Effect of Grammar on Security of Long Passwords[*]

Ashwini Rao **Birendra Jha**[†] **Gananand Kini**

May 2012
CMU-ISR-12-113

Institute for Software Research
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

The authors are grateful to Dr. Lorrie Cranor and Dr. Lujo Bauer for providing the long password dataset. The authors thank Edward Schwartz and Michelle Mazurek for reviewing the initial manuscript. This research was supported in part by Nicolas Minnici Graduate Fellowship and Prabhu and Poonam Goel Graduate Fellowship.

[*]Extended version of [33]
[†]Massachusetts Institute of Technology, USA

Keywords: Password, Passphrase, Cracking, Grammar, Policy

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 as high as 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.

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 [25]. 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. Examples include minimum 16 character passwords [24] and sentence-like or phrase-like passphrases [7, 14, 3, 15, 30]. 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" [7]. 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 [24] 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 the effect of structural patterns on password security. Studies on password security so far have focused only on structural dependencies at the character level [38, 36, 24].

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

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

Table 2: Brown Corpus statistics

Words	1161192
Unique Words	49815
Sentences	57340
Characters per Word	4.26
Words per Sentence	20.25
Unique Characters	58
Content Genres	15

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.

Organization: We discuss the effect of grammatical structures on password search space in Section 3 and on password guessing effort in Section 4. In Section 5, we examine the shortcomings of state-of-the-art password crackers in cracking long passwords and describe the advantages of our new grammar-aware cracking algorithm. We experimentally evaluate the efficiency of existing crackers and our grammar-aware cracker using long password datasets. In Section 6, we discuss the impact of our results on current password policies. We conclude in Section 7.

2 Background and Related Work

Parts-of-Speech Tagging: Part-of-Speech (POS) tagging is the process of assigning a part of speech to each word in a sentence [28]. In English language, parts of speech are noun, verb, adjective etc. For example, the parts of speech for a sentence "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 [22] 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 [28]. Two examples are the Google Web Corpus [20] and the Brown Corpus [26]. 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. Table 2 contains the statistics of the Brown Corpus. Sentences in the Brown Corpus are POS tagged. The Brown simple POS tag set consists of 30 tag types. Fig. 1 shows the unique word counts for popular tag types.

Password Security: Current password security primarily focuses on the relationships at character level. In [36] the authors estimate the password search space using Shannon entropy [35] at the character level. Password crackers that enumerate password search space use zeroth or higher order Markov models trained on character probability distribution [9, 29]. In [29] authors assume

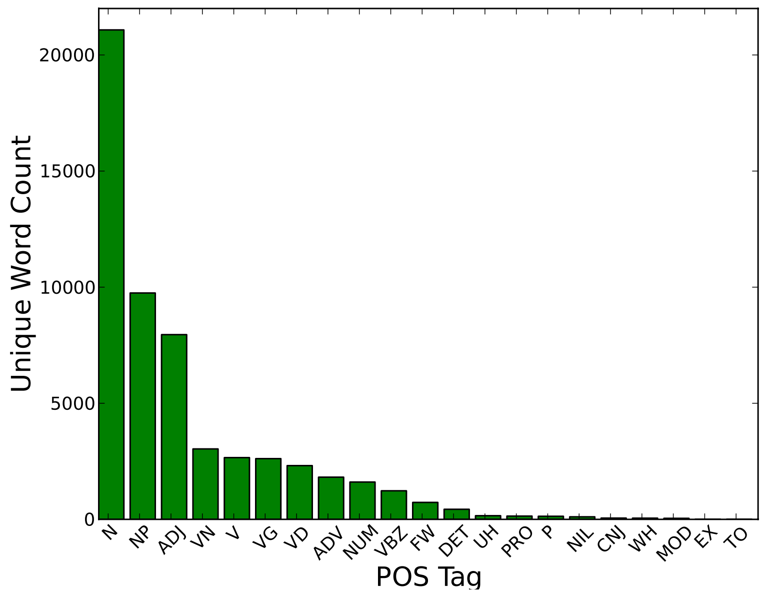

Figure 1: Count of unique words (*Unique Word Count*) for top 21 Parts-of-Speech tags (*POS Tag*) in the Brown Corpus. There are 30 tags in the simple Brown POS tag set. *N, NP, ADJ,* . . . correspond to the *Noun, Noun Proper, Adjective,* . . . . Note that the word counts are unevenly distributed among different tag types. This has important implications on password search space and guessing effort.

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 [19] 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 [30].

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 highlight that the reduction in search space could be as high as 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, of, run, king, queen, handsome $, \dots\}$ that contains $numw$ unique words, the size of the password search space of all possible word sequences is

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

If we consider a word to be any sequence of characters, not just an element from dictionary D, then the password search space is bigger. For example, "llmmnn" that is not present in a standard English dictionary. 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 [28]. Tools such as English language parsers and generators approximately model the rules of grammar using Context Free Grammar (CFG) or, more powerful, Context Sensitive Grammar (CSG). A CFG is required to recursively generate infinitely long sentences. However, for long passwords, it is unlikely that we will need to generate infinitely long sentences. For finite length sentences, recursion is not required and a Regular Language will suffice. Regular Language 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 here to model the rules of grammar.

We consider each grammatical rule as a sequence of POS tags. We can extract POS tag sequences from any POS-tagged corpus that is representative of a long password dataset. Our approach of generating grammatical rules is similar to expanding the CFG rewrite rules up to a finite length. However, its advantage is that we do not have to maintain complex CFG rewrite rules. Modifying the grammar is as simple as adding or deleting a POS tag sequence from the set.

