mask can be counted just like a digit or symbol string, yielding an observed probability for each

case mask which is the ratio of the number of occurrences of that particular mask to the total number of masks of that length. As before, probability smoothing techniques can be used to assign probabilities to masks which were not observed in the training data.

Compared to the approach of using explicit **U**s and **L**s in this work, case masks have the advantage of providing more comprehensive coverage of the possible password space. For example, suppose that during training the password "Base*ball" was encountered. If case masks were used, the structure $\mathbf{A_4S_1A_4}$ and case masks **ULLL** and **LLLL** would be recorded, whereas with the use of explicit **U**s and **L**s, only the structure $\mathbf{U_1L_3S_1L_4}$ is recorded. At this point, a grammar utilizing case masks could generate the password guesses "base*Ball", "base*ball" and "Base*Ball" because the case masks can be applied to any string of length 4. In contrast, a grammar that uses explicit **U**s and **L**s can only generate guesses with structures that have been observed in the training data; consequently, unless a password with the structure $\mathbf{L_4S_1U_1L_3}$ is observed, the guess "base*Ball" cannot be generated.

As a possible offset to this advantage of case masks is that their more comprehensive coverage must come at the price of longer search times, since the number of structure and case mask combinations must be at least as large as the number of structures observed when building structures with **U**s and **L**s rather than case masks. Additionally, it is possible that the probability of observing a particular case mask is dependent upon where it is in the password. Using the prior example, the use of case masks assumes that for the structure $\mathbf{A_4S_1A_4}$, the case mask **ULLL** has the same probability of occurring in the first alphabetic string as in the second. However, this assumption may be incorrect: if, for example, the first letter of the password is the most frequently capitalized, then the probability of seeing the case mask **ULLL** is different at the start of the password than in the middle.

Table 4-3 is similar to Table 4-2, showing the results when Algorithm 2 and Algorithm 3 are used as exhibited, and when the algorithms are modified to use case masks. As can be seen

the differences are fairly slight, with the use of case masks causing a modest decrease in the percent of passwords in rockyou-2 that can be guessed, at a small increase in the run time. Based on these results, the advantage appears to be with the explicit use **U**s and **L**s as done in this thesis; however, the advantage is sufficiently small that it cannot be deemed definitive and tests with different data may yield different results.

Table 4-3. Impact of using Case Masks on Guess Rates

	Explicit **U**s and **L**s		Case Masks	
Minimum GP	Percent	Run Time	Percent	Run Time
10^{-9}	25.4	15 sec	25.3	15 sec
10^{-10}	40.0	2 min	39.8	2 min
10^{-11}	56.5	16 min	56.3	18 min
10^{-12}	69.0	2.2 hrs	68.7	2.5 hrs
10^{-13}	77.5	17.3 hrs	77.2	19 hrs
10^{-14} (est)	83.7	*6 days*	83.4	*7 days*
10^{-15} (est)	88.4	*1.5 months*	88.1	*2 months*
10^{-16} (est)	91.8	*1 year*	91.5	*1.3 years*

Making Passwords Stronger – the Strengthening Algorithm

The basic strengthening algorithm used in this paper is presented as Algorithm 4. The algorithm references a *strengthening database*, which refers to the collection of password structures, Markov chains and dictionaries, together with their associated probabilities, as described in the section Measuring Password Strength. If these same items are used in an attack, rather than in a strengthening system, they are referred as a *cracking database*.

Note that not all passwords can be strengthened to the targeted GP (TGP). In an actual deployment, if the passwords could not be strengthened, the user would need to choose a different password. Here, however, the password is simply removed from the analysis pool.

The way in which the strengthening database is built is important, as will be seen in Chapter 6. Passwords in subset A (the original training data) are not strengthened and are fully processed into the strengthening database, meaning that all steps shown in Algorithm 1 are performed. In contrast, when subsets B and C are strengthened, both the original and strengthened passwords are only partially processed into the strengthening database, meaning that only the Markov chain's probability tables are updated, and the dictionaries are not; in other words, steps 3 and 4 in Algorithm 1 are not performed.

Data: password list X, threshold guess probability TGP, number of edits N
Result: strengthened password list X'
Create empty strengthening database SDB;
foreach *password PW* ∈ *subset A of X* **do**
  fully process PW into SDB;
end
foreach *password PW* ∈ *subset B and C of X* **do**
  PW' = PW;
  thisGP = GP(PW');
  while *thisGP > TGP and maximum number of attempts not exceeded* **do**
    make N edits to PW, yielding PW';
    thisGP = GP(PW');
  end
  if *thisGP* ≤ *TGP* **then**
    output PW' to X';
    partially process PW' into SDB;
  end
  partially process PW into SDB;
end

Algorithm 4: Basic Strengthening Algorithm

For the analysis in this thesis, it is assumed that any password successfully strengthened is accepted by the user. In practice, the user would have the opportunity to accept the strengthened password or try again. Assuming that all strengthened passwords are accepted is

likely a best-case scenario: it is possible, if not probable, that the subset of passwords approved by the user may share or lack particular features, rendering those passwords more susceptible to guessing attacks. For instance, users may be more accepting of an "X" inserted into their password than a "|" or a different letter.

Algorithm 4 is similar to the strengthening algorithm used by Houshmand and Aggarwal, with no major conceptual differences:

- There is a training phase to build an initial strengthening database
- Passwords are evaluated against this database
- Both the original user password and its strengthened counterpart are incorporated into the database every invocation.

At an implementation level, there is a noteworthy difference: in Houshmand's and Aggarwal's work, every password is "fully processed" into the strengthening database. However, their GP calculations utilized a static dictionary for alphabetic strings, rather than a dictionary dynamically built from the training set. Additionally, the processing of the digit and symbol strings incorporated some noise into the frequencies in order to mitigate the impact of a leaked strengthening database. Hence, the impact of a leak of their strengthening database is not directly comparable to results presented in Chapter 6.

By way of contrast, the strengthening algorithm in (Forget A. , Chiasson, van Oorschot, & Biddle, 2008) was one of three variants:

- **Preload** -- the user was presented with a password form with 2-4 slots pre-loaded with random characters
- **Replace** -- the users first chose their own password, and the system then replaced 2-4 characters in the password with randomly chosen ones
- **Insert** -- the users first chose their own password, and the system then inserted 2-4 random characters

The largest conceptual difference of these three approaches from Algorithm 4 is that passwords are altered without regard to their initial strength. Hence, while the ending password is almost certainly more resistant to guessing than the initial password, there is no mechanism to guide the ending password to a given level of strength.

Strengthened Datasets

A set of strengthened passwords was created by strengthening rockyou-1 and rockyou-2 according to Algorithm 4, applying one edit and targeting a GP of 10^{-16} or stronger. A second set of strengthened passwords was created by running the original rockyou-1 and rockyou-2 sets through the algorithm again, this time applying two edits. An edit could consist of replacing one character with another, or inserting a character into the password at any point with the decision to replace or edit being randomly determined. As with Persuasive Text Passwords (PTP), no restrictions were placed on the characters used in the edits – any printable character (ASCII 32 to 126, inclusive) could be used (Forget A. , Chiasson, van Oorschot, & Biddle, 2008), (Forget A. , Chiasson, van Oorschot, & Biddle, 2008).

In contrast, Houshmand and Aggarwal placed constraints on the edits: a character inserted into the middle of a string of digits or symbols had to be of the same type, and no changes could be made to an alpha string other than changing the case of one of the letters (Houshmand & Aggarwal, 2012). Thus, “password123” could be transformed to “passWord123”, “password1273” or “password-123” but not “pass-word123” or “password12:3”.

Chapter 5

Results: Successful Attacks on Strengthened Passwords

The goal of the strengthening algorithm is to generate passwords which will take a PCFG-based algorithm a long time to guess. This chapter first explores how well Algorithm 4 meets that goal by subjecting the strengthened passwords to two distinct PCFG-based attacks. However, an adversary needn't confine himself to a PCFG-based algorithm. Thus, a second type of attack, guided brute force (GBF) search, is analyzed. It is demonstrated that GBF attacks are highly effective in the absence of any safeguards. The chapter concludes with methods that can be taken to reduce the effectiveness of this second form of attack.

