Pitfalls in the Automated Strengthening of Passwords

David Schmidt
Department of Computer Science & Engineering
Pennsylvania State University
dave.schmidt.pa@gmail.com

Trent Jaeger
Department of Computer Science & Engineering
Pennsylvania State University
tjaeger@cse.psu.edu

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 paper shows that passwords that were strengthened against the best known cracking algorithms are still susceptible to attack, provided the adversary knows the strengthening algorithm. We propose two attacks: (1) by strengthening the data 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, we find that the proposed strengthening algorithms do not yet satisfy Kerckhoffs's principle.

Categories and Subject Descriptors

K.6.5 [**Management of Computing and Information Systems**]: Security and Protection—*authentication*

General Terms

Security, Passwords, Experimentation

Keywords

Password checking, Password creation policies

ACSAC '13 December 9-13, 2013, New Orleans, Louisiana USA

http://dx.doi.org/10.1145/2523649.2523651

1. INTRODUCTION

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 [4].

To motivate users to consider the security of their passwords, many websites utilize password meters to provide feedback regarding the guessability 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 over three decades ago identified several commonly used guessing techniques, such as dictionary words, common proper nouns, and common numbers [19]. More recently, researchers have developed guessing algorithms based on Markov chains [20] and probabilistic context free grammars [26] (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 [9, 10, 12]. 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 we may be overlooking possible vulnerabilities in such methods. 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 that 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 paper makes the following contributions:

- In Section 5.1.1, we demonstrate 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 Section 5.2 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 Section 5.1.2. We find that the leakage of the strengthening data typically retained would enable an adversary to crack 25% of the passwords in less than a day. We find that if we do not store information about the system's actual passwords, leakage of the strengthening data provides adversaries no advantage for recovering passwords, except for a modest impact on the quality of strengthening.

The outline of this paper is as follows. In Section 2 and Section 3, there is an introduction to state-of-the-art password crackers and a discussion of various approaches used in password strengthening. Section 4 discusses the data and the algorithms used to measure cracking time and adaptively strengthen passwords. In Section 5, the effectiveness of two different methods of cracking strengthened passwords is analyzed. Lastly in Section 6, the results are summarized and directions for future work are given.

2. BACKGROUND

It is well known that users tend to create weak passwords [6, 28]. Adams and Sasse [2] point out that this is to be expected as users are typically unaware of what makes a password more secure. 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.

2.1 Markov Chains

Narayanan and Shmatikov [20] describe a password cracking algorithm based on Markov models. 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'. The concept can be extended to character sequences of arbitrary length, called n-grams; the 3-gram (or tri-gram) 'thr' is far more likely to be followed by a vowel than by a consonant. 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 the publicly available John the Ripper program [22]. 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 [18].

2.2 Probabilistic Context-Free Grammars

Weir et al. [26] proposed using a probabilistic context-free grammar (PCFG) for password cracking. 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. $S_1U_1L_6D_2$ represents a special character followed by an uppercase letter, then 6 lowercase letters, and ending with two digits). PCFG attacks 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.

2.3 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 in [21], 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). 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 ease of cracking. Since password distributions are different from website to website [21], 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. 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 in [5] as method of improving security.

3. RELATED WORK

There is a large and growing body of literature on the insecure password choices that users tend to make [2, 6, 3, 28, 11]. This tendency is understandable as users must balance the competing goals of security and memorability [8, 28]. Passwords that are more random (through composition policies or system generation) or contain more characters are harder for password crackers to guess [14]. However, users have trouble remembering random or complex passwords [7, 16] and resort to insecure workarounds, such as writing down the password or following predictable patterns to meet password requirements [13, 15, 24, 27, 3, 23].

Composition policies such as requiring a digit, a symbol and/or an uppercase letter are very commonly used techniques to increase the security of user's passwords. In [15, 27, 24], the authors found that in order to satisfy the composition requirements, users tend to make incremental changes to an old password, rather than creating new ones. The authors of [14] report that the composition policies of *basic*16 and *comprehensive*8 exhibit the best resistance to cracking attacks. The first policy allows any password of length 16 or longer. The second policy mandates that all four character classes be present, and, after stripping out the digits and symbols, the remaining letters cannot spell a word.