Let T be the set of all POS tags

$$T = \{\text{Noun, Verb, Adjective}, \dots\} \ .$$

Each word w in dictionary D can have one or more POS tags. For example, *run* can be both a verb and a noun. If the tagger does not recognize a word, it can assign a default tag such as Noun. Each POS tag t in T has an associated dictionary of words

$$D(t) = \{w \in D,\ \text{tag}(w) = t\}\ .$$

For example, $D(\text{Noun}) = \{\text{king}, \text{queen}, \ldots\}$. A tag sequence ts of length n is a sequence of POS tags defined as

$$ts = t_1 \| \ldots \| t_n,\ t_i \in T,\ n \geq 1\ ,$$

For example, ts of length 3 can be "Determiner Adjective Noun", "Determiner Determiner Noun" etc. Not all tag sequences occur in a given corpus. The corpus could be a natural language corpus such as the Brown Corpus, the long password corpus introduced in Table 1, or any passphrase corpus. For a given corpus, we consider all tag sequences that occur in the corpus as grammatical and call them *tag-rules*. We use the notation ts^{grammar} for tag-rules. For example, "Determiner Determiner Noun" is not present in the Brown Corpus so it is not a tag-rule if we consider the Brown Corpus.

A tag sequence $ts = t_1 \| \ldots \| t_n$ of length n generates a set $\mathcal{S}(ts)$ of unique word sequences each of length n,

$$\mathcal{S}(ts) = \{w_1 \| \ldots \| w_n,\ w_i \in D(t_i)\}\ .$$

For example, the tag sequence "Determiner Adjective Noun" generates the set {"the handsome king", "the beautiful queen" , . . . , }.

Let $ts(n)$ denote the set of all tag sequences of length n. For a set T of size 30, $ts(3)$ has $30 \times 30 \times 30$ tag sequences. Let $ws(n)$ denote the set of all unique word sequences of length n generated by $ts(n)$.

$$\begin{aligned} ts(n) &= \{t_1 \| \ldots \| t_n\}, \\ ws(n) &= \{ws \in \mathcal{S}(ts)\ , \forall ts \in ts(n)\}\ . \end{aligned}$$

Note that the count of all unique word sequences up to length n is equal to $\mathcal{G}(\text{word})$ in (1), i.e.,

$$\mathcal{G}(\text{word}) = \mathit{count}\left(\bigcup_{i=1}^{n} ws(i)\right)\ .$$

We consider $ws^{\text{grammar}}(n)$ as the set of all word sequences of length n generated using rules of grammar, i.e., from tag-rules of length n, $ts^{\text{grammar}}(n)$. Henceforth, we refer to a word sequence generated using a tag-rule as a *phrase*. The size of the password search space of phrases is

$$\mathcal{G}(\text{grammar}) = \mathit{count}\left(\bigcup_{i=1}^{n} ws^{\text{grammar}}(i)\right) \tag{3}$$

Since each word can be associated with more than one tag, multiple tag-rules can generate the same phrase. The set union operator discounts repeated phrases.

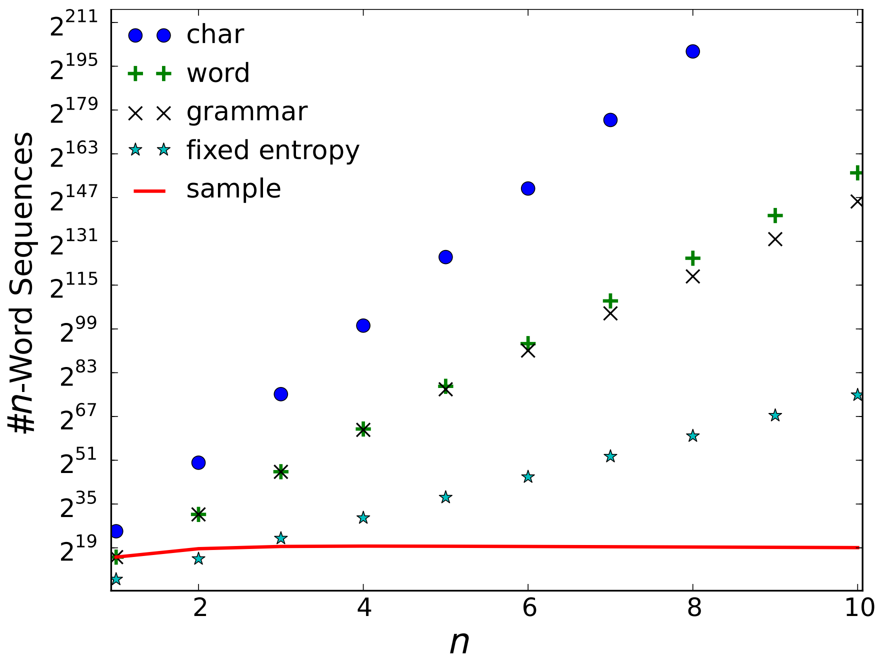

Figure 2: Comparison of the size of password search space treating password as a sequence of characters (*char*), a sequence of words (*word*), and a sequence of words generated using grammatical structures (*grammar*). Numbers are based on the Brown Corpus statistics. Note that *char* is much greater than both *word* and *grammar*. Although not obvious, *word* is significantly larger than *grammar* as the values are in log scale. Table 3 further emphasizes the difference. The gap between *word* and *grammar* widens as the number of words n in the password 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.

3.2 Numerical Evaluation

We need a corpus to numerically evaluate the password search space model proposed in Section 3. Here, we consider the Brown Corpus, a balanced corpus that contains a representative set of 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. For example, we find that 84% of the long passwords from "Simple" category in Table 1 were generated using tag-rules from the Brown Corpus. Hence, using the Brown Corpus should provide useful insights into the effect of structure on the password search space.

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

To evaluate the password search space of word sequences generated using tag-rules, $\mathcal{G}(\text{grammar})$ in (3), we need a set of POS tags, dictionary for each POS tag, and the set of tag-rules. We use the simple Brown 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 follows. Recall from Section 2 that sentences in Brown Corpus are POS tagged. First, we remove punctuation symbols and associated tags from the tagged sentences. Then we create n-gram of word and tag pairs. Finally, we get the set of unique tag-rules by extracting the tags from the word tag pair in each n-gram. In Fig. 3 (Top) we plot the number of tag-rules against all possible tag sequences. We observe that the number of tag-rules is much less than the number of all possible tag sequences and the difference increases with length of the tag-rules. Fig. 3 (Bottom) plots number of word sequences generated by tag-rules and the number of word sequences generated by all tag sequences. We plot the same two curves as *grammar* and *word* in Fig. 2. From Fig. 3 we see that few tag-rules generate a majority of the password search space.