Resistance to PCFG-based Attacks

Algorithm 4 guarantees that every password which was output met the required GP threshold, *at the time it was strengthened*. However, when processed on a different data set (including the same strengthening database at a later point in time), the calculated GP will vary. Thus, it is possible that a password deemed secure by the strengthening algorithm might be judged as weak when using a different database to measure strength. In the remainder of this section, the viability of attacking the strengthened passwords using out-of-band data is explored. The impact of an accidental leak of the strengthening database is investigated in Chapter 6.

A well-known attack vector is to train a password cracking program on a previously leaked data set such as rockyou, and use the derived data as the basis for an attack. However, when attempting to crack strengthened passwords, it would seem that a better course of action would be to take the rockyou set, strengthen it using the known or derived strengthening algorithm, and then use those strengthened passwords as the training set for a password cracking

program. If the newly strengthened passwords are statistically similar to the passwords that are to be guessed, this would be an effective attack.

In analyzing the effectiveness of this attack, rockyou-1 was used as the basis to crack the passwords in rockyou-2. The data needed by Algorithm 2 and Algorithm 3 to mount a guessing attack was calculated in two ways: from the original, unstrengthened rockyou-1 set and also from the strengthened rockyou-1 set. In Table 5-1, a breakdown of the GPs of the strengthened passwords under these attack scenarios is given; the rows labeled Weak reflect the first scenario (using the original rockyou-1 passwords), while the rows labeled Strong reflect the second scenario (using strengthened rockyou-1 passwords rather than the original passwords). If the strengthening algorithm were perfect, the worst GP under any scenario would be the level targeted, or 10^{-16}, and Table 5-1 would be all 0s. Results are shown for both 1 and 2 edits.

Table 5-1. GPs Calculated from External Data

Cracking Data	*# Edits*	10^{-13}	10^{-14}	10^{-15}
Weak	1	1.3%	2.2%	3.2%
Weak	2	0.3%	0.5%	0.8%
Strong	1	2.5%	4.6%	18.0%
Strong	2	0.4%	1.3%	7.6%

The top panel in the table shows that, as expected, using weak passwords as the basis for a PCFG-based attack on strengthened passwords is largely ineffective. Even when only 1 edit is applied, only 1.3% of the passwords could be cracked in less than a day (GP of 10^{-13}), and only 2.2% could be cracked in less than a week. This panel illustrates that the strengthening algorithm was effective in guarding against a PCFG-based attack which uses data from typical, weak passwords.

The data in the bottom panel shows that if the PCFG attack uses data derived from passwords which were strengthened using the same algorithm as the passwords which are to be guessed, one edit no longer suffices. In this scenario, 2.5% of the passwords could be cracked in less than a day, and 4.6% in less than a week. However, if two edits are applied rather than one, the strengthening algorithm is still largely effective.

Recall that Algorithm 4 placed no constraints on the editing process – inserted or replacement characters could be any printable character. If constraints along the lines of what was used in (Houshmand & Aggarwal, 2012) were used, the results are comparable to the figures in Table 5-1 and Table 6-1, with no meaningful changes. However, as discussed below, the restricted editing does have a significant impact in a different context.

Resistance to Brute Force Attacks

A second form of attack against strengthened passwords would be for an attacker to try and guess the original, unstrengthened password, and then to test all possible variants. This attack is a brute force approach, guided by the passwords generated with Algorithm 2. While this GBF search is far slower than a PCFG-based attack, it is nonetheless fast enough to be practical as will be illustrated shortly.

Detailed examination of the strengthened passwords shows one reason why a GBF attack could be successful: oftentimes, the change in GP from the original password to the strengthened one is several orders of magnitude more than what should be possible. As a somewhat extreme example, “12qwaszx” has a GP of 1.04×10^{-8}; this is quite weak, due to “qwaszx” being a keyboard pattern used frequently enough to be included in the cracking database's dictionary, and the relatively common structure of $\mathbf{D_2L_6}$. The strengthened password “12qwasJx” has GP of 1.75×10^{-17}, which reflects the extreme improbability of the alphabetic Markov chain within

the cracking database to generate the string "qwasjx". While the calculations from the Markov chain's transition probability tables are mathematically correct, it is entirely unreasonable, even nonsensical, to assert that changing a single character within an 8-character password actually makes it more secure by 9 orders of magnitude – particularly since the number of possible variants is only 1,607[7]. A similar state of affairs is shown in (Houshmand & Aggarwal, 2012): there the authors present an example that shows transforming the password "life45!" to "life^45!" decreases the GP by four orders of magnitude, from 1.1×10^{-12} to 1.8×10^{-16}.[8]

In order to determine how many passwords could potentially be guessed using this brute force approach, the GPs of the original passwords which were strengthened need to be calculated. Passwords with GPs no stronger than 10^{-12} can be guessed in about two hours and thus strengthened passwords built from passwords with this level of GP or weaker may be vulnerable to a brute force attack. Table 5-2 shows a breakdown of the GPs of the original passwords which were successfully strengthened by Algorithm 4 using one or two edits. Results are also shown for full editing (no restrictions on characters used, denoted "Yes") or restricted editing (as in (Houshmand & Aggarwal, 2012), denoted "No").

Table 5-2. GPs of Original Passwords, before Strengthening

# Edits	*Full Editing?*	10^{-9}	10^{-10}	10^{-11}	10^{-12}
1	Yes	13.3%	22.2%	35.2%	54.0%
1	No	2.4%	4.4%	6.4%	11.4%
2	Yes	26.5%	41.6%	60.3%	74.5%
2	No	4.9%	9.5%	25.0%	49.0%

[7] There are 855 variants from inserting one of 95 characters at 9 possible locations, and an additional 752 variants from changing any of the eight characters to one of 94 others.

[8] Here, the GP of "life45!" is not meaningful due to its length of 7 and our requirement of 8 characters for the training data. However, measuring the GP of "life456!" and "life^456!" shows an identical drop in GP of four orders of magnitude: 3.65×10^{-9} to 4.86×10^{-13}.

The percentages in Table 5-2 reveal that a GBF attack as described above would be effective in cracking a significant number of strengthened passwords. With one edit and no restrictions on editing, 13% of the strengthened passwords had original passwords with a GP weaker than 10^{-9}. As observed in Table 4-1, all passwords with this level of GP can be generated in seconds. Thus, the only barrier to guessing a large percentage of the strengthened passwords is the computation time to generate and test all possible variants of the weak passwords. In this regard, the passwords strengthened with two edits may be more secure. Despite the higher percentage of weak original passwords, using two edits will have a significant impact on the run time.

Notably, if the strengthening process restricts the edits that can be made, the likelihood of a password with a low GP being successfully strengthened is significantly less than in the case with unrestricted edits since it is harder to strengthen a password with the restrictions in place. As a side effect, this means that restricted editing actually makes the passwords less susceptible to a GBF attack. An explicit exploration of the impact of stronger initial passwords on the effectiveness of GBF guessing will be done in the section Counter-measures.

To determine run times for GBF attacks, the guessing algorithm shown in Algorithm 3 was modified so that whenever *currentGuess* was checked against the list of passwords, all permissible variants of *currentGuess* were generated and tested as well. The results are presented in Table 5-3, below.

As was seen in Table 5-2, restricting the edits ("Full Editing?" column of "No") greatly reduces the number of weak passwords that can be strengthened, which reduces the effectiveness of a GBF attack. Table 5-3 shows this advantage is offset to a degree. Here, it is evident that reduction in the size of the search space that the GBF attack must explore has a large impact on run time. As exhibited, when compared to unrestricted editing ("Full Editing?" column of "Yes") an additional order of magnitude of GPs can be explored when restricted edits were used.

