Effect of Grammar on Security of Long Passwords

Ashwini Rao
Carnegie Mellon University
arao@cmu.edu

Birendra Jha
Massachusetts Institute of Technology
bjha@mit.edu

Gananand Kini
Carnegie Mellon University
ganu@cmu.edu

ABSTRACT

Use of long sentence-like or phrase-like passwords such as "abiggerbetterpassword" and "thecommunistfairy" is increasing. In this paper, we study the role of grammatical structures underlying such passwords in diminishing the security of passwords. We show that the results of the study have direct bearing on the design of secure password policies, and on password crackers used for enforcing password security. Using an analytical model based on Parts-of-Speech tagging we show that the decrease in search space due to the presence of grammatical structures can be more than 50%. A significant result of our work is that the strength of long passwords does not increase uniformly with length. We show that using a better dictionary e.g. Google Web Corpus, we can crack more long passwords than previously shown (20.5% vs. 6%). We develop a proof-of-concept grammar-aware cracking algorithm to improve the cracking efficiency of long passwords. In a performance evaluation on a long password dataset, 10% of the total dataset was exclusively cracked by our algorithm and not by state-of-the-art password crackers.

Categories and Subject Descriptors: D.4.6 [**Operating Systems**]: Security and Protection–*Authentication*

General Terms: Security, Algorithms, Performance

Keywords: Password; Passphrase; Cracking; Grammar; Policy

1. INTRODUCTION

Text-based password authentication is a widely deployed user authentication mechanism. Use of text-based passwords involves a trade-off between usability and security. System assigned passwords and user-selected passwords subject to complex constraints (e.g. including mixed-case, symbols and digits) are harder to guess, but less usable[22]. Conversely, simple, memorable user-selected passwords offer poor resilience to guessing.

To obtain a good compromise between security and usability, researchers and organizations are recommending the use of longer user-selected passwords with simpler composition requirements, e.g. minimum 16 character passwords[21] and sentence-like or phrase-like passphrases[6, 11, 3, 12, 27]. In the minimum 16 character password policy, the only restriction is that passwords cannot contain spaces. An example of a passphrase policy is "choose a password that contains at least 15 characters and at least four words with spaces between the words"[6]. The increase in the length of the password supposedly makes the password difficult to guess.

To memorize longer passwords users may rely on memory aids such as rules of English language grammar. Users may use memory aids voluntarily or due to policy recommendations. Our analysis of a set of 1434 passwords of 16 characters or more from a published study[21] shows that more than 18% of users voluntarily chose passwords that contain grammatical structures. Each of these passwords contains a sequence of two or more dictionary words. An example is "abiggerbetterpassword" that contains the grammatical structure "Determiner Adjective Adjective Noun". Table 1 provides more examples. In addition to grammatical structures we also found other types of structures such as postal addresses, email addresses and URLs. Given the evidence of use of structural patterns in long passwords, we are motivated to investigate its effect on password security. Studies on password security so far have focused only on structural dependencies at the character level[35, 33, 21].

Main Contributions: (1) We propose an analytical framework to estimate the decrease in search space due to the presence of grammatical structures in long passwords. We use a simple natural language processing technique, Parts-of-Speech (POS) tagging, to model the grammatical structures. (2) We show that the strength of a long password does not necessarily increase with the number of characters or words in the password. Due to the presence of structures, two passwords of similar length may differ in strength by orders of magnitude. (3) We develop a novel cracking algorithm to increase the cracking efficiency of long passwords. Our cracking algorithm automatically combines multiple words using our POS tagging framework to generate password guesses. (4) We show that it is necessary to analyze the distribution of grammatical structures underlying password values in addition to the distribution of password values themselves to quantify the decrease in guessing effort.

2. BACKGROUND AND RELATED WORK

Parts-of-Speech Tagging: It is the process of assigning a part of speech to each word in a sentence[25]. In English language, parts of speech are noun, verb, adjective etc. For example, the parts of speech for a sentence

CODASPY'13, February 18–20, 2013, San Antonio, Texas, USA.

Table 1: Examples of phrases in long password dataset

Category	Password Example	Phrase	Total
Simple	abiggerbetterpassword	a bigger better password	178
Substitution	thereisnomored0ts	there is no more dots	20
Extra Symbol	longestpasswordever8	longest password ever	70
Total out of 1434			268

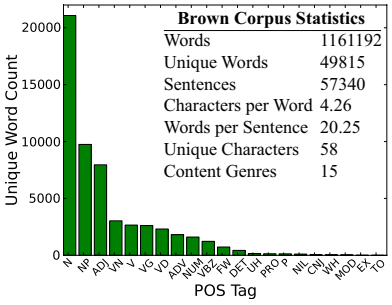

Figure 1: Brown Corpus statistics and count of unique words (*Unique Word Count*) for top 21 Parts-of-Speech tags (*POS Tag*) in the Brown Corpus. *N, NP, ADJ,...* correspond to *Noun, Noun Proper, Adjective,...* The uneven distribution of word counts among POS tags has important implications on password search space and guessing effort.

"She runs fast" are "Pronoun Verb Adverb". Given a sequence of words ($\text{word}_1\text{word}_2 \ldots \text{word}_n$), a POS tagger such as CLAWS[19] can output a sequence of tags, one tag per word ($(\text{word}_1, \text{tag}_1)$ $(\text{word}_2, \text{tag}_2)$ $\ldots (\text{word}_n, \text{tag}_n)$).