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 [27, 17, 3], 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. In [25], the authors 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.

Automated methods for improving a user-selected password have been suggested. One such approach is Persuasive Text Passwords (PTP), as described in [9, 10]. 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 [27], although memorability of the new password is an issue. The authors conclude that two randomly inserted characters was the best tradeoff between increase in security versus memorability. The authors of [12] describe a very similar approach in terms of strengthening user passwords by using a PCFG database obtained by training on a previously leaked set of passwords together with a set of editing rules.

The authors in [21] use Markov chains to estimate the strength of a user-supplied password and use that estimate of strength in an adaptive strength meter shown to the user. 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.

The work in this article extends the above research in the following ways. A password-scoring mechanism such as that described in [21] 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 [9, 10, 12], 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.

4. EXPERIMENTAL SETUP

This section first describes the data used. It then describes the method used to estimate password strength and how that measure of strength is used within a password guessing algorithm. Additionally, the algorithm used to strengthen passwords is described.

4.1 Data

Multiple leaked password lists exist in the public domain. Using actual passwords lists such as those found at [1] is the best way to study password distributions and cracking algorithms in a large scale manner. This paper uses the passwords from the rockyou website; rockyou had no composition policy and a stated minimum length of 5.

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, it should either be successful in strengthening the user's password or it should inform the user that the password cannot be strengthened. Hence, stringent initial constraints on the password should be unnecessary.

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 of these sets were randomly divided into subsets A, B and C in order to bootstrap the algorithm; see Section 4.4.[1]

4.2 Measuring Password Strength

To measure password strength, a program to calculate the "guess probability" (GP) of a password was implemented. As described in [26], GP is computed by a PCFG as the product of the base structure (i.e., L_6D_2) probability and the probabilities of the strings which fill each variable (i.e., L_6 and D_2), with the strings selected from an input dictionary. Consequently, only strings that are present in the dictionary can be guessed. To avoid this restriction, the GP calculation used in this paper uses both a dictionary and a Markov chain that are built from the training data. Since each Markov chain can produce any string within its domain (digits, symbols, etc.), the Markov chain can be thought of as a compact representation of a comprehensive dictionary, capable of generating a complete list of strings and 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, an n-gram chain for alphabetic strings of length n or longer and a 1-gram chain for alphabetic strings shorter than n.[2] 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 the frequencies with which they occur. By tracking the observed frequency for each string, the dictionaries capture sequences that appear more frequently than the product of the corresponding transition probabilities would indicate, thereby significantly increasing the number of passwords that can be guessed in a set interval of time.

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

$$GP_{MC}(token) = max(ObservedFrequency(token), prob(token_{1,n}) * \prod_{i=n}^{L-1} TP(token_{i-n+1,i}, token_{i-n+2,i+1})) \quad (1)$$

where $token_{i,j}$ is the substring from characters i to j and TP is the transition probability from one n-gram to another. For common tokens, GP_{MC} is $ObservedFrequency$, which is the ratio of the number of occurrences of *token* to the total number of strings seen of that same length. 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. All probabilities referenced are computed from the training set.[3]

The strength calculator combines the probabilities from the dictionaries, Markov chains and PCFG to compute the password's GP as:

$$GP(password) = SP(password) * \prod_{s \in SS(password)} GP_{MC}(s) \quad (2)$$

where SP is the observed probability of the password structure (e.g., $L_4S_2D_1L_4$) within the PCFG, and SS returns substrings from the password based off the structure (i.e. "pass**1word" would return {"pass", "**", "1", "word"}).