Table 5-3. Run Times for Guided Brute Force Attack

Minimum GP	*# Edits*	*Full Editing?*	*Run Time, 12 cores*
10^{-9}	1	Yes	1.2 hours
10^{-9}	1	No	8 minutes
10^{-10}	1	Yes	12.7 hours
10^{-10}	1	No	1.3 hours
10^{-11}	1	Yes	1 week (est)
10^{-11}	1	No	16.2 hours
10^{-9}	2	Yes	Guessed 5.4% in 24 hours
10^{-9}	2	No	20.4 hours

The runs which applied two edits would clearly take a long time to finish. However, the runs for passwords with GPs of 10^{-9} and weaker were started and allowed to run for 24 hours before being halted. Even though the search space with two edits is too large to exhaustively search, a GBF attack was still capable of guessing roughly 5% of the strengthened passwords in 24 hours when searching at a GP level of 10^{-9}. Since the goal of the strengthening algorithm was to ensure that no password could be guessed within weeks, while the strengthened passwords are significantly harder to guess than the original passwords, the strengthening algorithm has nevertheless not met its goal.

The GBF search actually guessed slightly more passwords than was indicated by Table 5-2 as some of the passwords which did not need strengthening were within one or two edits of a weak password. For example, the password “K3ybo@rd” would be judged secure by the

strengthening process, but it would be guessed when all 2-edit variants of "Keyboard" were generated[9].

Counter-measures

In order for a strengthening algorithm to perform as desired, the effectiveness of GBF attacks must be significantly reduced. One way to accomplish this goal would be to increase the number of edits as this increases the number of possible variants for each password by several orders of magnitude, making GBF attacks significantly more time-consuming. However, as noted by Forget, et al., going beyond two *random* edits significantly impairs memorability, so two random edits should be considered the limit when using Algorithm 4 (Forget A. , Chiasson, van Oorschot, & Biddle, 2008), (Forget A. , Chiasson, van Oorschot, & Biddle, 2008). A second method would be to require the initial password to be stronger, thus harder to guess, making a GBF attack take longer; in turn, the strengthened password will also be harder to guess. In other words, the user is required to clear an intermediate GP hurdle, with Algorithm 4 then decreasing the GP to a secure level with the resulting password resistant to GBF attack.

Although this approach requires more effort on the part of the user, tools could be provided to assist the user with the initial password selection. For instance, in the event the original password's GP did not meet the preliminary threshold, the user could be given high-level, non-specific feedback on how to make the initial password stronger such as making their password longer or using more character classes. This is, by itself, a difficult challenge and it has already been noted that stringent password scoring is frustrating to users. Further, the strength of these initial passwords is potentially illusory as users may, as with simple composition policies,

[9] The GP of "Keyboard" is very weak – 9.47×10^{-8} – whereas the GP of "K3ybo@rd" is quite strong at 1.81×10^{-17}. This again illustrates that applying simple mangling rules to weak passwords can produce passwords which are judged to be "secure".

find ways to meet the requirement but produce passwords that fall into patterns which could be discovered and exploited. However, absent any *a priori* knowledge about how users would adjust their password to score well against an adaptive PCFG-based metric, it is unclear how an attacker could ease the burden of guessing the initial password.

To test the impact of requiring a stronger initial password, the original rockyou-1 and rockyou-2 sets were screened for passwords with a minimum GP of 10^{-12}, which reduced each set to just over 500K passwords. These subsets were strengthened to a GP of 10^{-16}, and the strengthened passwords were analyzed for resistance to both PCFG-based and GBF attacks. Table 5-4 shows the GPs for the strengthened passwords in rockyou-2, when using the strengthened passwords in rockyou-1 as the training set for the cracking algorithm. When applying one edit, less than 1.5% of the passwords were vulnerable to guessing within a week (GP of 10^{-14}), which compares favorably to the 4.6% figure in the bottom panel of Table 5-1. Similarly, the results for two edits are also reasonably low at all GPs shown. This confirms the hypothesis that starting with an initially stronger password improves the resistance of the strengthened password to a PCFG-based attack.

Table 5-4. GPs of Strengthened Passwords, starting from Screened Subset

# Edits	*Full Editing?*	*10^{-13}*	*10^{-14}*	*10^{-15}*
1	Yes	0.5%	0.9%	2.2%
1	No	0.6%	1.4%	3.5%
2	Yes	0.4%	0.7%	1.3%
2	No	0.5%	0.9%	2.0%

While the starting passwords in this section were initially screened to have a GP of 10^{-12} or stronger, it is not anticipated that 100% of the initial, unstrengthened passwords will meet this threshold if a different database is used to measure GPs.

Table 5-5. GPs for Original Passwords Needing Strengthening, Screened Subset

# Edits	*Full Editing?*	10^{-9}	10^{-10}	10^{-11}	10^{-12}
1	Yes	0.0%	0.1%	1.2%	12.7%
1	No	0.0%	0.1%	0.7%	5.5%
2	Yes	0.0%	0.1%	1.2%	13.9%
2	No	0.0%	0.1%	1.2%	13.6%

As seen in Table 5-5, when cracking passwords using a data set trained on the screened rockyou-1 passwords, only 1.2% of the initial passwords in rockyou-2 have GPs one order of magnitude weaker than the targeted threshold of 10^{-12}. This is good news – compared to Table 5-2, all of the percentages are markedly lower and as a consequence, it will be harder to guess the strengthened password which means that a GBF attack will take significantly longer.

As noted in the section Resistance to Brute Force Attacks, the limiting factor on the efficacy of GBF attacks is the run time. As seen previously, the run times rapidly climb, making exhaustive search impractical for all but the lowest GP thresholds. When working with initially stronger passwords, the scenario is similar, with the run times in Table 5-6 similar to those in Table 5-3.

Table 5-6. GBF Run Times, Screened Subset

Minimum GP	*# Edits*	*Full Editing?*	*Run Time, 12 cores*
10^{-9}	1	Yes	20 minutes
10^{-9}	1	No	3 minutes
10^{-10}	1	Yes	6.5 hours
10^{-10}	1	No	1 hour
10^{-11}	1	Yes	5 days (est)
10^{-11}	1	No	17 hours
10^{-12}	1	Yes	Guessed 0.3% in 24 hours
10^{-12}	1	No	Guessed 0.6% in 24 hours
10^{-12}	2	Yes	3 passwords in 24 hours
10^{-12}	2	No	28 passwords in 24 hours

The run times for the lowest GP thresholds are quite short, but, due to the low number of passwords that are found this is not the issue that it was in the section Resistance to Brute Force Attacks. However, the last panel of Table 5-6 is cause for cautious optimism. Here, as in Table 5-3, the attacks did not run to completion but were halted after a reasonable period of time[10]. Unlike Table 5-3 however, essentially none of the strengthened passwords were guessed within a day. Although multi-day runs are required to better assess the true level of security, applying two edits to a password that is already fairly strong appears to adequately defeat a GBF attack, leaving a would-be attacker with a PCFG-based attack as the best option (cf. Table 5-4).

[10] For the partial runs in Table 5-6, only passwords with a GP stronger than 10^{-11} were modified due to the low probability of success when modifying weaker passwords.

Chapter 6

What if the Strengthening Database is Leaked?

The efficacy of attacking strengthened passwords using only data that is disjoint from the actual system passwords was explored in Chapter 5. In addition to these attacks, it is possible that a site's strengthening database could be accidentally leaked. Ideally, the leakage of that data should not be sufficient to crack a large percentage of the user passwords. Since the strengthening database contains statistics on the actual system passwords, this seems to be a worst-case scenario.

This chapter begins with the implications of a leak of the entire strengthening database. In this case, the dictionaries, probabilities of the password structures and the Markov chains are all completely known to the adversary, who would then use Algorithm 2 and Algorithm 3 to recover the strengthened passwords. The chapter continues with an examination of how noise could be placed into the strengthening database to reduce the number of recoverable passwords in the event of a leak. If an adversary is able to obtain copies of the strengthening database at multiple points in time, different challenges are presented, and these are analyzed in the last section.

PCFG-based Attacks, using a Leaked Strengthening Database

As stated above, in the event the strengthening database is obtained by an adversary, using Algorithm 2 and Algorithm 3 to recover the system passwords would be a logical choice. Thus, the GPs of the system passwords, as calculated when using those algorithms together with the leaked strengthening database, provide a measure of how many passwords can be recovered in a particular timeframe. In Table 6-1 the GPs for the strengthened passwords are shown for

three different scenarios, each representing a variation of Algorithm 4. The first variant, shown in the top panel, utilizes Algorithm 4 as presented. The rows in the middle panel show the consequences of fully processing all original and strengthened passwords; in other words, Algorithm 4 would be modified so that every token in both the original and strengthened passwords would enter the dictionaries. The bottom panel shows the results if Algorithm 4 were modified so that it fully processed all original passwords but still only partially processed the strengthened passwords.