From Fig. 2 we can compare the password search space sizes of *char*, *word*, and *grammar*. We observe that *char* grows exponentially compared to *word* and *grammar*. We see that *grammar* is significantly smaller than *word* (notice the log scale). To emphasize the decrease in password search space due to the presence of grammar, we tabulate the ratio of *grammar* to *word* in Table 3. The decrease can be as large as 50% for a password of length 5 words. The gap between *word* and *grammar* widens as the number of words n in the password increases.

4 Distribution Analysis

When the 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

Table 3: Percent decrease in password search space when passwords are generated using grammatical structures. *word* is password search space of all word sequences and *grammar* is password search space of word sequences generated using grammatical structures, from Fig. 2. n is the number of words in the password. Note the significant decrease in password search space due to the presence of grammatical structures e.g. for $n = 5$ the decrease is more than 50%.

n	1	2	3	4	5
$\frac{grammar}{word}$ %	100	99.92	96.90	80.66	46.95
n	6	7	8	9	10
$\frac{grammar}{word}$ %	17.17	4.28	0.99	0.25	0.07

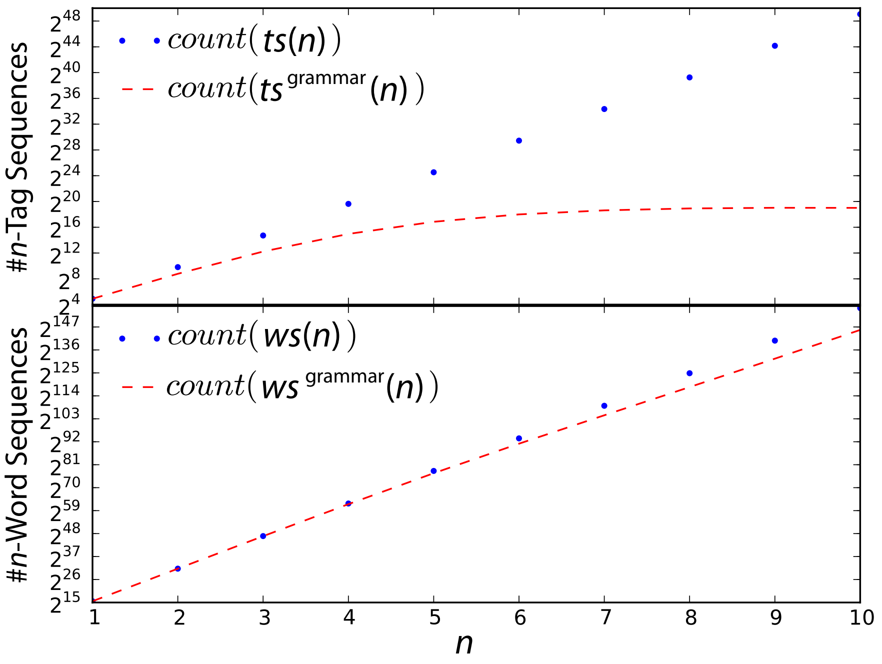

Figure 3: Top: comparison of the number of tag sequences present in the Brown Corpus $count(ts^{\text{grammar}}(n))$ with the number of possible tag sequences $count(ts(n))$. Each sequence contains n tags. We call tag sequences present in a given corpus as tag-rules. Bottom: comparison of the number of unique word sequences generated by tag-rules $count(ws^{\text{grammar}}(n))$ with the number of unique word sequences generated by all possible tag sequences $count(ws(n))$. Each sequence contains n words. The former is the password search space of word sequences generated using grammatical structures and the latter is the password search space of all word sequences. Interestingly, few tag-rules generate a large portion of the password search space.

password search space. So far, research involving analysis of password distributions [19, 18, 34] 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. When guessing effort decreases, an attacker can potentially crack more passwords for a given number of guesses. We estimate this increase in number of passwords cracked using a novel optimization framework.

4.1 Reduction in Guessing Effort due to Structure

An uniform distribution maximizes the guessing effort [31]. Conversely, a non-uniform distribution reduces the guessing effort. Usually, user-chosen password distributions are not uniform [18]. Given a set of user chosen passwords, for example {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, [34] ensures that password values do not repeat often and [18] 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”. An attacker aware of such a distribution can reduce her guessing effort by enumerating values for the structure “Adjective Noun” and ignoring all other structures. Hence, uniqueness of password values is a necessary, but not sufficient condition for ensuring uniformity. We also have to consider the distribution of structures.

Recall from Section 3 that we can compute the size of the search space of each grammatical structure (also referred to as a *tag-rule*). The size of the search space of individual tag-rules vary unevenly e.g. the size of “Noun Noun” is greater than the size of “Adjective Noun”. In Fig. 4 we group the tag-rules by their search space size. We observe that some of the tag-rules have very small search spaces e.g. for tag-rules of length 3 (*3-gram*), 8.9% of the rules have $10 - 19$ bits of strength. The effort required to guess a password generated by a tag-rule is a function of the size of the tag-rule search space. It is paramount to ensure that users do not use a disproportionate number of weak tag rules to generate their passwords.

4.2 Enforcing Uniformity

The conditions required to ensure a uniform distribution over a set of password values, V, generated using tag-rules are (a) password values have to be unique; and (b) each tag-rule should have proportional representation. As we will see in Section 4.3, unless these conditions are satisfied the distribution is not uniform thereby reducing the guessing effort. Intuitively, proportional representation implies that a tag-rule with a larger search space should generate more password values in V. We define the true probability, p_T, of a tag-rule as the size of the search space of a tag-rule divided by the size of the password search space. Each password value in V maps to a tag-rule. Let TR