4.3 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 1 was used to generate all passwords at or below a threshold GP ($minGP$) using password structures, dictionaries and Markov chains as described in Section 4.2.

```
Data: minGP, cracking data
Result: output list of recovered passwords
for each password structure SS do
    try all passwords with GP ≥ minGP using Alg. 2;
end
```

Algorithm 1: Cracking Program

```
Data: currentGuess, cumulativeGP, password structure
      SS, minGP, list of valid passwords
Result: output list of recovered passwords
if SS is empty then
    if currentGuess is a valid password then
        output G;
    end
end
else
    S = first element of SS;
    MC = the Markov chain used to guess S;
    for each word W ∈ MC's dictionary do
        if cumulativeGP * GP_MC(W) ≥ minGP then
            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) ≥ minGP/cumulativeGP;
    for each word W in L do
        recursive call with currentGuess + W,
        cumulativeGP * GP_MC(W), SS − S;
    end
end
```

Algorithm 2: Guess Generating Algorithm

Algorithm 1 invokes the guess generating function (Algorithm 2) for each password structure (e.g., $L_8D_1S_1$). This initial call to Algorithm 2 uses an empty *currentGuess*, $SP(SS)$ for *cumulativeGP* and the structure SS.

Whenever the guess generator (Algorithm 2) is called for a particular password structure (e.g., $L_8D_1S_1$), it fills in the first component of the structure (L_8) and makes a recursive call to fill the remaining parts of the structure (D_1S_1). At each stage of both algorithms, selections are made in descending probability order: the most common password

[1] All experiments presented in this paper were repeated on a second random division of rockyou, with the results from the second sets matching the first.

[2] Tri-grams were used for alphabetic strings throughout the analysis as they appear to be the most effective.

[3] 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.

Table 1: Run times for Algorithm 1

Min GP	# Guesses	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)

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 1 to utilize multiple processors is a simple matter of splitting the password structures into P groups, where P is the number of processors to use. This approach to distributing 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.

Both the number of guesses actually made and the run time for a range of GPs are shown in Table 1[4]. 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 on these results, a GP of 10^{-15} can be considered fairly secure. Consequently, in the remainder of this paper, passwords will be strengthened to a GP of 10^{-16} to allow for an additional margin of safety.

4.4 Strengthening Algorithm

The basic strengthening algorithm used in this paper is presented as Algorithm 3 which references a *strengthening database*. The strengthening database refers to the collection of password structures, Markov chains and dictionaries and their associated probabilities as described in 4.2. If these same items are used in an attack, rather than in a strengthening system, they are referred to as a *cracking database*.

The way in which the strengthening database is built is important, as will be detailed in Section 5.1.2. Passwords in subset A (the original training data) are not strengthened and are fully processed into the strengthening database, meaning that both the probability tables and dictionaries are updated as described in Section 4.2. 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. The rationale for this will be detailed in Section 5.1.2.

For the analysis in this paper, 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 all strengthened passwords are accepted is likely a best-case scenario: it is possible 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.

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

```
Data: password list X, threshold guess probability TP,
      number of edits N
Result: strengthed password list X'
create empty strengthening database SDB;
for each password PW in subset A of X do
    fully process PW into SDB;
end
for each password PW in subsets B and C of X do
    PW' = PW;
    thisGP = GP(PW');
    while thisGP > TP and maximum number of
    attempts not exceeded do
        make N edits to PW, yielding PW';
        thisGP = GP(PW');
    end
    if thisGP ≤ TP then
        output PW' to X';
        partially process PW' into SDB;
    end
    partially process PW into SDB;
end
```

Algorithm 3: Basic Strengthening Algorithm

Algorithm 3 is similar to the strengthening algorithm used in [12], with no major conceptual differences: there is a training phase to build an initial strengthening database, passwords are evaluated against this database, and 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 [12], every password is "fully processed" into the strengthening database. However, those GP calculations utilized a static dictionary for alphabetic strings, rather than a dictionary dynamically built from the observed passwords. Hence, the impact of the strengthening database being leaked is not directly comparable to results presented in this paper.

While the strengthening algorithm in [10] could also apply random edits to an initial password, there was a large conceptual difference: passwords were altered without regard to their initial strength. Consequently, while the ending password was almost certainly more secure than the initial password, there was no mechanism to guide the ending password to a given level of strength.