Table 6-1. GPs Calculated from Leaked Data

Password Processing	*# Edits*	10^{-13}	10^{-14}	10^{-15}
Original partial, strong partial	1	0.0%	0.1%	1.5%
	2	0.0%	0.0%	0.6%
Original full, strong full	1	58.2%	67.3%	75.6%
	2	28.2%	38.6%	50.0%
Original full, strong partial	1	21.2%	24.4%	28.7%
	2	6.4%	8.3%	10.3%

The top panel is rather surprising as it shows that the strengthening database for Algorithm 4, as presented, provides less information to an attacker than does the publicly available Weak data set of Table 5-1. This lack of success is attributable to the use of full versus partial processing of the passwords in Algorithm 4, and the remaining rows illustrate the importance of this differential in processing. As the percentages for both the middle and bottom panels show, either alternative method to building the strengthening database has the risk that, in

the event of a leak of the strengthening database, a large percentage of passwords would be guessed in under a day, even with two edits.

As described in Chapter 4, Making Passwords Stronger – the Strengthening Algorithm, the strengthening database is built from both the original, weak passwords and the revised, strengthened passwords. Consequently, it is not surprising that the rows in the top panel exhibit lower percentages than the Strong rows in Table 5-1. In the latter case, the data used in cracking reflects only the statistics of the passwords that are to be guessed, rather than both weak and strong passwords. It is, however, not immediately apparent why the leaked database would be less effective than using a database built solely from weak passwords. The answer has to do with the role of the dictionaries in computing GP_{MC}(token) and thus the computed GP for the password. Because the dictionaries are built only from the training data in Algorithm 4, the dictionaries in the strengthening database are much smaller than they are in the databases built from either the original or strengthened passwords. The larger dictionaries give those data sets an advantage in guessing.

As an example, when the password "pearlharbor1" in the rockyou-2 set was run through the strengthening process, the dictionary in the strengthening database did not contain the string "pearlharbor". As a result, GP_{MC}(pearlharbor) was computed using the Markov chain's probability tables, and GP(pearlharbor1) calculated to 2.5×10^{-17}, and therefore not in need of strengthening. However, the rockyou-1 set contained two passwords which contained the string "pearlharbor", so the dictionary in the cracking database computed from rockyou-1 contained "pearlharbor". Thus, GP_{MC}(pearlharbor) was much larger (easier to guess) and the cracking database computed from rockyou-1 would quickly generate the password "pearlharbor1", even though the cracking database was built from unstrengthened passwords.

It is clear that a larger dictionary is advantageous in guessing. Similarly, when strengthening, the use of a larger dictionary should produce passwords which are more resistant

to guessing as the scenario just described would not have occurred if the strengthening dictionary contained “pearlharbor”. Thus, while the partial versus full processing of the actual passwords has the benefit of not providing an attacker any advantage in the event of a leak, there is also a cost: the strengthened passwords must, due to the smaller dictionary being used, be more susceptible to an attack, such as a PCFG-based one, that utilizes a dictionary.

Table 6-2 shows the results of a PCFG-based attack on the strengthened passwords under the three full/partial processing combinations used in Table 6-1: partial processing of both the original and strengthened passwords, full processing on both passwords and fully processed original passwords but partially processed strengthened passwords; these correspond to the top, middle and bottom panels, respectively. In each of these cases, the data used to mount the attack is calculated exclusively from passwords strengthened from the rockyou-1 set as was done in the Strong rows of Table 5-1 and thus the top panel duplicates those percentages.

Table 6-2 – GPs of Strengthened Passwords under Different Dictionaries

Password Processing	*# Edits*	10^{-13}	10^{-14}	10^{-15}
Original partial, strong partial	1	2.5%	4.6%	18.0%
	2	0.4%	1.3%	7.6%
Original full, strong full	1	2.0%	3.7%	16.4%
	2	0.3%	1.1%	6.4%
Original full, strong partial	1	2.4%	4.4%	17.7%
	2	0.3%	1.3%	7.5%

The alternative constructions do produce strengthened passwords which are more resistant to a PCFG-based attack. However, the differential in the number of recoverable

passwords is slight, particularly if two edits are used, so the quality of the strengthened passwords is very nearly identical irrespective of which combination of full and partial processing is used by the strengthening algorithm. Nonetheless, if the probability of a leak is deemed sufficiently small, the slightly superior resilience to cracking seen in the alternative constructions could be judged as a risk worth taking.

Obfuscating the Strengthening Database with Noise

Prior work has analyzed the risks associated with the data used by an automated password strengthening system being leaked. Castelluccia, et al., in their work on adaptive Markov models, propose adding noise to the n-gram counts from which the transition probabilities are calculated (Castelluccia, Duermuth, & Perito, 2012). Within their analytical framework, they show that using their approach to adding noise, a bound of 1.3 bits of Shannon entropy per password is lost in the event of leak, assuming reasonable parameters. The addition of noise does have a downside, however. Castelluccia, et al., exhibit an ROC curve comparing the performance of the Markov model to the Markov model with noise, and the addition of noise lowers the ROC curve slightly. In the context of this thesis, this would be termed a loss of quality in the strengthened passwords.

As exhibited in the prior section, the number of passwords which could be successfully recovered in a reasonably short period of time is already quite low (cf. Table 6-1). Thus, the reduction in the quality of the strengthened passwords is critically important – any uptick in the percent of strengthened passwords which could be recovered by a PCFG-based or GBF attack may outweigh a reduction in the number of passwords that could be recovered in the event that the strengthening database is leaked.

To better understand the tradeoff between these competing factors, Algorithm 1was modified to add noise in a manner consistent with Castelluccia, et al. The modified algorithm is presented below. Note that the only change is the addition of steps 7 and 8.

Data: token T of length k, dictionary D, n-gram Markov chain with alphabet Σ
Result: updates the dictionary and Markov chain probability tables
Increment the total number of tokens seen; // step 1
Increment the number of tokens of length k seen; // step 2
if *T is not in D* **then**
  add T to D; // step 3
end
Increment the tally for T in D; // step 4
Increment count of $T_{1,n}$ beginning a token; // step 5
for *j from 1 to k-n* **do**
  Increment the transition count from $T_{j,j+n-1}$ to $T_{j+1,j+n}$; // step 6
end
foreach n-gram W in Σ^n
  Increment count of W beginning a token with probability γ // step 7
  foreach n-gram X such that $W_{2,n} = X_{1,n-1}$ // n-gram X can follow W
    Increment transition count from W to X with probability γ // step 8
  end
end

Algorithm 5: Token Processing with Random Count Increases

In this framework, the noise that is added can only increase the counts. Since the counts which are incremented are selected with equal probability, this has the effect of pushing the calculated probabilities towards a flatter, equi-probable distribution. In keeping with Castelluccia, et al., tests were run with $\gamma = 2^{-20}$; an additional test was run with $\gamma = 2^{-16}$ in order to see the impact of a higher level of noise.

To illustrate the effect of the noise in these experiments on the transition probabilities, after processing the rockyou-2 set with both levels of noise, the transition probabilities for the 3-gram "str" were compared to the baseline case of no noise. The results for the five most and five least common transitions are presented in Table 6-3.

Table 6-3 – Transition probabilities from 3-gram "str", varying noise levels

γ	*e*	*a*	*o*	*i*	*u*	*f*	*j*	*x*	*q*	*v*
0	27.58	25.10	21.97	16.03	5.52	0.03	0.02	0.02	0.02	0.01
2^{-20}	27.45	25.01	21.87	15.99	5.50	0.04	0.06	0.04	0.03	0.04
2^{-16}	24.77	22.76	20.06	14.66	5.31	0.44	0.54	0.60	0.43	0.48

As expected, the more noise that is added, the more the transition probabilities tend towards an equal distribution, and the greatest impact on the transition probabilities is seen at the extremes. For instance, the reduction in the transition probability for "e" is greater than the reduction for "u", both in absolute and percentage terms. Similarly, the probabilities for the uncommon transitions increase, particularly at the higher level of noise, which would have a notable impact on the GP of any password which contained those transitions.