Natural Language Corpora: The field of natural language processing commonly uses collection of real data samples or corpus to train and test tools[25]. Two examples are the Google Web Corpus[17] and the Brown Corpus[23]. The Google Web Corpus is a corpus of 1 trillion word tokens of English text collected from web pages and it contains 1 to 5 word n-grams and their frequency counts. The Brown Corpus is a corpus of printed English language of more than 1 million words. It contains articles from 15 genres such as fiction, government, news, and user reviews. Because of the presence of multiple genres, the Brown Corpus is considered a balanced corpus that well represents the entire printed English language. Sentences in the Brown Corpus are POS tagged. The Simple Brown POS tag set consists of 30 tag types. Fig. 1 contains the statistics of the Brown Corpus and the unique word counts for popular tag types.

Password Security: Current password security primarily focuses on the relationships at character level. In[33] the authors estimate the password search space using Shannon entropy[32] at the character level. Password crackers that enumerate password search space use zeroth or higher order Markov models trained on character probability distribution[7, 26]. In[26] authors assume that for memorability, user models a password as a sequence of characters whose distribution is similar to the distribution of characters in her native language. In[16] authors studied the linguistic properties of Amazon Payphrase[1] dataset where majority of the Payphrases are a sequence of two words. Authors investigate whether users choose their Payphrases as a sequence of words which occurs as-is in an existing natural language corpus. Further, they conjecture about guessing effort of passphrases if the distribution of passphrases are identical to the distribution of phrases in a natural language corpus such as Google Web Corpus. In this work, we assume that users model their password as a sequence of words following the rules of grammar such as "Determiner Adjective Noun". The sequence need not occur as-is in a natural language corpus. An example of such password is "the communist fairy" that occurs in the long-password dataset presented in Table 1 but not in the Google Web Corpus. By making a relaxed assumption we model a more powerful adversary who can attack defenses such as use of nonsensical phrases[27].

3. SEARCH SPACE ANALYSIS

The password search space is the set of all possible unique password values. In this section we investigate how the presence of grammatical structures modifies the password search space. One can consider a password value as a sequence of characters, a sequence of words, or a sequence of words generated using the rules of grammar. We propose an analytical framework to estimate the size of the password search space under each of the three assumptions. By comparing the three estimated sizes we can understand the level of reduction in the size of the password search space when grammar structures are present. Via numerical evaluation, we show that the reduction in search space could be 50% or more.

3.1 Computing Search Space Size

Consider a password that contains up to n words. If the words are from a dictionary D={the, run, king, handsome,...} that contains $numw$ unique words, the size of the password search space of all possible word sequences is

$$\mathcal{G}(\text{word}) = \sum_{i=1}^{n} numw^i \tag{1}$$

Consider a word as any sequence of characters, for example "llmmnn", and not only the elements in a standard dictionary. Now, the password search space is bigger than $\mathcal{G}(\text{word})$. Let $numc$ be the number of unique characters possible and $avgc$ be the average number of characters per word. We can approximate the size of the password search space of all possible character sequences as

$$\mathcal{G}(\text{char}) = \sum_{i=1}^{n} numc^{avgc \times i} \tag{2}$$

Let us now consider a password as a sequence of words created using grammatical rules. For example, a user may pick "thehandsomeking" based on the grammatical rule "Determiner Adjective Noun". To estimate the search space under this assumption, we need to define the set of valid grammatical rules. Modeling rules of natural language grammar is a difficult problem[25]. English language parsers and generators use Context Free Grammar (CFG) or more powerful Context Sensitive Grammar (CSG) to approximately model the rules of grammar. A CFG or CSG can recursively generate infinitely long sentences. For long passwords,

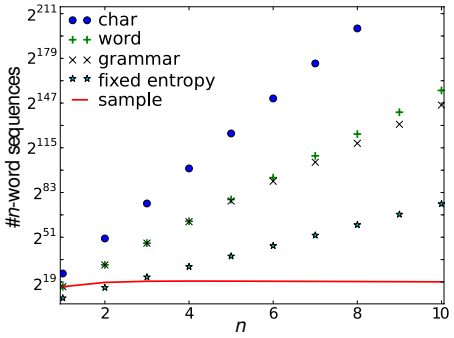

Figure 2: Comparison of the size of password search space treating password as a sequence of characters (*char*), a sequence of words (*word*), and a sequence of words generated using grammatical structures (*grammar*). Numbers are based on the Brown Corpus statistics. n is the number of words in the password. The difference between *word* and *grammar* widens as n increases. We also plot search space estimation using 1.75 bits per character of Shannon entropy (*fixed entropy*) and the actual number of unique n-word sequences in the Brown Corpus (*sample*). We explain their significance in Section 6.3.

it is unlikely that we will need to generate infinitely long sentences. For finite length sentences, a Regular Language is sufficient and it reduces computational complexity from $O(n^3)$ to $O(n)$. Parts-Of-Speech (POS) tagging technique described in Section 2 is equivalent to using a Regular Language, and we use it to model the rules of grammar.

We consider each grammatical rule as a sequence of POS tags. We extract POS tag sequences from a POS-tagged corpus that is representative of a long password dataset. This approach of generating grammatical rules is similar to expanding the CFG rewrite rules up to a finite length sans the complexity of the CFG rewrite rules. We can modify the grammar by including or excluding POS tag sequences.

Consider the set of all POS tags {Noun, Verb, Adjective, Determiner,...}. Each POS tag is associated with a dictionary of words e.g. dictionary of "Noun" is {king, queen,...}. Examples of POS tag sequences include "Determiner Adjective Noun" and "Determiner Determiner Noun". We consider a POS tag sequence as grammatical if it is observed in a corpus. We refer to a grammatical tag sequence as a *tag-rule*. "Determiner Adjective Noun" is a tag-rule because it is present in the Brown Corpus. However, "Determiner Determiner Noun" is not a tag-rule because it is not present in the corpus. The search space of a tag-rule is the set of word sequences it generates. For example, the tag-rule "Determiner Adjective Noun" generates the set of word sequences {"the handsome king", "the beautiful queen",...}. The size of the search space of a tag-rule is equal to the product of the size of dictionaries of individual tags in the sequence. The size of the grammatical password search space, $\mathcal{G}$(grammar), is equal to the sum of the sizes of search space of the tag-rules. For precise equations, refer to [30].