4.5 Strengthened Datasets

A set of strengthened passwords was created by strengthening rockyou-1 and rockyou-2 according to Algorithm 3, 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 Per-

suasive Text Passwords (PTP) [9, 10], no restrictions were placed on the characters used in the edits – any printable character (ASCII 32 to 126, inclusive) could be used.

In contrast, the authors of [12] 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. Thus, "password123" could be transformed to "passwoRd123", "password1213" or "password!123" but not "pass5word123" or "password12h3".

5. RESULTS

This section first explores the ability of the strengthening algorithm to prevent passwords from being guessed by a PCFG-based attack. A second type of attack, guided brute force (GBF) search, is then explored. The section concludes with methods that can be taken to reduce the effectiveness of this second form of attack.

5.1 Resistance to PCFG-based Attacks

Algorithm 3 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 database might be judged as weak when using a different set of data to measure strength. In this section, two possibilities for PCFG-based attacks are explored. In section 5.1.1, the viability of attacking the strengthened passwords using out-of-band data is explored, while in section 5.1.2 the impact of an accidental leak of the strengthening database is investigated.

5.1.1 Attacking with Derived Data

A well-known attack vector is to train a password cracking program on a leaked data set such as rockyou, and use the derived data as the basis for an attack. However, 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 only 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 1 to mount a guessing attack was calculated in two ways: from the original, unstrengthened rockyou-1 set and also from the rockyou-1 set that was strengthened by Algorithm 3. In Table 2, a breakdown of the GPs of the strengthened passwords under these attack scenarios is given; the rows labeled Weak reflect the first scenario (using only the original rockyou-1 passwords), while the rows labeled Strong reflect the second scenario (using only strengthened rockyou-1 passwords). If the strengthening algorithm were perfect, the worst GP under any scenario would be the level targeted, or 10^{-16} and Table 2 would be all 0s. Results are shown for both 1 and 2 edits.

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.

Table 2: GPs using Derived Data

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

Table 3: GPs using Leaked Data

Full	Partial	Edits	% 10^{-13}	% 10^{-14}	% 10^{-15}
–	All	1	0.0	0.1	1.5
–	All	2	0.0	0.0	0.6
All	–	1	58.2	67.3	75.6
All	–	2	28.2	38.6	50.0
Orig.	Str.	1	21.2	24.4	28.7
Orig.	Str.	2	6.4	8.3	10.3

Moving to the bottom panel, we see 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, with only 1.3% of the strengthened passwords recoverable in a week.

5.1.2 Attacking with Leaked Data

Alternatively, 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 passwords, this seems to be a worst-case scenario.

In Table 3 the GPs for the strengthened passwords are shown for three different scenarios, each representing a variation of Algorithm 3. The top panel utilizes Algorithm 3 as presented. The middle panel shows the consequences of fully processing all original and strengthened passwords; in other words, Algorithm 3 would be modified so that every string in both the original and strengthened passwords would enter the dictionaries. The bottom panel shows the results if Algorithm 3 were modified so that it fully processed all original passwords but still only partially processed the strengthened passwords. In all scenarios, the training phase (subset A) fully processes both the original and unstrengthened passwords; partial processing can only occur in the strengthening phase (subsets B and C), if at all.

The top panel is rather surprising as it shows that the strengthening database for Algorithm 3, as presented, provides less information to an attacker than does the Weak data set of Table 2. This lack of success is attributable to the use of full versus partial processing of the passwords in Algorithm 3, 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 Section 4.4, 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 2. In the latter case, the data used in cracking only reflects 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 password GPs. Because the dictionaries are built only from the training data in Algorithm 3, 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 generate the password "pearlharbor1" quickly.