This flattening of the transition probabilities has important ramifications for the calculated GPs, and consequently the quality of strengthening that is done. Focusing on the bottom row of Table 6-3, a password which contained the string "stre" would, assuming no other probabilities were changed by the noise, have a calculated GP 10% lower than in the scenario where noise was not added. Similarly, a password which contained the string "strx" would have its GP increased by a factor of 30. These changes in strength are purely an artifact of the noise as an attacker's ability to crack the password is unchanged. However, the implications are potentially troublesome: in the first instance, a password with a common string will appear stronger than it is, which could lead to an inaccurate classification of a password as strong. In the second instance, a password which is actually strong might not be classified as such, with the result that the user ends up with a less strong password.

To make a more informed assessment of the tradeoff between better protection of the system passwords in the event the strengthening database was leaked and the (assumed) reduction

in the quality of strengthening, rockyou-2 was strengthened as before, using Algorithm 4, but with noise added to the strengthening database in accordance with Algorithm 5. Runs were made using noise probability 2^{-20} and 2^{-16} in order to better understand the impact of this parameter. The strengthened passwords were then analyzed for resistance to guessing using the attacks of Chapter 4 as well as in the scenario of a leaked strengthening database. The results are summarized in Table 6-4.

Table 6-4 – GPs under varying noise levels and attack vectors

#Edits,	*PCFG, weak*			*PCFG, strong*			*GBF*			*Database Leaked*		
Noise(γ)	10^{-13}	10^{-14}	10^{-15}	10^{-13}	10^{-14}	10^{-15}	10^{-9}	10^{-10}	10^{-11}	10^{-13}	10^{-14}	10^{-15}
1, 0	1.3	2.2	3.2	2.5	4.6	18.0	13.3	22.2	35.2	0.0	0.1	1.5
1, 2^{-20}	1.3	2.2	3.3	2.5	4.7	18.0	13.3	22.2	35.2	0.0	0.1	1.3
1, 2^{-16}	1.6	2.5	3.9	2.7	5.2	20.5	13.7	23.1	36.3	0.0	0.0	0.7
2, 0	0.3	0.5	0.8	0.4	1.3	7.6	26.5	41.6	60.3	0.0	0.0	0.6
2, 2^{-20}	0.3	0.5	0.8	0.4	1.3	7.6	26.5	41.6	60.3	0.0	0.0	0.6
2, 2^{-16}	0.4	0.6	1.1	0.4	1.5	8.4	26.7	41.9	60.8	0.0	0.0	0.4

The “PCFG, weak” panel contains the percent of passwords at the indicated GP level using PCFG-based attack, with the cracking database built solely from weak passwords while the “PCFG, strong” panel is similar, but the cracking database is built from passwords which were strengthened by Algorithm 4. These panels correspond to the top and bottom panels of Table 5-1, respectively. The panel labeled “GBF” corresponds to the data presented in Table 5-2, the percent of original passwords at the indicated GP level, prior to strengthening; these passwords are recoverable by a GBF attack. The rightmost panel shows the GPs of the system passwords as calculated by the strengthening database; this mirrors the data in Table 6-1.

At the lower level of noise ($\gamma = 2^{-20}$) and with two edits made by the strengthening algorithm, the rows in the table are identical: there is no reduction in passwords recoverable in the event of a leak, but there are also no additional passwords recoverable by the attack methods used in this thesis. With $\gamma = 2^{-16}$ and two edits, the differences are marginal, but the uptick in the number of passwords recoverable in a GBF attack does not seem to warrant the reduction in passwords that would be safe in the event of a leak – particularly since this reduction is only evident at a GP of 10^{-15}, a level that is fairly secure.

When only one edit is used by the strengthening algorithm, the story is similar. The number of passwords recoverable in the event of a leak decreases only at a fairly secure GP level, whereas there is an increase the number of passwords which can be guessed by other means, particularly at $\gamma = 2^{-16}$. Thus, given that the leak of the strengthening database used by the algorithms in this thesis provides an adversary with less information than can be obtained by other means (such as a GBF attack or PCFG-attack based on strengthened passwords), it is unclear that the addition of noise is warranted when only these attack vectors are considered.

There are, however, alternative attack vectors which could be undertaken by an adversary in possession of the strengthening database. Castelluccia, et al., point out that in their adaptive Markov model, passwords which contain very rare n-grams could be easily reconstructed. Their reasoning is as follows: assuming 5-grams as the authors used, the password "qkyvsnjpo"[11] could probably be reconstructed from the strengthening database: there would be very few "qkyvs" 5-grams and very few 5-grams of the form "kyvsα", where α is an arbitrary letter; thus the next character in the password could be narrowed down to a very small number of choices, if not a single choice. Thus, passwords which contain rare n-grams may, ironically, be the most at risk if the n-gram counts (equivalently, the transition probabilities) are leaked. Determining the full implications on the ability of an adversary to reconstruct passwords in this manner, with and

[11] Using rockyou-1 as the training set, this password is secure with GP 3.7×10^{-17}

without noise being added to the counts, is beyond the scope of this thesis. It is noted that the adversary's task is complicated to a small degree by the fact that there are multiple Markov chains maintained within our strengthening database, rather than just one chain for the entire password as in Castelluccia, et al. Consequently, the strings which are re-constructed from the rare n-grams may be sub-strings of a password (tokens), rather than an actual password. However, by adding the re-constructed tokens to the dictionaries within the leaked strengthening database prior to running Algorithm 2, an attacker would still easily uncover those passwords which contained the re-constructed tokens.

Instead of always incrementing the counts when adding noise, the counts could be either increased or decreased by some random quantity. Drawing from the ideas of differential privacy (Dwork, McSherry, Nissim, & Smith, 2006), the noise added in this case is drawn from a Laplacian distribution and is applied to each count within the strengthening database; Algorithm 6 shows the details. Additionally, the strengthening algorithm was updated so that when a strengthened password was partially processed into the strengthening database, Algorithm 6 was run on each of the database's four Markov chains with probability 2^{-4}. Ideally, the noise should be added after each strengthened password is partially processed; however, given the sizes of rockyou-1 and rockyou-2, that would have required a run time of over a week. By applying the algorithm at the indicated probability, the run time was reduced to a manageable level.

Data: n-gram Markov chain with alphabet Σ
Result: adds Laplacian noise to the Markov chain
foreach n-gram W in Σ^n
  σ = random Laplacian(0,1) deviate;
  add σ to count for W beginning a token, flooring sum at 0;
  foreach n-gram X such that $W_{2,n} = X_{1,n-1}$ // n-gram X can follow W
    σ = random Laplacian(0,1) deviate;
    add σ to transition count from W to X, flooring sum at 0;
  end
end

Algorithm 6: Adding Laplacian Noise to a Markov Chain

When counts are reduced, rather than only increased, there would appear to be additional risks to the quality of strengthening. This is because for rarer transitions, the counts are, by definition, small; for these transitions, a decrease in the already small count could cause the transition probability to decrease significantly, making a GP appear much stronger than it is. If this reduction in transition probability makes the GP for an insecure password appear to be secure, this password is at risk of being easily guessed. As a result, it is expected that using Laplacian noise would have a larger negative impact on the quality of strengthening than only increasing the counts does. Specifically, it is expected that there will be a greater number of weak original passwords which are judged to be successfully strengthened, and thus a GBF attack would recover a greater percentage of passwords.