3.2 Numerical Evaluation

We need a corpus to numerically evaluate the password search space model proposed in Section 3.1. We use the Brown Corpus, a balanced corpus that contains representative grammatical structures (tag-rules) for English language. We believe users will model their long passwords using tag-rules similar to the tag-rules in the Brown Corpus; we find that 84% of the long passwords from "Simple" category in Table 1 were generated using tag-rules from the Brown Corpus. Using the Brown Corpus should provide useful insights into the effect of structure on the password search space.

Table 2: *grammar* password search space as a percentage of *word* password search space (from Fig. 2). Note the significant decrease in password search space due to the presence of grammatical structures.

#words	1	2	3	4	5
$\frac{grammar}{word}$ %	100	99.92	96.90	80.66	46.95

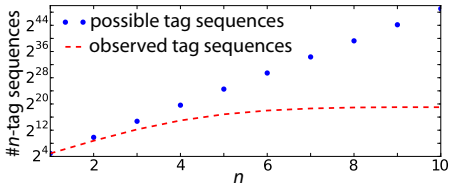

Figure 3: Comparison of the number of tag sequences observed in the Brown Corpus with the number of tag sequences possible with 30 POS tags. The *observed tag sequences*, which are a small fraction of the *possible tag sequences*, generate the *grammar* password search space in Fig. 2.

To evaluate the size of password search space of character sequences, $\mathcal{G}$(char) in (2), and word sequences, $\mathcal{G}$(word) in (1), we use the character and word statistics from Fig. 1. The number of unique characters $numc$=58, number of unique words in the dictionary $numw$=49815, and average number of characters in a word $avgc$=4.26. In Fig. 2 we plot the size of the password search space $char$=$58^{4.26\times i}$ and $word$=49815^i as a function of number of words i in the password.

To evaluate the password search space of word sequences generated by tag-rules, $\mathcal{G}$(grammar), we need a set of POS tags, a dictionary for each POS tag, and the set of tag-rules. We use the Simple Brown Corpus POS tag set with 30 tags. We get the dictionary for each tag from the Brown Corpus (Fig. 1). We extract the tag-rules from the Brown Corpus as explained in[30]. In Fig. 3, we plot the number of observed tag sequences (tag-rules) and possible tag sequences. Number of observed tag sequences is much less than the number of possible tag sequences, and the difference increases with length. Observed tag sequences and possible tag sequences generate the *grammar* and *word* search spaces in Fig. 2.

From Fig. 2 we can compare the password search space sizes of *char*, *word*, and *grammar*. Note that $char \gg word > grammar$. To emphasize the decrease in password search space due to the presence of grammar, we tabulate the ratio of *grammar* to *word* in Table 2. Observe that for a password of length 5 words, the decrease is more than 50%.

4. DISTRIBUTION ANALYSIS

When password values have underlying grammatical structures, it is important to understand the role of these structures in decreasing the guessing effort. Guessing effort can be defined as the number of values an attacker has to enumerate to guess a password. Guessing effort is a function of (a) size of the password search space, which is the set of all

possible unique password values and (b) distribution of password values, which depends on how users choose password values from the password search space. So far, research involving analysis of password distributions[16, 15, 31] has not considered the effect of underlying grammatical structures.

In Section 3 we showed that the grammatical structures reduce the password search space, which implies reduced guessing effort. This is because the maximum number of values an attacker has to enumerate is equal to the size of the search space. In this section we show that the distribution of grammatical structures can also reduce the guessing effort. This reduction is in addition to the reduction due to the distribution of the password values themselves.

4.1 Reduction in Guessing Effort

A uniform distribution maximizes the guessing effort[28]. Conversely, a non-uniform distribution reduces the guessing effort. Usually, user-chosen password distributions are not uniform[15]. Given a set of user chosen passwords such as {mypassword, mypassword, iloveu}, non-uniformity is evident as password values are not unique. In the past, research has associated uniformity solely with uniqueness of password values. For example, [31] ensures that password values do not repeat often and [15] computes the distribution by counting the repetition of password values. However, for passwords generated using grammatical structures, underlying structure may cause non-uniformity even if the password values are unique. For example, the values in the set {"tangy food", "pretty cat", "naughty kid"} are unique, but all values are generated using "Adjective Noun". Hence, uniqueness of password values is a necessary, but not sufficient condition to ensure uniformity.

Grammatical structures, or tag-rules, split the password search space unevenly; the size of the search space of individual tag-rules are different e.g. the size of "Noun Noun" is greater than the size of "Adjective Noun". Recall from Section 3 that the search space of a tag-rule is the number of word sequences it generates. The effort required to guess a password generated by a tag-rule is a function of the size of its search space. Given a set of user chosen passwords, we can analyze how the underlying tag-rules are distributed. If users are using certain rules more often than the others, an attacker can use this information to reduce her guessing effort. For example, if the password set contains only the tag-rule "Adjective Noun" then the attacker need not enumerate other tag-rules. Specifically, if the users are choosing weaker tag-rules more often than the stronger tag-rules, reduction in guessing effort can be higher. In Fig. 4 we group the tag-rules from the Brown Corpus by their search space size expressed in bits or $\log_2$. Observe that some tag-rules have small search spaces e.g. for tag-rules of length 3 (*3-gram*), 8.9% of the rules have 10-19 bits of strength. Further, our analysis of unique 2-word and 3-word sequences from the Brown Corpus shows that weaker tag-rules occur more often than stronger tag-rules. More details about this analysis is available in[30].