Clearly, 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 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 a PCFG-based attack. Table 4 shows the results of a PCFG-based attack on the strengthened passwords under the three full/partial processing combinations used in Table 3: partial processing of both the original and strengthened passwords, corresponding to the top panel of Table 3; full processing on both passwords, corresponding to the middle panel; and fully processed original passwords but partially processed strengthened passwords, corresponding to the bottom panel. In each of these cases, the data used to mount the attack is calculated exclusively from passwords strengthened from the rockyou-1 set; this corresponds to the Strong rows in Table 2.

The alternative constructions do produce strengthened passwords which are more resistant to a PCFG-based attack. However, the differential in the percent of recoverable passwords is slight, particularly if two edits are used. Nonethless, if the probability of a leak is deemed sufficiently small, the superior resilience to cracking seen in the alternative constructions could be judged as a risk worth taking.

Recall that Algorithm 3 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 [12] were used, the results are comparable to the figures in Tables 2 and 3, with no meaningful changes. However, as discussed below, the restricted editing does have a significant impact in a different context.

Table 4: GPs using Different Dictionaries

Full	Partial	Edits	% 10^{-13}	% 10^{-14}	% 10^{-15}
–	All	1	2.5	4.6	18.0
–	All	2	0.4	1.3	7.6
All	–	1	2.0	3.7	16.4
All	–	2	0.3	1.1	6.4
Orig.	Str.	1	2.4	4.4	17.7
Orig.	Str.	2	0.3	1.3	7.5

5.2 Resistance to Brute Force Attacks

A second form of attack against strengthened passwords would be for an attacker to try to guess the original, unstrengthened password, and then to test all possible variants. This attack is a brute force approach, but it is guided by the passwords generated by Algorithm 1. 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 D_2L_6. The strengthened password "12qwasJx" has GP of 1.75×10^{-17}, reflecting the extreme improbability of the 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 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.[5] A similar state of affairs is shown in [12]: there the authors present an example that shows transforming the password "life45!" to "life^45!" decreases the GP by four orders of magnitude – 1.1×10^{-12} to 1.8×10^{-16}.

In order to determine how many passwords could potentially be guessed using this brute force approach, we need to know the GPs of the passwords which were strengthened. Passwords with GPs no stronger than 10^{-12} can be guessed in about two hours, and thus strengthened passwords built from these passwords may be vulnerable to a brute force attack. Table 5 shows a breakdown of the GPs of the passwords which were strengthened by Algorithm 3 using one or two edits. Results are also shown for full editing (no restrictions on characters used, denoted "Y") or restricted editing (as in [12], denoted "N").

The percentages in Table 5 reveal that a GBF attack as

[5] 855 variants from inserting one of 95 characters at 9 possible locations, plus 752 variants from changing any of the eight characters to one of 94 others.

Table 5: GPs for Passwords Needing Strengthening

Edits	Full?	% 10^{-9}	% 10^{-10}	% 10^{-11}	% 10^{-12}
1	Y	13.3	22.2	35.2	54.0
1	N	2.4	4.4	6.4	11.4
2	Y	26.5	41.6	60.3	74.5
2	N	4.9	9.5	25.0	49.0

Table 6: Guided Brute Force Run Times

Min GP	Edits	Full?	Run Time, 12 cores
10^{-9}	1	Y	1.2 hours
10^{-9}	1	N	8 minutes
10^{-10}	1	Y	12.7 hours
10^{-10}	1	N	1.3 hours
10^{-11}	1	Y	1 week (est)
10^{-11}	1	N	16.2 hours
10^{-9}	2	Y	Guessed 5.4% in 24 hours
10^{-9}	2	N	20.4 hours

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 1, a search of all passwords at this level of GP can be done in a minute. 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 more difficult 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 is presented in Section 5.3.

To determine run times for GBF attacks, the guessing algorithm shown in Algorithm 2 was modified so that whenever *currentGuess* was checked against the list of passwords, all variants of *currentGuess* were generated and tested as well. The results are presented in Table 6.

As was seen in Table 5, restricting the edits ("Full?" column of "N") greatly reduces the number of weak passwords that can be strengthened, which reduces the effectiveness of a GBF attack. Table 6 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?" column of "Y") an additional order of magnitude of GPs can be explored when restricted edits were used.