Table 6-5– GPs under varying noise levels and attack vectors, Laplacian noise

#Edits,	*PCFG, weak*			*PCFG, strong*			*GBF*			*Database Leaked*		
Noise	10^{-13}	10^{-14}	10^{-15}	10^{-13}	10^{-14}	10^{-15}	10^{-9}	10^{-10}	10^{-11}	10^{-13}	10^{-14}	10^{-15}
1, None	1.3	2.2	3.2	2.5	4.6	18.0	13.3	22.2	35.2	0.0	0.1	1.5
1, Lapl.	4.7	10.9	21.4	14.4	34.1	58.4	24.5	38.6	54.2	0.1	1.2	6.8
2, None	0.3	0.5	0.8	0.4	1.3	7.6	26.5	41.6	60.3	0.0	0.0	0.6
2, Lapl.	2.7	5.2	9.4	2.0	9.1	29.2	28.4	45.0	63.9	0.0	0.2	1.6

Tests along the lines of those exhibited in Table 6-4 were performed using Laplacian noise, and the results, detailed in Table 6-5, confirmed this hypothesis. Examination of the strengthening results showed the potential problem posed by reducing transition probabilities is quite real: as one example, the password “alejandro” was successfully strengthened to “alNjandro” when noise was added to the strengthening database. If noise were not added to the strengthening database, “alNjandro” has a calculated GP of 1.76×10^{-16}, not quite under the

required 10^{-16} threshold. With noise, the GP calculated to 3.80×10^{-17}, satisfying the 10^{-16} hurdle. The primary source of difference between the two GP calculations came from the transition probability from "lnj" to "a". Without noise being added, the transition probability was 12.96%, whereas in the "with noise" scenario the probability was 3.15%. That represents a four-fold increase in strength purely as an artifact of the noise. Given that other, equally large or larger percentage reductions in transition probabilities occur, the overall increase in the percent of passwords recoverable by a GBF attack is not surprising.

What is initially surprising is the increase in the percent of passwords that can be recovered in the event of a one-time leak of the strengthening database. However, the same reasoning as in the above paragraph explains this as well. Passwords modified by Algorithm 4 can meet the strength requirement of purely as a result of the noise due to n-gram transitions within the password being artificially lower at the time the password was strengthened. When those transitions revert to a level in keeping with their actual rate of occurrence, the GP of the password, when measured by the strengthening database at that point in time, would be higher and thus more readily recovered in the event of a leak.

Taken together – that there is an increase in the number of passwords recoverable by PCFG-based and GBF attacks, *and* an increase in the number of passwords which are recoverable in the event of a one-time leak – these two results clearly show that, in the context of a one-time leak and the attacks analyzed here, applying Laplacian noise is inferior to adding no noise, as well as only incrementing the counts.

Multiple or Ongoing Leaks

As discussed in the previous section, Castelluccia, et al., showed that only 1.3 bits of Shannon entropy per password were lost, on average, by an adversary who obtained the transition

counts used with their adaptive Markov chain. However, as pointed out by Heysham, there is the potential for a much greater information loss if the transition counts are leaked more than once (Heysham, 2013). By using two leaked data sets, the difference between the transition counts can be taken, revealing those n-grams that were used in the passwords which were added to the database. If noise is added to the transition counts, only increasing the transition counts as in Castelluccia, et al., the n-grams from the noise will also be found in the difference. However, since the actual n-grams will fit together to form strings – "mus", "usi", and "sic" can transition from one 3-gram to the next, forming the word "music" – the n-grams which were added as noise will tend to not fit into any string of transitions and can consequently be detected as noise by an adversary. If, on the other hand, Laplacian noise is used, some actual transitions will be missing, creating more work for the adversary.

The analysis of Heysham and Castellucia, et. al, is centered on an adaptive Markov chain, whereas the strengthening algorithm used here uses a PCFG-based approach, with password structures and multiple Markov chains. Nonetheless, Heysham's approach is applicable, as if the strengthening database is leaked more than once, the difference in counts between the two instances can be taken, yielding a new cracking database. As the strengthening database consists of password structures and associated counts, dictionaries with counts for each word, and Markov chains which contain counts of starting n-grams and counts of transitions from one n-gram to another, the difference between two databases is defined in the obvious way and is easy to calculate.

To determine the efficacy of this difference database, the following experiment was run. Algorithm 4 was run on rockyou-2, with the strengthening database at the completion of subset B saved as the base. After the next 100 passwords in subset C which were strengthened into rockyou-2, the strengthening database was saved; this is the base+100 scenario. Then, after the

next 900 passwords from subset C were strengthened, the strengthening database was saved, yielding a base+1K scenario. Similarly, base+10K and base+100K scenarios were generated.

The difference between the base+100 and base strengthening databases was calculated to obtain a cracking database for the additional 100 passwords present in the base+100 scenario. Cracking databases were created for the additional passwords in each of the other scenarios as well. Using each cracking database against its respective list of passwords, GPs were calculated. The results are shown in Table 6-6.

Table 6-6 – GPs when multiple leaks of strengthening database, no noise

#Edits	*Base+100*			*Base+1K*			*Base+10K*			*Base+100K*		
	10^{-12}	10^{-13}	10^{-14}	10^{-12}	10^{-13}	10^{-14}	10^{-12}	10^{-13}	10^{-14}	10^{-12}	10^{-13}	10^{-14}
1	91.0	96.0	97.0	57.3	70.8	81.4	9.0	18.7	32.5	0.1	0.6	2.6
2	79.0	90.0	95.0	45.5	59.3	72.3	6.3	13.1	24.2	0.1	0.3	1.5

The data clearly shows that if the strengthening database is leaked on multiple occasions, then the passwords which were added in the time interval between leaks can be recovered with high probability, even if the passwords were strengthened with two edits. While the Base+100K scenario shows that the advantage to an adversary decreases markedly if a large number of passwords were added between leaks, this may be of little practical value. If an adversary has the ability to obtain the strengthening database, getting multiple copies in a short time interval would seem to be the most probable course of action an attacker would take.

Additionally, the security implications are worsened by two observations. First, as Algorithm 4 only partially processes the original and strengthened passwords once it is no longer in training mode, the dictionaries within the strengthening database will be unchanged from the base case to the second leak. Hence, when the difference between the two leaked databases is

computed, the dictionaries will be empty. With no dictionaries to use within Algorithm 2, the run time for the algorithm is decreased, as is seen in Table 4-2 when using only Markov chains rather than Markov chains and dictionaries.

Second, it was noted in conjunction with Algorithm 1 that all starting prefix counts and transition counts within the Markov chains were initialized to 1, rather than 0, so that no computed probability would be 0. This allows for a full search of the password space, subject to the minimum GP parameter in Algorithm 2. However, when the difference between the two databases is computed, the resulting cracking database does contain many 0 probabilities, further speeding up the run time of Algorithm 2. The net result is that when running Algorithm 2 with a GP of 10^{-14}, the Base+100 scenario takes 7.5 hours while the Base+1K scenario takes under a day. Even without this second speed-up, a GP of 10^{-14} cannot be considered very secure as the estimated run time for Algorithm 2 at this GP is 3 days if no dictionaries are used (cf. Table 4-2).

This second observation also helps explain why the difference of the two leaked strengthening databases is so effective. Only data points used by the passwords which were added between leaks will have a non-zero associated probability – in other words, we know with certainty which password structures were used and which were not, which n-grams started tokens and which did not and which transitions from one n-gram to another were observed and which were not. Since it is known which password structures, starting n-grams and transitions do not occur, the search space is dramatically reduced. By the same reasoning, given a small number of passwords added between leaks (e.g., 100), the non-zero probabilities in the Markov chain will tend to be larger, yielding larger GPs. For instance, if the n-gram "str" was observed 20 times between leaks, the smallest that any transition probability from "str" can be is 5%. This is almost as high as the 5th largest transition in Table 6-3.

Obfuscation Revisited

As demonstrated in the prior section, the addition of noise to the strengthening database can, to some degree, reduce the number of passwords which can be recovered from a leaked database, albeit at a reduction in the quality of the strengthened passwords. Here, we investigate if the addition of noise to the strengthening database has an impact on the number of passwords which can be recovered in the event of multiple leaks. In the case where the noise is only allowed to increase the counts, the results are mixed. When only an additional 100 passwords are added to the base case, the number of recoverable passwords is essentially unchanged, irrespective of number of edits or noise parameter γ. As the number of passwords added to the database between leaks increases, there is an observable decline in the number of recoverable passwords, particularly at the higher noise level. However, even at the higher noise level and with 10,000 passwords added between leaks, 677 of the 10,000 passwords could be recovered in only half a week. Full details are presented in Table 6-7.