To ensure uniform distribution over a set of password values (a) password values have to be unique, and (b) each tag-rule should have *proportional representation*. Intuitively, proportional representation implies that a tag-rule with a larger search space should generate more password values in the set. Unless these conditions are satisfied, the distribution is not uniform, and the guessing effort decreases.

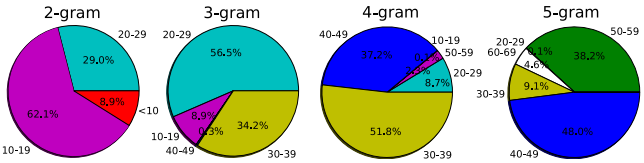

Figure 4: Tag-rules of length 2 to 5 grouped by the size of their search space (in bits or $\log_2$). Numbers outside indicate the range of bits, and numbers inside indicate the percentage of tag-rules with those many bits. Tag-rules divide the password search space unevenly, and many tag-rules have low strength. For example, 8.9% of *3-gram* tag-rules have 10-19 bits of strength.

Table 3: Data set ExSet

Password	Phrase
Ihave3cats	I have 3 cats
Ihave4dogs	I have 4 dogs
Ihave5fish	I have 5 fish
Ihad1cat.	I had 1 cat.
Ihad1goat	I had 1 goat

We prove this in[30] by posing the problem of computing an attacker's guessing effort as an optimization problem.

5. PASSWORD CRACKERS

We investigate whether state-of-the-art cracking tools such as John the Ripper (JTR)[7], Hashcat[5], and Weir Algorithm[35] can crack long passwords efficiently. This is important because crackers are used in auditing user passwords and estimating the strength of password policies[21]. We discuss the shortcomings of these crackers in cracking long passwords. We show ways to improve cracking efficiency for both long passwords in general and long passwords generated using grammatical structures. We propose a novel algorithm to improve cracking efficiency. Our cracking algorithm uses the POS tag framework introduced in Section 3.

5.1 Shortcomings of Current Crackers

A password cracker tries to recover a plain text value of a password hash value. The cracker generates candidate password guesses, hashes them, and compares them with the available hashes until a match is found. A heuristic dictionary-based cracker[7, 5, 35] uses a dictionary of values to generate candidate passwords. A dictionary may contain leaked passwords[21], words from many languages[10], common quotes, music lyrics, movie titles[24] etc. Cracker may use the dictionary values as-is or transform them by applying mangling rules. An example of a mangling rule is "capitalize first alphabet", which transforms a dictionary value "password" to "Password". Alternatively, an intelligent brute-force cracker, eventually, enumerates the entire password search space[7]. Below we explain the main shortcomings of current crackers in cracking long passwords using the example data set, ExSet in Table 3 and a dictionary, ExD={I, have, had, cats, dogs, fish, cat, goat}.

JTR in Wordlist mode and Hashcat are dictionary-based crackers. Their mangling rules can combine a single dictionary value in different ways, for example "catscats" or "catsstac" from "cats". They can append, prefix or insert specific strings to a dictionary value, and delete parts of the dictionary value. However, JTR can not combine multiple

values from the dictionary to form longer passwords. To crack passwords such as "Ihave3cats" from ExSet using dictionary ExD, user has to (1) write multiple mangling rules for example "prefix I", "append had" or "prefix Ihave" or (2) explicitly add the value "Ihave3cats" to the dictionary ExD. To add longer values to the dictionary, user has to generate the values himself or collect them from existing sources such as books, Web etc. Hashcat can combine up to two values from the input dictionary, but for more values it has issues similar to JTR.

Weir Algorithm is another dictionary based technique. It improves cracking efficiency by improving the order in which mangling rules are applied to the values in the dictionary. Weir Algorithm generates a set of base structures from a training corpus. A base structure in Weir Algorithm is a sequence of "L", "D", and "S" that denote "Letter", "Digit", and "Special Symbol". Each base structure is assigned a probability. Weir Algorithm learns the digits and special symbols to insert into "D" and "S" from the training corpus. For letter sequences in a base structure, Weir Algorithm tries to fit values from the dictionary whose length exactly matches the length of the letter sequence. For example, for "LLLL" in a base structure "LLLLDLLLS", it tries to fit values {have, cats, dogs, fish, goat} from dictionary ExD. It cannot combine shorter values such as "I" and "had" to form a longer value "Ihad" that fits the sequence "LLLL". This shortcoming is similar to that of JTR and Hashcat. Although trained on passwords in dataset ExSet, Weir Algorithm cannot crack any password from dataset ExSet using the dictionary ExD; it cannot create "Ihave" and "Ihad" from "I", "have" and "had".

To force Weir Algorithm to generate longer values, we can train it with passwords containing words separated by a single space. Weir Algorithm treats space as a special symbol. We have to remove spaces from the generated password guesses. If we train Weir Algorithm on phrases listed in the dataset ExSet, it generates a base structure "LSLLLSDSLLLL". Now, it can crack passwords such as "Ihad3cats" using dictionary ExD. However, this approach generates guesses such as {Ihad1had, Ifish5have, Icats3fish ...} that may be in the search space of all word sequences, $\mathcal{G}$(word), but not in the search space of word sequences generated using grammatical structures, $\mathcal{G}$(grammar). From Section 3, $\mathcal{G}$(word) can exceed $\mathcal{G}$(grammar) by more than 50% for 5-word length.