The runs which applied two edits would clearly take a long time to finish. However, the runs for passwords with GPs no stronger than 10^{-9} 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, the strengthening algorithm has therefore not met its goal, despite the fact that the strengthened passwords are significantly harder to guess than the original passwords. Additional, longer runs are required to better assess the overall security risks.

The GBF search actually guessed slightly more passwords than was indicated by Table 5 as some of the passwords which did not need strengthening were within one or two edits of a weak password. For example, the user 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. Because of these "extra" hits, the completed run in the bottom panel of Table 6 guessed 5.1% of the passwords, rather than 4.9% shown in Table 5.

5.3 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 in [9, 10], going beyond two *random* edits significantly impairs memorability, so two random edits should be considered the limit when using Algorithm 3. A second method would be to require the initial password to be stronger, thus harder to guess, making a GBF attack take longer. In short, this would require the user to clear an intermediate GP hurdle, with Algorithm 3 then decreasing the GP to a secure level with the resulting password resistant to GBF attack.

Although this approach requires more of the user, tools could be provided to assist with the initial selection. For instance, in the event the original password's GP did not meet a preliminary threshold, the user could be given high-level, non-specific feedback on how to make the initial password stronger such as to make the password longer or to use more character classes. This is admittedly a difficult challenge, as we have already 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, find ways to satisfy the requirement but still 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 7 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 vulnerable to guessing within a week (GP of 10^{-14}), which compares favorably to the 4.6% figure in the bottom

Table 7: GPs for Strengthened Passwords, Screened Subset

Edits	Full?	% 10^{-13}	% 10^{-14}	% 10^{-15}
1	Y	0.5	0.9	2.2
1	N	0.6	1.4	3.5
2	Y	0.4	0.7	1.3
2	N	0.5	0.9	2.0

Table 8: GPs for Passwords Needing Strengthening, Screened Subset

Edits	Full?	% 10^{-9}	% 10^{-10}	% 10^{-11}	% 10^{-12}
1	Y	0.0	0.1	1.2	12.7
1	N	0.0	0.1	0.7	5.5
2	Y	0.0	0.1	1.2	13.9
2	N	0.0	0.1	1.2	13.6

panel of Table 2. 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.

While the starting passwords in this section were initially screened to have a GP of 10^{-12} or stronger, recalling the "pearlharbor1" example, we do not anticipate that 100% of the initial, unstrengthened passwords to meet this threshold if a different database is used to measure GPs. As seen in Table 8, 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, all of the percentages are markedly lower. Consequently, it will be harder to guess the password which was strengthened which means that a GBF attack will take significantly longer.

As noted in Section 5.2, 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 9 similar to those in Table 6.

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 Section 5.2. However, the last panel of Table 9 is cause for cautious optimism. Here, as in Table 6, the attacks did not run to completion but were halted after a reasonable period of time.[6]. Unlike Table 6, we see that essentially none of the strengthened passwords were guessed within a day. Although multi-day runs are required to better assess the 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 7).

Table 9: GBF Run Times, Screened Subset

Min GP	Edits	Full?	Run Time, 12 cores
10^{-9}	1	Y	20 minutes
10^{-9}	1	N	3 minutes
10^{-10}	1	Y	6.5 hours
10^{-10}	1	N	1 hour
10^{-11}	1	Y	5 days (est)
10^{-11}	1	N	17 hours
10^{-12}	1	Y	Guessed 0.3% in 24 hours
10^{-12}	1	N	Guessed 0.6% in 24 hours
10^{-12}	2	Y	3 passwords in 24 hours
10^{-12}	2	N	28 passwords in 24 hours

6. CONCLUSION

After strengthening two large sets of passwords (rockyou-1 and rockyou-2), the strengthened passwords 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.