Table 6-7 – GPs when multiple leaks of strengthening database, increasing noise

#Edits	*Base+100*			*Base+1K*			*Base+10K*			*Base+100K*		
γ	10^{-12}	10^{-13}	10^{-14}	10^{-12}	10^{-13}	10^{-14}	10^{-12}	10^{-13}	10^{-14}	10^{-12}	10^{-13}	10^{-14}
1, 0	91.0	96.0	97.0	57.3	70.8	81.4	9.0	18.7	32.5	0.1	0.6	2.6
1, 2^{-20}	89.0	96.0	97.0	53.8	69.1	80.9	6.1	15.1	30.1	0.0	0.0	0.6
1, 2^{-16}	90.0	94.0	96.0	46.9	64.9	78.3	0.0	1.1	8.9	0.0	0.0	0.0
2, 0	79.0	90.0	95.0	45.5	59.3	72.3	6.3	13.1	24.2	0.1	0.3	1.5
2, 2^{-20}	83.0	91.0	97.0	43.6	58.3	72.4	5.4	11.5	22.0	0.0	0.0	0.4
2, 2^{-16}	79.0	92.0	98.0	36.3	52.4	66.2	0.1	1.2	6.8	0.0	0.0	0.0

Whereas the noise used in Table 6-7 could only increment the counts, we can alter the noise to either increase or decrease the counts. While in the preceding section we saw no benefit to this approach – the quality of the strengthened passwords suffered more using Laplacian noise than it did using only increasing counts, and there was actually an increase in the number of recoverable passwords when Laplacian noise was used. Nonetheless, it is possible that Laplacian noise could help in the context of multiple leaks: some of the transitions that were encountered between leaks could be effectively erased, thereby making the recovery of the password more difficult or even impossible if the transition probability calculated from the difference of the leaked strengthening databases is 0%.

Table 6-8 shows this is the case – even with only 100 passwords added in the time interval between leaks, at most one password can be recovered in a day.

Table 6-8– GPs when multiple leaks of strengthening database, Laplacian noise

#Edits	*Base+100*			*Base+1K*			*Base+10K*			*Base+100K*		
Noise	10^{-12}	10^{-13}	10^{-14}	10^{-12}	10^{-13}	10^{-14}	10^{-12}	10^{-13}	10^{-14}	10^{-12}	10^{-13}	10^{-14}
1, No	91.0	96.0	97.0	57.3	70.8	81.4	9.0	18.7	32.5	0.1	0.6	2.6
1, Lap	0.0	0.0	0.0	0.0	0.0	0.3	0.0	0.0	0.1	0.0	0.0	0.4
2, No	79.0	90.0	95.0	45.5	59.3	72.3	6.3	13.1	24.2	0.1	0.3	1.5
2, Lap	0.0	0.0	1.0	0.0	0.0	0.1	0.0	0.0	0.0	0.0	0.0	0.0

Thus, the use Laplacian noise does essentially eliminate the issue of extracting information from the differences between two leaked strengthening databases. However, as Laplacian noise notably reduces the quality of the strengthened passwords, this technique is not effective at stopping other forms of attack.

Chapter 7

Conclusion and Future Directions

This thesis describes the problem of users typically selecting weak passwords and the various means that have been developed to help users select more secure passwords. Strengthening algorithms appear to be the best choice currently available in this regard, as the user still gets to choose the initial password and the ultimate password is closely related to this choice. However, since the user's password is subject to random edits, memorability is still an issue, and so this paper only analyzed the results from a strengthening algorithm that applied either one or two edits.

After strengthening two large sets of passwords (rockyou-1 and rockyou-2), the results from rockyou-1 were used to create a cracking database for the strengthened passwords from rockyou-2. For a PCFG-based cracking algorithm, the results clearly show that in order to reasonably guarantee that a password cannot be cracked in under a day, the strengthening algorithm should apply two edits to the user's original password rather than just one. If only one edit is applied, an attacker who builds a cracking database from passwords that were strengthened using the same algorithm would be able to guess 2.5% of the strengthened passwords in under a day, and 4.6% in under a week (cf. Table 5-1).

It was also shown that an alternative type of attack could be even more successful. Since a strengthened password is derived from a user-selected password, the original password is likely to be weak and therefore easily guessed. Thus, systematic guessing of all possible variations of passwords that can be quickly generated by a PCFG-based algorithm is a potentially effective attack, with the effectiveness limited only by the computing time required. However, even though the search space is too large to search exhaustively when two edits are applied, a successful attack need not run to completion; if a large number of passwords are recovered in a

manageable time, frame, the attack is successful. This scenario is exactly what was found to be possible, with over 5% of passwords strengthened with two random edits being guessed in the first 24 hours (cf. Table 5-3). However, requiring the initial password to be significantly stronger, with an initial GP of 10^{-12} or better, seemed to completely thwart a GBF attack when two edits were used, with essentially none of the passwords guessed within the first 24 hours (cf. Table 5-6). Still, roughly 1% of the strengthened passwords remain vulnerable to a PCFG-based attack even with the stronger initial password (cf. Table 5-4).

Although the use of a significantly stronger initial password was successful in markedly reducing the effectiveness of a guided brute force attack, a GP of 10^{-12} is a fairly high hurdle, with only 7% of the passwords in the entire rockyou list meeting this criterion. Since users are not predisposed to producing passwords possessing this level of strength, one area of future research would be the use of non-specific feedback (telling the user to include more special characters, for example) or specific feedback (e.g., telling the user not to use a particular string such as "pearlharbor" or to not put all the special characters at the end of the password) which may help guide users to selecting stronger initial passwords.

When examining the ramifications of a leak of the strengthening database, the number of passwords which can be recovered from a one-time leak is quite small, provided the dictionaries in the strengthening database are not updated with tokens from actual user passwords or their strengthened counterparts. However, not updating the dictionaries has a cost associated with it – by not updating the dictionaries, slightly more passwords are recoverable by PCFG-based and GBF attacks than would be the case if the dictionaries were updated with all tokens.

Adding noise to the database had mixed results in the context of a one-time leak. Using Laplacian noise made more passwords recoverable by PCFG-based and GBF attacks while also increasing the number of passwords recoverable from the strengthening database in the event of a single leak. Using noise as outlined in Castelluccia, et al., which only increased the transition

counts, had a mild negative impact on the quality of strengthening but did reduce the number of passwords recoverable from a one-time leak.

If multiple leaks of the database are considered, the implications are much more severe. If less than 1,000 passwords are entered into the database between the leaks, over 50% of those passwords can be recovered in less than a day by using the difference between two leaked strengthening databases as the cracking database. Experiments exploring the impact of noise in the context of multiple leaks showed that Laplacian noise all but eliminated the number of passwords recoverable from the difference of two leaked datasets. Adding noise in the style of Castelluccia, et al., did reduce the number of passwords recoverable from the difference, but the percentages were still high, particularly when fewer than 1,000 passwords were added between leaks. Finding a technique which is effective against recovering passwords from the difference of two leaked databases while simultaneously not having a large negative impact on the quality of the strengthened passwords is an important area of future research.

More efficient algorithms to search the strengthened space could also be investigated, in order to better understand the security risk posed by guided brute-force attacks. The algorithm used for this paper was straightforward and its efficiency could be improved. In particular, there was some redundancy as edits to the structures $\mathbf{L_7D_1}$ and $\mathbf{L_6D_2}$ both generate some passwords of the form $\mathbf{L_7D_2}$.

Finally, ways to bolster strengthening algorithms should be investigated: for example, rather than simply randomly replacing or inserting characters in the user's password, additional transformations could be tried. While past work has shown that two *random* edits should be considered an upper limit with respect to maintaining memorability (Forget A. , Chiasson, van Oorschot, & Biddle, 2008) and (Forget A. , Chiasson, van Oorschot, & Biddle, 2008), other types of edits may still be feasible. Examples would be to insert or weave a 3-5 letter word into the user's password (transforming “password” to “passCATword” or “CpassAwordT”), or to reverse

one of the strings within the password (transforming “rock&roll” to “rock&llor”). It may also be beneficial to have the strengthening algorithm reject any starting password which is within two edits of a weak password[12]; this could be instead of, or in addition to, a GP threshold on the initial password. Once again, user studies would be required in order to determine how to maximize acceptance of this type of password composition policy.