JTR Incremental mode, an intelligent brute-force cracker, uses a Markov model trained on 3-gram letter frequency distribution to generate password guesses. Based on our experiments, letter frequencies are effective for conventional short passwords consisting of sequence of characters but not for longer passwords with multiple words.

5.2 Evaluation on Long Password Dataset

We evaluate the cracking efficiency of JTR, Hashcat and Weir Algorithm with experiments on a published long password dataset[21]. We introduce this dataset in Section 1. It contains 1434 passwords of minimum length 16 characters, and was collected as part of a field study. Subjects could create their passwords using any character except the space character. To test the cracking efficiency on long passwords, we use the complete long password dataset, henceforth referred to as P16. To test the cracking efficiency on long passwords with underlying grammatical structures, we use a subset from P16 containing simple phrases, henceforth, P16S. We are not aware of datasets that exclusively contain user-selected long passwords with underlying grammatical structures. P16S contains 144 passwords. To create P16S, we manually examine each password in P16 using tools such as the Microsoft Word Breaker[34], and identify passwords with multiple words. We initially include all passwords with two or more words (e.g. "compromisedemail", "thereisnomored0ts") except those that contain repetitions of a single word (e.g. "elephantelephant"). We further categorize the passwords into three groups: simple phrase, phrase with symbol substitution and phrase with extra symbols. Table 1 shows example for each category. For our experiments, we use passwords with simple phrases containing 2 to 5 words. One of our experiments uses the Google Web Corpus that contains n-grams of length up to 5. Hence, testing on passwords with more than 5 words will not permit fair comparison. Table 4 describes our dictionaries;"L" contains publicly available datasets, "GW" has data from the Google Web Corpus and "B" has data from the Brown Corpus.

Table 4: Dictionaries used for evaluating crackers. Dictionary "L" is a large dictionary combining the datasets Myspace, Rockyou, Brown, 1gram, Dic-0294, Basic Full, Basic Alphabetic, Free Full, Free Alphabetic, Paid, Alphabetic, Paid Lowercase. In column *Name* "-x" indicates minimum length of the words in the dictionary.

Name	#Words	Description
L-8	35267653	Minimum length 8 values from L
LASCII	39251222	All length ASCII values from L
GW25-8	3625636435	Google Web Corpus 2-5 grams
B210	5942441	Brown 2-10 gram word sequences

Table 5: Performance of crackers on long passwords (P16) and long passwords with underlying grammatical structures (P16S). *Experiment* is the name of the experiment. *NM* indicates no mangling. *%P16 Cracked* is % of passwords cracked out of 1434 passwords in P16. *%P16S Cracked* is % of passwords cracked out of 144 passwords in P16S. *Guesses* is the total number of password guesses. *SC?* indicates if experimental session completed after *Guesses*.

Experiment	%Cracked		Total	SC?
	P16	P16S	Guesses	
JTR L-8	13.6	6.9	2.31E10	Yes
JTR L-8 NM	8.5	4.8	3.42E7	Yes
JTR GW25-8	20.5	34.7	2.48E12	Yes
JTR GW25-8 NM	11.08	27.7	3.4E9	Yes
Weir LASCII	12	4.8	1.07E12	No
Weir Space LASCII	7.6	3.4	1.26E12	No
JTR Incremental	0	0	2.48E12	No

We tabulate our experimental results in Table 5. We allow all experiments to make up to 2.5E12 guesses; we indicate if an experiment terminated earlier in the "SC?" column. We terminated two experiments before 2.5E12 guesses due to their excessive memory consumption. For brevity, we omit the less significant experimental results from Table 5. In our experiments, we try to overcome the main shortcoming of current crackers: They do not generate longer values automatically; user has to generate and add longer values to the dictionary. Specifically, we do the following:

Use a better dictionary of long values: given the evidence that, for longer passwords, users choose word sequences, we

hypothesize that a corpus of word sequences is a better dictionary. Experiments on long password datasets until now have not tested this hypothesis. We test the hypothesis using word-grams in the Google Web Corpus and the Brown Corpus, but other corpora may also be explored. Using the Google Web Corpus as dictionary, we crack 20.5% of long passwords from dataset P16 ("JTR GW25-8"). For the same number of guesses, published experiments, using publicly available datasets similar to dictionary "L", crack 6% from P16[21]. Using the Brown Corpus, we crack only 0.3% from P16 ("JTR B210").

Use workarounds to generate longer values automatically: we use the workaround for Weir Algorithm explained in Section 5.1. From experiment "Weir Space LASCII", we find that this approach generates large number of guesses and fails to improve cracking efficiency of long passwords.

JTR L-8, JTR L-8 NM, JTR GW25-8, JTR GW25-8 NM: we run JTR in word mode using dictionaries L-8 and GW25-8. We run JTR with and without mangling rules. Dictionary values have minimum 8 characters as mangling rules can concatenate them to form 16-character length guesses. JTR GW25-8 outperforms other experiments. Using 2.48E12 guesses it cracks 20.5% passwords from P16 and 34.7% passwords from P16S. Even with lower number of guesses it outperforms other experiments.

Weir LASCII: we train the Weir Algorithm using minimum 16-character passwords from the Myspace and Rockyou, and run it using the dictionary LASCII. We terminated this experiment before 2.5E12 guesses as the memory consumption became unwieldy. Weir LASCII does not match the performance of JTR GW25-8.

Weir Space LASCII: we train Weir Algorithm on sequences of 1 to 10 words from the Brown Corpus and run it using the dictionary LASCII. The words were separated with a single space. We strip spaces from the generated guesses and check if they crack any passwords. Weir Space LASCII does not match the performance of JTR GW25-8.