It was also shown that an alternative type of attack could be even more successful. Since the original password is likely to be weak and therefore easily guessed, systematic guessing of all possible variations of passwords that can be quickly generated by a PCFG-based algorithm is an effective attack, limited only by the computing time required. Even though the search space is too large to search exhaustively when two edits are applied, we nonetheless found that 5% of passwords strengthened with two random edits could be successfully guessed in the first 24 hours. 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 guessable within 24 hours. Still, roughly 1% of the strengthened passwords remain vulnerable to a PCFG-based attack even with the stronger initial password.

Although the use of a significantly stronger initial password was successful in markedly reducing the effectiveness of a GBF attack, a GP of 10^{-12} is a fairly high hurdle, with only 7% of the passwords in the rockyou list meeting this criterion. Since users do not seem predisposed to producing passwords possessing this level of strength, one area of future research would be the use of specific feedback (e.g., telling the user not to use "pearlharbor") which may help guide users to selecting stronger initial passwords. More efficient algorithms to search the strengthened space could also be investigated, in order to better understand the security risk posed by GBF attacks. The efficiency of the algorithm used in this paper could be improved. In particular, there was some redundancy as edits to the structures L_7D_1 and L_6D_2 both generate some passwords of the form 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 ([9, 10]) has shown that two *random* edits should be considered an upper limit with respect to maintaining memorability, other 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 a string within the password (transforming "rock&roll" to "rock&llor"). It may also be benefi-

[6]For the partial runs in Table 9, only passwords with a GP stronger than 10^{-11} were modified due to the low probability of success when modifying weaker passwords

cial to have the strengthening algorithm reject any starting password which is within two edits of a weak password[7]; this could be instead of, or in addition to, a GP threshold on the initial password. Once again, user studies are required to determine user acceptance of this policy.

7. ACKNOWLEDGEMENTS

The first author wishes to thank Gregory Ference and Megan Heysham, with whom he partnered on a preliminary project which spawned the ideas contained herein.

8. REFERENCES

[1] Publicly available leaked password databases : http://www.skullsec..org/wiki/index.php/passwords.

[2] Adams, A., and Sasse, M. A. Users are not the enemy. *Commun. ACM 42*, 12 (Dec. 1999), 40–46.

[3] Bonneau, J. The science of guessing: analyzing an anonymized corpus of 70 million passwords. In *IEEE Symp. Sec. and Priv.* (2012).

[4] Bonneau, J., Herley, C., Oorschot, P. C. v., and Stajano, F. The quest to replace passwords: A framework for comparative evaluation of web authentication schemes. In *Proc. of the 2012 IEEE Symp. on Sec. and Priv.* (Washington, DC, USA, 2012), SP '12, IEEE Computer Society, pp. 553–567.

[5] Bonneau, J., Just, M., and Matthews, G. What's in a Name? Evaluating Statistical Attacks on Personal Knowledge Questions. In *Financial Cryptography and Data Sec. '10* (2010).

[6] Florencio, D., and Herley, C. A large-scale study of web password habits. In *Proc. of the 16th Int. Conf. on WWW* (New York, NY, USA, 2007), WWW '07, ACM, pp. 657–666.

[7] Forget, A., and Biddle, R. Memorability of persuasive passwords. In *CHI '08 Extended Abstracts on Human Factors in Computing Sys.* (New York, NY, USA, 2008), CHI EA '08, ACM, pp. 3759–3764.

[8] Forget, A., Chiasson, S., and Biddle, R. Helping users create better passwords: is this the right approach? In *Proc. of the 3rd Symp. on Usable Priv. and Sec.* (New York, NY, USA, 2007), SOUPS '07, ACM, pp. 151–152.

[9] Forget, A., and et al. Improving text passwords through persuasion. In *Proc. of the 4th Symp. on Usable Priv. and Sec.* (New York, NY, USA, 2008), SOUPS '08, ACM, pp. 1–12.

[10] Forget, A., and et al. Persuasion for stronger passwords: Motivation and pilot study. In *Proc. of the 3rd Int. Conf. on Persuasive Technology* (Berlin, Heidelberg, 2008), PERSUASIVE '08, Springer-Verlag, pp. 140–150.