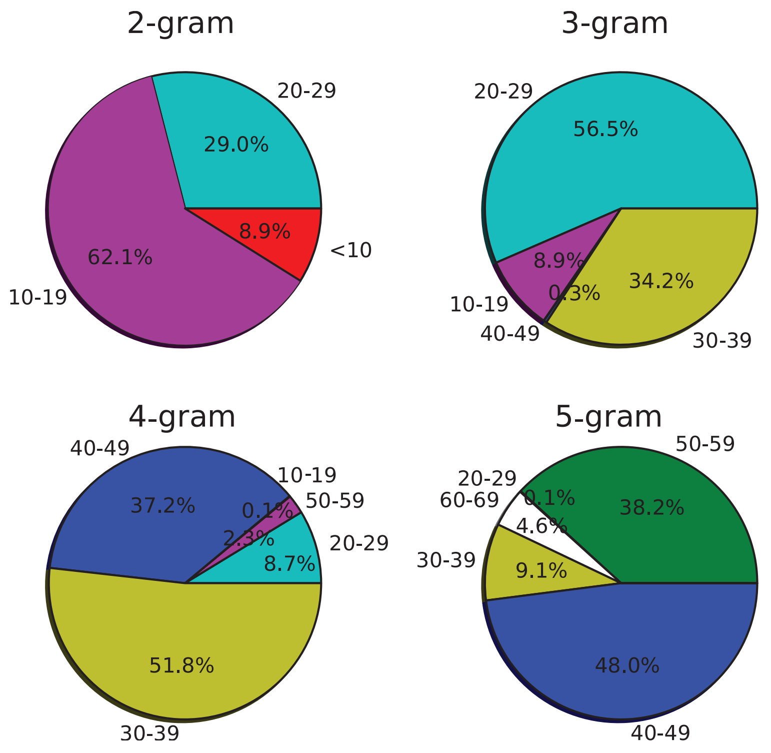

Figure 4: Tag-rules of length 2 to 5 grouped by the size of their search space (in bits). The search space of a tag-rule, ts, is the number of word sequences it generates, which in bits is $\log_2 count(\mathcal{S}(ts))$. Numbers outside the pie chart indicate the range of bits. Numbers inside the pie indicate the percentage of tag-rules with those many bits. Note that the tag-rules divide the password search space unevenly. A significant number of tag-rules have very low strength in bits. For example, for tag-rules of length 3 (*3-gram*), 8.9% of the rules have $10 - 19$ bits of strength.

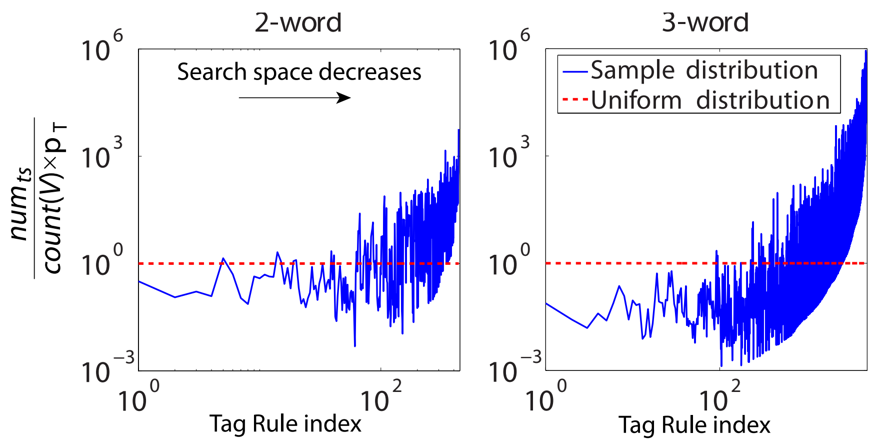

Figure 5: Visualizing disproportional representation of tag-rules using unique 2-word and 3-word phrases from the Brown Corpus. The tag-rules are plotted in descending order of search space size. For each tag-rule, we plot $\frac{num_{ts}}{count(V)\times p_T}$, which should be equal to 1 (dotted red line) if the tag-rule has proportional representation. We can see that in the Brown Corpus samples, weaker tag-rules occur disproportionately more often than stronger tag-rules.

be the set of tag-rules corresponding to the password values in V. A tag-rule, ts, has proportional representation if the number of times it occurs in TR, num_{ts}, is equal to $count(V) \times p_T$.

To visualize disproportional representation of tag-rules, we consider the set of unique 2-word and 3-word sequences in the Brown Corpus. We compute the tag-rule set TR for both. For each tag-rule, ts, in the Brown Corpus we compute its true probability p_T. In Fig. 5, for each tag-rule ts, we plot $\frac{num_{ts}}{count(V)\times p_T}$ for each tag-rule ts, which should be equal to one (dotted red line) if the tag-rule has proportional representation. The tag-rules are indexed in descending order of their search space size.

4.3 Estimating Decrease in Guessing Effort

Let V be a set of unique values. We define $gain$ as the number of values an attacker can guess from V by making G guesses. We can pose the problem of computing $gain$ as an optimization problem,

$$\text{maximize } \textit{gain} = \sum_{i=1}^{count\left(TS^{\text{grammar}}\right)} r_i v_i \tag{4}$$

$$\text{subject to} \sum_{i=1}^{count\left(TS^{\text{grammar}}\right)} r_i g_i \leq G \tag{5}$$

$$0 \leq r_i \leq 1$$

$$TS^{\text{grammar}} = \bigcup_{j=1}^{n} ts^{\text{grammar}}(j)$$

$$WS^{\text{grammar}} = \bigcup_{j=1}^{n} ws^{\text{grammar}}(j)\ .$$

Here, TS^{grammar} is the set of tag-rules of lengths up to n. WS^{grammar} is the set of word sequences generated by TS^{grammar}. For ith tag-rule $ts_i \in TS^{\text{grammar}}$, $g_i = count\left(\mathcal{S}(ts_i)\right)$ is the number of word sequences generated by ts_i, i.e, the search space of ts_i. v_i is the number of values guessed from V using ts_i. r_i is the unknown weight assigned to ts_i and $r_i g_i$ controls the number of values enumerated from the search space of ts_i. For a given V, if the tag-rules have proportional representation, the maximum gain $gain^{\text{uniform}}$ is $(count(V) \times G)/count\left(WS^{\text{grammar}}\right)$ for a given number of guesses G. For same G, when the tag-rules have a skewed representation, the $gain$ of the attacker increases ($gain^{\text{skew}} \geq gain^{\text{uniform}}$). We prove this in Appendix.