JTR Incremental: we train JTR Incremental mode on minimum 16-character passwords from Myspace and Rockyou datasets. We configure it to generate passwords between 16 and 23 characters in length inclusive. JTR Incremental mode experiment fails to crack any passwords.

Experimental results show that using a good dictionary of long password values can improve cracking efficiency. However, relying only on existing sources to build dictionaries may not be ideal. Existing sources contain values that people use often and by relying on them we may fail to crack passwords that contain uncommon and nonsensical phrases e.g. "the communist fairy" in P16S. Building a Markov model based on word gram frequencies as opposed to letter gram frequencies may be useful. However, training these word gram models on existing sources can run into similar issues. We explore a novel technique to automatically generate longer password values.

5.3 Grammar Aware Cracking

A cracker should ideally emulate user behavior to generate password candidates; if passwords contain grammatical structures, crackers should use grammatical structures. We develop proof-of-concept of such a cracker using the POS tag framework in Section 3. It automatically combines words into longer password values using sequences of POS tags. Main challenges in our approach are (1) identify a set of POS tag sequences that are grammatical (tag-rules). It is possible to identify tag-rules from existing corpus e.g. the Brown Corpus or long password datasets and (2) build a dictionary for individual POS tags. The level of difficulty in building tag dictionaries depends on the type of POS tag. Closed tags such as "Determiner" and "Conjunction" (e.g. *the*, *and*) contain small number of values that do not change much with time. Open tags such as "Noun" and "Noun Proper" have large dictionaries and also grow with time.

Table 6: Performance of grammar-aware cracker on long passwords with underlying grammatical structures (P16S). *%Cracked BWeb* is % of P16S cracked using dictionary BWeb. *%Cracked with BWeb90* is % of P16S cracked with dictionary BWeb90. *%Exclusive* is % of P16S cracked by grammar-aware cracker, but not by other crackers in Table 5.

Guesses	%Cracked with BWeb	%Cracked with BWeb90	%Exclusive
5E10	9.7	18.7	4.8
1E12	14.5	25	9
2.5E12	15.2	27	10.4
10E12	20.1	29.1	11.8
40E12	25.6	35.4	13.8

Our main goal is to evaluate the value of a grammar aware cracking approach. Will a grammar-aware cracker allow an attacker to crack passwords that can not otherwise be cracked? We assume that an attacker has access to a good set of tag-rules. We believe that assuming otherwise leads to a security-through-obscurity model. As use of long passwords increases, it is likely that long password data sets will become public, and attackers can use tag-rules from these datasets. To simulate a scenario that provides maximum advantage to an attacker, we extract tag-rules from P16S long password dataset used in Section 5.2. We tag the passwords in P16S using the CLAWS[19] POS tagger.

First, we build a dictionary for each POS tag using the Brown Corpus and a small web text corpus[9]. We include the web text corpus because the Brown Corpus, compiled in year 1961, does not contain Internet related words. We use all words, including repetitions, from the Brown Corpus and all words that occurred at least 10 times in the web text corpus (373 words). Words in the Brown Corpus are tagged. We tag the words in the web text corpus using CLAWS. If a word has a noun tag, we assign it to the noun dictionary and so on. We refer to this first set of tag dictionaries as BWeb. Next, we build an alternate set of tag dictionaries from BWeb. To reduce the size of a tag dictionary, we compute the cumulative probability distribution of word frequencies within the dictionary and discard words that do not meet the 0.9 cumulative probability cutoff. We apply this reduction process to all POS tags except "Noun", "Proper Noun", "Adjective", and "Cardinal Number". We call this alternate set BWeb90. Intuition for BWeb90 is that users use few popular words for closed tags, but not for open tags.

The inputs to the grammar aware cracker are a set of tag-rules, a set of tag dictionaries, and the maximum number of guesses it can make. The cracker computes the size of each tag-rule (Section 3), sorts the tag-rules by their size, and selects the subset of smallest tag-rules whose sizes add up to the number of guesses. The cracker generates word sequences from identified tag-rules using the tag dictionaries, and outputs them as password candidates.

Table 7: Comparison of passphrase strength; strength is not a direct function of length. *Tag-Rule* lists the tag-rule that can generate the passphrase. *Guesses* is based on the size of the search space of *Tag-Rule* and effort required to mangle the phrase. *Time* estimates the time required to guess a given passphrase.

Passphrase	Tag-Rule	Guesses	Time
Th3r3 can only b3 #1!	EX MOD V DET PRO	1.3E12	22 min
Hammered asinine requirements.	VD ADJ N	12.6E12	3.5 h
Superman is $uper str0ng!	NP V ADJ ADV	12.3E15	142 d
My passw0rd is $uper str0ng!	PRO NP V ADJ ADV	1.7E18	56 yr

Table 6 details the performance of our grammar-aware cracker. The columns "%Cracked with BWeb" and "%Cracked with BWeb90" list percentage of dataset P16S cracked using dictionaries BWeb and BWeb90 respectively. The "Exclusive" column lists the percentage of P16S that was exclusively cracked by grammar-aware cracker, but not by other crackers in Table 5. For 5E10 guesses, grammar-aware cracker cracked 18.7% of passwords, which is better than "Weir Algorithm" (4.8%) and "JTR Incremental" (0%). Although not listed in Table 6, it performs better than "JTR L-8" for 2.3E10 guesses. At 2.5E12 guesses it cracks 27% passwords, but that is not better than the performance of "JTR GW25-8" (34.7%). However, grammar-aware cracker cracks 10% new passwords from P16S dataset. This 10% was not cracked by "JTR GW25-8" or other experiments. Grammar-aware cracker consumes <10MB of storage versus >50GB for "JTR GW25-8". It is scalable as number of guesses increases, easier for operations such as network transfer and dictionary sorting, and provides flexibility in targeting different user groups e.g. use different "Proper Noun" dictionary for users from North America and Asia. From initial results, we believe that grammar-aware approach has potential to improve cracking efficiency of long passwords.