[12] This is reminiscent of the `comprehensive8` composition policy of (Kelley, et al., 2012).

Bibliography

Abadi, M., Mark, T., Lomas, A., & Needham, R. (1997). *Strengthening Passwords.* Palo Alto, CA: Digital Equipment Corporation; SRC Technical Note.

Adams, A., & Sasse, M. A. (1999, December). Users are not the enemy. *Communications of the ACM, 42*(12), 40-46.

Bonneau, J. (2012). The science of guessing: analyzing an anonymized corpus of 70 million passwords. *IEEE Symposium on Security and Privacy.* San Francisco, CA, USA: IEEE Computer Society.

Bonneau, J., Herley, C., van Oorschot, P. C., & Stajano, F. (2012). The quest to replace passwords: A framework for comparative evaluation of web authentication schemes. *IEEE Symposium on Security and Privacy* (pp. 553-567). San Francisco, CA, USA: IEEE Computer Society.

Bonneau, J., Just, M., & Matthews, G. (2010). What's in a Name? Evaluating Statistical Attack on Personal Knowledge Questions. *Financial Cryptography and Data Security '10.*

Castelluccia, C., Perito, D., & Duermuth, M. (2012). Adaptive Password-Strength Meters from Markov Models. *Network and Distributed Systems Security Symposium*.

Dell'Amico, M., Michiardi, P., & Roudier, Y. (2010). Password Strength: an empirical analysis. *Proceedings of the 29th conference on Information Communications* (pp. 983-991). San Diego, CA: IEEE Press.

Florencio, D., & Herley, C. (2007). A large-scale study of web password habits. *Proceedings of the 16th international conference on World Wide Web* (pp. 657-666). New York, NY, USA: ACM.

Forget, A., & Biddle, R. (2008). Memorability of persuasive passwords. *CHI '08 Extended Abstracts on Human Factors in Computing Systems* (pp. 3759-3764). New York, NY, USA: ACM.

Forget, A., Chiasson, S., & Biddle, R. (2007). Helping users create better passwords: is this the right approach? *Proceedings of the 3rd symposium on Usable privacy and security* (pp. 151-152). New York, NY, USA: ACM.

Forget, A., Chiasson, S., van Oorschot, P. C., & Biddle, R. (2008). Improving text passwords through persuasion. *Proceedings of the 4th symposium on Usable privacy and security* (pp. 1-12). New York, NY, USA: ACM.

Forget, A., Chiasson, S., van Oorschot, P., & Biddle, R. (2008). Persuasion for stronger passwords: Motivation and pilot study. *Proceedings of the 3rd international conference on Persuasive Technology* (pp. 140-150). Berlin, Heidelberg: Springer-Verlag.

Garrison, C. P. (2006). Encouraging good passwords. *Proceedings of the 3rd annual conference on Information security curriculum development* (pp. 109-112). New York, NY, USA: ACM.

Houshmand, S., & Aggarwal, S. (2012). Building better passwords using probabilistic techniques. *Proceedings of the 28th Annual Computer Security Applications Conference* (pp. 109-118). New York, NY, USA: ACM.

IEEE Computer Society. (2012). *IEEE Symposium on Security and Privacy, SP 2012, 21-23 May 2012, San Francisco, CA, USA.*

Inglesant, P. G., & Sasse, M. A. (2010). The true cost of unusable password policies: password use in the wild. *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (pp. 383-392). New York, NY, USA: ACM.

Kelley, P. G., Komanduri, S., Mazurek, M. L., Shay, R., Vidas, T., Bauer, L., et al. (2012). Guess again (and again and again): Measuring password strength by simulating password-cracking algorithms. *IEEE Symposium on Security and Privacy*, (pp. 523-537).

Komanduri, S., Shay, R., Kelley, P. G., Mazurek, M. L., Bauer, L., Christin, N., et al. (2011). Of passwords and people: measuring the effect of password-composition policies. *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (pp. 2595-2604). New York, NY, USA: ACM.

Kuo, C., Romanosky, S., & Cranor, L. F. (2006). Human selection of mnemonic phrase-based passwords. *Proceedings of the second symposium on Usable privacy and security* (pp. 67-78). New York, NY, USA: ACM.

Leonhard, M. D., & Venkatakrishnan, V. N. (2007). A comparative study of three random password generators. *EIT '07: Proceedings of the 2007 IEEE International Conference on Electro/Information Technology.*

Ma, W., Campbell, J., Tran, D., & Kleeman, D. (2010). Password entropy and password quality. *Proceedings of the 2010 Fourth International Conference on Network and System Security* (pp. 583-587). Washington, DC, USA: IEEE Computer Society.

Marechal, S. (2008). Advances in password cracking. *Journal of Computer Virology, 4*(1), 73-81.

Morris, R., & Thompson, K. (1979). Password security: A case history. *Communications of the ACM, 22*, 594-597.

Narayanan, A., & Shmatikov, V. (2005). Fast dictionary attacks on passwords using time-space tradeoff. *Proceedings of the 12th ACM conference on Computer and communications security* (pp. 364-373). New York, NY, USA: ACM.

Peslyak, A. (n.d.). *John the Ripper password cracker*. Retrieved from www.openwall.com/john

Pinkas, B., & Sander, T. (2002). Securing passwords against dictionary attacks. *Proceedings of the 9th ACM conference on Computer and communications security* (pp. 161-170). New York, NY, USA: ACM.

Shay, R., & Bertino, E. (2009). A comprehensive simulation tool for the analysis of password policies. *International Journal Information Security, 8*(4), 275-289.

Shay, R., Kelley, P. G., Komanduri, S., Mazurek, M. L., Ur, B., Vidas, T., et al. (2012). Correct horse battery staple: exploring the usability of system-assigned passphrases. *Proceedings of the Eighth Symposium on Usable Privacy and Security* (pp. 7:1-7:20). New York, NY, USA: ACM.

Shay, R., Komanduri, S., Kelley, P. G., Leon, P. G., Mazurek, M. L., Bauer, L., et al. (2010). Encountering stronger password requirements: user attitudes and behaviors. *Proceedings of the Sixth Symposium on Usable Privacy and Security* (pp. 2:1-2:20). New York, NY, USA: ACM.

Skull security. (n.d.). Retrieved 4 29, 2013, from Publicly available leaked password databases: http://www.skullsecurity.org/wiki/index.php/passwords

Ur, B., Kelley, P. G., Komanduri, S., Lee, J., Maass, M., Mazurek, M. L., et al. (2012). How does your password measure up? The effect of strength meters on password creation. *Proceedings of the 21st USENIX conference on Security symposium* (pp. 1-5). Berkeley, CA, USA: USENIX Association.

Vu, K.-P. L., Proctor, R. W., Bhargav-Spantzel, A., Tai, B.-L. B., Cook, J., & Eugene Schultz, E. (2007). Improving password security and memorability to protect personal and organizational information. *International Journal of Human-Computer Studies, 65*(8), 744-757.

Weir, M., Aggarwal, S., Collins, M., & Stern, H. (2010). Testing metrics for password creation policies by attacking large sets of revealed passwords. *Proceedings of the 17th ACM*

conference on Computer and communications security (pp. 162-175). New York, NY, USA: ACM.

Weir, M., Aggarwal, S., d. Medeiros, B., & Glodek, B. (2009). Password cracking using probabilistic context-free grammars. *Proceedings of the 2009 30th IEEE Symposium on Security and Privacy* (pp. 391-405). Washington, DC, USA: IEEE Computer Society.

Xu, Y., Reynaga, G., Chiasson, S., Frahm, J.-M., Monrose, F., & Van Oorschot, P. (2012). Security and usability challenges of moving-object captchas: decoding codewords in motion. *Proceedings of the 21st USENIX conference on Security symposium* (pp. 1-4). Berkeley, CA, USA: USENIX Association.

Yan, J., Blackwell, A., Anderson, R., & Grant, A. (2004). Password memorability and security: Empirical results. *IEEE Security and Privacy, 2*(5), 25-31.