5 Password crackers

We investigate whether state-of-the-art cracking tools such as John the Ripper (JTR) [9], Hashcat [6], and Weir Algorithm [38] can crack long passwords efficiently. This is important because crackers are used in auditing user passwords and estimating the strength of password policies [24]. 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 dictionary-based cracker [9, 6, 38] uses a dictionary of values to generate candidate passwords. A dictionary may contain leaked passwords [24], words from many languages [13], common quotes, music lyrics, movie titles [27] etc. Cracker may use the dictionary

Table 4: 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

Table 5: Weir Algorithm base structures for ExSet

Trained on Passwords		Trained on Phrases	
Structure	Prob	Structure	Prob
LLLLLDLLLL	0.6	LSLLLLSDSLLLL	0.6
LLLLDLLLS	0.2	LSLLLSDSLLLS	0.2
LLLLDLLLL	0.2	LSLLLSDSLLLL	0.2

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". Below we explain the main shortcomings of current crackers in cracking long passwords using the example data set, ExSet in Table 4 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 provides a combination mode that can automatically combine any two values from the input dictionary. For more than two values, Hashcat faces same issues as JTR.

Weir Algorithm is another dictionary based technique that improves the cracking efficiency by improving the order in which mangling rules are applied to the values in the dictionary. Weir Algorithm requires a training corpus of passwords from which it creates a set of base structures. A base structure in Weir Algorithm is a sequence of "L", "D", and "S" which stand for "Letter", "Digit", and "Special Symbol". Each base structure is associated with a probability. Weir Algorithm generates the base structures listed in Table 5 when trained on passwords from dataset ExSet. 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 the base structure "LLLLDLLLS", it tries to fit values {have, cats, dogs, fish, goat} from dictionary ExD. It cannot, however, combine shorter values such as "I" and "had" to form a longer value "Ihad". With the base structures in Table 5 Weir Algorithm fails to crack any password from dataset ExSet because “Ihave” and “Ihad” are not in dictionary ExD. Weir Algorithm cannot combine values from the dictionary to form longer values. The user has to manually generate longer values and add them to the dictionary as in the case of JTR and Hashcat.

To force Weir Algorithm to generate longer values, we can train it with passwords containing words separated by a single space. We have to then strip out the spaces from the generated password guesses. Weir Algorithm treats space as a special symbol. In our example, if we train Weir Algorithm on the phrases listed in dataset ExSet, it generates base structures e.g. “LSLLLS-DSLLLL” and is able to crack passwords such as “Ihad3cats” using dictionary ExD. However, note that this is an approximation of generating all possible word combinations from the input dictionary, and is not an optimal approach for targeting long passwords generated using grammatical structures. For example, Weir Algorithm generates password guesses {Ihad1had, Ifish5have, Icats3fish ...} that may not be useful. From Section 3 we know that the search space of all word sequences can be more than 50% larger than the search space of word sequences generated using grammatical structures, and that the gap increases with length (Table 3).

An alternative to heuristic dictionary technique is intelligent brute-force technique. An intelligent brute-force technique such as JTR Incremental mode eventually enumerates the entire password search space. JTR Incremental mode uses a Markov model trained on 3-gram letter frequency distribution to generate password guesses. We observe from our experiments that using letter frequencies is effective for conventional short length passwords that are mostly a sequence of characters, but not for longer passwords consisting of multiple words.

5.2 Evaluation on Long Password Dataset

We evaluate the cracking efficiency of JTR, Hashcat and Weir Algorithm via a set of experiments involving a published long password dataset [24]. We briefly described this long password data set in Section 1. The dataset 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. We are not aware of any datasets (public or otherwise) that only contain user-selected, long passwords with underlying grammatical structures. We manually examine each password in P16 using tools such as the Microsoft Word Breaker [11, 37], 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 all passwords from the “Simple” category. Our dataset contains 178 passwords of which 144 are 2 to 5 words in length. We henceforth refer to this dataset as P16S. The crackers use the dictionaries described in Table 6. The dictionary “L” contains publicly available datasets, the dictionary “GW” contains data from the Google Web Corpus and the dictionary “B” contains data from the Brown Corpus.

We tabulate the experimental results in Table 7. We allow all experiments to make up to $2.5E12$ guesses. We indicate if an experiment terminated before making $2.5E12$ in the “Session Com-

Table 6: 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
GW1-8	6603610	Google Web Corpus 1-gram
GW25-8	3625636435	Google Web Corpus 2-5 grams
B210	5942441	Brown 2-10 gram word sequences

Table 7: Performance of crackers on long passwords. *Experiment* lists the name of the experiment. For JTR experiments, *NM* indicates that mangling rules were not used. *%P16 Cracked* is the percentage of passwords cracked out of 1434 long passwords in P16. *%P16S 2-5 Cracked* is percentage of passwords cracked out of 144 long passwords of length 2 to 5 words in P16S. P16S is a subset of P16 that contains long passwords with underlying grammatical structures. We tested on passwords of length 2 to 5 as Google Web Corpus has n-grams of length up to 5. *Guesses* is the total number of password guesses generated. *Session Completed* indicates if the experiment completed after *Guesses*.

Experiment	%P16 Cracked	%P16S 2-5 Cracked	Total Guesses	Session Completed
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

pleted" column. We manually terminated two experiments before $2.5E12$ guesses due to their excessive memory consumption. For brevity, we omit the less significant experimental results from Table 7. In our experiments, we try to overcome the main shortcoming of current crackers, i.e., they do not generate longer values automatically, and expect the user to add longer values to the dictionary. Specifically, we do the following:

1. *Use a better dictionary of long values*: we use the Google Web Corpus as a dictionary. Experiments on long password datasets until now have not used the Google Web Corpus, but depend on other publicly available datasets similar to dictionary "L". In experiment "JTR GW25-8", using Google Web Corpus we cracked 20.5% of long passwords from dataset P16. For the same number of guesses, published experiments crack 6% [24]. For long passwords with grammatical structures, we crack 27.7% of passwords with Google Web Corpus compared to 6.9% with dictionary "L".

2. *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 exceedingly 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 4 dictionaries: L-8, B210, GW1-8 and GW25-8. We first run JTR without mangling rules, and then run it again with the standard JTR mangling rules. Dictionary values have minimum of 8 characters. Mangling rules can concatenate two values to form 16 character length guesses. JTR experiment using GW25-8 performs better than other experiments by cracking 20.5% of long passwords and 34.7% of long passwords with grammatical structures using 2.48E12 guesses. Even with lower number of guesses it outperforms other experiments.