6. POLICY IMPLICATIONS

In this section, we highlight the need for policy makers to understand the impact of grammatical structures on security of long passwords. We examine some of the implicit assumptions within passphrase policies in light of the results on search space and guessing from Sections 3, 4 and 5. Many policies consider a passphrase as a long sequence of characters or words[8, 2, 3, 6] and estimate security metrics such as size of search space and guessing effort accordingly[6, 14]. However, when users choose sentence-like or phrase-like passphrases, due to grammatical structures the search space and guessing effort will decrease. Further, because of structure, the strength of the passphrase does not increase uniformly with the length i.e. a longer passphrase is not necessarily stronger than a shorter passphrase.

6.1 Passphrase Strength and Length Relation

Consider the passphrase examples in Table 7. The examples "Th3r3 can only b3 #1!", "Superman is $uper str0ng!" and "My passw0rd is $uper str0ng!" are from technology and academic websites[4, 12, 3]. The example "Hammered asinine requirements." is a synthetic example based on a recommendation to use nonsensical phrase[27]. The tag-rule column lists one of the grammatical structures that can generate the passphrase. The number of guesses is an estimate of the total number of passphrases the tag-rule can generate. It considers the size of the tag-rule search space and a fixed number of additional guesses required to mangle the phrase. The time column estimates the time required to guess passphrases generated by the tag-rule using a guessing rate of 1 billion guesses per second. This rate is realistic considering that current state-of-the-art GPU accelerated machines achieve up to 33 billion comparisons per second and can be built with less than USD 3000[13]. From the guessing effort and time estimates, we see that passphrase strength is not a direct function of the number of words or characters in the passphrase. The passphrase "Th3r3 can only b3 #1!" has more words than "Hammered asinine requirements.", but is one order of magnitude weaker. Similarly, "Hammered asinine requirements." has more characters than "Superman is $uper str0ng!", but is one order of magnitude weaker. Underlying structures and not just the number of characters or words determine the strength of a passphrase. Passphrase policies that do not consider this[3] may unwittingly allow passphrases such as "Th3r3 can only be #1!" and "My passw0rd is $uper str0ng!" that differ in strength by three orders of magnitude.

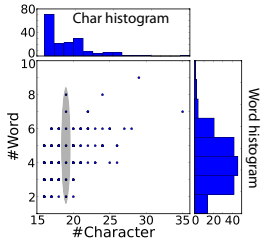

Figure 5: User behavior on long passwords containing simple phrases. Total number of characters in the passwords tends to remain the same as the number of words increases (shaded oval). Users seem to meet the policy requirement of minimum 16 characters with fewer long words ("compromisedemail") or more short words ("thosedamnhackers").

6.2 User Behavior and Passphrase Policy

While studying the long password data set[21], we observed that users tend to choose fewer long words or more short words to generate a password that meets the policy requirement of minimum 16 characters. For example, the passwords "compromisedemail" and "thosedamnhackers" contain 16 characters, but two and three words respectively. Fig. 5 plots the word and character statistics for long passwords in simple category from Table 1. All passwords in the shaded oval have 19 characters (x-axis) and between two to eight words (y-axis). Observe similar behavior for passwords with 16, 17, etc. characters. We found some explanation for this behavior in the field of cognitive psychology. Jahnke and Nowaczyk say regarding experiments on word-length and short-term memory[20, Chap. 4], "Strings of short words (e.g. *cup*, *war*) given in tests of immediate memory are much more likely to be recalled correctly than are equally

long strings of equally familiar long words (e.g. *opportunity*, *university*). Stated alternatively, subjects can remember for immediate recall about as many words as they can say in 2 s, and obviously they can say more short than long words in that time." From Section 6.1, we know that two passwords of equal length (same number of characters), but different number of words may vary in strength by orders of magnitude. Hence, passphrase policies such as "choose a password that contains at least 15 characters and at least 4 words with spaces between the words"[6] may allow weaker passphrases unless they consider user behavior and effect of structure.

6.3 Passphrase Entropy

To determine size of passphrase search space, subsequently passphrase security, some evaluations[29] use fixed entropy estimate of the English language[32]. We explain below, why this approach may be incorrect. Informally, entropy measures how much the values emitted by a source can be compressed. When values repeat more often, compression is higher and entropy is lower. Since search space of a source is the set of unique value it emits, we can use entropy to estimate size of the search space. However, an accurate estimate of entropy is necessary to do so. It is incorrect to use estimation derived from one source (e.g. English language) for another (e.g. long passwords) as they can have different distribution of values. To illustrate this point, in Fig. 2, we plot the search space estimate using the equation $2^{1.75\times4.26\times i}$ where i is the number of words in the password. We use fixed entropy estimate of 1.75 bits per character for printed English from[18], which derived the estimate using a 3-word gram language model trained on large amounts of printed English sources and tested on the Brown Corpus. We use 4.26 average word length statistic from the Brown Corpus (Fig. 1). We observe from Fig. 2 that the estimation for 3-word phrases closely matches the true number of unique 3-word phrases in the Brown Corpus. For other lengths, there is varying degrees of inaccuracy.

7. CONCLUSIONS

Long passwords is a promising user authentication mechanism. However, to achieve the level of security and usability envisioned with long passwords, we have to understand the effect of structures present in them. Further, we have to make policies and enforcement tools cognizant of the effect of structures. As a first step, we developed some techniques to achieve these goals. We studied grammatical structures, but other types of structures such as postal addresses, email addresses and URLs present within long passwords may have similar impact on security. More research is necessary to fully understand the effect of structures on long passwords.