[11] Garrison, C. P. Encouraging good passwords. In *Proc. of the 3rd Annual Conf. on Information Sec. Curriculum Development* (New York, NY, USA, 2006), InfoSecCD '06, ACM, pp. 109–112.

[12] Houshmand, S., and Aggarwal, S. Building better passwords using probabilistic techniques. In *Proc. of the 28th Annual Computer Sec. Applications Conf.* (New York, NY, USA, 2012), ACSAC '12, ACM, pp. 109–118.

[7]This is reminiscent of the *comprehensive*8 policy of [14]

[13] Inglesant, P. G., and Sasse, M. A. The true cost of unusable password policies: password use in the wild. In *Proc. of the SIGCHI Conf. on Human Factors in Computing Sys.* (New York, NY, USA, 2010), CHI '10, ACM, pp. 383–392.

[14] Kelley, P. G., and et al. Guess again (and again and again): Measuring password strength by simulating password-cracking algorithms. In *IEEE Symp. on Sec. and Priv.* (2012), pp. 523–537.

[15] Komanduri, S., and et al. Of passwords and people: measuring the effect of password-composition policies. In *Proc. of the SIGCHI Conf. on Human Factors in Computing Sys.* (New York, NY, USA, 2011), CHI '11, ACM, pp. 2595–2604.

[16] Leonhard, M. D., and Venkatakrishnan, V. N. A comparative study of three random password generators. In *EIT'07: Proc. 2007 IEEE Int. Conf. on Electro/Information Tech.* (2007).

[17] Ma, W., and et al. Password entropy and password quality. In *Proc. of the 2010 Fourth Int. Conf. on Network and System Sec.* (Washington, DC, USA, 2010), NSS '10, IEEE Computer Society, pp. 583–587.

[18] Marechal, S. Advances in password cracking. *Journal in Computer Virology 4* (2008), 73–81.

[19] Morris, R., and Thompson, K. Password sec.: A case history. *Comm. of the ACM 22* (1979), 594–597.

[20] Narayanan, A., and Shmatikov, V. Fast dictionary attacks on passwords using time-space tradeoff. In *Proc. of the 12th ACM Conf. on Computer and Comm. Sec.* (New York, NY, USA, 2005), CCS '05, ACM, pp. 364–372.

[21] Perito, D., Castelluccia, C., and Duermuth, M. Adaptive password-strength meters from markov models. In *Network and Dist. Sys. Sec. Symp.* (2012).

[22] Peslyak, A. John the ripper.

[23] Shay, R., and Bertino, E. A comprehensive simulation tool for the analysis of password policies. *Int. J. Inf. Secur. 8*, 4 (Aug. 2009), 275–289.

[24] Shay, R., and et al. Encountering stronger password requirements: user attitudes and behaviors. In *Proc. of the 6th Symp. on Usable Priv. and Sec.* (New York, NY, USA, 2010), SOUPS '10, ACM, pp. 2:1–2:20.

[25] Ur, B., and et al. How does your password measure up? the effect of strength meters on password creation. In *Proc. of the 21st USENIX Conf. on Sec. Symp.* (Berkeley, CA, USA, 2012), Sec.'12, USENIX Association, pp. 5–5.

[26] Weir, M., Aggarwal, S., Medeiros, B. d., and Glodek, B. Password cracking using probabilistic context-free grammars. In *Proc. of the 2009 30th IEEE Symp. on Sec. and Priv.* (Washington, DC, USA, 2009), SP '09, IEEE Computer Society, pp. 391–405.

[27] Weir, M., and et al. Testing metrics for password creation policies by attacking large sets of revealed passwords. In *Proc. of the 17th ACM Conf. on Computer and Comm. Sec.* (New York, NY, USA, 2010), CCS '10, ACM, pp. 162–175.

[28] Yan, J., and et al. Password memorability and sec.: Empirical results. *IEEE Sec. and Priv. 2*, 5 (Sept. 2004), 25–31.