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

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

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 uses Markov model at a character level. JTR Incremental mode experiment fails to crack any passwords.

From our experiments we see that using a good dictionary of long password values can improve cracking efficiency of long passwords e.g. 20.5% versus 6%. However, relying only on existing sources such as Google Web Corpus 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". 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 in the following section.

5.3 Grammar Aware Cracking

A cracker should ideally emulate user behavior to generate password candidates. We develop a proof-of-concept cracker that generates long passwords using grammatical structures. We use the POS tag framework introduced in Section 3 to automatically combine words into longer password values. The main challenges in our approach are: (1) identify a set of grammatical structures (tag-rules) that users prefer. It is possible to identify such tag-rules from existing corpus e.g. the Brown Corpus or long password datasets; and (2) build a dictionary for each individual POS tag in the tag-rules. The level of difficulty involved 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. On the other hand, open tags such as “Noun” and “Noun Proper” have large dictionaries and also grow with time.

Our main goal is to evaluate the value in pursuing 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 will be able to study specific 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 the CLAWS [22] POS tagger.

Having a set of tag-rules, we proceed to build dictionaries for individual tags. First, we built a dictionary for each tag using words present in the Brown Corpus and a small web text corpus [12]. We included web text corpus as the Brown Corpus was compiled in year 1961 does not contain Internet related words. We used all words from the Brown Corpus and all words that occurred at least

Table 8: Performance of grammar-aware cracker against passwords of length 2-5 words in P16S. *%Cracked with BWeb* is the percentage of passwords cracked using dictionary BWeb and *%Cracked with BWeb90* is the percentage of passwords cracked with dictionary BWeb90. *%Exclusive* is the percentage of 2-5 word phrases cracked by grammar-aware cracker, but not by *JTR GW25-8* or other experiments in Table 7.

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

Table 9: Comparison of passphrase strength. It can be seen that strength is not a direct function of length. Column *Tag-Rule* lists the tag-rule that can generate the passphrase. Column *Guesses* is the based on the size of the search space of the corresponding tag-rule and the effort required to mangle the phrase. Column *Time* is an estimate of the time required for guessing the given password.

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

10 times in the web text corpus (373 words). The words in the Brown Corpus are already tagged, and we identified the tags for the words in the web text corpus using CLAWS. If a word has a noun tag, then we assign it to the noun dictionary and so on. We refer to this first set of tag dictionaries as BWeb. Next, we built an alternate set of tag dictionaries that we refer to as BWeb90. We reduced the size of the dictionary for all POS tags except "Noun", "Proper Noun", "Adjective", and "Cardinal Number". We computed the cumulative probability distribution of word frequencies within each tag dictionary and discarded words that did not meet the 0.9 cumulative probability cutoff. Our intuition for BWeb90 is that users often use few words for closed tags such as "Determiner" and "Conjunction".

The inputs to the grammar aware cracker are a set of tag-rules, a set of individual tag dictionaries, and the maximum number of guesses it can make. The cracker computes the size of each tag-rule (refer to Section 3 for details), sorts the tag-rules by their size, and selects subset of smallest tag-rules whose sizes add up to the number of guesses. The cracker then generates password candidates using the subset of tag-rules and the tag dictionaries.

In Table 8, we tabulate the performance of our grammar-aware cracker. The columns "%Cracked with BWeb" and "%Cracked with BWeb90" list the percentage of 2-5 word passwords from P16S cracked using the dictionaries BWeb and BWeb90 respectively. The "Exclusive" column lists the percentage of the total dataset that was exclusively cracked by grammar-aware cracker, but not by other crackers. We compare the perfromance of grammar-aware cracker with the perfromance of other crackers on P16S2-5 in Table 7. For 5E10 guesses, grammar-aware cracker cracked 18.7% of passwords, which is better than "Weir Algorithm" (4.8%) and "JTR Incremental" (0%) experiments. Although not listed in Table 8, 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 P16S2-5 dataset. This 10% was not cracked by "JTR GW25-8" or other experiments. Further, grammar-aware cracker consumes $<$ 10MB of storage compared to $>$ 50GB for JTR experiment using Google Web Corpus. It is highly scalable as number of guesses increases, and easier for operations such as network transfer and dictionary sorting. It also provides flexibility in targeting different user groups e.g. we can use different "Proper Noun" dictionaries for users from North America versus Asia. We feel that the initial results look promising and that using a grammar-aware cracker can indeed improve the 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 so they can implement secure password policies. We consider the example of passphrase policies. We reexamine 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 [10, 2, 3, 7] and estimate security metrics such as size of search space and guessing effort accordingly [8, 17]. 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 Relationship between Passphrase Strength and Length

Consider the passphrase examples in Table 9. The examples, “Th3r3 can only b3 #1!”, “Superman is $uper str0ng!” and “My passw0rd is $uper str0ng!” are from technology and academic websites [5, 15, 3]. The example, “Hammered asinine requirements.” is a synthetic example based on a recommendation to use nonsensical phrase [30]. The tag-rule column lists one of the grammatical structures that can generate the corresponding passphrase. The number of guesses lists the total number of passphrases the corresponding tag-rule can generate, that is the size of the tag-rule search space. The guessing estimates are based on the grammar aware cracking approach from Section 5. The time column lists the total time required to guess all the passphrases generated by the tag-rule, and are based on a guessing rate of 1 billion guesses per second. This rate is realistic if we consider that current state of the art GPU accelerated machines can achieve up to 33 billion comparisons per second and can be built with less than USD 3000 [16]. Looking at the guessing effort and time estimates, it is clear 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. The examples, “Th3r3 can only be #1!” and “My passw0rd is $uper str0ng!” vary in strength by three orders of magnitude, but satisfy the composition requirements of the same passphrase policy [3, 4]. Underlying structures, and not just the number of characters or words determine the overall strength of a passphrase.