8. REFERENCES

[1] Amazon payphrase. www.amazon.com/payphrase, 2011.

[2] Bitcoin passphrase policy. https://en.bitcoin.it/wiki/Securing_your_wallet#Password_Strength, 2012.

[3] Carnegie Mellon University passphrase policy and FAQ. http://www.cmu.edu/iso/governance/guidelines/password-management.html;http://www.cmu.edu/computing/doc/accounts/passwords/faq.html#8, 2012.

[4] Cheap GPUs rendering strong passwords useless. http://it.slashdot.org/story/11/06/05/2028256/cheap-gpus-rendering-strong-passwords-useless, 2012.

[5] Hashcat advanced password recovery. http://hashcat.net/oclhashcat-plus/, 2012.

[6] Indiana University passphrase policy and strength estimation. http://kb.iu.edu/data/acpu.html#atiu;http://protect.iu.edu/cybersecurity/safeonline/passphrases, 2012.

[7] John The Ripper password cracker. http://www.openwall.com/john/, 2012.

[8] Massachusetts Institute of Technology passphrase policy. http://ist.mit.edu/security/passwords, 2012.

[9] NLTK Web Text Corpus. http://nltk.googlecode.com/svn/trunk/nltk_data/index.xml, 2012.

[10] Openwall wordlists collection. http://www.openwall.com/wordlists/, 2012.

[11] University of Maryland passphrase policy. http://www.security.umd.edu/protection/passwords.html, 2012.

[12] University of Minnesota passphrase policy. http://www.oit.umn.edu/security/topics/choose-password/index.htm, 2012.

[13] Whitepixel GPU Hash Auditing. http://whitepixel.zorinaq.com/, 2012.

[14] T. Baekdal. Passphrase usability and strength estimation. http://www.baekdal.com/insights/password-security-usability, 2012.

[15] J. Bonneau. The science of guessing: analyzing an anonymized corpus of 70 million passwords. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 538–552, 2012.

[16] J. Bonneau and E. Shutova. Linguistic properties of multi-word passphrases. In *Workshop on Usable Security*, pages 1–13, 2012.

[17] T. Brants and A. Franz. *Web 1T 5-gram Version 1. Linguistic Data Consortium*. Philadelphia, PA, 2006.

[18] P. F. Brown, V. J. D. Pietra, R. L. Mercer, S. A. D. Pietra, and J. C. Lai. An estimate of an upper bound for the entropy of English. *Comput. Linguist.*, 18(1):31–40, 1992.

[19] CLAWS. Part-Of-Speech tagger for English. http://ucrel.lancs.ac.uk/claws/, 2012.

[20] J. C. Jahnke and R. H. Nowaczyk. *Cognition*. Prentice-Hall, Inc., Upper Saddle River, New Jersey, USA, 1998.

[21] P. G. Kelley *et al*. Guess again (and again and again): Measuring password strength by simulating password-cracking algorithms. In *Proceedings of the IEEE Symposium on Security and Privacy*, 2012.

[22] S. Komanduri *et al*. Of passwords and people: Measuring the effect of password-composition policies. In *Proceedings of the Annual Conference on Human factors in Computing Systems*, pages 2595–2604. ACM, 2011.

[23] H. Kucera and W. N. Francis. *Computational analysis of present-day American English*. Brown University Press, Providence, RI, 1967.

[24] C. Kuo, S. Romanosky, and L. F. Cranor. Human selection of mnemonic phrase-based passwords. In *Proceedings of the Symposium on Usable Privacy and Security*, pages 67–78. ACM, 2006.

[25] C. D. Manning and H. Schütze. *Foundations of Statistical Natural Language Processing*. MIT Press, Cambridge, MA, USA, 1999.

[26] A. Narayanan and V. Shmatikov. Fast dictionary attacks on passwords using time-space tradeoff. In *Proceedings of the ACM Conference on Computer and Communications Security*, pages 364–372, 2005.

[27] PGP. FAQ: How do I choose a good password or phrase? http://www.unix-ag.uni-kl.de/~conrad/krypto/passphrase-faq.html, 2012.

[28] J. O. Pliam. On the incomparability of entropy and marginal guesswork in brute-force attacks. In *Proceedings of the International Conference on Progress in Cryptology*, pages 67–79. Springer-Verlag, 2000.

[29] S. N. Porter. A password extension for improved human factors. *Computers & Security*, 1(1):54–56, 1982.

[30] A. Rao, B. Jha, and G. Kini. Effect of Grammar on Security of Long Passwords. Technical Report CMU-ISR-12-113, ISR, Carnegie Mellon University, 2012.

[31] S. Schechter, C. Herley, and M. Mitzenmacher. Popularity is everything: A new approach to protecting passwords from statistical-guessing attacks. In *Proceedings of the USENIX conference on Hot topics in Security*, pages 1–8, 2010.

[32] C. E. Shannon. Prediction and entropy of printed English. *Bell Systems Technical Journal*, 30:50–64, 1951.

[33] R. Shay *et al*. Encountering stronger password requirements: User attitudes and behaviors. In *Proceedings of the Symposium on Usable Privacy and Security*, pages 1–20. ACM, 2010.

[34] K. Wang, C. Thrasher, and B.-J. P. Hsu. Web scale NLP: A case study on url word breaking. In *Proceedings of the International Conference on World Wide Web*, pages 357–366. ACM, 2011.

[35] M. Weir, S. Aggarwal, B. de Medeiros, and B. Glodek. Password cracking using probabilistic context-free grammars. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 391–405, 2009.