6.2 User Behavior and Passphrase Policy

While studying the long password data set [24], 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. Consider the word and character statistics from the P16S long password dataset in

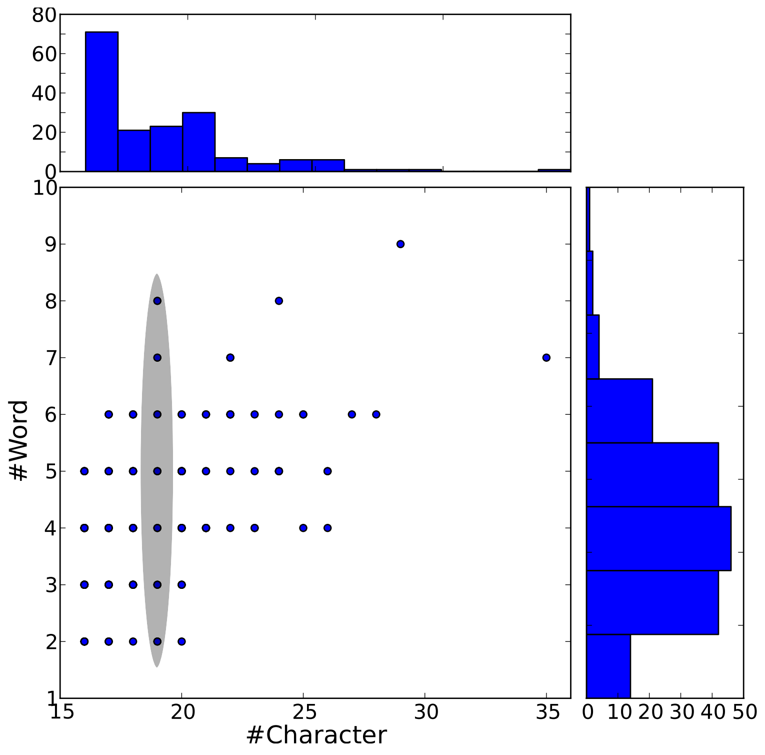

Figure 6: Word and character statistics for long passwords in dataset P16S. We also plot the histograms for number of characters (Top) and number of words (Side). Note how total number of characters in the passwords tends to remain the same as the number of words increases (shaded oval). Users seem to meet the policy requirement of minimum 16 characters with fewer long words or more short words. The passwords “compromisedemail” and “thosedamnhackers” both have 16 characters, but 2 and 3 words respectively.

Fig. 6. The shaded oval in the figure highlights the passwords with 19 characters (x-axis), and between two to eight words (y-axis). We can observe similar behavior for passwords with 16, 17, etc. characters. For example, the passwords “compromisedemail” and “thosedamnhackers” contain 16 characters, but two and three words respectively. From our results, 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. Interestingly, we found some explanation for this user behavior in the field of cognitive psychology. As Jahnke and Nowaczyk explain regarding experiments on word-length and short-term memory [23, 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.” Passphrase policies such as “choose a password that contains at least 15 characters and at least four words with spaces between the words” [7] may unwittingly allow weaker passphrases unless they consider user behavior and effect of structure.

6.3 Passphrase Entropy

Some passphrase security evaluations [32] use the concept of entropy of English language [35]. Informally, entropy measures how much the values emitted by a source can be compressed. Higher compression is achieved when values repeat more often. Entropy can be a useful measure to estimate search space. However, entropy estimations have to be based on representative probability distributions of the source emitting the values. It is incorrect to use estimation derived for one source for another without verifying if the two sources have similar probability distributions. To illustrate this point, in Fig. 2, we plot the search space estimation using fixed entropy estimation equation $2^{1.75 \times 4.26 \times i}$ where i is the number of words in the password. We use the estimate of 1.75 bits per character for printed English [21], and 4.26 average word length statistics from the Brown Corpus 2. In [21], a 3-word gram language model was trained on large amounts of printed English sources, and was tested on the Brown Corpus. We observe from Fig. 2 that the estimation for 3-word phrases closely matches the true number of unique 3-word phrases in Brown Corpus. However, for other lengths there is varying degrees of inaccuracy. This is because of the difference in probability distribution used by the estimation model and the true probability distribution of n-word phrases. To our knowledge there is no study of probability distribution of user chosen passphrases. Passphrases may also have different probability distributions for different user groups and policies. Hence using single entropy estimation derived from printed English may be incorrect.

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.

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. http://www.cmu.edu/iso/governance/guidelines/password-management.html, 2012.

[4] Carnegie mellon university passphrase policy faq. http://www.cmu.edu/computing/doc/accounts/passwords/faq.html#8, 2012.

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

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

[7] Indiana university passphrase policy. http://kb.iu.edu/data/acpu.html#atiu, 2012.

[8] Indiana university passphrase policy strength estimation. http://protect.iu.edu/cybersecurity/safeonline/passphrases, 2012.

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

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

[11] Microsoft word breaker for web. http://research.microsoft.com/en-us/um/people/kuansanw/wordbreaker/, 2012.

[12] NLTK web text corpus. http://nltk.googlecode.com/svn/trunk/nltk_data/index.xml, 2012.

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

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

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

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

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

[18] Joseph Bonneau. The science of guessing: analyzing an anonymized corpus of 70 million passwords. In *2012 IEEE Symposium on Security and Privacy*, May 2012.

[19] Joseph Bonneau and Ekaterina Shutova. Linguistic properties of multi-word passphrases. In *USEC '12: Workshop on Usable Security*, March 2012.

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

[21] Peter F. Brown, Vincent J. Della Pietra, Robert L. Mercer, Stephen A. Della Pietra, and Jennifer C. Lai. An estimate of an upper bound for the entropy of english. *Comput. Linguist.*, 18(1):31–40, March 1992.

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

[23] John C. Jahnke and Ronald H. Nowaczyk. *Cognition.* Prentice-Hall, Inc., Upper Saddle River, New Jersey, USA, 1998.

[24] Patrick Gage Kelley, Saranga Komanduri, Michelle L. Mazurek, Richard Shay, Tim Vidas, Lujo Bauer, Nicolas Christin, Lorrie Faith Cranor, and Julio Lopez. Guess again (and again and again): Measuring password strength by simulating password-cracking algorithms. In *Proceedings of the 2012 IEEE Symposium on Security and Privacy*, May 2012.

[25] Saranga Komanduri, Richard Shay, Patrick Gage Kelley, Michelle L. Mazurek, Lujo Bauer, Nicolas Christin, Lorrie Faith Cranor, and Serge Egelman. Of passwords and people: measuring the effect of password-composition policies. In *Proceedings of the 2011 annual conference on Human factors in computing systems*, CHI '11, pages 2595–2604, New York, NY, USA, 2011. ACM.

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

[27] Cynthia Kuo, Sasha Romanosky, and Lorrie Faith Cranor. Human selection of mnemonic phrase-based passwords. In *Proceedings of the second symposium on Usable privacy and security*, SOUPS, pages 67–78, New York, NY, USA, 2006. ACM.

[28] Christopher D. Manning and Hinrich Schütze. *Foundations of Statistical Natural Language Processing.* MIT Press, Cambridge, MA, USA, 1999.

[29] Arvind Narayanan and Vitaly Shmatikov. Fast dictionary attacks on passwords using time-space tradeoff. In *Proceedings of the 12th ACM conference on Computer and communications security*, CCS, pages 364–372, New York, NY, USA, 2005. ACM.

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

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

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

[33] Ashwini Rao, Birendra Jha, and Gananand Kini. Effect of Grammar on Security of Long Passwords. In *Proceedings of the Third ACM Conference on Data and Application Security and Privacy*, CODASPY'13, 2013.

[34] Stuart Schechter, Cormac Herley, and Michael Mitzenmacher. Popularity is everything: A new approach to protecting passwords from statistical-guessing attacks. In *Proceedings of the 5th USENIX conference on Hot topics in security*, HotSec, pages 1–8, Berkeley, CA, USA, 2010. USENIX Association.

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

[36] Richard Shay, Saranga Komanduri, Patrick Gage Kelley, Pedro Giovanni Leon, Michelle L. Mazurek, Lujo Bauer, Nicolas Christin, and Lorrie Faith Cranor. Encountering stronger password requirements: User attitudes and behaviors. In *Proceedings of the Sixth Symposium on Usable Privacy and Security*, SOUPS, pages 2:1–2:20, New York, NY, USA, 2010. ACM.

[37] Kuansan Wang, Christopher Thrasher, and Bo-June Paul Hsu. Web scale nlp: a case study on url word breaking. In *Proceedings of the 20th international conference on World wide web*, WWW '11, pages 357–366, New York, NY, USA, 2011. ACM.

[38] Matt Weir, Sudhir Aggarwal, Breno de Medeiros, and Bill Glodek. Password cracking using probabilistic context-free grammars. In *Proceedings of the 2009 30th IEEE Symposium on Security and Privacy*, SP, pages 391–405, Washington, DC, USA, 2009. IEEE Computer Society.

A Increase in gain of an attacker

For a representative uniform sample set, the maximum gain of an attacker depends only on the number of guesses to be made, G, and the ratio of the search space size to the sample size. From (4),

$$\begin{aligned}
&\sum_{i=1}^{count\left(TS^{\text{grammar}}\right)} r_i v_i \\
&= \frac{count(V)}{count\left(WS^{\text{grammar}}\right)} \sum_{i=1}^{count\left(TS^{\text{grammar}}\right)} r_i g_i && (6)\\
&= \frac{count(V)}{count\left(WS^{\text{grammar}}\right)} G, && (7)
\end{aligned}$$

where the set of weights, $R^U = \{r_i\}$, is not unique i.e. more than one choice of $\{r_i\}$ can yield the same maximum gain. Hence, without loss of generality, we can choose a rule set R^U of length $k \leq count\left(TS^{\text{grammar}}\right)$ such that for $r_i \in R^U$,

$$\begin{aligned}
\sum_{i=1}^{k} r_i g_i &= G && (8)\\
r_i &= 1,\ 1 \leq i \leq k-1, && (9)\\
0 < r_i &\leq 1,\ i = k && (10)
\end{aligned}$$

Let v_i^U and v_i^S denote the values in the representative uniform distribution and the skewed sample distribution. The following steps prove that $gain^{\text{skew}} \geq gain^{\text{uniform}}$.

1. If $v_i^S \geq v_i^U, \forall i$ then $r_i v_i^S \geq r_i v_i^U, \forall i$. Hence, $gain^{\text{skew}} \geq gain^{\text{uniform}}$. Proved. Else go to next step.

2. Find a new rule set $R^p = \{r_i\}, 1 \leq i \leq k'$ such that $\sum_{i=1}^{k'} r_i g_i = G$ and $v_i^S \geq v_i^U, \forall i$. If found, $gain^{\text{skew}} \geq gain^{\text{uniform}}$. Proved. Else R^p only contains rules with $v_i^S \geq v_i^U$ and the guessing constraint is not satisfied. Go to next step.

3. At this point, we could not find a rule set where for each rule we see an increase in gain compared to the uniform distribution. So, in our rule set we must include some rules for which $v_i^S < v_i^U$. We will show that the apparent loss in gain due to this will be more than balanced by the increase in gain due to other rules for which $v_i^S > v_i^U$.

   There exists a set $R' \subseteq TS^{\text{grammar}}$ where $R' \supset R^p$. For a uniform distribution, R' is an instance of R^U satisfying the guessing constraint. With a new skewed distribution, we know that for each rule of R^p in R', $v_i^S \geq v_i^U$. For remaining rules in R', $v_i^S < v_i^U$. For these remaining rules, $r_i = 1$ for all but one element and $0 < r_i \leq 1$ for the last element. The decrease in gain due to the decrease in v_j^S values in $R' - R^p$ are compensated by the increase in v_i^S of rules contained in R^p. $r_i = 1$ for all rules in R_p and all except one rule in $R' - R^p$. For each rule j with $r_j = 1$ in $R' - R^p$, the decrease in v_j^S is compensated by an increase in v_i^S for some rule $i \in R^p$. The last rule in $R' - R^p$ for which $r_j < 1$, the decrease in v_j^S is compensated by v_i^S of some rule i in R^p with corresponding $r_i = 1$. Hence, the overall gain $gain^{\text{skew}}$ of R' is greater than or equal to $gain^{\text{uniform}}$. Recall that the rules in set R' already satisfies the guessing constraint. Proved.